NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EVASION OF HONEYPOT DETECTION MECHANISMS THROUGH IMPROVED INTERACTIVITY OF ICS-BASED SYSTEMS

by

Jeffrey T. Dougherty

September 2020

Thesis Advisor: Thuy D. Nguyen
Co-Advisor: Neil C. Rowe

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
EVASION OF HONEYPOT DETECTION MECHANISMS THROUGH IMPROVED INTERACTIVITY OF ICS-BASED SYSTEMS

5. FUNDING NUMBERS

6. AUTHOR(S) Jeffrey T. Dougherty

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

In recent years critical-infrastructure systems, particularly smart electrical grids, have become dependent on computer control systems and thus increasingly vulnerable to cyber attack. Attempts to defend these systems with deceptive decoys (i.e., honeypots) have been hampered by unconvincing feedback to attackers about the state of the physical processes the honeypots supposedly control. We constructed a high-interaction honeypot for a microgrid that used a physics-based electrical-grid simulation to provide realistic responses to intruders. To create a diverse data corpus, we deployed the honeypot in two configurations, one with a supervisory control and data acquisition (SCADA) human-machine interface layer and one without. Both honeypots elicited real attacks and successfully evaded the Shodan search engine's Honeyscore, a widely known automated honeypot detection system. The collected data was analyzed to determine traffic patterns and attackers' behavior, and the results suggest promising avenues for further research. This work contributes to the research and development of intelligent behavior-based intrusion detection systems to better defend national critical infrastructure and DOD systems, both afloat and ashore.

14. SUBJECT TERMS
honeypot, industrial control systems, ICS, cybersecurity, cyberdeception

15. NUMBER OF PAGES
107

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

EVASION OF HONEYPOT DETECTION MECHANISMS THROUGH IMPROVED INTERACTIVITY OF ICS-BASED SYSTEMS

Jeffrey T. Dougherty
Civilian, CyberCorps – Scholarship For Service
BA, Williams College, 2005

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2020

Approved by: Thuy D. Nguyen
Advisor

Neil C. Rowe
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In recent years critical-infrastructure systems, particularly smart electrical grids, have become dependent on computer control systems and thus increasingly vulnerable to cyber attack. Attempts to defend these systems with deceptive decoys (i.e., honeypots) have been hampered by unconvincing feedback to attackers about the state of the physical processes the honeypots supposedly control. We constructed a high-interaction honeypot for a microgrid that used a physics-based electrical-grid simulation to provide realistic responses to intruders. To create a diverse data corpus, we deployed the honeypot in two configurations, one with a supervisory control and data acquisition (SCADA) human-machine interface layer and one without. Both honeypots elicited real attacks and successfully evaded the Shodan search engine's Honeyscore, a widely known automated honeypot detection system. The collected data was analyzed to determine traffic patterns and attackers' behavior, and the results suggest promising avenues for further research. This work contributes to the research and development of intelligent behavior-based intrusion detection systems to better defend national critical infrastructure and DOD systems, both afloat and ashore.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

APCI	Application Protocol Control Information
API	application programming interface
APDU	Application Protocol Data Unit
ASDU	Application Service Data Unit
COT	Cause of Transmission
DCS	distributed-control system
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICS	industrial-control system
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IOA	information object address
IP	Internet Protocol
LN	logical node
MMS	Manufacturing Message Specification
PCAP	packet capture
RDP	Remote Desktop Protocol
SCADA	supervisory control and data acquisition
TCP	Transmission Control Protocol
TI	Type Identification
UDP	User Datagram Protocol
VM	virtual machine

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Mom, Dad, Matt, and Joanna their unwavering love and support throughout life, even when things seemed darkest. I'm proud to be your brother and son, and I love you all. I would also like to thank my SFS cohort and fellow Computer Science students for getting me through the past two years with as much as possible of my sanity intact. Monterey would have been a much bleaker place without your support and good humor.

I would like to thank Professors Nguyen and Rowe for turning me on to such a deeply interesting project, letting me take it in my own direction, and helping steer me over the rocks and shoals of turning it into a thesis. You've been the best advisors I could have asked for.

Thanks are also due to Dr. José Romero-Mariona, Anthony Shaw, and all the other wonderful folks at Code 71770 of Naval Information Warfare Center Pacific for their support over the summer of 2019, for showing remarkable flexibility and grace when their summer intern showed up with a project already in hand, and for generously giving their permission to use portions of my unpublished report to them in this thesis. I hope our groups' future collaboration on this project is long and fruitful.

This material is based upon activities supported by the National Science Foundation under Agreement No. 1565443. Any opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In recent years, the critical-infrastructure systems that provide essential services to Americans have become increasingly complex, interdependent, and reliant on computerized industrial control systems (ICS) [1]. As our national dependence on these control systems has grown, so has the frequency and complexity of cyberattacks conducted against them [2]. Protection of these critical-infrastructure systems has been identified as a major concern in both the United States' National Cyber Strategy [3] and SECNAVINST 3501.1D [4].

A. MOTIVATION

During the initial development of ICS systems from the 1950s, little attention was paid to identifying and managing risk from potential security flaws because the systems were not connected to large networks [1]. In more recent years, there has been increasing recognition that "in today's world, reliability requires cybersecurity" [5]. A major challenge, however, is that ICS systems are often constructed with a planned lifetime of 20 to 30 years, meaning that legacy systems with well-known vulnerabilities will continue to be used in critical systems for some time [1].

In recent years, electrical-grid systems have become a major user of ICSs to control power generation, transmission, and distribution [6]. While this has enabled greatly increased efficiency, it has also left power grids increasingly vulnerable to cyber attack. A prominent example of this occurred in December 2015, when hackers believed to be linked to the Russian government used malware called CrashOverride to cut electrical power service to over 230,000 people in the Ukraine [7]. Due to the inherent difficulties in securing the legacy systems used in many power systems and in other vital infrastructure, researchers are examining novel ways to enhance their security. One such method is the deployment of honeypots.

A honeypot is a heavily monitored system designed to resemble a real system as closely as possible, but having no real functions other than deceiving an intruder into believing they have actually penetrated a system which the honeypot resembles [8].

Previous studies of hacker behavior using honeypots imitating ICS systems have been hampered by the relatively low fidelity of available honeypots. Most ICS honeypots are either commercial products, requiring fees and licensing to deploy [9], or are relatively low-fidelity [10]. “Low-fidelity” for a honeypot means that it opens a network port corresponding to an ICS protocol, but does not offer a hacker much interaction over that protocol. This makes such a honeypot easy to detect by automated systems or alert hackers, substantially reducing its value.

A past attempt to overcome these limitations was a system called GridPot, originally described in a Ph.D. dissertation [11]. GridPot was an open-source honeypot with four components: an outward-facing Web-based user interface, a virtual signal unit, a simulated power grid, and a modified version of an ICS honeypot called Conpot [12] simulating ICS-connected devices on a grid. The signal unit received commands from the attacker, which were then passed to simulated electrical devices, and from there to the simulated grid. The grid changed the state of its simulation and provided feedback to the operator via a user interface, providing dynamic responses to the attacker’s commands and making the honeypot much more realistic.

Unfortunately, the publicly available code for GridPot does not offer the full range of capabilities described in the dissertation. A GridPot instance implemented by Kendrick and Rucker at our school [13] produced only a low-fidelity honeypot like Conpot because its simulation capabilities were limited. Investigation of the public code base revealed several limitations in GridPot’s implementation. However, it was a potential basis for development of a higher-fidelity electrical grid honeypot.

B. RESEARCH PLAN

Our plan had two phases. In Phase 1, we constructed a honeypot capable of generating dynamic feedback to an attacker based on the state of a physics-based power grid simulation. Our Phase 1 honeypot interacted over both the Hypertext Transfer Protocol (HTTP) [14] and the ICS control protocol IEC 104, but lacked any user interface. In Phase 2, we added an interface to the honeypot by connecting it to a supervisory application running on a separate virtual machine. The two phases were separated to let us collect data

from Phase 1 while we completed Phase 2. We gathered information on attacker interactions with both phases of the honeypot and compared our results to those obtained by our group's previous version of GridPot [13].

C. THESIS OUTLINE

The remaining chapters are organized as follows. Chapter II gives general background about the electrical grid, industrial control systems, the protocols they use for communication, and honeypots in general. Chapter III provides more specific background on the electrical-grid simulation software, model, networking protocols, and honeypot we used as the basis for our development. Chapter IV discusses our implementation of both phases of the honeypot in detail and describes our data collection plan for each phase. Chapter V shows our analysis of the data collected for both phases, and Chapter VI gives our conclusions and identifies some possible further research.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. INDUSTRIAL CONTROL SYSTEMS AND THE ELECTRICAL GRID

1. The Electrical Power Grid and GridLAB-D

An electrical-power grid is a complex system with four main functions: generation, transmission, distribution, and consumption [15]. First, power is generated at a relatively small number of plants by means such as coal, natural gas, nuclear, or solar collection. It is then transmitted from nearby electrical substations, at very high voltages (typically 155 kV or higher) to minimize losses in transit due to wire resistance, until it reaches local distribution substations. At those substations power is stepped down to lower voltages and delivered over local transmission lines to individual consumers in homes, office buildings, or factories [16]. Typically, power is stepped down a final time just before delivery, minimizing losses in the distribution steps.

This thesis simulated the power grid through the IEEE 13-Node Model, originally developed by the Test Feeder Working Group under the Analytic Methods for Power Systems technical committee of the Institute of Electrical and Electronics Engineers (IEEE)'s Power and Energy Society. This is an old, relatively simple, and well-characterized model of distribution from a central step-down substation to local customers [17]. The advantages of using this model were that it was available in the power-grid simulator GridLAB-D [18] (see Chapter III for details), required relatively few computational resources, and offered a variety of electrical devices to simulate in a fairly realistic environment. For this project, we judged these advantages more important than the risk that an attacker knowledgeable about the electrical industry might recognize this widely published grid model and realize our honeypot was fake.

2. Industrial Control Systems

Industrial control systems (ICSs) are information systems for automating industrial processes, an umbrella term that can refer to any means of production and distribution of finished products [19]. ICS are used in many sectors of industry and vital infrastructure in the United States, including all types of manufacturing, transportation of both people and

goods, and essential utilities such as water and the electrical grid [1]. ICSs are divided into two broad categories: distributed control systems (DCSs), devices such as programmable controllers that generally run a single piece of machinery, and supervisory control and data acquisition (SCADA) systems, which aggregate data from multiple DCSs in a given area for purposes including automated analysis, centralized human monitoring of the entire system, and shared storage space for data and system logs.

3. Electrical Grid Industrial Control Protocols

Modern electrical-grid ICSs mostly use either the Distributed Network Protocol 3 (DNP3), the IEC 60870-5 protocol, or the IEC 61850 protocol. DNP3 and IEC 60870-5 are both older protocols, originating in an effort by the IEEE to develop a protocol standard for the industry during the 1980s. The IEC 61850 standard originated about a decade later as another attempt to develop a standard protocol [20]. Although current data are scarce, a 2014 study found that DNP3 predominated in North American power grid automation, while utilities elsewhere used a mix of IEC 61850 and IEC 60870-5 protocols [21] with IEC 60870-5 predominating for communication between substations and remote-control stations.

By itself, the IEC 60870-5 standard assumes dedicated wired connections between devices, and thus does not route messages. The IEC 60870-5-104 companion standard, commonly called IEC 104, extends IEC 60870-5 to include routing by the Transfer Control Protocol/Internet Protocol (TCP/IP) protocol stack [21]. IEC 61850 is another important high-level protocol, with the companion standard IEC 61850-8-1 defining a mapping to a transport-level protocol called the Manufacturing Message Specification (MMS) [22].

4. Protocol Selection and Rationale

When the project began our plan was to handle control messaging with IEC 61850 as did the original GridPot implementation [11]. However, we later chose to use IEC 104 instead, for the following reasons.

a. IEC 61850

IEC 61850 was used in the original GridPot, and is emerging as the de facto industry standard for substation automation [23]. It thus appeared to be the best chance for creating a realistic ICS honeypot. However, IEC 61850 is substantially more complex than previous electrical grid ICS protocols such as those described in the following paragraphs. It specifies not only the content of messages but also the data model used to store and process information on each end of the connection [23]. Every device in a grid using IEC 61850 is described by a logical node (LN), implemented as software classes that define devices with specific functions such as switch controllers, circuit breakers, and transformers [23]. While this approach enables standard syntax and easy validation of commands, it also means that in addition to implementing the required protocol stack, IEC 61850 products must also implement every logical node required for every device in the power grid in which the products are used.

b. DNP3

Like IEC 104, DNP3 offered the possibility of simplification compared to IEC 61850. It is also common in North-American applications, and in one publication appeared to attract more attention from attackers than an IEC 104 honeypot [24]. However, we could not find any open-source SCADA applications supporting DNP3, which drove us towards IEC 104.

c. IEC 104

The IEC 104 protocol is much less prescriptive than IEC 61850, eliminating configuration of the data model as a possible source of error. Since the open-source honeypot Conpot [12] and an open-source SCADA application both supported full IEC 104 messaging, we decided that the IEC 104 protocol offered the best potential for simulating control messaging in our honeypot. Chapter III gives more explanation of our reasoning.

B. HONEYPOTS

As discussed in Section I.A, a honeypot is a heavily monitored system designed to serve as a decoy for unauthorized network users [25]. A honeypot tries to resemble a real system as closely as possible to deceive an intruder into believing it is a real system [8].

1. Honeypot Types

Honeypots are deployed for two main purposes: research and production. Research honeypots gather intelligence on intruders, such as their origin, tactics, and data of interest. Production honeypots seek to improve the security of a system by mirroring it as closely as possible. Since the honeypot is a decoy, an intrusion-detection system will produce few false positives, making it a useful early warning system for the network administrator [26]. Production honeypots can also slow intruders by presenting many similar decoy systems that they must sort through to find a real target.

As mentioned in Chapter I, honeypots can also be categorized by how realistically they respond to an intruder's probes and queries over the network, or their level of interaction. Low-interaction honeypots emulate the network characteristics of a service, but either ignore attacker queries or provide a limited repertoire of preset responses [27]. Examples of low-interaction honeypots are Honeyd [28], Specter [28], Gastopf [27], and Dionaea [27]. Medium-interaction honeypots more accurately simulate a specific service or protocol, responding to a range of attacker queries as would a real system offering that service or protocol [28]. While they are more complex to implement and maintain than low-interaction honeypots, the additional realism they offer can better fool an attacker. Examples of medium-interaction honeypots are Mwcollect, Nepenthes, and Honeytrap [28]. Finally, high-interaction honeypots run the operating system of the host they try to imitate, often by acting as a transparent proxy for actual hardware [27]. While high-interaction honeypots offer the highest potential for both research and production purposes, they also expose the network on which they reside to substantially more risk than other types, as the host for which they are acting as a proxy can actually be exploited by the attacker.

2. Existing ICS and SCADA Honeypots

We wanted to create a honeypot that could imitate an ICS and attached SCADA system, and was free of licensing issues that would limit deployment for research or production. Surveys of the field have found few such honeypots for ICS systems [9], [10]; many security companies concentrate their research on high-interaction honeypots by combining virtual hosts and actual hardware. Options we considered were as follows.

HoneyPoint is a commercial honeypot that can imitate an ICS/SCADA network [9]. However, price and licensing issues make it unsuitable for us.

Cisco's SCADA HoneyNet project [29] used multiple virtual hosts to simulate an entire industrial control network, including both local controllers and a SCADA system. Although it emulated a specific controller, it did not try to simulate the physical process the controller was supposedly managing, so that an intruder who actually checked the controller's data values would not see them changing realistically [30]. Similar low-interaction ICS honeypots that fail to simulate physical processes include CryPLH [31], GasPot [32], and CamouflageNet [33].

Antonioli et al. proposed a design for a high-interaction honeypot that would both emulate an ICS device and simulate the physical process that device was supposedly controlling. Results from the simulation would be fed into the emulated ICS devices to provide realistic output for an attacker [34]. However, their work was apparently not implemented.

HoneyPhy is an ICS honeypot framework that includes a physics-based simulation of the processes its devices are supposedly controlling, providing an attacker with realistic data on the state of a physical process. Two implementations of the framework were described in a 2017 Master's thesis [35], one implementing a simple Heating, Ventilating, and Cooling (HVAC) system, and the other a water-treatment plant. There have been no signs of active development since then, and no code is available online.

SHaPe was described as a plugin for the Web honeypot Dionaea that emulated an ICS using IEC 61850 for communication [36]. While the code for SHaPe is publicly available, it has not been updated since shortly after initial publication and is incompatible

with the current version of Dionaea. While SHaPe could be a possible basis for future development, we found more promising candidates available.

Conpot is a well-known and well-documented open-source ICS honeypot [34]. It has a modular framework in which a common core and data bus support protocols and associated configuration templates, which are bound to specific Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) ports. Protocol servers in Conpot define how the honeypot should respond to specific datagrams in packet sequences; templates are XML documents setting specific parameters such as the number and type of data points offered by the emulated device. By default, Conpot is a low-interaction honeypot without physical-process simulation, like several honeypots described in this chapter. However, it forms the basis for GridPot, the honeypot we selected for further development.

