Towards High-Interaction Virtual ICS Honeypots-in-a-Box

Daniele Antonioli
daniele_antonioli@sutd.edu.sg

Anand Agrawal
agrawal_anand@sutd.edu.sg

Nils Ole Tippenhauer
nils_tippenhauer@sutd.edu.sg

Singapore University of Technology and Design (SUTD)
8 Somapah Road
487372 Singapore

ABSTRACT

In this work, we address the problem of designing and implementing honeypots for Industrial Control Systems (ICS). Honeypots are vulnerable systems that are set up with the intent to be probed and compromised by attackers. Analysis of those attacks then allows the defender to learn about novel attacks and general strategy of the attacker. Honeypots for ICS systems need to satisfy both traditional ICT requirements, such as cost and maintainability, and more specific ICS requirements, such as time and determinism.

We propose the design of a virtual, high-interaction and server-based ICS honeypot to satisfy the requirements, and the deployment of a realistic, cost-effective, and maintainable ICS honeypot. An attacker model is introduced to complete the problem statement and requirements.

Based on our design and the MiniCPS framework, we implemented a honeypot mimicking a water treatment testbed. To the best of our knowledge, the presented honeypot implementation is the first academic work targeting Ethernet/IP based ICS honeypots, the first ICS virtual honeypot that is high-interactive without the use of full virtualization technologies (such as a network of virtual machines), and the first ICS honeypot that can be managed with a Software-Defined Network (SDN) controller.

Keywords

Honeypots; cyber-physical systems; industrial control systems; security;

1. INTRODUCTION

Industrial Control System (ICS) security is a promising research topic, because it combines traditional cyber-security threats with control system security ones [32, 33]. Attacks targeting ICS, such as the sophisticated stuxnet worm, are becoming more frequent, and can have serious consequences (e.g., economical damages, environmental catastrophes and loss of human lives [8, 28]).

It is fundamental to harden the security of an ICS, especially in this decade where often ICS devices are facing the Internet on a public IP. Typical defense mechanisms include the use of industrial firewalls to segment the network architecture, and Intrusion Detection Systems to monitor the network traffic, and react in case of suspicious activity.

CPS-SPC '16, October 28, 2016, Vienna, Austria.
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4568-2/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2994487.2994493

In security research, *honeypots* are vulnerable systems that are set up by defenders with the intent to be probed and compromised by attackers. Monitoring systems will then record traces of the attacks and actions taken. In that context, honeypots are able to provide detailed information about the attacker's activities, and to defend-against, or slow-down, the ongoing attack. Honeypots are extensively used in traditional ICT systems, but they are rarely deployed in the ICS domain, mainly because of the very high associated costs, and maintenance's complexity. So far, little academic work has been done in the domain of ICS honeypot design and implementation.

In this work, we propose a design for realistic virtual ICS honeypots. Our design addresses the main challenges for ICS honeypots related to ICT and ICS requirements (e.g., time, determinism, and operating cost). We present an attacker model for the ICS honeypot that captures the goals, the skills, the resources, and the entry points of the attacker. According to the requirements of the attacker model we then propose our architecture design. We classify the presented ICS honeypot as server-based, and high-interaction honeypot (to satisfy the realism constraints), and virtual (to satisfy the cost and maintainability constraints).

We then present an implementation based on our ICS honeypot design. The implementation leverages the MiniCPS framework, which combines lightweight virtual network emulation with physical process, and ICS devices simulation, to help researchers simulating Cyber-Physical Systems. To the best of our knowledge, the presented honeypot implementation is the first academic work targeting Ethernet/IP based ICS, the first ICS virtual honeypot that is high-interactive without the use of full virtualization technologies, such as a network of virtual machines, and the first ICS honeypot that can be managed with a Software-Defined Network controller.

To show the effectiveness of the implemented honeypot, the paper presents the evaluation of a honeypot that is mimicking a water treatment testbed. The attacks on that system were conducted in the context of a cyber-security Capture-The-Flag event. The evaluation confirms that honeypots are a potential solution also in the ICS domain and that they can be integrated in an ICS defense-in-depth scheme.

Honeypot development is a broad topic, and the paper is focusing on the honeypot's core functionalities such as: the network, the physical process, the physical devices and the data retrieval. In particular, the paper is not focusing on the data post-processing part (e. g., no data analytics).

The rest of the paper is organized as follows: in Section 2, we introduce ICS networks, ICS honeypots, and the MiniCPS framework. The honeypot's requirements, attacker model, and proposed design are presented in Section 3, and the honeypot's implementation core components, and additional benefits are presented in Section 4. In Section 5, we present the evaluation of the implemented system. Related work is summarized in Section 6. We conclude the paper in Section 7.

2. BACKGROUND

We now briefly summarize ICS networks, and ICS honeypots Then, we introduce the MiniCPS framework.

2.1 ICS Networks

Industrial Control Systems (ICS) are used to supervise and control systems such as critical infrastructure (electric power, and water), and public transportation systems (trains, and planes). In this work, we assume the system consists of supervisory components, such as human-machine interfaces and servers, programmable logic controllers, sensors, and actuators. All those components are interconnected through a network with a specific topology. We provide the network topology of a generic ICS network as an example in Figure 1.

Programmable logic controllers. PLCs are the core controllers of an ICS. Each device runs a program, also known as control logic, that is able to perform many tasks such as: reading values from a sensor, requesting specific values from other PLCs, and driving an actuator. If the ICS can be divided into stages, then each PLC typically controls one of these stages.

Sensors and actuators. Those components interface with the physical process, and they are either directly connected, or indirectly connected, via remote input/output units (RIOs) or PLCs, to the network.

Network Devices. An ICS uses different types of network devices. Industrial switches and firewalls are deployed to segment the network into layers (e. g., DMZ, and control network). Gateway devices are used to translate one protocol into another (e. g., Modbus into Modbus/TCP). Remote Terminal Units (RTUs) are deployed on the filed to collect and send data back to the SCADA system. We also note that industrial Ethernet switches are often focused on electrical reliability, rather than IP-layer functionality (e.g. the Rockwell Automation Stratix 5900 switch).

Network Topology. Traditionally, ICS follows standard like RS-232 to connect together different components. Additionally, alternative field bus schemes, such as RS-485 and PROFIBUS, have been used. In specific situations where reliability is a major concern, ring [5] topologies are widely used in practise due to ease of deployment, and single-point-of-failure tolerance with very low reaction time.

Industrial protocols. In recent years, ICS networks are transitioning to traditional ICT technology like Ethernet (IEEE 802.3), and TCP/IP. However, the need for reliability, and interoperability with existing equipment led to the development of customized industrial protocols, such as Ethernet rings, and Ethernet/IP (ENIP) [21]. We now discuss the main ENIP features as an illustrative example of an industrial protocol. ENIP is a Real Time Ethernet (RTE) field bus based on TCP/IP, with a custom application layer designed to meet typical industrial communications requirements, like time constraints, and packet determinism. Technically, ENIP is an Ethernet based implementation of the Common Industrial Protocol, and it is defined in the IEC 61784-1 standard [10]. A PLC uses CIP messages to obtain sensor readings, to query another component, to set configuration options, to update its firmware, and even

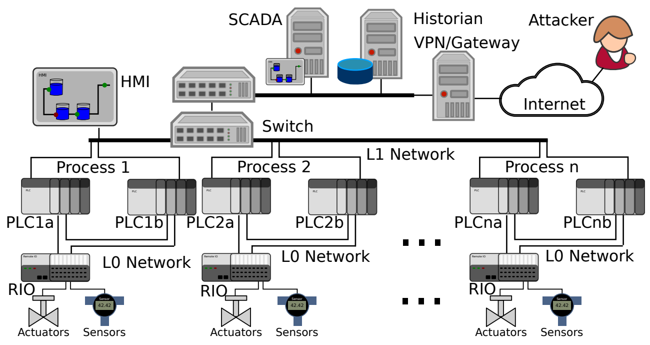

Figure 1: Example local network topology of a plant control network.

download a new control logic. In that model, sensor readings and control values are represented by tags, that are like variable names in a programming languages. CIP uses a request-response model, and such requests can operate on tags, and on the meta-data associated with the tags.

Topology layers. ICS networks typically are layered in different zones (more details in [20, 24]). On the lowest layer, controller devices are connected to sensors and actuators, or to remote input/output (RIO) devices, capable of converting raw sensors and actuators signals into Ethernet-based packets. The next layer will connect together the controller devices, and the additional devices such as: Human-Machine-Interface (HMI), engineering workstation, and historian server. For simplicity, all these devices are often kept in the same IP-layer sub-network, although more complex topologies are possible.

2.2 ICS Honeypots

A honeypot is a system *intended* to be probed, attacked and compromised [29]. Historically, the idea behind the development of modern honeypots comes from the nineties, where skilled computer programmers and system administrators were playing in real time with the attackers, to gain information about their targets, techniques, exploited vulnerabilities and ultimately for fun [4, 31].

Honeypots can be classified by means of different orthogonal features. *Real* honeypots use real physical devices to replicate the target system. They are the most realistic solution, however in the ICS domain their deployment is too expensive in terms of money, maintenance and space. On the other hand, *virtual* honeypots utilize virtualization technologies, such as PLC emulators, to reproduce a system, and they offer a compact, and low-cost solution. Hybrid honeypots include a mixture of virtual and real devices, and they might be an interesting cost-effective solution for the ICS domain.

One of the core aspect of an honeypot is its level of realism. *Low-interaction* honeypots simulate only specific systems services (such as a telnet daemon), providing a narrow attack surface. Indeed they are easy to develop, configure, and secure against the attacker. However, the effectiveness of a low-interaction honeypot in the ICS domain is questionable, mainly because the ultimate targets of an ICS attack are the physical process and the ICS devices, and these parts are not simulated by low-interaction honeypots. In contrast, *high-interaction* honeypots use real services running on real Operating Systems, such as a webserver running on Linux listening to port 80, or simulate the services and the relevant parts of an Operating System. High-interaction honeypots provides a realistic environment for the attacker, a large attack surface, and they are tricky to implement, and secure against a motivated attacker. To the best of our knowledge, in the context of ICS honeypot there is no standard definition regarding high-interaction honeypots. The paper defines an high-interaction ICS honeypot as an honeypot able to simulate both the physical process, and the ICS devices' control logic, and to emulate the ICS network using industrial protocol stacks, and network topologies.

Honeypots have different roles with respect to the attacker. *Server-based* honeypots expose, over an insecure channel, a number of vulnerable services, that are passively listen to well-known ports. In simple words, a server-based honeypot is passively waiting to be attacked. In contrast, a *client-based* honeypot acts as vulnerable client application, such as a web browser, and it actively looks for an attack from a malicious webserver.

Honeypots are used in different contexts. *Research* honeypots implement well-known vulnerabilities to lure attackers, and to study their behaviours, or (less often) for educational and security training purposes. *Production* honeypots are supposed to be more secure, and they are deployed to defend a system against an attacker. In the best case, the production honeypot will prevent the attacker to sabotage the real system, in the average case it will slow-down the attack, and hopefully will increase the attacker frustration, and in the worst case it will help the attacker to complete his job.

It is important to emphasize that ICS honeypots present additional requirements compared to traditional ones, most importantly *time* and *determinism* constraints. An ICS device has to complete a sequence of tasks within a critical time interval, and in a precise order, that's way it uses a Real Time Operating System with a deterministic scheduler. In the same manner, ICS packets are sent over the network with a specific order, and they had to reach their destinations within a time period, that's why industrial protocols, such as Ethernet/IP, are extended with special application layer features to address these requirements. An ICS honeypot has to take into account these factors with great care, otherwise the attacker can easily detect the honeypot with simple tests.

There are many academic and industrial projects involving honeypots. In the domain of traditional network security we have *honeynets*, that are networked honeypots able to communicate among themselves in a NIDS fashion [11]. In the context of ICS and SCADA the most well known (still active) project is Conpot [34], an open-source ICS/SCADA honeypot, that is part of a large-scale project called The Honeynet Project [30].

2.3 MiniCPS Framework

MiniCPS [2] is a toolkit for security research on Industrial Control System (ICS) security. It builds on top of a lightweight Linux network emulator called Mininet [15], and it extends its application to the ICS domain.

MiniCPS combines network emulation, physical process simulation, and ICS devices simulation to build a real-time, ICS simulation in-a-Box. It is a framework written in Python, and it provides an high-level, object-oriented, public API. MiniCPS is developed using modern and agile techniques, like distributed source version control, test-driven development, build and documentation automation, and it is free and open source (MIT license) [1].

In this work, we propose to extend MiniCPS in the context of ICS honeypots. Those honeypots traditionally lack a realistic network sub-systems or focus on the simulation of a single ICS device, like a PLC. With the help of MiniCPS, we can reproduce the exact ICS network topology with PLCs, HMI, middle boxes, etc.. We can use the same network configuration as the real ICS, to take care of the attacker host enumeration, and fingerprinting phase (e. g., same IP, MAC, and net masks). From the emulated network the attacker may discover real ICS services, listening to standard ports, such as ssh or VPN servers. Furthermore, MiniCPS supports link shaping, meaning that each host can be configured with a custom link bandwidth, packet loss rate, and time latency.