GridPot is a medium-interaction ICS honeypot that simulates both industrial control devices on a simulated power grid and the flow of electricity through the grid [11]. It includes two major components, a power-grid simulator called GridLAB-D and a honeypot based on Conpot. The changes to Conpot included class templates for GridLAB-D devices such as switches and transformers, and the ability to get and set GridLAB-D variables using GridLAB-D's HTTP-based interface. As the honeypot ran, it periodically polled GridLAB-D for updated values from the simulation, then updated the corresponding variables on its simulated devices with the current simulation data. Thus, an attacker monitoring GridPot's "devices" would see their values change realistically over time. Although GridPot had several limitations (discussed in Chapter III), it was a useful starting place for development.

3. SCADA Software

As previously discussed, SCADA software provides a supervisory interface allowing an operator to view data from and send commands to multiple ICS devices. Since our proposed design included a SCADA interface linked to our honeypot, we evaluated some open-source SCADA software packages.

ScadaBR and the closely related Scada-LTS were used by other projects involving deception in SCADA systems [37]. However, neither package supports IEC 61850 or IEC 104. They do support IEC 101, a related protocol that uses a different transport layer on

top of IEC 60870-5, but making IEC 104 work with IEC 101 required special routing software [38]. To avoid these complications, we examined other alternatives.

OpenSCADA is another prominent open-source project. Unfortunately, its IEC 61850 implementation only supports MMS, without any provision for embedded IEC 61850 information. It also does not support IEC 104 and thus was rejected as inadequate for our needs.

Our final choice was an open-source SCADA application that natively supports both IEC 104 and IEC 61850 [39]. With this application we could configure measurement and control points corresponding to the IEC 104 variables configured on Conpot, enabling receipt of updates and transmit commands to the virtual IEC 104 device running on Conpot. The name of this application is withheld to prevent fingerprinting.

Although the project providing this application is open-source, we were unable to build from source code and had to rely on compiled binaries provided by the project maintainer. These binaries work only on Windows, forcing us to use a Windows Virtual Machine (VM) to host the SCADA system rather than Linux. The project maintainer created these binaries by compiling source with Visual Studio 6.0 [40], an obsolete version that is difficult to find or run on modern systems. The maintainer has also said that although it should be possible to compile the software for Linux, it has never been done to his knowledge [41], and no resources are available for doing so. This may complicate future work which seeks to integrate our version of GridPot with Docker [42].

THIS PAGE INTENTIONALLY LEFT BLANK

III. POWER GRID HONEYPOTS

This chapter discusses the approaches we took for our honeypot implementation, including the reasoning behind our choices of networking protocols and physical model for our power grid simulation. We also describe the Conpot and GridPot components that are relevant for our implementation (discussed in Chapter IV).

A. GRIDLAB-D AND POWER FLOW ANALYSIS

1. GridLAB-D

GridLAB-D is an open-source simulation package intended to model the flow of electricity through a distribution system [43]. It uses C++ syntax to describe systems of interconnected components such as power sources, transformers, switches, and voltage meters [44] and an algorithm called PowerFlow [45] to model the flow of electricity through those components. A running GridLAB-D simulation can be monitored and controlled through an integral HTTP server running in a separate thread, using specifically formatted requests to either get or set the value of a variable in the simulation. By default, this interface listens on TCP port 6267, assigned to GridLAB-D by the Internet Assigned Numbers Authority (IANA) [46], to avoid interfering with other HTTP servers running on the same host. This interface is a key to GridPot's function.

2. The IEEE 13-Node Model

The model we simulated in GridLAB-D was the IEEE 13-Node Model with houses added (Figure 1). Our model expands the standard 13-Node Model discussed in Chapter II by including houses with associated power meters and street-level transformers to provide them with 120 V current.

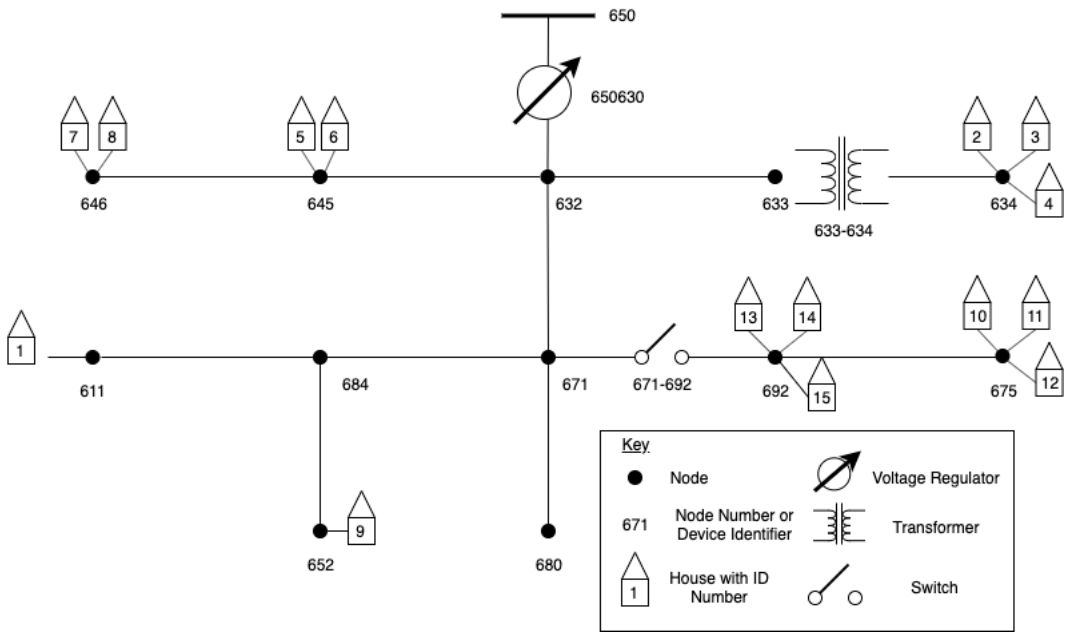

Figure 1. The IEEE 13-Node Model With Houses. Adapted from [47].

The components of the PowerFlow simulation are power lines and nodes which represent line junctions and the electrical buses interconnecting them [48]. Nodes are identified by 3-digit numbers, with lines and most in-line electrical devices deriving their names from the nodes they interconnect, as in Figure 1. In-line devices included in the model are a voltage regulator, which takes power from a notional upstream source and outputs it to the rest of the grid at a given output voltage, a transformer that steps down high-voltage current between nodes 633 and 634, and a switch that can disconnect nodes 692 and 675 from the rest of the grid. The model also includes houses numbered from 1 to 15, each of which has an associated local transformer and a power meter.

B. NETWORKING PROTOCOLS

1. HTTP

The Hypertext Transfer Protocol (HTTP) is a protocol allowing a server and a client on the World Wide Web to exchange messages, operating at the application layer of the OSI networking model. [14]. An HTTP exchange consists of a client using several defined HTTP methods to interact with resources stored on a server. The most often used methods on the Web are the GET method, which requests that the server transfer a copy of the

resource to the client, and the POST method, which requests that enclosed data from the client be appended to the resource stored on the server. Less often used methods include OPTIONS, which requests information about what methods the server will permit for interaction with a given resource, and HEAD, which returns metadata for a resource on the remote server. HTTP is assigned by default to Transmission Control Protocol (TCP) port 80. Conpot's integral HTTP server can handle GET, POST, HEAD, and OPTIONS requests, as well as an additional method called TRACE for debugging purposes.

2. IEC 61850

The IEC 61850 protocol is relatively new, with the first compliant applications appearing on the market around 2005 [20]. IEC 61850 is more than a communications protocol; it specifies an object-oriented information model to standardize how electrical-grid devices are described. Every device is described by one of more logical nodes (LNs), software classes that define devices with specific functions such as switch controllers, circuit breakers, and transformers[23]. A standard syntax describes the data and control points associated with each logical node, permitting interoperability between devices and applications from different manufacturers [49].

The disadvantage of this approach is that before any device can communicate by IEC 61850, its functional components must be described compliant with the logical node standard [23]. This creates a major implementation challenge for anyone trying to create their own IEC 61850 application. Perhaps for this reason, no open-source honeypots or SCADA software packages we found currently support IEC 61850, with the partial exception of the original GridPot. These disadvantages, and the ready availability of an IEC 104 honeypot as part of Conpot and open-source SCADA software supporting IEC 104, lead us to switch to using IEC 60870-5-104 (IEC 104) for our power-grid honeypot.

3. IEC 60870-5-104 (IEC 104)

IEC 60870 is an older networking protocol [20] intended to control ICS within the electrical industry. IEC 60780-5-104 is a companion standard that implements IEC 60870 messages as an application-layer protocol on top of the TCP/IP stack [50]. This permits

easy IEC 60870 messaging over the public Internet, virtual private networks, and private packet-switched networks, and is commonly called IEC 104.

IEC 104 messages start with a six-byte header called Application Protocol Control Information (APCI) [51] (Figure 2).

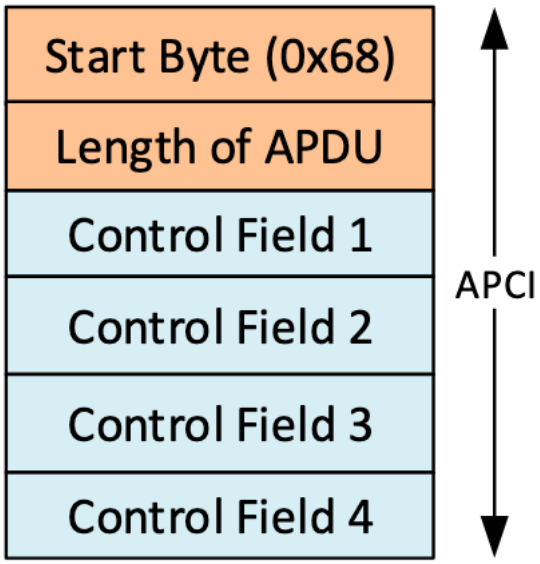

Figure 2. Structure of the APCI for IEC 104. Source: [51].

The APCI header consists of a “magic” byte with value 0x68 that identifies it as an IEC 104 message, a length byte, and four bytes of control information which depend on the message’s type. The last two bits of the third APCI byte identify the message as one of three IEC 104 frame types as follows:

1. I-frames (last two bits 00 or 10) exchange information between the central control station and slave devices. The control bytes are used as sequential numbering and acknowledgement fields to permit detection and retransmission of lost I-frames.

2. S-frames (last two bits 01) transmit commands from the central station to remote devices.

3. U-frames (last two bits 11) enable and disable transmission of data from a remote station to the central controller, and also exchange link-layer keep-alive messages to verify that the network is still functioning when no other IEC 104 frames are being transmitted.

I-frames and S-frames contain additional data, referred to as an Application Service Data Unit (ASDU), which consist of a header followed by one or more information objects. The structure of the ASDU, shown in Figure 3, contains several fields relevant to our work. The complete frame, consisting of the APCI header and the ASDU if present, is referred to as an Application Protocol Data Unit (APDU).

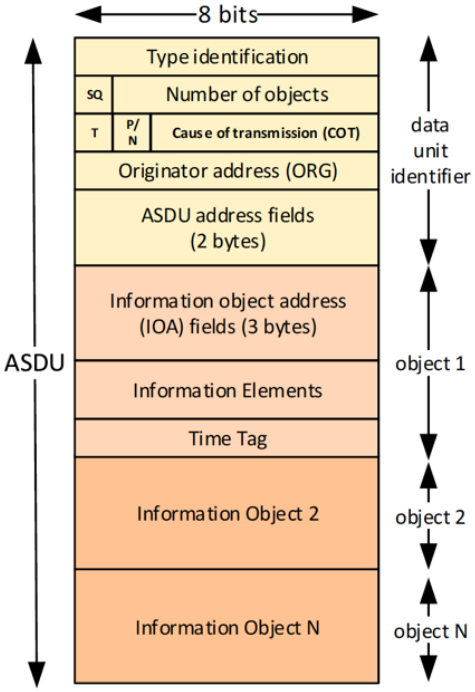

Figure 3. Structure of the ASDU. Source: [51].

The ASDU address field, two bytes in length, is implementation-specific. Although it normally indicates a specific physical device on the network, it can also indicate logical groups of values within a device [52]. There are also “broadcast” ASDU addresses, which can be attached to certain command types (see discussion of Type Identification) to elicit

responses from all ASDU addresses on a given IEC 104 network. The Conpot IEC 104 server assumes that the IEC 104 server has a single ASDU address specified in the server's template XML file.

Each value in an IEC 104 network is called an information object. It is identified by a three-byte Information Object Address (IOA), the first two bytes of which must be unique over the IEC 104 network, for a total of 65535 possible IOAs. The third byte is optional and is used for disambiguating IOAs of the same device.[51] By convention, Conpot stores IOAs as strings in the format:

<decimal value of first two bytes>_<decimal value of last byte>

For example, an IOA with bytes 0x00 0x6b 0x03 would be written as "107_3".

The Type Identification (TI) field's meaning varies depending on whether the message is going in the monitor direction (from subordinate device to the master control station) or the control direction (from the master control station to subordinate device). In the monitor direction, it tells the master station what type of IOA it reports on. IEC 104 types not only include basic data types such as Boolean, integer, or normalized values, but also specify whether the value has a short time tag, a long one, or none. In the control direction, the TI field identifies the command being sent, and functions like an instruction [53]. Importantly, there is only one TI field for each ASDU. Hence, while multiple IOAs can report or be commanded by the same ASDU, they must all be of the same type.

As mentioned previously, some monitor commands can be broadcast to an entire IEC 104 network. The most important of these for our discussion is the "General Interrogation" command (TI value 100). When this command is sent to a network, all devices on the network generate I-frames reporting the status of their IOAs. This provides a newly logged on control station with both a catalog of IOAs it may interrogate and an overview of the status of the network.

Except for some system-configuration messages, most other commands must be addressed to a single IOA with a command type matching its data type. For example, a properly formatted command to change the value of a non-timestamped floating-point IOA

(TI value 13) must not only be addressed correctly, but it must be of the non-timestamped floating-point command type (TI value 50) [54]. This provides additional checking that commands are addressed properly. Although details appear to be implementation-specific, Conpot implements this check by mapping each changeable informational IOA to a separate command IOA of the appropriate type. The advantage of this arrangement is that it provides detail about both the reception and execution of commands: a SCADA application knows that the command has been received when the command IOA reports a change in value, and knows that the command has been executed when the informational IOA reports a change in value.

The final field of interest is the Cause of Transmission (COT) field, which indicates why the datagram is being sent. This field in the APDU must match its Type Identification and IOA. For example, the General Interrogation command must be sent with a Cause of Transmission value of 6 (Activation), and responses to it must be coded with a value of 20 (Response to General Interrogation). This provides another layer of error checking, and provides a way for an overloaded master station to distinguish command responses from more routine traffic.

4. Microsoft Remote Desktop Protocol

The Microsoft Remote Desktop Protocol (RDP) is one of a family of remote access protocols that allow a user to stream an operating system desktop from a remote machine to their local system [55]. As the same suggests, RDP is a proprietary protocol bundled with the Microsoft Windows operating system.

RDP is a TCP-based protocol that establishes encrypted connections over TCP port 3389 [56] using the RC4 cipher and either a 56 or 128-bit key [57]. Once a Windows host has the remote desktop access enabled, it responds to incoming connections on port 3389 by negotiating with the remote system to allow mouse movements and keyboard strokes inside the remote connection window on the client to be transmitted to the server as though they came from a user using the server's own input devices [58]. User authentication and login then takes place using the normal Windows login process.

However, this use of Windows login for authentication left a large security flaw in the remote desktop. Versions of Windows previous to Windows 10 had a preinstalled account called Guest, intended for temporary users, which by default had a blank password. Although Microsoft has not commented on the Guest account's removal from Windows 10, many observers believe its removal was for security reasons [59].

We used Remote Desktop Protocol to give attackers access to Phase 2 of the honeypot. Attackers would connect to a Windows virtual machine that was linked to our honeypot, and on logging into the Guest desktop would be presented with a SCADA user interface offering information about our notional power grid and opportunities to control it. Since we wished to make access to the SCADA interface as easy as possible, we used Windows 8.1 with the Remote Desktop Protocol for the Guest account enabled, an intentionally vulnerable configuration.

C. CONPOT

1. Description and Components

Conpot is a well-known and well-described open-source ICS honeypot [12]. Written in the Python language, Conpot is a low-interaction honeypot without physical process simulation, but it served as the basis for the honeypot part of GridPot. Conpot has a modular structure with the following components.

a. Core

This module has three functions that are relevant to this thesis. During startup, it reads a master configuration file to determine which protocol servers to create and which ports are assigned to each server. It also initializes a virtual data bus, which allows protocol servers to read and write data without generating real network activity. During operation, the core listens for incoming traffic on all ports and passes any traffic on a server-assigned port to the appropriate server. The core also assigns each combination of remote IP and TCP/UDP ports a unique identifier consisting of 32 hexadecimal digits, which is appended to all log entries associated with that remote socket.

b. Protocol Servers

These modules recognize, and sometimes respond to, traffic using a specific application-layer protocol. For example, the default Conpot HTTP server can parse an incoming packet to determine the HTTP request method and can, for simple requests, generate and send HTTP replies. A major advantage of using the current version of Conpot for our honeypot is that it already contains an IEC 104 protocol server.

c. Templates

Each protocol server has a template, which is an XML-format document containing setup information for that server. For example, the template of an HTTP server describes which version of HTTP it uses, the response codes it returns for given requests, and the location of any Hypertext Markup Language (HTML) documents it provides in response to URL requests. The templates aid obfuscation of a honeypot by allowing the Conpot administrator to change the names and strings used by a server without changing the server's inner workings.

d. Loggers

These are independent workers that receive notifications of events from running protocol servers and generate a single log file in one of several formats, including syslog and JSON. The Conpot core automatically logs the start and end of sessions, but once the session has been passed to a protocol server, that server must decide what else is logged. For our research, we used a relatively simple text logger that output any events sent to it as text to both standard output stream (stdout) and a file on disk. Although less structured than the other options available, this logger also captures output to the standard error stream (stderr), giving additional information for troubleshooting.