MiniCPS's public API can reduce the honeypot's development time, and increase the portability of the developed code across different simulation experiments, involving different physical processes and industrial protocols. The public API is built against four core methods: `set`, `get`, `send` and `receive`. Each device in the simulated ICS inherits (a subset of) these methods according to its functionality. For example, a PLC is able to `get` (read) a sensor value, and `set` (write) an actuator command, additionally a PLC can `send` (serve) a packet over the wire, or `receive` (request) a packet from the wire.

3. HIGH-INTERACTION, VIRTUAL ICS HONEYPOT DESIGN

3.1 Problem Statement

In this work, we address the problem of designing an ICS honeypot with the following requirements:

- Realistic, with multiple services supported.
- Low cost, in terms of hardware, software, and deployment time.
- Reconfigurable, to allow extensibility, scalability, and secure maintenance.
- Targeting the ICS domain, dealing with physical processes, physical devices, industrial protocols, time, and determinism constraints.
- Usable both for research, and production.

To the best of our knowledge, there exists no related work that presents a solution able to satisfy the outlined requirements. Given the ICS honeypot classification criteria, and related tradeoff presented in Section 2, we are proposing the design of a *virtual*, *high-interaction*, *server-based* ICS honeypot, to solve our problem. In the remaining part of the section we will present the reference attacker model, and the related system architecture.

3.2 Attacker Model

In this section we present the reference attacker model. We assume that attacker reaches the honeypot over the Internet, e. g., finding an honeypot's Internet facing device using general purpose search engines, such as Google, or more targeted ones such as Shodan [19]. Once connected to the ICS internal network, the attacker is able to fingerprint the target ICS system, using tools such as `nmap`, and `xprobe2`. As result, the attacker is able to obtain basic system information, such as the number of devices, their addresses, their ports status, and the type of industrial protocol.

We assume that the attacker has a basic knowledge about not-well documented industrial protocols, such as Ethernet/IP, and extensive knowledge about well documented ones, such as Modbus, and DNP3. The attacker may also be familiar with the underlying physical process, and control logic.

We assume that the attacker connects to the honeypot only through the intentionally vulnerable interfaces, and that the provided interfaces define the initial attacker surface. For example, if the attacker connects to a VPN server, using default credentials, then he can only interact over the network. Nothing prevents the attacker from escalating his privileges within the honeypot, for example once connected to the internal network, the attacker may discover an intentionally vulnerable gateway device, and get a root shell on that device.

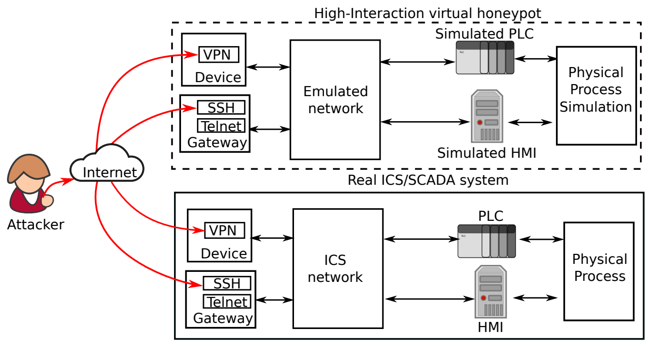

Figure 2: High-Interaction Virtual ICS Honeypot vs. Real ICS Architecture.

We assume that the attacker interacts with the physical processes, and the ICS devices both as a fair, and malicious device. For example, the attacker as a malicious device may send malformed packet, unauthorized commands, perform Man-in-the-Middle attacks (both passive and active), and try to Denial-of-Service the honeypot.

We are limiting the attacker model to what we think is a reasonable scenario. We understand that there are more powerful attacker models, such as the ones profiling a disgruntled employee or an insider threat, however we reserve the option to extend the presented honeypot design, and implementation to deal with these kind of attacks in future work.

3.3 System Architecture

The main contribution of the paper is the design, implementation, and evaluation of a realistic, low-cost, and reconfigurable honeypot targeted to ICS. In this section we present our design points according to the requirements presented in Section 3.1, and the attacker model presented in Section 3.2.

We classify the presented honeypot as follows:

- Virtual (lightweight virtualization).
- High-interaction.
- Server-based.
- Targets ICS requirements.
- Research and Production usage.

The use of virtualization allows us to implement a low-cost solution, that is easy to configure, reproduce, deploy, and maintain. High interaction is crucial to support multiple services and to keep the attacker busy as long as possible. Our honeypot is server-based, because it has to expose realistic services, listening on standard ports that are accessible from outside the ICS internal network perimeter. We envision that our honeypot could be used both in research and production environments. Researchers could use the honeypot system to learn about novel threats in the wild, while plant operators could use the honeypot system to detect specific threats targeted to their system, or mitigate ongoing attacks.

Figure 2 shows an high-level comparison between the presented honeypot architecture and a real ICS architecture. As dictated by the attacker model, our attacker comes over the Internet, and he can access the fake ICS internal network using two different vulnerable interfaces that he may discover during the attack reconnaissance phase. The first interface will give the attacker access to the honeypot over the network, in the figure we are using a vulnerable VPN server as an example. The second interface will give the attacker a command line interface on an ICS device connected to the internal network, in Figure 2 we are using a vulnerable gateway device as an example.

An emulated network enables us to reproduce the same network topology as the real ICS, with the same number of hosts, addresses, and link characteristics. The emulated hosts send packet over the virtual network using real protocol stacks (e. g., Ethernet/IP or ARP). A set of simulated devices reproduces the control logic of the ICS system, and a physical process simulation mimics the real physical process. With those modular settings, we can separate the individual device control logics, and the physical process simulation in different sub-systems, allowing to reuse the blocks according to the honeypot initial configuration.

The ICS block is represented with dashed lines, to underline the fact that the proposed design *physically* separate the honeypot network from the real ICS one. In contrast, traditional honeypots are deployed inside the internal network, using unallocated IP addresses, and their separation from the real system is logical, typically by means of a firewall, a router or an ARP proxy. The physical separation between the honeypot and the real ICS provides an additional layer of security for free, meaning that an attacker who gained (privileged) access to the honeypot is not connected to the real ICS network, but to a virtualized emulated replica.

We explicitly focus on the design, and implementation of the core honeypot functionalities such as physical layer and network layer interaction, or data collection. In particular, we are not focusing on the data post-processing and analysis, and we refer to existing frameworks such as [23].

3.4 Qualitative Metrics

We stated that *realism* is one of our goals. Our assumption is that realism is required to attract interesting attackers. In particular, more knowledgeable attackers might recognize that they are interacting with a honeypot more quickly if the realism of the honeypot is lower. In practise, it can be hard to measure realism in a quantitative way. In the following, we propose some qualitative metrics to determine to which degree our honeypot represents a realistic system faithfully. To the best of our knowledge, there do not exist common metrics for such an evaluation so far.

We will use the following metrics later in the evaluation section to specify a summary of our honeypot prototype capabilities (e. g., honeypot provides feature 1, and does not provide feature 2). The following metrics are complementary to the proposed attacker model, and to the set of already presented requirements. We decided to divide the metrics into two categories: *network* and *physical*, and each category into sub-categories.

Our network metrics:

- Network Parameters: IP, MAC and netmask addresses are identical to a real systems.
- Link Shaping: average packet loss, delay, and bandwidth can be set to comparable values as found in real systems.
- Infrastructure: The network topology is matching the real system exactly.
- Protocol: communications between devices in the honeypot use standard-compliant implementations of industrial, and common protocols (e. g., ARP, HTTP, DHCP).
- Advanced traffic properties: perfect sequence of messages and delay, matching a real system identically.

Our physical metrics:

- Process: The physical process simulation uses a realistic mathematical model of the process (with time steps < 1 minute).
- Devices: The honeypot provides real time simulation of sensor readings, actuator driving, and control logic.
- Human operator: the honeynet allows to simulate interactions of human operations.

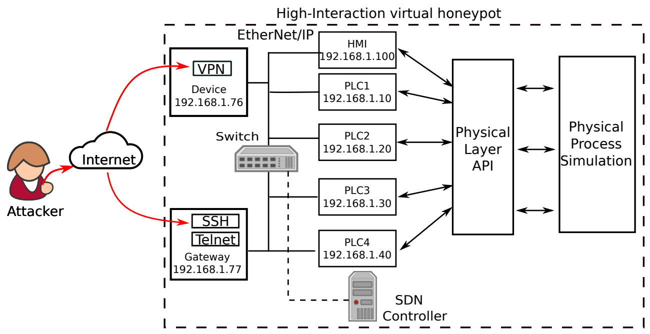

Figure 3: Honeypot Implementation Block Scheme.

- Advanced process: Simulation fast state change (e. g., transient, time steps < 1 s).

4. HONEYPOT IMPLEMENTATION WITH MINICPS

Our honeypot implementation is based on the MiniCPS framework described in Section 2.3. To the best of our knowledge, the presented honeypot implementation is the first academic work targeting Ethernet/IP based ICS, the first ICS virtual honeypot that is high-interactive without the use of full virtualization technologies, such as a network of virtual machines, and the first ICS honeypot that can be managed with a Software-Defined Network controller.

Figure 3 presents the basic building blocks of our implementation, using Ethernet/IP as the reference industrial protocol. The vulnerable, Internet-facing devices are connected to the internal network, and they are the attacker's baits. A virtual switch is distributing Ethernet/IP traffic in the internal network, enabling ARP poisoning attacks, packet sniffing, and malicious command delivery. Every simulated host is connected to the virtual network, and it is exposing real services. For example, PLC1 may expose an Ethernet/IP server for explicit messaging, listening on standard TCP port 44818, that is addressable with realistic tag names, and preloaded with realistic tag values. Another example is a Human Machine Interface (HMI), that exposes an HTTP configuration interface through a webserver listening on port 80.

We now provide details on the implementation of the vulnerable VPN endpoint, the vulnerable gateway device, the simulated ICS devices, the network emulation, and the data collection subsystem. Finally, we describe additional benefits discovered during implementation time.

4.1 Vulnerable VPN Endpoint

Virtual Private Network (VPN) are widely used in ICS network to establish a secure channel between a host located outside the control network, and a network interface inside the control network.

Our target hardware platform is an Allen-Bradley Stratix 5900 Router, with firewall capabilities. The target device runs an IPv4, OpenConnect (Cisco) VPN server, reachable from the Internet with weak credentials: user is `admin`, and password is `admin`. Given this vulnerable VPN server, the attacker is able to get an IP in the internal network, and interact with the honeypot through the virtual network interface associated with that IP.

We support the OpenConnect VPN server using one of its open source implementations: `ocserv`. Our emulated network has a dedicated firewall host with IP `192.168.1.76` that is listening on default port 443.

4.2 Vulnerable Gateway Device

Gateway devices are used in ICS control network to translate industrial protocols, such as back and forth from Modbus to Modbus/TCP, and extend the inter-device communication capabilities of the whole ICS (similar to NAT).

Our target hardware platform is Moxa OnCell IP gateway, that is able to connect to the testbed over cellular networks using different technologies such as: GPRS, EDGE, UMTS, and HSDPA. The target device has two configuration ports open: a telnet server is listening on port 23, and a ssh server is listening on port 22. The ssh service is configured with weak credentials: username is `admin` and password is `admin`. The telnet service is configured with plaintext-based unencrypted authentication, with the same weak credentials as the ssh server. Given such a configuration, the attacker is able to get a command shell on the gateway device, that is directly connected to ICS internal network.

We support ssh and telnet servers through `sshd`, and `telnetd`. Our emulated network has a dedicated 3G Gateway host, with IP `192.168.1.77`, that is listening on port 22 and port 23. The honeypot shell is chrooted, and the fake file system mimics the one of the gateway device. In the ssh case, the chroot jail is specified directly in the sshd's server configuration file, taking advantage of OpenSSH's convenient `ChrootDirectory` feature [6].

4.3 Network Emulation

The network emulation, and the virtual network hosts isolation is implemented by Mininet using a low-level feature of the Linux kernel, called container-based virtualization. Container-based virtualization takes advantage of Linux network namespaces, and virtual Ethernet links, known as veth, to isolate subsets of processes. Each collection of processes is called a container, and it has a complete virtualized Linux network stack associated (e. g., IP, ARP, and route tables). Each container interface is connected to the software switch's virtual interface through a veth. The net effect is an emulated virtual network, this is the reason why the proposed honeypot runs *in-a-Box*.

The link shaping feature is implemented using another low level Linux kernel functionality that can be accessed through the `tc` program. Tc allows to monitor, and manipulate the network traffic control setting for each active network interface. Indeed, it is easy for use to set custom bandwidths, delays, and packet loss for each container in our honeypot.

Finally, the proposed network emulation implementation is able to run any (industrial) protocol stack available for Linux. The paper focuses on Ethernet/IP (ENIP), a modern object-oriented industrial protocol. ENIP is supported through the `cpppo` Python module [14].

4.4 Physical Process and Devices Simulation

The honeypot is simulating a water treatment physical process, an HMI, and four PLCs using a collection of python scripts. The PLCs logic mirrors the one described in the water treatment testbed operational manual, with the same control flow acting on real tag names, values, and types. Interlocks are simulated as well: for example, PLC1's logic depends upon values stored on PLC2, and PLC3.