2. Detection by Adversaries

As a freely available ICS honeypot, Conpot has been used extensively in honeypot research [5, 6, 7]. Unfortunately, this has also made it easier to create signatures to detect it. A 2016 study of multiple Conpot instances deployed in the Amazon AWS cloud found that Conpot opened additional ports not typically found open on ICS, potentially allowing

automated detection [12]. As with any other security measure, the more widely Conpot is deployed, the easier time adversaries will have developing signatures to identify it.

D. GRIDPOT

GridPot is a medium-interaction to high-interaction honeypot that simulates both the flow of electricity through a power grid and the functioning of electrical devices on that grid. First described in [11], the original version of GridPot included two major subsystems. One was GridLAB-D, running as a standalone process with an HTTP server to simulate electrical devices in a grid. The other was a modified version of Conpot capable of interacting with the GridLAB-D server.

By default, the only port GridPot opens to the Internet is port 80, which connects users to the standard Conpot HTTP protocol server. The GridPot template contains an HTML document provided by that HTTP server in response to incoming requests. The document contains Conpot and GridPot code that causes the HTML document to display a primitive interface showing the status of GridPot's variables. The HTTP server obtains this data by using the GridPot data bus to call methods of the GL_obj class.

GL_obj is a base class that contains methods that create an object in GridPot's memory corresponding to an object in GridLAB-D. GL_obj maps its internal variables to those in the GridLAB-D object, and contains methods allowing the GL_obj object to get and set those variables through GridLAB-D's HTTP interface. As previously mentioned, it also contains methods for returning HTML snippets with information about the status of its corresponding device in the GridLAB-D simulation All device objects in GridPot are instances of a specific device class descended from GL_obj which specifies the possible behavior of that type of device. For example, the GL_SWITCH class translates Boolean values for switch closure to the open and closed syntax used by GridLAB-D. A detailed discussion of our modifications to the GL_obj object is provided in Section IV.B.2.a.

E. PREVIOUS WORK AT NPS

Initial deployment of an Internet-facing ICS honeypot by our research group began with a five-month deployment of Conpot v 0.51 [60]. From October 2017 to February 2018,

a Conpot instance was deployed on a host outside our school's firewall with servers open for the protocols HTTP, SNMP, Modbus, S7Comm, BACnet, and IPMI. The first two of these are common communication protocols, while the last four are ICS protocols. The honeypot logged an average of 10.5 attacks per day during the deployment period, with irregular traffic patterns. Most of the traffic occurred on HTTP and Modbus, and showed bursts of high activity with intervening periods of little to no traffic [60].

A subsequent deployment of GridPot compared the Conpot data against a similar period with the higher-interaction GridPot honeypot [18]. As previously noted, the published GridPot software fell short of what it claimed it could do. Changes to the deployed honeypot were limited to modifying it to work with the modified IEEE 13-Node model with houses, as we have discussed, and changing some of Conpot's default template values to provide some obfuscation. Data collection on HTTP, Modbus, and S7Com protocols occurred over an 19-day period in April of 2019 [13].

As previously mentioned, our research goal was to create a more interactive and higher fidelity version of GridPot and to compare our data against the data collected in previous work [13]. The next chapter discuss the design and implementation of our enhancements to GridPot.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. METHODOLOGY

This chapter discusses the design and execution of our experimental work, including both phases of our implementation plan, as well as data collection methods and challenges we encountered. The two finished honeypots we deployed as part of this experiment are discussed in Sections IV.B and IV.D. Their deployment, on a standalone host outside our school's firewall, is described in IV.C and IV.E.

A. EXPERIMENTAL DESIGN

To maximize our chances of getting useful data, we divided our experiment into two phases. Each phase would end with a deployable honeypot with more sophisticated feedback mechanisms than GridPot, letting us to collect data with our Phase 1 honeypot while we developed Phase 2. The goal of Phase 1 was to produce an ICS honeypot that could accept IEC 104 commands from a remote user, translate them through Conpot to change the values of variables in the linked GridLAB-D simulation, and tell the user by IEC 104 messaging of changes produced by their input and other changes to the simulation state. The goal of Phase 2 was to connect the honeypot from Phase 1 with SCADA software that would provide a graphical interface for the electrical system simulated by the honeypot.

1. Phase 1 Design

Since Phase 1's development updated Conpot to include an IEC 104 server, we dispensed with a separate receiving device such as the virtual GEbrick (discussed in IV.B.1.a) used in the original GridPot design. Figure 4 shows how a user command flows through the original GridPot structure.

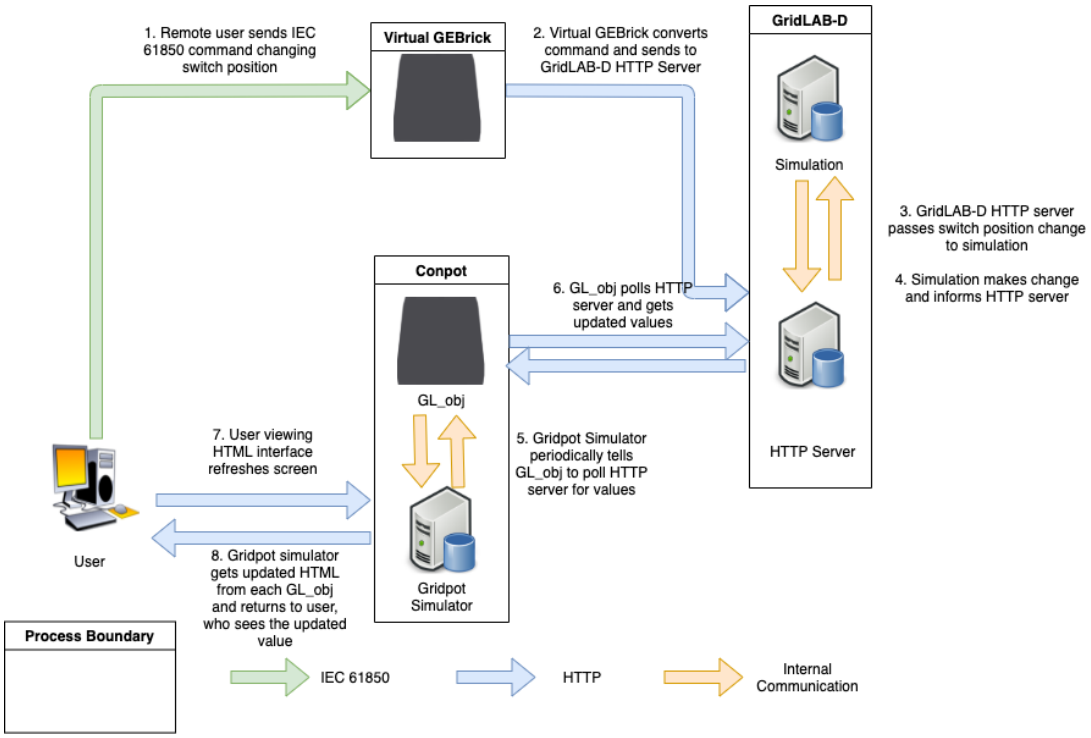

Figure 4. Flow of a User Command Through Original GridPot Structure

For comparison, Figure 5 shows the flow of the process used for our GridPot design: The remote user sends an IEC 104 (instead of IEC 61850) command and sees that command yield a realistic response. The user's command is directed to Conpot's IEC 104 server, which calls a GridPot simulator method that passes the command to the associated GridPot information object. The object then translates the IEC 104 command into an HTTP request and passes it to the GridPot HTTP server, from where it is passed to the GridLAB-D simulation which makes the change. The next time the GridPot simulator tells that GridPot object to poll its variables, the object passes that new value to the simulator, which notices the change. The simulator then uses its pointer to the IEC 104 server to prompt it to send a packet telling the user of the new value.

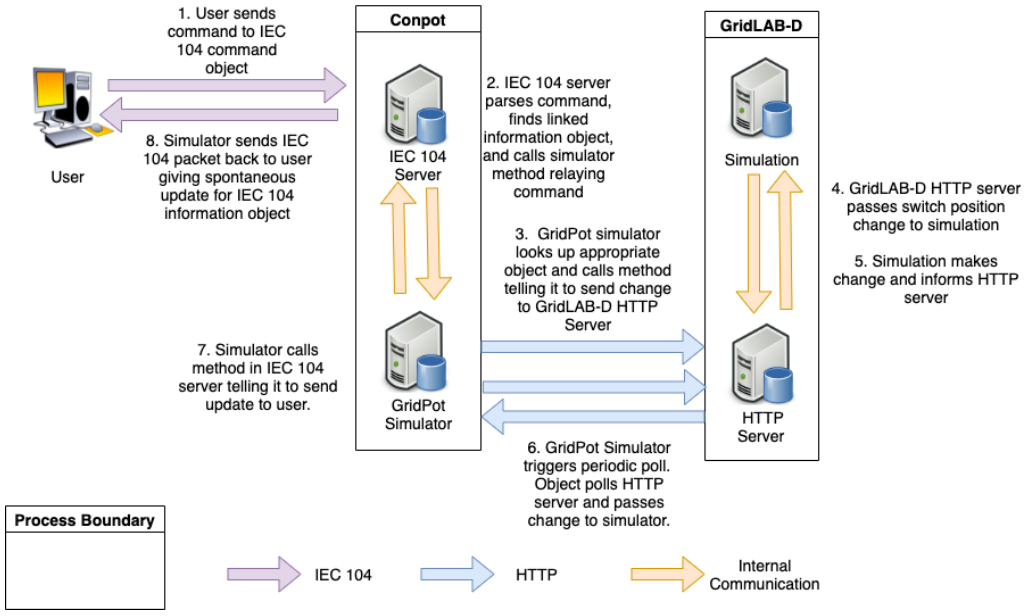

Figure 5. Design of the Phase 1 Honeypot System

2. Phase 2 Design

Phase 2 added another deception by adding a SCADA station with graphical user interface to the honeypot, obviating the need for a remote user to have an IEC 104 client application. Phase 2 allowed remote users to connect to a virtual machine running our open-source SCADA application on a Windows operating system through Microsoft's Remote Desktop Protocol. That virtual machine communicated with a Linux virtual machine running a backend system functionally identical to the Phase 1 GridPot. The backend then provided realistic feedback to the SCADA application on the Windows virtual machine. Figure 6 shows the Phase 2 design.

Figure 6 shows the same process as Figure 5, although for clarity steps unchanged from Figure 4 have been aggregated into step 3 in Figure 5. Instead of interacting directly with Conpot's IEC 104 server, the intruder uses the remote desktop application to interact with the human-machine interface software running on the Windows system. This provides the intruder with an interface closer to the one they would experience if they broke into an actual ICS control station. The desktop transmits the user's keystrokes and mouse clicks to the interface software, which translates them into IEC 104 messages and sends them over a virtual network to a separate Linux virtual machine running GridPot. Importantly, this

means that if the remote user captures packets on the Windows host's inward-facing virtual network interface controller, they will see only IEC 104 traffic of the type expected between a real SCADA control station and its associated IEC 104 server. Once the traffic reaches GridPot's IEC 104 server, it is processed identically to IEC 104 traffic in Phase 1. When the IEC 104 server sends its packet containing updates to the information object, the packet goes back over the virtual network to the interface software, which processes it and presents the results to the remote user. The changes in the view are then transmitted back to the remote user's screen by the Remote Desktop Protocol.

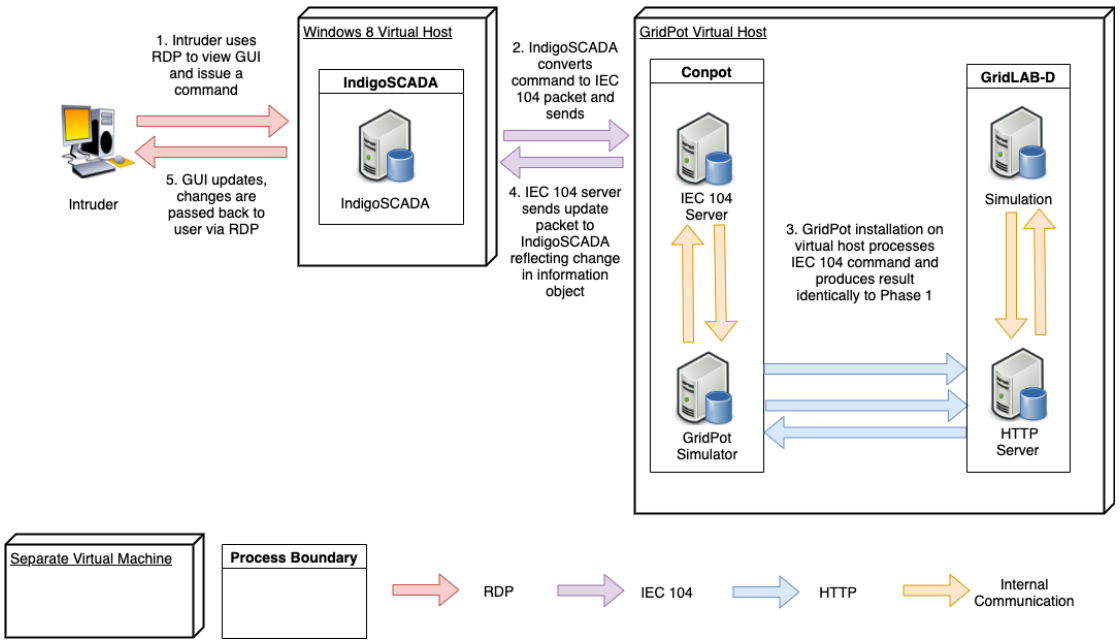

Figure 6. Design of Phase 2 Honeypot System

B. PHASE 1 IMPLEMENTATION

Changes to the Conpot component of GridPot in Phase 1 were of two major types: Conpot upgrades and changes to the GridPot simulator.

1. Modifications to Conpot IEC 104 Server

As mentioned, the first change we made to Conpot was to update the version built into GridPot to the newest version available. This also required updating the version of the

Python language used in Conpot from 2.7 to 3.6. The major advantage of this upgrade was that the newer version of Conpot contained an IEC 104 server. This server interprets and responds to some IEC 104 messages, eliminating the need to implement our own server. In its original form, the Conpot IEC 104 server could respond to a general interrogation by listing informational Information Object Addresses (IOAs) and their current values. We expanded on those capabilities substantially.

First, we added the capability to optionally label IEC 104 IOAs defined in the server's template as either "GridPot" or "Python" variables. "GridPot" variables map to a value in the GridLAB-D simulation. If the corresponding GridLAB-D value has changed when polled by its corresponding GridPot object, the change is automatically routed through the GridPot simulator to the IEC 104 server where the data is updated. Similarly, any commands sent to an IEC 104 command object are automatically passed to the GridPot simulator and on to GridLAB-D.

"Python" variables, by contrast, map to variables in a GridPot object not in the GridLAB-D simulation. These variables are not polled by the simulator because they can only change from user commands. For example, a GridPot switch might have an "enable" command that must be set to True before the position of the switch can be changed. This capability is not modeled in GridLAB-D, so the command cannot use a GridPot variable. However, we can map the command to a Python variable, then write the code defining switch behavior so that the GridPot variable for position will not be changed unless the enable Python variable's value is true.

Second, we modified the Conpot IEC 104 server to accept certain types of incoming commands not handled by the original GridPot, and to pass their values to the GridPot simulator for action if their IOAs mentioned GridPot or Python variables to. We modified existing code for handling IEC 104 type-45 (set single-point Boolean), type-46 (set double-point Boolean), type-49 (set scaled value), and type-100 (general interrogation) commands to let them interface without requiring a SCADA application, and added the capability to handle type-63 (set floating-point value with time tag) commands as well [51]. We chose to handle type-63 commands rather than modify the existing code for type-50 (set floating point) commands because our SCADA application sent type-63 commands when the user

issued commands with floating-point numbers. Table 1 shows what the Conpot IEC 104 server could handle after our changes.

Table 1. IEC 104 Command Types Handled by Conpot Server

Message Type	**Description**	**New/Modified/ Unchanged**
45	Set Single-Point Boolean	Modified
46	Set Double-Point Boolean	Modified
49	Set Scaled Value	Modified
50	Set Floating-Point Value	Unchanged
63	Set Floating-Point Value With Time Tag	New
100	General Interrogation	Modified

Third, we enabled the Conpot IEC 104 server to send data updates to a remote user without prompting. When a remote socket interrogates an IEC 104 server, its session is now marked as "active" until the remote socket disconnects or times out. While a session is active, the GridPot simulator will continue to poll the objects it controls (such as switches) for changes to their variables, and will pass these changes to the IEC 104 server as with Phase 1. If the IEC 104 server receives updates, it generates packets with the new values, labels them as spontaneous updates (COT value 3), and sends them to the remote socket. When no active sessions remain, the IEC 104 server tells the GridPot simulator to stop polling its virtual devices.