The physical process simulation script simulates only the hydraulic part of the system, in real-time. Technically, each water tank has an inflow pipe, and an outflow pipe, both are modeled according to the equation of continuity from the domain of hydraulics (pressurized liquids). Where present, a drain orefice is modeled using the Bernoulli's principle for the trajectories [35].

MiniCPS allows us to parametrize the simulation time of a water treatment simulation. We are not using this feature in the honeypot because the attacker may realize that the response time is too fast compared to the real water treatment. However, we used this feature during other types of experiments (e. g., Man-in-the-Middle attacks) to generate data faster than the usual. It is important to note that the speed-up factor is bounded by the capability of the Linux kernel scheduler to manage concurrent processes. In our case, the speed requirements are not very high as the physical process of the simulated water treatment system is relatively slow.

4.5 Data Collection Subsystem

The data collection subsystem involves different types of data acquisition, that depend upon the attacker activities inside the honeypot. In case of the vulnerable gateway device, that is providing a shell to the attacker, we use standard shell logging techniques: a log file is storing a set of records, and each record contains the timestamp, the username, and the issued command. We are monitoring every user to deal with an attacker able to escalate honeypot's privileges. The log file is periodically copied to a safe location, outside the honeypot.

For the vulnerable VPN server we use standard network traffic logging techniques. Multiple `tcpdump` daemons are attached to the vulnerable network interfaces, and they are generating pcap capture files. Eventually, these files can be post-processed using more sophisticated network analysis tools, such as `wireshark`.

Additionally, the honeypot includes a software keylogger program, running with root privileges, and masked from user-space memory. The keylogger is generating a log file with all the entered keystrokes, and it is able to deal with more motivated attackers, that for example might use obfuscation, and encryption techniques to transfer their malicious payloads. This is a key difference between the effectiveness of a honeypot versus a (signature-based) NIDS. The NIDS is not able to recognize an encrypted malware sent by the attacker to the target machine, in contrast the honeypot keylogger will log the decryption phase of the malware on the target machine, detecting the attack, and providing precious information about the attacker's tactics.

4.6 Implementation Benefits and Risks

It is important to notice that a good implementation yields additional benefits, that typically are not captured during the design phase. In this section we will present some of the additional benefits provided by the MiniCPS framework.

In the problem statement, we target an honeypot usable both for research, and production. MiniCPS is based on lightweight virtualization, and allows us to configure the security level of the honeypot *parametrically*, at start-up time. By security level, we mean the amount of vulnerabilities deliberately included in our services, indeed a research honeypot will be pre-configured with a medium, or low security level, and a production honeypot will be pre-configured with a high security level. For example the latest version of an ssh server is installed in the high-security honeypot, and an older vulnerable version is installed in the low/medium-security honeypot. One can even think to patch a service, introducing trivial vulnerabilities, like default ssh admin credentials, for the low-security honeypot. Additionally, if the attacker manages to exploit the high-security production honeypot, we will most probably discover a new vulnerability, or a new exploitation technique.

MiniCPS allows to extend our honeypot from a virtual to an *hybrid* configuration. As discussed in Section 2, an hybrid honeypot presents a mixture of real and virtual devices and it is a cost-effective solution in the ICS context. Technically, a hybrid honeypot can be categorized as an hardware-in-the-loop simulation, and this setting increase the level of realism of the honeypot and also the complexity of its design and implementation. We have access to spare PLCs in our lab, and in the future we plan to perform experiments with a hybrid honeypot and compare the results against our virtual honeypot.

MiniCPS allows to connect different ICS instances together and to a real network. This feature enables the possibility to deploy a virtual ICS *honeynet*, that is a network of (virtual) ICS honeypots that can be accessed over the Internet and can collaborate to manage more advanced attack scenarios. For example, an honeynet might be able to manage multiple attackers attacking at the same time, redirecting each attack to a dedicated honeypot instance.

Finally, MiniCPS supports *Software Defined Network (SDN)* development natively [9]. In the default setting, the honeypot SDN controller is idle and it is not visible by the attacker. Nothing prevents us to actively using the honeypot SDN controller. For example, we may develop specific ICS control plane logics, able to extend the functionalities of our honeypot, such as data analytics, deep packet inspection, and detection mechanisms.

It is worth mentioning that the following implementation introduces the typical risks of a high-interaction honeypot. For example, if the attacker is able to escalate privileges inside the honeypot, we consider that honeypot useless (e. g., the attacker may send false values to the data collection subsystem). The attacker may also be able to penetrate the honeypot using a side channel, but this scenario is not captured by the presented attacker model, indeed is out of scope for this paper. Finally, we understand that it is really difficult to protect the honeypot against a knowledgeable attacker, but still the physical isolation between the honeypot, and the real ICS system will protect the real ICS anyway.

5. EVALUATION

5.1 Evaluation Context

In this section, we present a preliminary evaluation of our honeypot in the context of a Capture-The-Flag (CTF) competition. The competition was a part of broader ICS security event, called SWaT Security Showdown (S3), and hosted by Singapore University of Technology and Design (SUTD) in July 2016.

CTF are educational cyber-security competitions, hosted, online and offline, by Universities, private companies, and non-profit organizations. There are two standard types of CTF: jeopardy-style and attack/defense. A jeopardy-style CTF involves a set of challenges, divided by category (e. g., reversing, exploiting, and cryptography), and each challenge is presented with a short description, a number of clues, and an amount of reward points. Each team scores points solving these challenges, and the solution usually consists in a message to be entered in the CTF's scoring system. An attack/defense CTF involves a set of machines running vulnerable services, given to the participating teams, and connected on the same LAN. To score points, each team has both, to defend its services from the other teams (e. g., by patching a vulnerable service), and to attack the services protected by the contender teams (e. g., by login as admin on a webserver). Usually the given machines settings are not known a-priori by the teams, this is a key point to augment the learning experience of the participants. In both cases, the CTF stars and ends at prescribed time, and the team that scores most points wins.

SWaT Security Showdown's CTF involved different instances of our honeypot, one for each participating team, pre-loaded with different CTF challenges. In particular, the honeypots were simulating the hydraulic part of a subset of the Secure Water Treatment (SWaT) testbed. SWaT is a state-of-the-art water treatment testbed located at SUTD, consisting of six stages managed by six control devices. There are many interesting details about the SWaT testbed, but they are out of scope, indeed they are omitted from the discussion. To understand the remaining part of this section, it is sufficient to know that: each honeypot included several simulated components, in particular: four Programmable Logic Controllers (PLCs), a Human Machine Interface (HMI), two water tanks (named Raw water tank, and Ultra-filtration tank), and a vulnerable gateway device, that Ethernet/IP was the spoken industrial protocol, and that we connected the simulated devices in a star topology, recreating one layer of the SWaT control network. Notice that, we exposed only one vulnerable interface over the Internet, because S3's CTF already assumed that the attacker knew the (vulnerable) ICS entry point.