Fourth, to aid handling of updates we changed the IEC 104 server to provide a pointer to itself to the GridPot simulator using the Conpot data bus, and accept a pointer to the simulator in return. When the IEC 104 server has completed startup, it pushes a pointer to itself onto the Conpot data bus and reads the pointer to the simulator previously pushed during the simulator's startup (see Section IV.B.2.c). While in theory this might cause a crash if the server tries to retrieve a nonexistent pointer, testing showed this was not a problem. Once the server has a pointer to the simulator, it uses that pointer to call a simulator method that prompts the simulator to read the pointer the server just pushed to the data bus. Since both server and simulator have pointers to each other, they can communicate by calling each other's methods.

Finally, we changed the IEC 104 server to communicate with our SCADA application. Although not required for Phase 1, we implemented it to test it with the other IEC 104 server changes. IEC 104 specifies that to change a variable, the remote station must send an "activation" command, to which the server responds with an "activation confirmation" message. The user then sends a command which causes the server to change the variable and send back both an I-frame with the updated value and an "activation termination" message for the variable. By default, the Conpot IEC 104 server receives the command, changes the variable, sends back the updated value, then sends the activation termination message. However, our SCADA application expects the updated value to come after the activation termination message and will not listen for it until the termination message has been received. Since we could not determine which behavior was correct, we added a Boolean variable in the simulator allowing the GridPot user to choose at startup whether to use the default Conpot sequencing or interface-compatible sequencing.

2. Modifications to GridPot Simulator

This section discusses changes made to the GridPot simulator and other GridPot-specific data structures and code.

a. Changes to GL_obj

The original GL_obj object and its derived classes have two major limitations. First, all variables within an object must correspond to a variable in the GridLAB-D simulation, so functionality not described in the GridLAB-D model is impossible. Second, devices simply report values from GridLAB-D, which can create unrealistic output. For example, the PowerFlow algorithm returns voltages as a complex number with a real and imaginary component, from which (differently depending on how the number is reported) must be calculated a real number for the actual "gauge" reading. GridPot simply reports the real and imaginary values, making it relatively easy for an attacker familiar with power systems to discover it is a simulator rather than a real grid.

Our changes to the GL_obj class substantially expanded its functionality and removed those limitations. We implemented methods that allowed for object variables not associated with the GridLAB-D simulation, similar to the "Python" variables implemented

in the Conpot IEC 104 server. We also improved handling of variables by storing them as instances of a new class called GL_104_datobj. This new class stored mappings between IEC 104 IOA numbers and GridLAB-D or Python variables, and also contained methods for converting between the strings GridLAB-D uses to store data and IEC 104 values, including complex numbers.

These changes substantially reduced the amount of device-specific code required for each subclass of GL_obj, since the only code needed for each device was the logic to differentiate its behavior from others. For example, when a switch is told to close, the HTTP command must read "status=CLOSED2" instead of "status=CLOSED" for the variable to change in the GridLAB-D simulation. Since this only applies to switches, sending "CLOSED2" instead of "CLOSED" is handled in the GL_switch class. This let us add basic power meters as GridPot devices by instantiating the base GL_obj class without any device-specific code.

b. The GridPot Simulator

The GridPot simulator has a Conpot protocol server that is initialized by the Conpot core like other protocol servers. At startup, the simulator parses a template file with the extension "gpm" that specifies what GridPot devices to create, their types, and the mapping between their internal variables and GridLAB-D simulation variables. Once the simulator is running, it periodically polls its subordinate devices to update their variables from the corresponding GridLAB-D values using an event-based loop, and, by default, print their status to stdout. The simulator also has methods letting it supply the status HTML document from its subordinate devices when prompted by the Conpot HTTP protocol server.

The major limitation in the original GridPot simulator is the lack of integration with any power grid control protocols. The integration with IEC 61850 described in [11] was achieved by using the open-source libiec61850 library [61] to create a device called *gebrick*. This device can interpret one specific command sent to one specific data point within a notional switch logical node, then passes the command to toggle the switch directly to GridLAB-D using its HTTP-based interface. The GridPot simulator's HTML

interface showed the change in switch position once the switch object polled for its new value, but the simulator itself was unaware of the IEC 61850 protocol. The gebrick device could not react to anything other than that one specific IEC 61850 command, and GridPot could neither generate a realistic IEC 61850 response on its own nor interface with any SCADA HMI. This imposes major obstacles to creation of a convincing simulation of a power grid. Figure 4 (in Section IV.A.1) shows how a user command flows through GridPot's original structure.

Our changes to the GridPot simulator and our additions to the IEC 104 protocol server enabled the two to interact. The GridPot simulator periodically polls its devices for updated values and orders the IEC 104 server to send updates as needed by calling methods within the server, with mechanisms in place to prevent multiple updates for a single IOA to be queued simultaneously. Incoming IEC 104 requests for information or commands are passed to the simulator by the server calling its methods, with requests or commands being passed on by the simulator to individual objects and results being sent back.

c. Modifications to GridPot Simulator Startup Sequence

We changed the startup sequence of the GridPot simulator to support interaction with the IEC 104 server and our new features of the simulator. The new startup sequence is as follows:

1. An instance of the GridPot simulator class is initialized. It reads a specified gpm file to determine what devices it must support, and initializes GridPot objects for each. For this project, the gpm files used in the original GridPot were extended with additional definitions of devices containing the following information: device type (currently switch, transformer, regulator, or meter), name in the GridLAB-D simulation, a list mapping GridLAB-D variables to IEC 104 IOAs for all GridPot variables in the object, and a list mapping Python variables to IEC 104 IOAs for all Python variables in the object. The entries use the format:

device_type.name.{gridlab_var:IEC_104_IOA,...}.{python_var:IEC_104_IOA, ...}

2. The simulator instantiates GridPot objects with the specified GridPot and Python variables. It also creates a hash table mapping the IEC 104 IOAs of GridPot and Python variables to the names of the GridPot objects corresponding to those variables. This simplifies routing of commands and requests for information.

3. The simulator establishes a connection pool to the GridLAB-D HTTP server for all queries by its GridPot objects, using the urllib3 Python library rather than the urllib2 library used in the original GridPot. This connection pool is available to all GridPot objects, obviating each object having its own HTTP connection to GridLAB-D.

4. The simulator pushes a pointer to itself on the Conpot data bus and waits for a call from the IEC 104 server. When that call arrives, the simulator pulls a pointer to the IEC 104 server off the Conpot data bus.

d. Modifications to Polling Loop

Since our changes to the GridPot simulator made it slower, we changed its architecture to try to improve its efficiency. We had the simulator poll its GridPot objects only when the IEC 104 server confirms an active connection. We also made the polling a multi-threaded process, allowing the simulator to simultaneously use multiple processors. The simulator's polling sequence was as follows:

1. The simulator checks a shared dictionary for the polling variable associated with each object. If it is set to True, object is still being polled from a previous polling cycle (perhaps due to a slow GridLAB-D). In this case, the simulator bypasses this object and goes to the next. Otherwise, the simulator instantiates a thread to poll the object. It repeats steps 1–2 until all GridPot objects have either been bypassed or had threads instantiated to check them. The main thread then waits for each polling thread to finish, with an arbitrary limit of two seconds.

2. Each object's polling thread sets the polling variable for its object to True in the shared dictionary, indicating that the object is being polled. It then reads the current value associated with each variable in the object, prompts the object to update its GridPot variables values through GridLAB-D and return their values, and checks whether the new value for each variable differs from the previous one. If a value has changed, the thread checks a second shared dictionary for a Boolean variable denoting whether that value is eligible for a spontaneous update. If that variable is set to True, the thread creates an instance of a class called ExchangeValue containing the changed variable's IOA, type, and new value, and places that instance in a shared set of ExchangeValue instances. It also change's the variable's spontaneous update Boolean in the second shared dictionary to False. Before returning, the thread sets the object's polling variable in the first shared dictionary to False.

3. When all device threads have either completed or timed out, the main simulator thread uses its pointer to the IEC 104 server to pass any ExchangeValue instances created by its threads to the server. The IEC 104 server then generates spontaneous updates for each ExchangeValue as described in IV.B.1. It then waits an arbitrary time of one second to begin the next polling cycle.

4. A variable's spontaneous update value is not reset to True until the GridPot simulator is told that it has received an IEC 104 packet updating the value has been sent. This ensures there will only be one spontaneous update value per variable waiting to be sent at any time.

C. PHASE 1 DEPLOYMENT

Phase 1 was deployed as a standalone honeypot.

1. Host Environments

We used two environments for this experiment, a development environment and a live environment. The host for the development environment was a Dell Precision M6800 laptop computer with a 64-bit Intel 2.8 GHz 8-core processor, 16 gigabytes memory, and a 750 gigabyte hard disk running the Ubuntu 18.04.01 LTS operating system. This host held the GridPot virtual machines, and was used for code development and testing of honeypot components. The host for the live honeypot was a Dell XPS 8910 desktop computer with an Intel 3.40 GHz quad-core CPU, 16 gigabytes memory, and a 925 gigabyte hard disk running the Windows 10 Home operating system.

The live honeypot host was connected to the Internet by an ISP outside our school's firewall. However, the IP address is part of a block belonging to our institution, something that may have discouraged more sophisticated attackers.

2. Virtual Environments

On both hosts, we ran the Phase 1 honeypot as a virtual machine under VirtualBox 5.2.22 with two virtual CPU cores, 10,240 megabytes memory, and 50 gigabytes of storage in a virtual disk. The test honeypot could access the Internet by a Network Address Translation (NAT) connection, and was also connected to an internal network to which we also connected a separate virtual machine running an IEC 104 client. This allowed the test machine to update software as needed from the Internet through the network-address connection, while disallowing incoming connections from the Internet to the test machine's servers since its Internet-connected IP was not publicly visible. The live-honeypot machine had a bridged adapter, and used the same MAC address as that of the host machine's network-interface controller through which the virtual machine was bridged. The controller's IP address, subnet, and gateway were manually configured to match the host machine's configuration. This allowed Conpot to record the remote IP addresses connecting with its open ports.

3. Data Collection

Data was collected for Phase 1 over two time periods. The first period ran with some interruptions between January 28 and May 14, 2020, and the second ran between June 17 and August 2, 2020. The Phase 1 GridPot, configured as described earlier, connected to public-facing external network and was left for Internet users to discover. Other than conducting a SHODAN scan [62] and Honeyscore evaluation [63] of the honeypot's IP address early in the first run, we did not publicize the honeypot's address or try to draw attention to it in any way.

a. Wireshark Packet Captures

We collected data in two ways during Phase 1. The first was with Wireshark, a network-protocol analyzer that captured incoming packets on GridPot's connection to the Internet [64]. To keep the packet capture (PCAP) files to a manageable size, we only recorded packets for the two services provided by GridPot, HTTP (TCP port 80) and IEC 104 (TCP port 2404). We lost Wireshark data from January 30 to February 3, when Wireshark was mistakenly set to capture on the wrong network interface. We also lost data sometime between May 3 and May 13, when a power outage caused the host computer to shut down. At the time, Wireshark required a human operator to manually save packets to disk; after this, Wireshark was reconfigured to save its PCAP files every three hours as in [13].

b. Conpot Logs

Our second data source was Conpot's logs, specifically its text logger. These logs have the advantage of being continuously written to disk, so we had Conpot log information for periods in which lost PCAP data.

We had more trouble with Conpot data collection than with Wireshark collection. In some crashes, the logger would cease logging, but when Conpot was manually stopped would suddenly output many events with timestamps corresponding to when it was stopped. This occurred five times between February 4 and May 3, and lost a total of 28

days' worth of Conpot logs. This problem was seen in [60] and they could not identify a cause.

Other crashes related to the IEC 104 server. The logger would output an error message about the IEC 104 server's packet "receive" function, then freeze. This happened twice between March 14 and March 25, and lost six days' worth of Conpot logs. We found that these crashes were due to malformed IEC 104 packets and added more consistency checks to the server, after which this type of crash ceased.

4. Data Analysis

Most analysis used the Python 3.6 Scapy packet handling library [65], and the Pandas data analysis library [8, 9].

a. Data Consolidation and Interpolation: Incoming Requests

To consolidate the data files from Phase 1, we wrote a Python program to parse PCAP files and Conpot log files for incoming HTTP and IEC 104 traffic, and record the timestamp, remote IP address, target TCP port, and HTTP method or IEC 104 frame type. For IEC 104 packets, the program determined whether they were simply directed to port 2404 (called "traffic") or if they contained valid frames (called "messages"). A valid frame was defined as a TCP segment directed to port 2404 with a payload starting with a byte value of 0x68 and a second byte correctly specifying the length of the message. The remote IP address for each request was also queried in GeoLite2, a free IP geolocation database[66].

PCAP files were our primary analysis data source since we had fewer problems collecting their data. Conpot logs were used as backup for time intervals in which we had no PCAP data. The resulting data was then saved into a Pandas DataFrame for analysis.

For comparison, we also got the raw data files from the previous GridPot study described in [13] and ran them through the same analysis software. We could not simply use the reported results in that study because they relied heavily on Conpot logs, which we could not do because of the data collection issues described in Section IV.B.3.b. Since the

previous dataset used different ICS protocols, we did not gather message type information from that dataset.

b. Data Consolidation and Interpolation: Sessions

To investigate which IP addresses contacted the honeypot more than once, we wrote a Python program to take our parsed data described in Section IV.C.4.a and count the number of sessions started by each IP address. For our purposes, we defined a session as all packets exchanged by a socket pair on a given date. For each session, we recorded the date, the remote IP address and port, the service the session was addressed to, the country of origin for the IP address, and the number of incoming requests recorded during that session. For IEC 104 sessions, we also recorded the number of correctly and incorrectly formatted messages.

c. Reconstructing IEC 104 Packets from Conpot Logs

To allow us to more closely inspect IEC 104 traffic using Wireshark, we wrote a Python program using Scapy to generate IEC 104 packets from Conpot log data for periods in which we lacked PCAP data. We could do this because Conpot logs for IEC 104 messages provide the sending and receiving sockets as well as the payloads. We could not do this for HTTP traffic because Conpot does not capture the full payload of HTTP messages.

D. PHASE 2 IMPLEMENTATION

As with our changes for Phase 1, the work we did in Phase 2 can also be broken into two main categories: the changes needed to supply a realistic user interface to an intruder, and configuration changes needed on both virtual machines to support Phase 2's operations.

1. SCADA Application Interface

Creating a realistic experience for an intruder to see involved configuring a SCADA application and designing a realistic user interface within that application.

a. Device and IOA Configuration

Our SCADA application ran on a Windows virtual machine (see Figure 6). We first created a virtual IEC 104 device and added it to the application's list of supported devices, then created IOAs within that virtual device for each information and command object from GridPot we wished to make visible at the SCADA interface. The SCADA application stores IOAs as integers, so we added code to convert IOAs from GridPot's format to integers.

We then connected GridPot and our SCADA application to the same VirtualBox internal network. When the SCADA application started, it sent an IEC 104 general interrogation to GridPot, which responded by supplying its IOAs, and began updates as described in Section IV.B.1 and IV.B.2.d. We verified that commands were passed correctly to GridPot and produced changes in the simulated devices that were reported back to the SCADA application.

b. Simulator Configuration and HMI Design

We did not simulate the full IEEE 13-Node Model because testing showed it slowed GridPot's operation unacceptably. Instead, the Phase 2 configuration used a voltage regulator, a switch, and seven power meters for residential customers.

We created two main displays within our SCADA application. The first display reported the state of all simulated devices in the system. The second showed the status of one specific simulated device, and allowed the intruder to send commands to that device.

2. SCADA Host and GridPot Configuration

The second category of work needed was to configure both the SCADA and GridPot virtual machines for Phase 2 operation. The SCADA host's configuration enabled access to the system by the remote desktop Guest account, which by default lacks a password [67] and is a popular target for attacks [68]. Providing an account without a password did require several changes to the local machine's security policy and to Windows firewall rules. Other than changes to the gpm file for our Phase 2 simulation model, no significant changes were made to GridPot between Phase 1 and Phase 2.

E. PHASE 2 DEPLOYMENT

1. Host and Virtual Environments

The machines for Phase 2 testing and deployment were the same as those used in Phase 1. To reduce the chance of attackers realizing the two phases were related, we used a different IP address for our live environment machine during Phase 2 deployment than we had during Phase 1.

Our Phase 2 GridPot virtual machine was deployed on VirtualBox 5.2.22 with two virtual processors, 4096 megabytes of memory, and 50 gigabytes of storage. One virtual network-interface controller attached to the machine, which was in turn attached to a VirtualBox internal network equipped with a DHCP server to allow communication with the Windows virtual machine hosting the SCADA application. The latter had four virtual processors, 6052 megabytes of memory, and 40 gigabytes of storage. It was equipped with two virtual network interface controllers: one attached to the same VirtualBox internal network as the Phase 2 GridPot virtual machine that provided communications with the GridPot virtual machine, and one a bridged adapter with its MAC address the same as the MAC address of the host machine network-interface controller that provided Internet connectivity. Since the Windows host could not respond to HTTP traffic, this controller forwarded incoming traffic addressed to TCP port 80 to the internal network-facing controller, and forwarded traffic from TCP port 80 on that controller back to the bridged adapter. This allowed the GridPot HTTP server to respond to these requests. Once both machines were running, we determined the GridPot virtual machine's IP address on the internal network and set the SCADA application's IEC 104 virtual device to connect to that address.

2. Data Collection

Data was collected for Phase 2 over two periods, both of which ended in system compromise. The first run was May 26 to June 7, 2020, and the second was June 17 to June 29. We analyzed three sources of data for Phase 2: PCAP packet captures, Conpot logs, and Windows event logs. Packet captures and Conpot logs were collected from the outset, while the Windows event logs were added later.

a. Wireshark Packet Captures

We deployed Wireshark on the Windows virtual machine to collect packets from the Internet-facing bridged adapter. As with Phase 1, we collected packets to or from TCP ports 80 (HTTP) and 3389 (RDP). We did not collect data from port 2404 during this phase, since the Internet-facing virtual machine did not have that port open. Wireshark saved its PCAP packet data every three hours. After our data collection issues on our first deployment of Phase 2, we also ran Wireshark on the GridPot virtual machine. This secondary source of data was less useful since all traffic to the GridPot machine passed through the Windows virtual machine, and thus lost the remote IP address originating the communications.

Since the Microsoft Remote Desktop Protocol is an encrypted protocol, we used the security tool Mimikatz and a procedure found at [69] to extract the Windows host's encryption keys, allowing Wireshark to capture and store remote desktop traffic in the clear.

b. Conpot Logging

As with Phase 1, Conpot's text logger on the GridPot host was secondary data source, although its usefulness was decreased because it too could not record the remote IP originating the traffic.

c. Windows Event Logs

We gathered the following logs for each live run:

- Windows System
- Windows Security
- Windows Firewall
- Microsoft-Windows-Terminal Services-LocalSessionManager
- Microsoft-Windows-Terminal Services- RemoteConnectionManager

- Microsoft-Windows-User Profile Service- Operational
- Microsoft-Windows-WindowsDefender-Operational
- Microsoft-Windows-User Profile Service-Operational

Both Phase 2 live runs ended with the discovery that an attacker had accessed the remote desktop Guest account and used it for malicious activities. After the first system compromise, which installed crypto-mining software in the Guest user's Desktop directory, we restricted the Guest's ability to write anywhere on the system drive. Unfortunately, this did not stop a second system compromise. During both compromises, the remote user halted Wireshark packet collection Although the data loss made detailed data analysis like Phase 1's impossible, we used the remaining packets and event logs to reconstruct what happened.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS AND DISCUSSION

This chapter discusses our analysis of the traffic patterns we observed on the Phase 1 honeypot and the system compromises that ended both runs of our Phase 2 honeypot.

A. SHODAN HONEYSCORE

When we used the Shodan command-line interface to request a Honeyscore evaluation of our live Phase 1 GridPot, it returned a score of 0.0, indicating that the algorithm found a 0% probability of Phase 1 GridPot being a honeypot. A score of 1.0 (or 100% probability of being a honeypot) was received by the previous version of GridPot [13], indicating that our version of GridPot is substantially better at evading a commonly used automated honeypot detection tool.

We also used the Shodan application programming interface (API) to request information on our deployed GridPot as well as on the host on which the older version of GridPot was deployed in [13]. When we requested host information on our deployed GridPot, Shodan did not identify it as a honeypot regardless of whether or not the IP's history was included in the results. The previous version of GridPot [13] was not tagged as a honeypot when IP history was excluded from the results, but was identified as a honeypot when IP history was included [63]. We conclude that our version of GridPot is better at evading this Shodan-based detection mechanism.

B. PHASE 1 RESULTS

1. Overall Traffic Characterization

Phase 1's first run (hereafter "Run 1") lasted 111 days and recorded traffic from 2,276 unique IP addresses from 121 countries. Phase 1's second run (hereafter "Run 2") lasted 46 days and recorded traffic from 588 unique IP addresses from 67 countries. By comparison, the single run by our research team of the previous version of GridPot (referred to as the "MZ dataset" here) lasted 19 days and recorded traffic from 508

unique IP addresses from 72 countries [13]. This is consistent with previous honeypot work which found that the Modbus and S7comm protocols implemented by previous versions of GridPot attracted more attention than IEC 104 [24].

a. IP Addresses with a Single Session versus. IP Addresses with Multiple Sessions

We compared the remote IP addresses that had only connected to our honeypot for a single session with remote IP addresses that had connected for more than one session. See Section IV.C.4.b for the definition of "session" used for this study.

As Figure 7 shows nearly all IP addresses only connected to the HTTP server once. Figure 7 also shows that only about 3% of IPs, the ones shown under the "Other" category in Figure 7, connected to the IEC 104 server on the Phase 1 runs. These statistics were consistent for the two Phase 1 runs. By contrast, the MZ dataset had a higher percentage of IP addresses connecting to their ICS servers, with slightly over 10% of their IP addresses connecting to ICS ports. This further supports the observation that IEC 104 is markedly less popular with honeypot scanners than Modbus and S7comm.

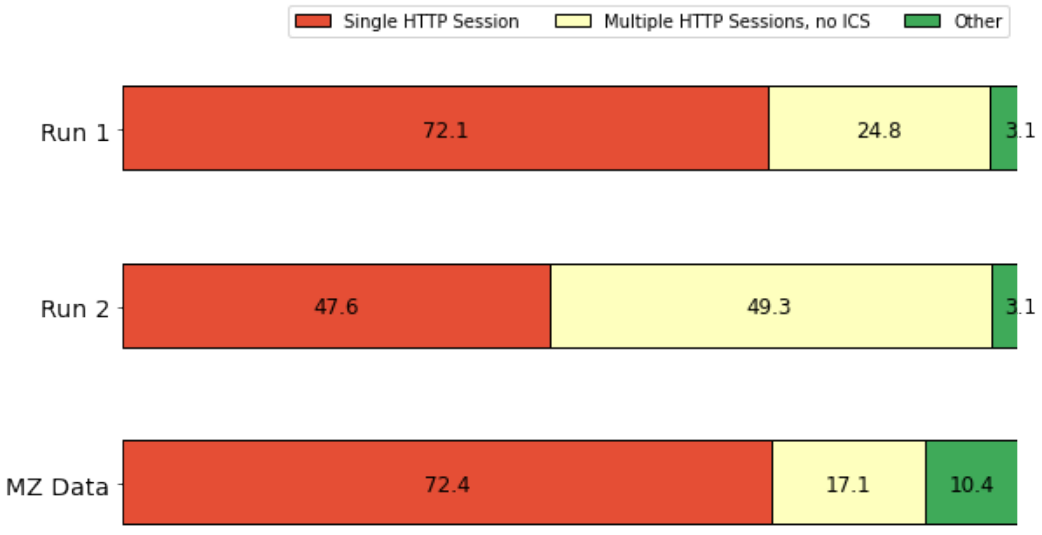

Figure 7. Percentage of IP Addresses by Number and Type of Sessions

Figure 8 shows the breakdown of the IP addresses grouped into "Other" in Figure 7. Interestingly, both Phase 1 runs had a much lower percentage of IP addresses connecting to ICS ports than the MZ data set, and a much higher percentage of those IP addresses connecting to both HTTP and ICS over the lifetime of the honeypot. This phenomenon needs to be observed further before we can hypothesize its cause.

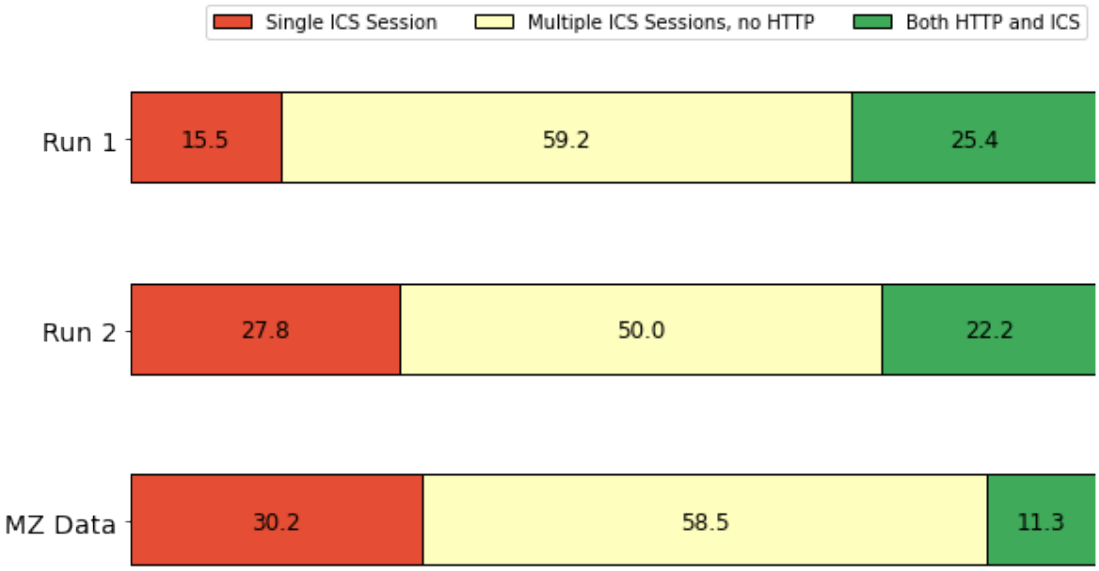

Figure 8. Breakdown of "Other" Category from Figure 7

We also studied whether the number of requests made per session between IP addresses that only conducted a single session for HTTP or IEC 104 was different from that of IP addresses that conducted multiple sessions for either protocol (Tables 2 and 3). We applied the resulting means and standard deviations to a two-tailed, two-sample T-test, a standard way of determining whether the means of two samples significantly differ [70]. For this test, a p value less than or equal to 0.05, meaning there is a 5% or lower chance that the variation between the two means could have happened by chance, is usually considered significant. The p-values shown in Tables 2 and 3 indicate that there was a statistically significant difference in the number of requests per session, with IP addresses contacting the honeypot multiple times making more requests per session than IP addresses that only connected to the honeypot once.

(Each p-value on Tables 2 and 3 refers to the comparison between a set of single-session IP addresses and the corresponding set of multiple-session IP addresses; thus we only show one p-value per pair.) This suggests that most single-session IP addresses conducted relatively simple port scans, while IP addresses that connected multiple times are more likely to perform a detailed investigation of the open port. Due to the technical problems with Conpot logging discussed in Section IV.C.3.b, we were unable to confirm or disprove this hypothesis.

Table 2. Requests per Session for HTTP, Run1, Run 2, and Combined

	Mean	Standard Deviation	Number of Addresses	T-Statistic	P-Value
Single-Session IP Addresses, Run 1	2.107	37.86	1767	-8.547	2.311×10^{-17}
Multiple-Session IP Addresses, Run 1	40.96	176.26	467	--	--
Single-Session IP Addresses, Run 2	1.44	0.78	468	-4.164	3.612×10^{-5}
Multiple-Session IP Addresses, Run 2	26.82	131.65	106	--	--
Single-Session IP Addresses, Combined	1.966	33.666	2235	-9.468	5.853×10^{-21}
Multiple-Session IP Addresses, Combined	38.347	168.987	573	--	--

Table 3. Requests per Session for IEC 104, Run 1, Run 2, and Combined

	Mean	Standard Deviation	Number Observed	T-Statistic	P-Value
Single-Session IP Addresses, Run 1	1.7	0.936	30	-5.466	1.014×10^{-6}
Multiple-Session IP Addresses, Run 1	15.0	13.07	30	--	--
Single-Session IP Addresses, Run 2	1.6	0.917	10	-3.429	0.003
Multiple-Session IP Addresses, Run 2	17.0	13.35	8	--	--
Single-Session IP Addresses, Combined	1.675	0.932	40	-5.466	1.014×10^{-6}
Multiple-Session IP Addresses, Combined	15.421	13.154	38	--	--

b. Inter-Session Times for IP Addresses with Multiple Sessions

We also studied how intersession times varied between HTTP and ICS protocols. We were interested to see if those multiple sessions from a single IP address were conducted in rapid sequence, which might indicate short but intense scanning on one occasion.

Our analysis showed substantial differences between the two protocols we examined and between the datasets. Both Run 1 (Figure 9) and the MZ dataset (Figure 11) showed that most HTTP sessions occurred fairly close to one another (10 seconds or less), with a substantial "tail" of sessions occurring further apart. However, in Run 2 (Figure 10) more sessions occurred 10,000 to 100,000 seconds apart (between three and thirty hours). More investigation is needed to determine whether other GridPot deployments see a similar effect as they spend more time online. For ICS protocols, both Phase 1 runs and the MZ dataset showed most sessions 10 seconds or less apart. Although there does appear to be more of a "tail" in the MZ dataset, we do not believe we have sufficient data to hypothesize

on its significance. Overall, it seems that most traffic was relatively short, intense scanning, with evidence of more repeat HTTP visits on Run 2.

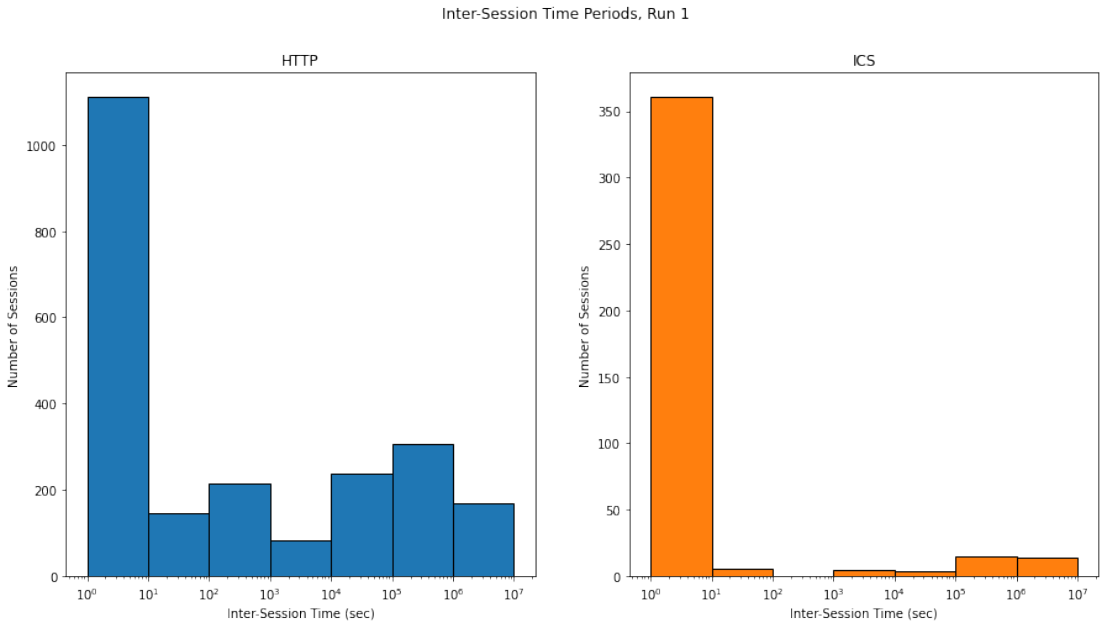

Figure 9. Inter-Session Time Periods for Run 1

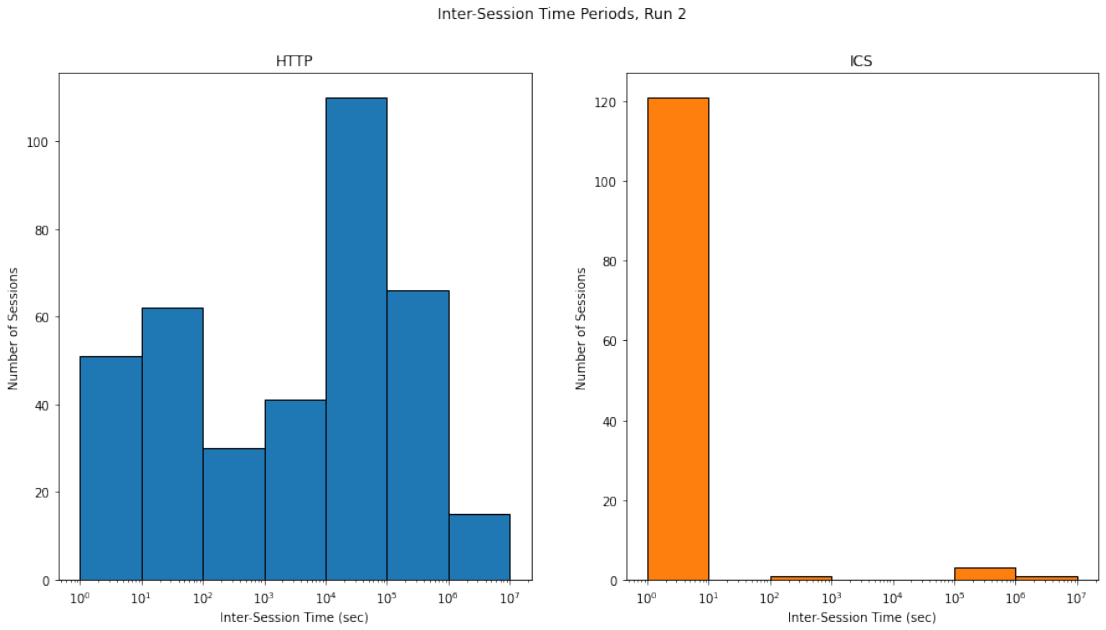

Figure 10. Inter-Session Time Periods for Run 2

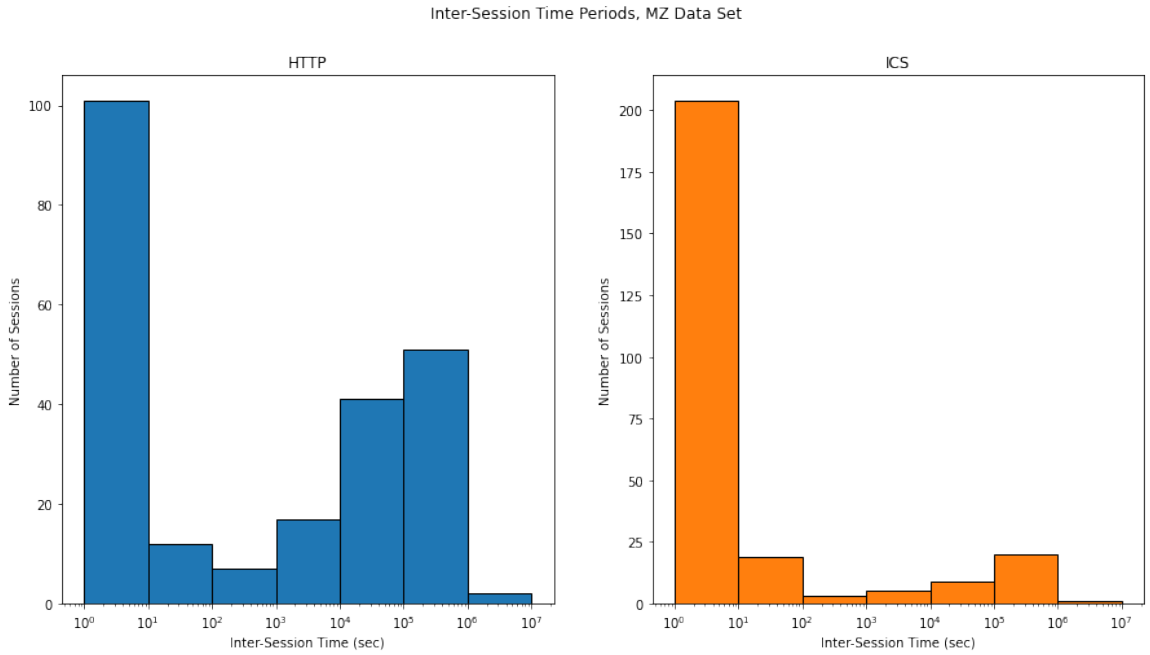

Figure 11. Inter-Session Time Periods for MZ Dataset

c. Distribution of IP Addresses by Country

We next investigated how the remote IP addresses broke down by country. The results, shown in Tables 4 and 5, show an interesting difference between the two Phase 1 runs. For Run 1, countries contributing 2% or more of the total number of IP addresses comprised 63.7% of the total number of IP addresses, while for Run 2 they comprised 73.0%. The countries contributing also varied significantly between the two runs. For example, Taiwan, the 7th-largest contributor for the first run with 3.0% of total IP addresses, was by far the largest contributor for the second run with 21.9% of total IP addresses. Nevertheless, both runs recorded the United States, Russia, and China as three of the top countries in terms of number of IP addresses. More research is needed to determine whether this variation between runs is normal.