5.2 CTF and Honeypot Setup

In this section we will provide a brief description abut the CTF's scoring system, and the honeypots' setup,

The CTF scoring system was implemented as a web application (webapp), using the flask Python framework [26]. The webapp authentication was based on username and passwords, and we used Let's Encrypt to encrypt the webapp's ingoing and outgoing traffic, via HTTPS [13]. Each challenge could be solved entering the flag using an HTML form field. We decided to log all the scoreboard's user input to understand common errors, and detect brute-force attempts.

The honeypot setup was the most complex task. For network security reasons, we decided to run all the honeypots "in the cloud", using Amazon Web Services' Elastic Compute Cloud (AWS EC2) instances. Each honeypot ran on a single Linux kernel, using an m3-type EC2 instance. We set up a single instance, tested it, and then replicated it, with minimal reconfiguration issues, to accomodate six teams.

We decided to use ssh as the vulnerable service on the gateway device, and we assumed that the attacker already had obtained the credential to access it. That is why we distributed a (different) private key for each team to login inside the honeypot as the `attacker` user. Each instance was running two different ssh servers, with different configurations. One server was used by us to access the virtual machine, and manage the honeypot. The other server was running inside a Linux container, and the attacker was chrooted to protect the honeypot file system and running processes. It is important to notice that, both servers were running on port 22 on their respective networks, however we had to use port forwarding (from port 2222 of the control network to port 22 of the honeypot network), to give the attacker a ssh connection inside the honeypot network.

5.3 Challenges Descriptions

This section describes in detail the five challenges involving our honeypot, mimicking a subset of the SWaT's testbed, more information about the settings may be found in Section 5.1. The challenges' design followed common best practices of jeopardy-style CTF: challenges were presented in increasing order of difficulty, and the solution of challenge x was providing useful knowledge to solve challenge $x+1$.

Network warm up. The first challenge involves a basic understanding of the SWaT network topology. The challenge description is: "Can you eavesdrop what PLC2 has to say to PLC3?". The goal of that challenge is to perform a passive Man-in-the-Middle attack between PLC2 and, PLC3.

Ethernet/IP warm up. The second challenge involves some understanding of the Ethernet/IP industrial protocol. The challenge description is: "`cpppo` can be used in the testbed to communicate using the Ethernet/IP protocol. Can you read the README:2 tag." In this case, the attacker has to understand which PLC stores the README:2 tag, know its IP, and know how to query an Ethernet/IP server. Python's cpppo module is suggested because it is an easy to use library to do the job.

Overflow the Raw water tank. The third challenge involves a basic understanding of a water treatment industrial control system. The challenge description is: "PLC1 is controlling the raw water tank. It is reading the water level value addressed by the tag LIT101:1. PLC1 is controlling a motorized input valve, addressed by MV101:1, that can be turned ON/OFF using respectively 1/2. PLC1 is also controlling an output pump, addressed by P101:1, that can be turned ON/OFF using respectively 1/2. The maximum tank level in m from the ground is 1.2. The goal is to overflow the raw water tank." In this case, the attacker has to understand how to overflow a tank, based on a provided list of sensors and actuators.

Denial of Service HMI. The fourth challenge involves a basic understanding of Denial-of-Service (DoS) attacks. The challenge description is: "The HMI (set to manual mode) is constantly sending to PLC3 the keep alive value 2. You can access this value using the MITM:3 tag stored in PLC3. Can you change this value to 3? If you see that the tag value has a stable 3 value, wait a little bit to get the flag." In this case, the attacker has to find a way to disrupt the communications between HMI and PLC3. It is not sufficient for the attacker to write the value three in the MITM:3 tag.

Overflow the Ultra-filtration tank. The fifth challenge involves an advanced overflow attack on the ultra-filtration tank. The challenge description is: "The HMI (set to manual mode) is sending commands to PLC4. PLC4 is controlling the water level using MV301:3 and P301:3. Both can be turned ON/OFF using respectively 1/2. You can query LIT301:3 to discover the actual water level. The maximum tank level in m from the ground is 1.2. The goal is to overflow the ultra-filtration tank." In this case, the attacker could not reuse the same techniques used for the third challenges, and he has to find a smarter way to overflow the tank.

5.4 Challenges Results

In this section we present a number of interesting statistics gathered during the CTF event. We anonymized the names of participating teams. Team 6 is explicitly excluded from the analysis, because its members managed to eploit a side channel attack unrelated to the honeypot, which allowed them to bypass our honeypot challenges.

After the CTF event we post-processed the log files, and in Table 1 we present several interesting results. Only one team was able to solve all the challenges, and the average number of solved challenge per team was three. The majority of the teams used traditional reconnaissance tools such as: `nmap` and `ping`, expected attack techniques such as: Man-in-the-Middle attacks, and used `cpppo` for Ethernet/IP interaction, as suggested by us in the challenge description.

There are a number of lessons learnt by us during the CTF event, we will present two of them. Firstly, crash recovery management. It is important to implement an automated, and reliable honeypot's crash recovery sub-system, because an attacker may break the honeypot in (unexpected) ways. In that case, the honeypot has to shutdown gracefully, and restart after a reasonable time interval with the same settings. We experienced down-time issues because the attackers used simple bash scripts containing infinite while loops, resulting in a DoS of some of our honeypots. We were not able to automatically restart the targeted honeypots, and as result the of-

Table 1: CTF Results Summary.

Teams	# Captured Flags	# Distinct Cmds	# Executed LOC	# Recon Tools	# Attack Tools	Most Used Tools*
Team 1	2	20	1074	3	1	{1, 2, 6, 8}
Team 2	5	30	2488	6	2	{1, 2, 3, 4, 5, 6, 7, 8}
Team 3	3	23	2045	5	2	{1, 2, 3, 4, 6, 7, 8}
Team 4	4	27	963	5	2	{1, 2, 3, 4, 6, 7, 8}
Team 5	1	3	52	1	0	{1}

# : Number Of, LOC : Lines Of Code

*{1: `ettercap`, 2: `nmap`, 3: `netstat`, 4: `tcpdump`, 5: `tshark` 6: `ifconfig`, 7: `cpppo`, 8: `ping`}

Table 2: Honeypot metrics evaluation summary.

Metric	By design	Implemented
Network		
IP, MAC and netmask	●	●
Packet loss	●	●
Packet delay	●	●
Bandwidth	●	●
Topology	●	●
Common protocols	●	●
Industrial protocol	●	◐
Advanced Traffic	●	◐
Physical		
Realistic math model	●	●
Sensor readings	●	●
Actuators driving	●	●
Control logic	●	●
Human operations	●	◐
Advanced Process	●	◐

Legend ●: full support, ◐: partial support.

fending teams had to wait a couple of hours before restarting to attack the system.