Table 4. Countries Contributing 2% or More of Total IP Addresses, Run 1

Country	Percentage of Total
United States	18.68
Brazil	8.97
China	8.22
Russia	5.98
India	3.34
Italy	3.21
Taiwan	3.03
Iran	2.95
Germany	2.55
Mexico	2.29
Netherlands	2.29
Ukraine	2.2

Table 5. Countries Contributing 2% or More of Total IP Addresses, Run 2

Country	Percentage of Total
Taiwan	21.94
United States	15.31
China	7.14
Russia	5.95
Brazil	5.27
Germany	5.1
India	3.06
Iran	2.38
Netherlands	2.38
Indonesia	2.21
Romania	2.21
Other	27.05

d. Incoming Requests per Day by Protocol

Figures 12 and 13 show the number of incoming requests per day for both Phase 1 runs in graphical form, where "IEC 104 Traffic" counts all traffic using IEC 104's well-known TCP port (2404) whereas "IEC 104 Messages" only counts traffic with correctly formatted IEC 104 data. HTTP traffic was recorded on most days when Phase 1 was active, although the amount varied greatly between days. By contrast, IEC 104 traffic had some gaps spanning multiple days during which no IEC 104 traffic at all was recorded.

Tables 6 and 7 present basic statistics for the number of incoming requests per day for both runs as well as for the MZ dataset. For the two Phase 1 runs in Table 7, the "All" row counts all traffic directed to TCP port 2404, while the "Valid" row counts only traffic with correctly formatted IEC 104 data units. For the MZ data in Table 7, the row indicates the combined number of incoming requests for either of the honeypot's open ICS protocol ports.

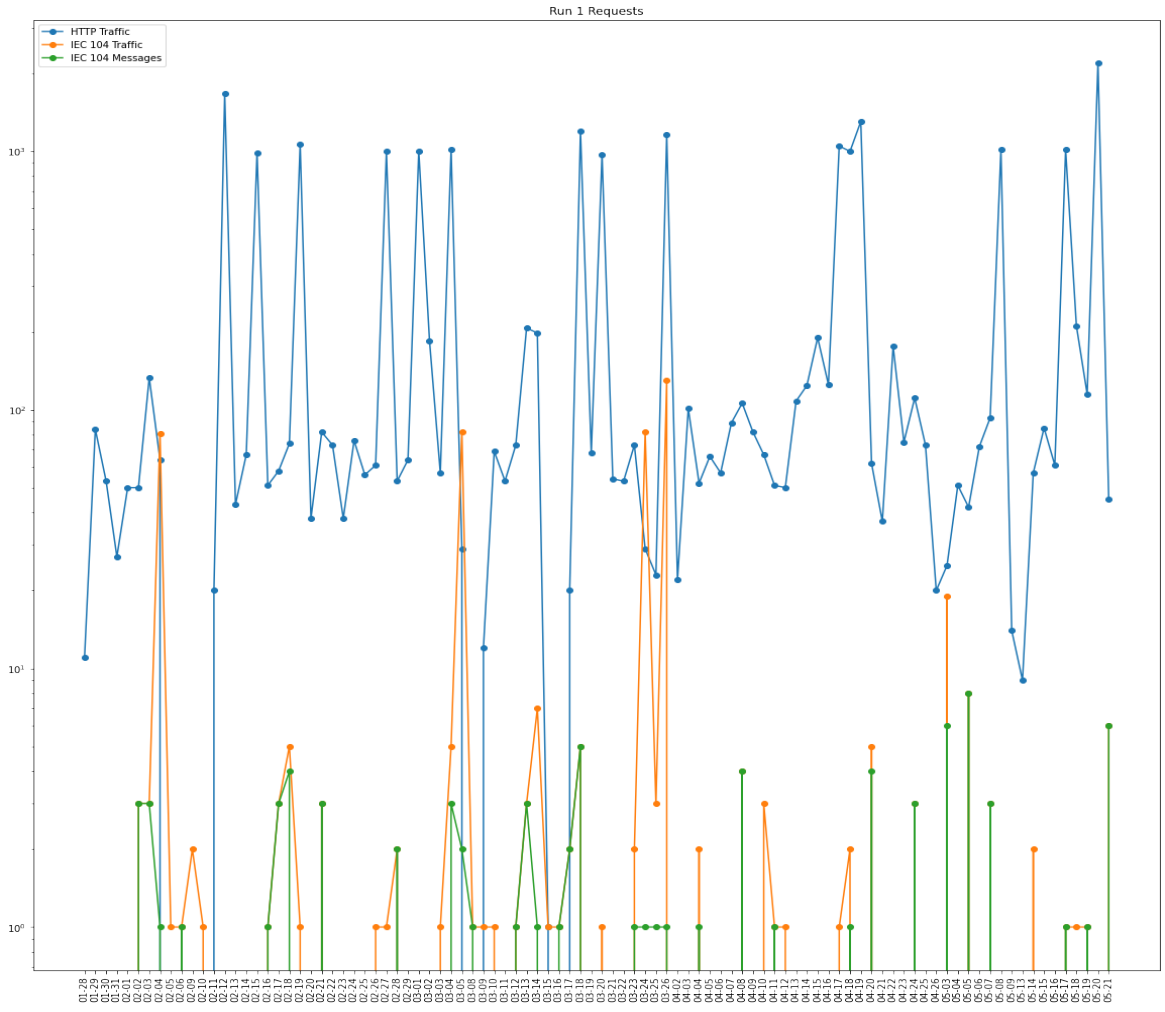

Figure 12. Requests per Day for Run 1

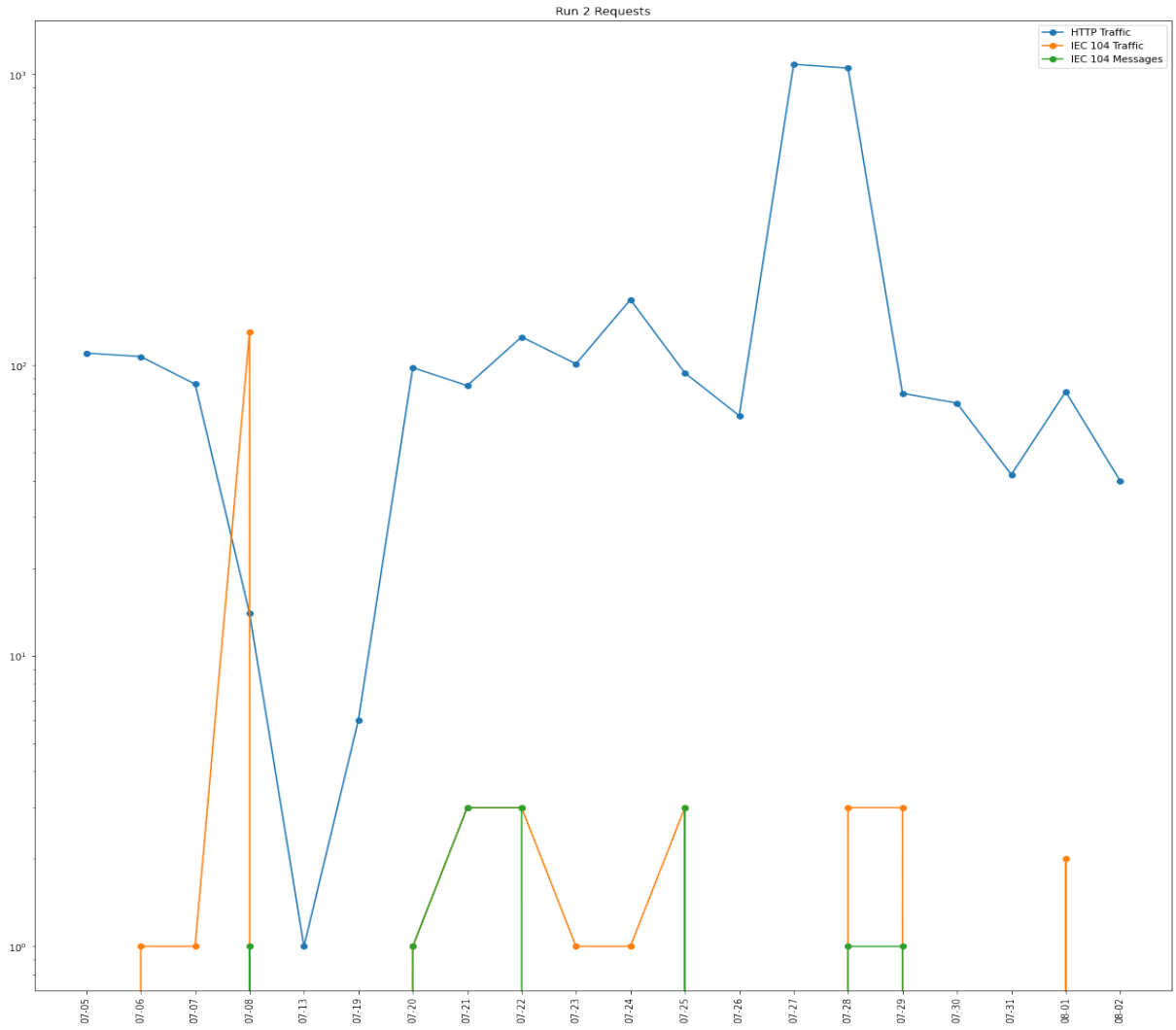

Figure 13. Requests per Day for Run 2

Table 6. Basic Statistics for Incoming HTTP Requests per Day

	Minimum	Median	Mean	Standard Deviation	Maximum
Run 1	0	67.5	253.92	433.91	2191
Run 2	0	85.5	175.75	300.1	1084
MZ	8	519.5	481.15	444.52	1185

Table 7. Basic Statistics for Incoming ICS Requests per Day

	Minimum	Median	Mean	Standard Deviation	Maximum
Run 1 - All	0	2	10.02	25.55	130
Run 1- Valid	0	2	2.44	1.77	8
Run 2 - All	0	2.5	12.67	35.39	130
Run 2- Valid	0	1	1.86	0.99	3
MZ	8	43.0	244.0	681.26	3079

Although the wide deviations from the mean prevent us from drawing definite conclusions, we can say that the MZ version of the honeypot drew more attention because it had popular ICS protocols despite the history of its site. Second, the increased popularity of the MZ honeypot's ICS portion correlates with the increased popularity of the HTTP portion. Finally, most traffic directed to IEC 104 ports contained invalid messages and probably represented basic port scanning.

e. Cosine Similarity Analysis of Traffic

We did cosine similarity analysis [71] on the proportion of HTTP traffic versus. IEC 104 traffic to test for a significant difference between our data and the MZ set as well as for the variation between weeks of our dataset. We compared two-component vectors, with the first number in the vector representing the proportion of traffic over the HTTP protocol and the second the proportion over ICS protocols. Proportions for the MZ dataset were taken from [63]. The result is a coefficient of similarity between the two vectors, with 1.0 representing identical vectors and 0.0 representing completely orthogonal vectors.

Cosine similarity analysis did not show a statistically significant difference between our data and the MZ dataset, with the two vectors having a similarity of 0.999. This indicates that, despite the higher overall proportion of ICS traffic in the MZ dataset, the two datasets do not differ significantly in the type of traffic they recorded.

The last column of Table 8 shows the deviation from the mean's values for each week, which lets us to identify periods in which traffic coming into the honeypot changed

significantly. For example, during Weeks 18 and 28 the proportion of IEC 104 traffic coming into the honeypot spiked markedly but returned to relatively normal in Weeks 19 and 29. The return to normal proportions caused another elevated standard deviation score, since those weeks still differed substantially from Weeks 18 and 28. This could indicate intensive scanning of the honeypot by someone interested in IEC 104.

Table 8. Cosine Similarity Between Proportions of HTTP and IEC 104 Traffic Recorded in Each Week of Phase 1

Week	HTTP Proportion	IEC 104 Proportion	Coefficient of Similarity	Standard Deviations From Mean
5	0.989	0.011	nan	nan
6	0.691	0.309	0.917	0.556
7	0.999	0.001	0.913	0.612
8	0.992	0.008	1.0	0.603
9	0.998	0.002	1.0	0.603
10	0.935	0.065	0.998	0.575
11	0.978	0.022	0.999	0.589
12	0.996	0.004	1.0	0.603
13	0.855	0.145	0.987	0.422
14	0.992	0.008	0.987	0.422
15	0.982	0.018	1.0	0.603
16	0.999	0.001	1.0	0.603
17	0.986	0.014	1.0	0.603
18	0.568	0.432	0.805	2.12
19	0.992	0.008	0.801	2.176
20	0.998	0.002	1.0	0.603
21	0.997	0.003	1.0	0.603
27	1.0	0.0	1.0	0.603
28	0.611	0.389	0.843	1.589
29	1.0	0.0	0.843	1.589
30	0.984	0.016	1.0	0.603
31	0.997	0.003	1.0	0.603

When we combined Table 8's data with a similar analysis conducted on the proportion of IEC 104 message traffic to IEC 104 malformed traffic recorded each week shown in Table 9, we can further differentiate traffic patterns. Week 28, one of the weeks that showed a spike in IEC 104 traffic in Table 8, also had significantly more malformed IEC 104 traffic than the norm. This indicates that any intensive scanning during Week 28 was either brute-force port scanning by an intruder uninterested in IEC 104, or deliberate fuzzing aimed at testing the IEC 104 server's response to malformed inputs. However, similar spikes in the proportion of malformed traffic in Weeks 6, 10, and 16 did not affect the proportion of IEC 104 traffic in Table 8, possibly indicating less intensive brute-force scanning.

By contrast, the spike in the proportion of IEC 104 traffic shown in Week 18 in Table 8 was accompanied by an increase in malformed traffic in Table 9 that, while significant, was not as unusual in terms of changes from previous weeks, indicating that this time the increase in traffic included more correctly formed IEC 104 messages. This could indicate scanning from a more protocol-aware intruder interested in finding actual IEC 104 servers.

Table 9. Cosine Similarity Between Proportions of IEC 104 Message Traffic vs. IEC 104 Malformed Traffic Recorded in Each Week of Phase 1

	Message Proportion	Malformed Proportion	Coefficient of Similarity	Standard Deviations From Mean
5	1.0	0.0	nan	nan
6	0.057	0.943	0.06	1.819
7	0.5	0.5	0.748	0.621
8	0.833	0.167	0.832	0.919
9	0.5	0.5	0.832	0.919
10	0.067	0.933	0.756	0.649
11	0.357	0.643	0.907	1.184
12	0.889	0.111	0.59	0.06
13	0.018	0.982	0.143	1.525
14	0.5	0.5	0.72	0.521
15	0.556	0.444	0.994	1.493
16	0.333	0.667	0.908	1.188
17	0.875	0.125	0.569	0.014
18	0.316	0.684	0.543	0.106
19	1.0	0.0	0.419	0.546
20	0.333	0.667	0.447	0.447
21	0.875	0.125	0.569	0.014
28	0.008	0.992	0.149	1.503
30	0.833	0.167	0.204	1.308
31	0.25	0.75	0.496	0.273

2. HTTP Traffic Characterization

Tables 10–12 show the source-country distribution of HTTP sessions established for both Phase 1 runs and the MZ dataset. Countries that contributed at least 2.0% of the total sessions were broken out individually, while all other countries were grouped into the “Other” category. All three datasets had the United States, Russia, and China as the top three countries by number of sessions.

Table 10. Source Countries Representing 2% or More of HTTP Sessions, Run 1

Country	Percentage of Total
China	24.68
United States	16.87
Russia	11.23
Brazil	4.1
Taiwan	3.06
Netherlands	2.87
Germany	2.65
Other	34.54

Table 11. Source Countries Representing 2% or More of HTTP Sessions, Run 2

Country	Percentage of Total
Russia	16.88
United States	14.76
China	12.97
Taiwan	11.99
Netherlands	6.53
Germany	4.4
Brazil	2.69
Romania	2.37
Seychelles	2.2
Luxembourg	2.2
Other	23.01

Table 12. Source Countries Representing 2% or More of HTTP Sessions, MZ Dataset

Country	Percentage of Total
United States	15.54
China	13.65
Russia	11.49
Brazil	10.95
India	4.59
Netherlands	3.38
Japan	3.38
Iran	3.38
Estonia	2.57
France	2.03
Other	29.04

Figure 14 shows the distribution of the method requested by incoming HTTP requests for both Phase 1 runs and the MZ dataset. Our version of GridPot attracted a much higher proportion of HTTP requests without a valid method (shown as “None” on the chart), as well as a lower ratio of POST requests to GET requests, although we cannot be certain whether this effect is statistically significant without more data.

Figure 14. Distribution of HTTP Method Types by Dataset

3. IEC 104 Traffic Characterization

Tables 13–15 show the distribution of ICS sessions by countries for both Phase 1 runs as well as the MZ dataset. For Tables 13 and 15, countries that contributed at least 1.0% of the total number of ICS sessions are broken out individually, while other countries were grouped into the "Other" category. Table 14 shows all countries contributing sessions, since only one country contributed less than 1.0% of the total number of ICS sessions for Run 2. Our version of GridPot attracted a much higher proportion of sessions from IP addresses in foreign countries, indicating that IEC 104-aware honeypots may be of interest to researchers in certain areas despite the fewer sessions they attract.

Table 13. Countries Contributing 1% or More of IEC 104 Sessions, Run 1

Country	Percentage of Total
United States	47.07
Germany	24.3
United Kingdom	14.75
China	7.59
Romania	1.95
Other	4.34

Table 14. Countries Contributing IEC 104 Sessions, Run 2

Country	Percentage of Total
Singapore	27.27
Germany	25.87
United States	21.68
United Kingdom	9.09
Canada	8.39
Russia	2.8
Romania	2.1
China	2.1
South Korea	0.7

Table 15. Countries Contributing 1% or More of ICS Sessions, MZ Dataset

Country	Percentage of Total
United States	70.17
Romania	9.94
Japan	7.4
United Kingdom	3.7
China	1.5
Russia	1.39
Hong Kong	1.39
Netherlands	1.16
Other	3.35

Figures 15 and 16 show the fraction of malformed IEC 104 packets for both Phase 1 runs. Nearly all incoming requests to port 2404 were invalid IEC 104 messages. Examination of PCAP files revealed that these packets were usually either messages formatted for another protocol (mainly HTTP) but sent to the IEC 104 port, or what appeared to be random bytes. Of the valid IEC 104 messages, most were U-frames, indicating simple start/stop messages or connectivity checks. Only a tiny fraction of incoming IEC 104 messages contained the I-frames intended for deeper interaction.

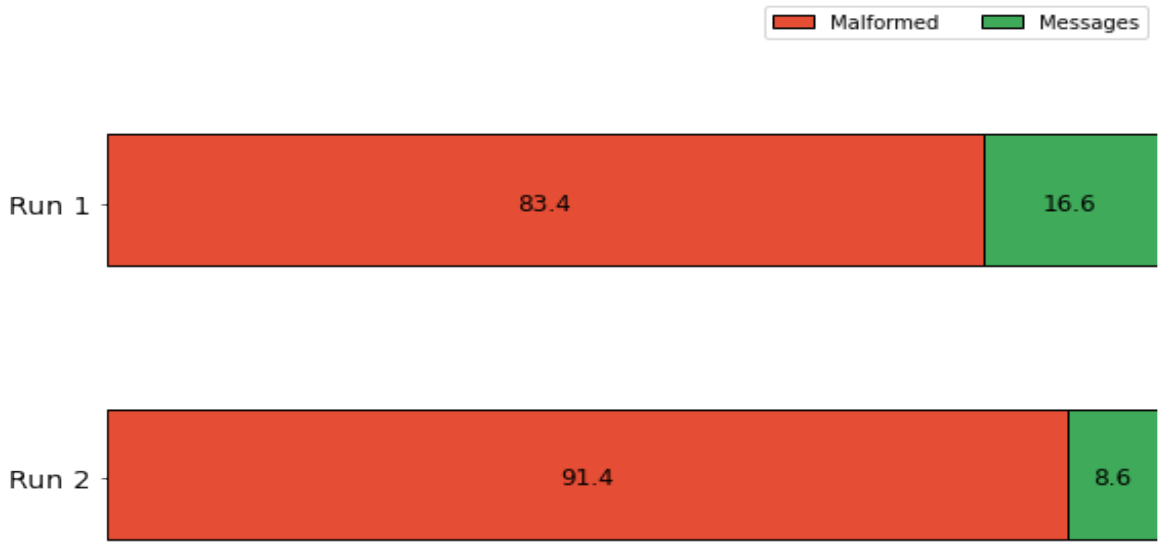

Figure 15. IEC 104 Distribution of Malformed Packets vs. Valid Messages

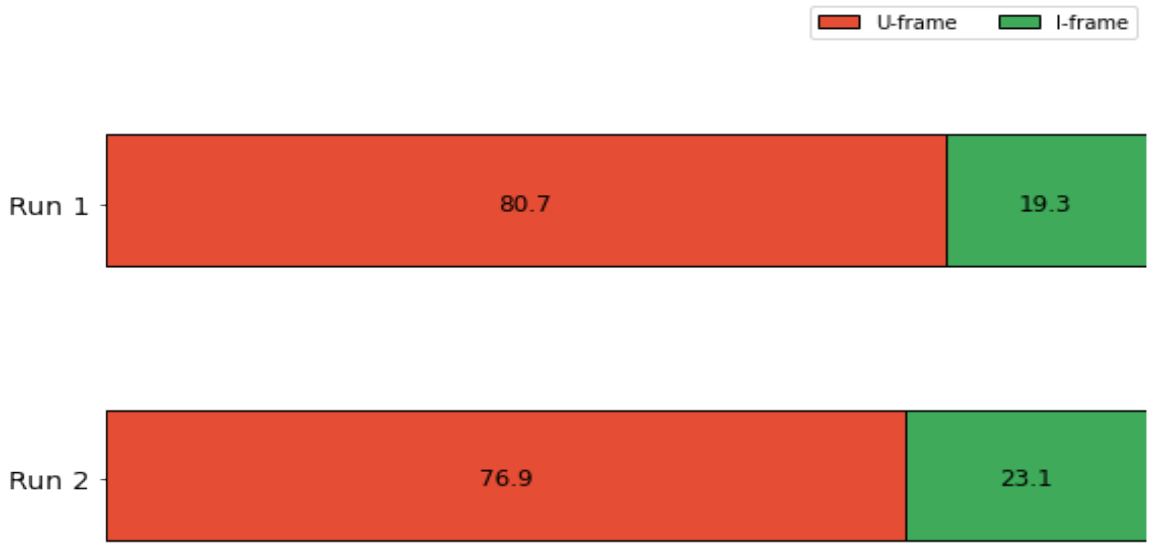

Figure 16. IEC 104 Message Frame Types

Table 16 shows behaviors among the top 10 IP addresses for the IEC 104 protocol. Specifically, three IP addresses submitted identical sequences of 39 malformed payloads, (Pattern A in Table 16) even though two were located in the United States and the third in Singapore. Two other IP addresses submitted a sequence of 38 malformed payloads that

differed from the first only by omitting a malformed cookie submission that was in the third from last position of Pattern A (Pattern B in Table 16), although one was in the United States and the other in Germany. Three of the five IP addresses showing patterns A and B were from the 172.104.0.0/16 IP address block, and two of those three were registered to the domain members.linode.com, suggesting a common origin.

The remaining five IP addresses in Table 8 all submitted slightly different sequences of payloads, which still had many payloads in common with Patterns A and B. (Pattern C in Table 16) They did, however, include a single valid IEC 104 message, a U-frame containing an activation command, and contained at least some common requests. This suggests a common scanning or fuzzing application with options that can slightly vary its output. Most of these IP addresses were also registered to members.linode.com, indicating a possible common origin for Patterns A, B, and C.

Table 16. Patterns of IEC 104 Payloads Among Top 10 IP Addresses by Number of Requests

Address	Country	Messages	Malformed	Pattern	Domain Name
104.237.128.197	United States	0	39	A	members.linode.com
172.104.25.33	United States	0	39	A	members.linode.com
172.104.174.197	Singapore	0	39	A	li.binaryedge.ninja
45.79.130.160	United States	0	38	B	members.linode.com
172.104.144.136	Germany	0	38	B	members.linode.com
139.162.170.48	Germany	1	32	C	li.binaryedge.ninja
209.97.180.161	United Kingdom	1	30	C	None
139.162.153.48	Germany	1	30	C	members.linode.com
172.105.76.241	Germany	1	30	C	members.linode.com
45.79.77.87	United States	1	30	C	li.binaryedge.ninja

4. Characterization of IP Addresses Conducting Deeper Interaction by IEC 104

During Phase 1, 17 IP addresses sent IEC 104 messages with valid ASDU data to GridPot, as shown in Table 17. The data in Table 17 are sorted from highest IP address to lowest. The pattern for each address was similar: the host did a three-way handshake with TCP port 2404, exchanged U-frames with the server to start data transfer, and then sent a single I-frame containing an IEC 104 general interrogation message. GridPot provided data from the GridLAB-D simulation, and continued updating the remote host until the IEC 104 server timed out, whereupon activity ceased. The remote hosts did not respond to GridPot's messages or end the TCP connection with a three-way signoff. We conclude that Phase 1 apparently did not attract visitors that would notice or be affected by our GridPot's increased responsiveness.

Table 17. Addresses Sending Valid 104 ASDU Messages to GridPot

IP Address	Country	Domain Name	Number of Sessions
198.20.87.98	United States	border.census.shodan.io	2
198.20.70.114	United States	census3.shodan.io	1
193.37.255.114	Slovakia	No Record	2
185.142.236.34	Netherlands	hat.census.shodan.io	2
185.165.190.34	Russia	red.census.shodan.io	1
185.234.219.205	Poland	No Record	1
185.181.102.18	Romania	turtle.census.shodan.io	1
93.174.95.106	Netherlands	battery.census.shodan.io	1
82.221.105.6	Iceland	census10.shodan.io	2
80.82.77.33	United Kingdom	sky.census.shodan.io	2
80.82.77.139	United Kingdom	dojo.census.shodan.io	1
71.6.146.186	United States	inspire.census.shodan.io	2

IP Address	Country	Domain Name	Number of Sessions
71.6.158.166	United States	ninja.census.shodan.io	1
71.6.135.131	United States	ubuntu20135131.aspadmin.net	1
66.240.236.119	United States	census6.shodan.io	1
66.240.192.138	United States	census6.shodan.io	1
43.245.222.176	Hong Kong	No Record	1

5. Behavioral Analysis

We conducted basic behavioral analysis on IP addresses that sent IEC 104 traffic to the honeypot. Our first step was to determine whether these addresses could be usefully categorized. Figure 17 shows that 87.7% of IP addresses sending IEC 104 traffic used either only malformed IEC 104 messages or only valid IEC 104 messages. The former could indicate either fuzzing with random bytes or probes of port 2404 with other protocol data, and the latter could be attempts to contact the IEC 104 servers.

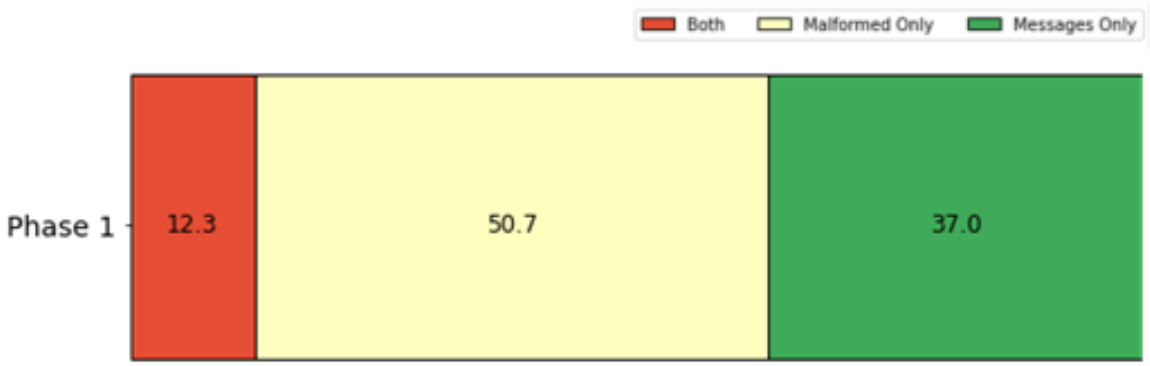

Figure 17. IP Addresses by Type of IEC 104 Traffic Submitted

Of the nine IP addresses that submitted both valid and malformed IEC 104 messages to our honeypot, shown on Table 18 five were the addresses grouped under Pattern C on Table 16. Of the four remaining IP addresses, one (173.255.249.78) only

contacted the honeypot a single time, sending a TCP SYN packet followed immediately by a U-frame with an activation command and no more transmissions. We are uncertain as to why this scanning activity was not repeated, but it may have been part of a horizontal scan of a large number of network blocks for open IEC 104 ports.

Table 18. IP Addresses Submitting Both Valid and Malformed IEC 104 Traffic

IP Address	HTTP Requests/ Sessions	IEC 104 Requests/Sessions	Remarks
139.162.170.48	1	30	Pattern C from Table 16, probes on 7/8/20
209.97.180.161	1	30	Pattern C from Table 16, probes on 3/5 and 3/9/20
139.162.153.48	1	30	Pattern C from Table 16, probes on 3/24 and 3/25/20
172.105.76.241	1	30	Pattern C from Table 16, probes on 2/8/20
45.79.77.87	1	30	Pattern C from Table 16, probes on 3/26 and 4/2/20
173.255.249.78	1	2	Only contacted honeypot once, 5/3/20
122.228.19.80	5	2	Probes on 2/18, 4/18, 5/3, 5A/5, and 7/20/20
223.71.167.166	10	4	Probes on 2/28, 3/4, 4/12, 4/4, 4/8, 4/20, 5/3, 5/5, 5/ 19, 7/2, and 7/28/20
164.52.24.178	1	6	Probes on 3/17/20

The three remaining IP addresses all began by sending a TCP SYN packet, then a TCP RST packet upon receiving the honeypot's SYN+ACK packet. From there, their requests diverged. Two of the three addresses, 122.228.19.80 and 223.71.167.166, submitted a series of probes over a long time period, each consisting of a TCP 3-way handshake followed by an IEC 104 activation U-frame, which the server responded to with

an IEC 104 confirmation U-frame. Both addresses also submitted a spurious interrogation response I-frame once, which was dropped by the server. 223.71.167.166 also attempted to submit random bytes to port 2404, which were also dropped by the server.

The last IP address, 164.52.24.178, completed its activity within a single day, 3/14/20. It did not complete a TCP 3-way handshake before sending any of its IEC 104 traffic, so none of its requests received a response from the server. After first making contact, it submitted a number of TCP packets directed to port 2404 that had payloads consisting of random bytes, possibly an attempt at fuzzing. It then submitted an activation U-frame, then another with crafted errors in a possible attempt to see how the IEC 104 server would respond. It then terminated communications.

Taken together, the behavioral similarities between these three addresses indicate a common scanning method mixing correctly and incorrectly formatted IEC 104 commands. Of particular interest is the long gaps between sessions for 122.228.19.80 and 223.71.167.166, indicating periodic rescanning possibly coupled with an attempt at fuzzing or exploiting a known vulnerability of a commercial IEC 104 server. The fact that 164.52.24.178 completed its activity comparatively quickly indicates that its focus may have been more oriented towards fuzzing or vulnerability exploitation of the server.

C. PHASE 2

As discussed in Chapter IV, we ended both Phase 2 runs when an attacker took control of the Windows virtual machine and stopped packet capture. However, examination of Windows logs enabled a reasonable reconstruction of what happened on each run.

1. Phase 2 First Run

The first Phase 2 run began on 5/26/2020. Our first indicator of compromise came on 6/7/2020 when we discovered a process called XMRig Miner using 100% of our CPU. The screenshot of the Guest desktop is shown in Figure 18. Further investigation established that packet capture had been ended, leaving only a few entries from 5/26 that recorded some HTTP traffic and one TCP SYN scan of the remote desktop port; and that a third-party software called Crack_by_NERO had been installed and used to password-

lock the Guest desktop. We also found a new folder on the Guest desktop called “xmrig-5.1.1,” containing the XMRig Miner executable, which appeared to be a crypto-currency miner, (Figure 18 shows a screenshot of XMRig Miner running), the desktop locker executable, and a then-unidentified executable called “SpoolerComp.exe.” The password locker achieved persistence by adding itself to the Windows Registry at HKEY_USERS/<Guest User ID>/Software/Microsoft/Windows/CurrentVersion/Run, which specifies programs that run automatically when a user logs in.