The second important lesson was related to side channel attacks mitigation. We believe that defending a system is generally more difficult than attacking it, because the attacker has to find one vulnerable hole, however the defender has to protect every holes (that he is aware of). During the CTF, we suffered a side channel attack on a machine that was not running on any of our honeypots. The attack disclosed (among other things) information about the event logistics, solutions to challenges, credentials for a web service. For future events, we will take into account every detail of the configuration process, and ensure that we protect sensible data with access control and delete unnecessary data from places accessible to the attacker. The last statement might sound trivial, but it can be tricky to implement in practice, especially when multiple people are configuring a complex virtualized system, running multiple services, connected over the Internet.

5.5 Evaluation using Qualitative Metrics

In Table 2, we present an evaluation summary of our honeypot prototype. As features, we use the metrics proposed in Section. 3.4. The table distinguishes between features that are enabled by the honeypot design, and the ones implemented in the evaluated prototype. For example, the honeypot design is capable of providing full industrial protocol support. However, if there is only a partial implementation of the protocol available for Linux (as in the case of Ethernet/IP) we have to indicate a partial support in the Implemented column.

Table 2 shows that the implemented honeypot satisfies all the basic metrics, and partially satisfies the more advanced ones (such the simulation of a human operation). Furthermore, the implemented honeypot supports different types of attack ranging from network attacks, such as: Man-in-the-Middle, port scanning, and service enumeration, to physical process related ones, like tank overflows and DoS attacks on devices.

6. RELATED WORK

It has been observed over the years Internet-facing ICS devices are vulnerable to cyber attacks and in respect to that security aspects of ICS devices have been discussed in [36, 37], in particular the author presents attack statistics and a robust attribution framework by deploying a honeypot architecture in the simulated virtualized ICS environment. We now review related work in detail, and position our work against it. To the best of our knowledge, our work and the *Conpot* share the most features. We summarize our findings in Table 3.

Table 3: Our Honeypot Features vs. Related Works.

Related Work	Real	Virtual	Low-Interaction	High-Interaction	Hybrid-Interaction	Research	Production	Live Project	Link Shaping	Light-weight	Emulated Network
Our work		●		●		●	●	●	●	●	●
Scott et al. [27]	●		●				●				
Buza et al. [3]		●		●			●				
Holczer et al. [12]		●		●			●				
Yin et al. [22]		●	●			●					
Conpot [34]		●	●			●	●	●	●		●
Provos et al. [25]		●	●			●	●				
Litchfield et al. [18]	●	●			●	●		●	●		●
Liljenstam et al. [17]		●				●			●		

Design and Implementation of ICS honeypot. In [27], Scott proposes a mapping and configuration of honeypot that is similar to our High-interaction Hybrid honeypot architecture in several ways. In general, for honeypots to work effectively, there are two major concerns. Firstly, it is needed to map the network attack surface of the target system, and choose which services to mimic in the honeypot. Secondly, The configuration and placement of the honeypot in the local network needs to ensure that controlling and monitoring of attacks can be performed and attacker activities can be logged by a secure channel.

PLC honeypot. On the topic of PLC honeypots, the authors of [3, 12] propose a high-interaction honeypot PLC solution for secure network design. The proposed implementation involves exploration and inspection of all the PLCs and the services (HTTP, HTTPS, ISO-TSAP, and SNMP) running on PLCs in a system that should be protected. Those services are then implemented and integrated in a Linux based Virtualized simulated environment acting as a honeypot. In contrast to our work, the authors do not consider any interactions with physical processes, and in general only network-based interactions with the honeypot (no shell or VPN access).

6.1 Honeypot frameworks

IoTPOT. In [22], a low interaction honeypot framework was proposed. The authors claim that telnet based attacks on IoT devices are increasing rapidly. The authors propose IoTPOT, a honeypot that emulates telnet interactions of IoT devices, in order to attract and analyze attacks against various IoT devices running on different CPU architectures such as ARM, MIPS, and PPC. In contrast to IoTPOT, our proposed honeypot is focusing more on ICS rather than IoT devices. Additionally, the proposed honeypot does not focus on a specific protocol, such as telnet and related attacks, but it deals with different industrial protocols and services, resulting in a much stronger attacker model, and much larger attacker surface.

Conpot. Conpot [34] is an open source project for industrial control system honeypots. The honeypots can be classified as low-interactive server side honeypots, that are implemented with following attributes in mind: a) Deployment of honeypot in network should be easy b) Modification and extension of it can be done easily. As Conpot is provided with a stack of several industrial protocols, it can interact with real devices such as HMI and PLCS, and is capable of emulating complex infrastructures.

In contrast to Conpot, our framework allows high-interaction honeypots. In addition, the Conpot project does not cover the host and network virtualization aspects of our system, or enable physical process simulation. Instead, Conpot is more related to providing libraries and tools to build simulated ICS components. Unlike our work in this paper, Conpot also provides analysis functionality of the communication and actions of the attacker. For the future, we believe that we can strengthen the device simulation aspects of our system with components from Conpot.

Honeyd. Honeyd [25] is an open source software framework to configure several virtual host/honeypots in an existing network. The honeypots can be configured with arbitrary services. The framework is useful for attack detection and to collect statistics about malware attack. Unfortunately, development of Honeyd seems to have stopped in 2013.

HoneyPhy. HoneyPhy [18] is physics-aware honeypot framework for Cyber-Physical Systems (CPS). The proposed work has a broader scope (CPS are a superset of ICS), and claims to provide realistic physical process, and devices simulation by means of a hybrid configuration. In contrast, our honeypot framework is purely virtual, and eventually can extended to a (more expensive) hybrid configuration.

6.2 Related ICS Simulation Frameworks

In [7], a framework similar to MiniCPS is proposed in the context of Software-Defined Networking (SDN), and network intrusion detection for ICS. In particular, the presented system also uses Mininet [15]. The authors do not discuss the use of Mininet framework in a honeypot setting, but they discuss about a virtual network layer built on the top of physical process, to monitor the network status with high level of granularity. Furthermore, they demonstrate how smart grids could be made resilient in catastrophic circumstances using SDN.

In [16], a simulator named RINSE (Real-time Immersive Network Simulation Environment for Network Security Exercises) is developed with similar intent of the proposed ICS honeypot. The simulator claims to provide realistic and scalable network simulation by using multi-resolution traffic model, and routing protocols simulations. An extension of RINSE is proposed in [17], where a stronger attacker model is introduced. The new attacker's capabilities include: Denial-of-Service, computer worms, and similar large-scale attacks involving large numbers of hosts, and high intensity traffic. In contrast to RINSE, the proposed ICS honeypot does not simulate the network stack, but it uses network emulation to provide real packets and realistic network characteristics, such as delay, packet loss, and bandwidth. Furthermore, RINSE is developed as a training platform for network security, on the other hand the presented ICS honeypot is designed and implemented for both research and production purposes.

7. CONCLUSION

In this work, we presented the design of a virtual, high-interaction, server-based ICS honeypot, which aims to provide a realistic, cost-effective, and maintainable solution to observe and capture the activities of the attackers. Based on that design and the MiniCPS framework, we implemented parts of the SWaT testbed as honeypot.

To the best of our knowledge, the presented honeypot implementation is the first academic work targeting Ethernet/IP based ICS honeypots, the first ICS virtual honeypot that is high-interactive without the use of full virtualization technologies (such as a network of virtual machines) and the first ICS honeypot that can be managed with a Software-Defined Network controller.

We evaluated our implementation in the context of a capture-the-flag event targeted to ICS, called SWaT Security Showdown. During the event, six teams attacked six instances of our honeypot, producing interesting results. We were able to implement a realistic scenario, run multiple services, and generate realistic traffic over the virtual network.

In the future, we plan to improve the crash management subsystem, the Ethernet/IP support, the logging capabilities over the network, and the SDN support.

8. REFERENCES

[1] D. Antonioli. MiniCPS public repository. https://github.com/scy-phy/minicps.

[2] D. Antonioli and N. O. Tippenhauer. MiniCPS: A toolkit for security research on CPS networks. In *Proceedings of the Workshop on Cyber-Physical Systems-Security and/or PrivaCy (CPS-SPC)*, pages 91–100. ACM, 2015.

[3] 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 *Proceedings of the Workshop on Smart Grid Security*, pages 181–192. Springer, 2014.

[4] B. Cheswick. An Evening with Berferd in Which a Cracker is Lured, Endured, and Studied. *In Proceedings of the Winter USENIX Conference*, pages 163–174, 1992.

[5] CISCO. Industrial ethernet: A control engineer's guide. www.cisco.com/web/strategy/docs/manufacturing/industrial_ethernet.pdf.

[6] M. Damien and F. Markus. Chroot in OpenSSH. http://undeadly.org/cgi?action=article&sid=20080220110039.

[7] X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk. Software-defined networking for smart grid resilience: Opportunities and challenges. In *In Proc. of The Cyber-Physical System Security Workshop (CPSS)*, April 2015.

[8] N. Falliere, L. Murchu, and E. Chien. W32. stuxnet dossier (Symantec Security Response), 2011.

[9] N. Feamster, J. Rexford, and E. Zegura. The road to SDN. *ACM Queue*, 11(12):20–40, 2013.

[10] B. Galloway and G. P. Hancke. Introduction to industrial control networks. *IEEE Communications Surveys & Tutorials*, 15(2):860–880, 2013.

[11] J. B. Grizzard, S. Krasser, and H. L. Owen. The Use of Honeynets to Increase Computer Network Security and User Awareness. *Journal of Security Education*, 1(2-3):23–37, 2005.

[12] T. Holczer, M. Félegyházi, and L. Buttyán. The design and implementation of a PLC honeypot for detecting cyber attacks against industrial control systems. https://www.crysys.hu/publications/files/HolczerFB2015CN.pdf, 2015.

[13] I. S. R. G. (ISRG). Let's Encrypt. https://letsencrypt.org/.

[14] P. Kundert. Communications protocol python parser and originator. https://github.com/pjkundert/cpppo. [Online; accessed 31-July-2016].

[15] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010.

[16] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier. RINSE: The real-time immersive network simulation environment for network security exercises. In *Proc. of Workshop on Principles of Advanced and Distributed Simulation (PADS)*, pages 119–128, 2005.

[17] M. Liljenstam, J. Liu, D. M. Nicol, Y. Yuan, G. Yan, and C. Grier. Rinse: The real-time immersive network simulation environment for network security exercises (extended version). *Simulation*, 82(1):43–59, 2006.

[18] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, and R. Beyah. Poster: Re-thinking the honeypot for cyber-physical systems. Poster at IEEE Symposium on Security and Privacy, 2016.

[19] J. C. Matherly. SHODAN the computer search engine. https://www.shodan.io. Accessed: 2016-08-01.

[20] J. R. Moyne and D. Tilbury. The emergence of industrial control networks for manufacturing control, diagnostics, and safety data. *Proceedings of the IEEE*, 95(1):29–47, Jan. 2007.

[21] ODVA. Ethernet/IP technology overview. https://www.odva.org/Home/ODVATECHNOLOGIES/EtherNetIP.aspx. Accessed: 2016-08-01.

[22] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow. IoTPOT: Analysing the Rise of IoT Compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT)*. USENIX Association, 2015.

[23] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, pages 2435–2463, 1999.

[24] T. Phinney. IEC 62443: Industrial network and system security. https://www.isa.org/pdfs/autowest/phinneydone/.

[25] N. Provos. A virtual honeypot framework. In *Proc. of the USENIX Security Symposium*, 2004.

[26] A. Ronacher. Flask: web development, one drop at a time. http://flask.pocoo.org/.

[27] C. Scott. Desigining and implementing a honeypot for SCADA network, 2014. White paper published by SANS Institute Infosec Reading Room.

[28] J. Slay and M. Miller. Lessons learned from the maroochy water breach. *IFIP International Federation for Information Processing*, 253:73–82, 2007.

[29] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley Reading, 2002.

[30] L. Spitzner. The honeynet project: Trapping the hackers. *IEEE Security & Privacy*, 1(2):15–23, 2003.

[31] C. Stoll. *The cuckoo's egg: tracking a spy through the maze of computer espionage*. Simon and Schuster, 2005.

[32] K. Stouffer, J. Falco, and K. Scarfone. Guide to Industrial Control Systems (ICS) Security. http://industryconsulting.org/pdfFiles/NISTDraft-SP800-82.pdf, 2006. Recommendations of the National Institute of Standards and Technology.

[33] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn. Guide to Industrial Control Systems (ICS) Security (Revision 2). http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf, 2015. Recommendations of the National Institute of Standards and Technology Revision 2.

[34] The Honeynet Project. Conpot. http://conpot.org/.

[35] Wikipedia. Continuity Equation. https://en.wikipedia.org/wiki/Continuity_equation. Accessed: 2016-08-01.

[36] K. Wilhoit. The SCADA that didn't cry wolf, 2013. Whitepaper.

[37] K. Wilhoit. Who's really attacking your ICS equipment? http://www.edgis-security.org/honeypot/whos-really-attacking-your-ics-devices/, 2013. Whitepaper.