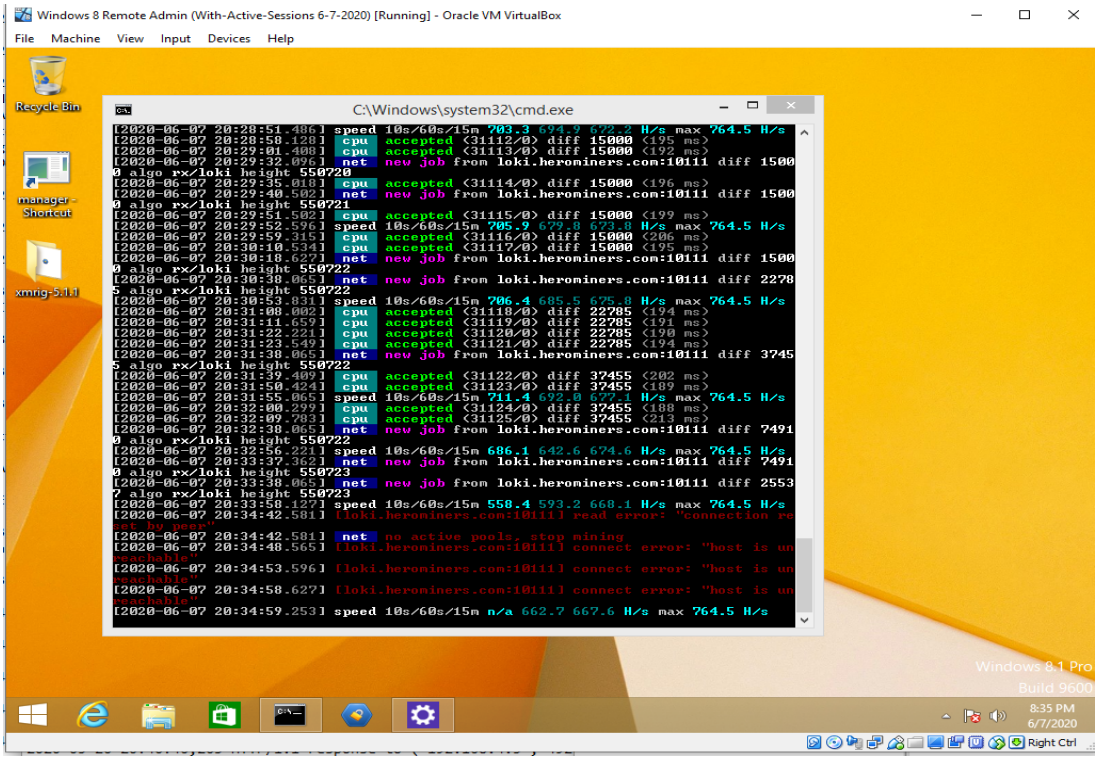

Figure 18. Guest Desktop at End of Phase 2 First Run

After viewing Windows logs, we inferred the following chain of events:

1. On 5/28/20 at about 9:25 AM (all times are local), the Windows Defender service noticed the file “SpoolerComp.exe” in the “xmrig-5.1.1” folder was infected with Win32/Neshta.A, a Trojan that uploads data from the infected system to a remote host. Simultaneously, the Microsoft-

Windows-Shell-Core-Operational log noted that Crack_by_NERO was added to the registry and the Windows application resolver cache. Therefore, we know the system was compromised no later than then, and it seems likely that all programs in the “xmrig-5.1.1” folder were installed together.

2. The last login recorded in the Microsoft-Windows-TerminalServices-LocalSessionManager log before this was a login to the Guest desktop by 5.121.24.239 at 9:23:07 AM. Notably, this login did not appear in the Microsoft-Windows-TerminalServices-RemoteConnectionManager-Admin log although it should have, possibly indicating the log was erased. On this basis, it seems likely, although not certain, that this session installed the illicit software.

3. An additional suspicious Guest desktop session began at 2:15 AM on 5/27/20, started by the IP address 174.34.157.86. When the session connected, the Administrator session we had left open to run Wireshark was ended and a new session started, supposedly from the local machine. The Guest session then enumerated the registry keys under Software/Microsoft/Windows/ CurrentVersion/Run, the same hive later used to give Crack_by_NERO persistence. While this is highly suggestive, we lack sufficient evidence to state that it is definitely connected to the incident in which the malware was installed.

Based on the information gathered above, we cleaned up the malware by deleting the “xmrig-5.1.1” folder on the Desktop and removing the malicious registry key.

2. Second Run

Our initial analysis of the first run concluded that the malware was installed in the Desktop folder because a Guest user had write access to it. We therefore disabled Guest’s write access to Desktop and other folders in their home directory, as well as setting our SCADA software to start at login to make it more apparent that the system was an ICS.

Phase 2 second run started on 6/17/2020. When we checked the honeypot on 6/29, we observed the following changes:

- Packet capture was no longer running.
- The Windows preinstalled Administrator account was visible despite it being set to invisible at startup.
- A new Administrator level account called “Admin” had been created.
- A new executable called “opera_portable_56.0.3051.36.exe” had been downloaded to the C:\Program Files directory, as well as several executables labeled as Opera installers.

On further investigation, we reconstructed these events:

1. The “Microsoft-Windows-User Profile Service – Operational” log showed events related to logging in the unauthorized “Admin” account on 6/25 at 5:27 AM (all times local). Therefore, the system was compromised by then.
2. The “Microsoft–Windows–TerminalServices–LocalSessionManager-UserEvents” log showed several logins to Guest from the IP address 88.200.137.9 between 5:27 AM and 5:34 AM. However, the “Microsoft–Windows–TerminalServices–RemoteConnectionManager–Admin” log, which should have logged all incoming remote desktop connections, had no record of a connection to that IP address being made, possibly indicating the log was erased.
3. Several other logins were recorded in the “LocalSessionManager–UserEvents” log for 88.200.137.9, before all logging to that file abruptly stopped at 8:37 AM.
4. At 5:36 AM, Windows Defender generated an alert about the opera_portable binary in a new directory called C:\ProgramData being infected with the Win32/Neshta.A virus, followed by a notification that

the file had been cleaned. At 5:44 AM, Windows Firewall rules were changed to block any incoming connections to a file at C:\ProgramData\opera\69.0.3686.36\opera.exe. It is not immediately clear why an attacker would block communications to malware they had installed,

5. At 4:53 PM, the "RemoteConnectionManager–Admin" log shows a Guest session was started, although the log contained no record of a remote desktop connection being made from any IP address. Also at 4:53 PM, Windows Defender generated an alert about a file at "C:\ProgramData\7z1900-x64.exe" being infected with Win32/Neshta.A and cleaned it. Shortly thereafter Windows Defender's anti-malware and anti-spyware real-time protection features were disabled.

We conclude that there were at least two penetrations of the system on 6/25. The first exploit used the Admin account and installed the infected opera.exe file, but was apparently thwarted by Windows Defender's anti-malware rules. This attack probably originated from the IP address 88.209.137.9. The second attack installed a different file in the same C:\ProgramData directory, which was again stopped by Windows Defender. Instead of giving up like the first time, someone or something deactivated Windows Defender and proceeded with the exploit. Although log erasure means we cannot say for certain whether these two attacks were related, the use of the same type of malware in the same unauthorized C:\ProgramData directory suggests it.

Interestingly, one remote user seemed to have been attracted by the SCADA interface. Conpot logs indicated that at 9:47 PM on 6/25, it received a series of messages corresponding to an IEC 104 connection being established, indicating that a user on the Windows machine had opened the interface. The connection remained open for approximately 3 minutes, timing out at 9:50 PM. Other than the U-frames and general interrogation messages establishing the connection, the only traffic from the Windows computer were S-frames exchanged to verify continued connectivity.

We stopped our analysis of Phase 2 data at this point because we concluded that one or more malicious users had clearly tampered with Windows logs. The

“LocalSessionManager–UserEvents” log showed several logins to the Guest account without a corresponding log of a remote connection, which was sufficient evidence for that conclusion. This suggests that remote users had either not attempted to cover their tracks or had done so only incompletely. To continue with Phase 2 data collection, we need a more secure deployment of Windows virtual machine. It is obvious that the version of Microsoft Remote Desktop Protocol we used has vulnerabilities allowing a remote user to sign on as Guest, then conduct privilege escalation to obtain admin-level access. Since Windows 8 uses an older version of Remote Desktop Protocol it is unlikely that these vulnerabilities will be fixed, requiring us to adopt a new approach. Some possibilities are discussed in Chapter VI.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

Current open-source honeypots offer only a low-fidelity simulation of industrial control systems because they lack the ability to give an attacker realistic feedback on the physical processes they are supposedly controlling. We have demonstrated a novel honeypot that offers dynamic feedback to attackers based on the state of a simulated electrical grid, and deployed it locally both with and without a supervisory graphical interface. Analysis of the data gathered during those deployments allows us to draw the following conclusions.

A. SUMMARY OF FINDINGS

Our data suggest that deployment of GridPot Phase 1 on a single local server is unlikely to draw much attention other than from automated scanners. While some of the IP addresses that sent ASDUs to our IEC 104 server could be identified as belonging to the SHODAN scanner, they all showed a similar pattern of communications consistent with an automated protocol scan. Nevertheless, we believe Phase 1 offers a basis for further development and deployment.

Our analysis of the number and type of sessions initiated by each IP address, both from our data and the MZ dataset, also indicates that the set of attackers interested in HTTP and the set of attackers interested in ICS protocols are largely disjoint. This implies that analyzing the behavior of the attackers by the protocols they use is important and should be included in future studies. The varying distribution of countries of origin for different ICS protocols was also interesting and suggests some possible locations for honeypot deployment in the cloud.

Our experiences in Phase 2 made it apparent that configuring a Windows virtual machine to provide a graphical interface to entice attackers to interact with our honeypot, while controlling their actions, is much more difficult than we anticipated. Although we believe that Phase 2 offers a chance to study hacker activity against a SCADA system in a level not previously available, additional development will be required before it can be a viable system.

We believe that GridPot offers the possibility for fruitful collaborations with researchers and practitioners interested in industrial control system security. Interested researchers can request access to our executables and installation instructions via Honeypot_Research@nps.edu.

B. FUTURE WORK

Several additional projects could be done using the data we have already collected. First, due to time constraints, only a preliminary analysis of the traffic patterns was performed. To better understand the intrusion patterns, a more exhaustive behavioral analysis of both HTTP and IEC 104 traffic should be done using the existing data corpus. The malware executables we recovered during Phase 2 could be analyzed, and our dataset could be subjected to machine learning techniques to find more patterns in the existing data.

Wider deployment of our Phase 1 GridPot using cloud computing is possible. With ICS systems increasingly moving into the cloud, using a cloud-based provider would not necessarily be suspicious, and would let us use an IP address not affiliated with our school. The ease of cloud deployment in multiple data centers also enables the study of geographical differences in attack patterns against both the HTTP and IEC 104 protocols. Since GridPot is already based on virtual machines, it should be relatively easy to adapt for cloud deployment.

Further development work should be done on GridPot itself. Our original design supported integer-based IEC 104 variables as well as floating-point variables, but this was not implemented. Implementing this would allow an attacker to switch between different gauge readings by changing the location on the virtual grid that data is being drawn from. Future work could also try to fix the persistent problems with our SCADA application and GridPot getting out of synchronization after a connection timeout.

Additional work is needed to deploy remote desktops safely. One approach would be to further secure our Windows remote desktop machine; with better security, it is possible we could restrict guest user access so that intruders could not interfere with data

collection. Alternately, our SCADA application could be run in Linux using an emulator and could switch to a different remote desktop protocol such as VNC.

We would also like to draw the right kind of attention to the honeypot. GridPot's dynamic output is designed to fool relatively sophisticated hackers, but so far it has mostly attracted attackers uninterested in ICS protocols. These problems were exacerbated in Phase 2, since the Internet-connected virtual machine lacked open ICS protocol ports. Careful attention to deployment or Web page design might be needed to attract attackers who would leave more useful data. Alternately, since DNP3 appears to be more popular with attackers, reimplementing GridPot to use DNP3 instead of IEC 104 may attract more attention from ICS-aware attackers.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] K. Stouffer, V. Pillitteri, S. Lightman, M. Abrams, and A. Hahn, “Guide to Industrial Control Systems (ICS) Security,” National Institute of Standards and Technology, Gaithersberg, MD, NIST SP 800-82 Revision 2, Jun. 2015 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf

[2] National Cybersecurity and Communications Integration Center, “Seven Steps to Effectively Defend Industrial Control Systems,” Department of Homeland Security, Arlington, VA, Dec. 2015.

[3] National Security Council, “National Cyber Strategy of the United States of America,” National Security Council, Washington, DC, Sep. 2018.

[4] Secretary of the Navy, “Department of the Navy Critical Infrastructure Protection Program,” Department of the Navy, Washington, DC, SECNAVINST 3501.1D, Jan. 2018.

[5] C. Hawk and A. Kaushiva, “Cybersecurity and the Smarter Grid,” *The Electricity Journal*, vol. 27, no. 8, pp. 84–95, Oct-2014.

[6] C. Glenn, D. Sterbentz, and A. Wright, “Cyber Threat and Vulnerability Analysis of the U.S. Electric Sector,” INL/EXT--16-40692, 1337873, Dec. 2016 [Online]. Available: http://www.osti.gov/servlets/purl/1337873/

[7] Electricity Information Sharing and Analysis Center, “Analysis of the Cyber Attack on the Ukrainian Power Grid: Defense Use Case,” SANS ICS, Washington, DC, Mar. 2016.

[8] R. M. Campbell, K. Padayachee, and T. Masombuka, “A survey of honeypot research: Trends and opportunities,” in *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2015, pp. 208–212. [Online]. doi: 10.1109/ICITST.2015.7412090

[9] M. Nawrocki, M. Wahlisch, Thomas Schmidt, C. Keil, and J. Schonfelder, “A Survey on Honeypot Software and Data Analysis,” *arxiv:160806249v1 [cs.CR]*, Aug. 2016.

[10] C. Dalamagkas *et al.*, “A Survey On Honeypots, Honeynets And Their Applications On Smart Grid,” presented at the 2019 IEEE Conference on Network Softwarization, Paris, France, 2019. [Online]. doi: 10.1109/NETSOFT.2019.8806693

[11] W. O. Redwood, “Cyber Physical System Vulnerability Research,” Ph.D Dissertation, Florida State University, Tallahassee, FL, USA, 2016.

[12] A. Jicha, Mark Patton, and H. Chen, “SCADA honeypots: An in-depth analysis of Conpot,” in *2016 IEEE Conference on Intelligence and Security Informatics*, 2016, pp. 196–198.

[13] M. M. Kendrick and Z. A. Rucker, “Energy-Grid Threat Analysis Using Honeypots,” M.S. Computer Science, Naval Postgraduate School, 2019 [Online]. Available: http://hdl.handle.net/10945/62843

[14] P. J. Leach, T. Berners-Lee, J. C. Mogul, L. Masinter, R. T. Fielding, and J. Gettys, “Hypertext Transfer Protocol -- HTTP/1.1,” Internet Engineering Task Force, 1999 [Online]. Available: https://tools.ietf.org/html/rfc2616

[15] S. Blume, “System Overview, Terminology, and Basic Concepts,” in *Electrical Power System Basics for the Nonelectrical Professional*, Hoboken: John Wiley & Sons, 2007, pp. 1–12.

[16] S. Blume, “Distribution,” in *Electrical Power System Basics for the Nonelectrical Professional*, Hoboken: John Wiley & Sons, 2007, pp. 1–12.

[17] K. P. Schneider *et al.*, “Analytic Considerations and Design Basis for the IEEE Distribution Test Feeders,” *IEEE Transactions on Power Systems*, vol. 33, no. 3, pp. 3181–3188, May 2018. [Online]. doi: 10.1109/TPWRS.2017.2760011

[18] GridLAB-D Project, “GridLAB-D Github Page,” 26-Jun-2020. [Online]. Available: https://github.com/gridlab-d/gridlab-d.

[19] Joint Task Transformation Initiative, “Security and Privacy Controls for Federal Information Systems and Organizations,” National Institute of Standards and Technology, SP 800–53r4.

[20] J. Hoogenboezem, “Distributed network protocols – the old and the new DNP 3 , IEC 60870-5 and IEC 61850,” presented at the IDC Technologies Industrial Automation Conference, 2012.

[21] C. Newton, “World Market for Substation Automation & Integration 2014–2016,” *Newton-Evans Research Company Market Trends Digest*, Mar. 2014.

[22] I. J. Shin, B. K. Song, and D.-S. Eom, “International Electronical Committee (IEC) 61850 Mapping with Constrained Application Protocol (CoAP) in Smart Grids Based European Telecommunications Standard Institute Machine-to-Machine (M2M) Environment,” 2017. [Online]. doi: 10.3390/en10030393

[23] C. R. Ozansoy, A. Zayegh, and A. Kalam, “The Application-View Model of the International Standard IEC 61850,” *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1132–1139, Jul. 2009. [Online]. doi: 10.1109/TPWRD.2008.2005657

[24] A. Vlad, S. Obermeier, and D.-Y. Yu, “ICS Threat Analysis Using a Large-Scale Honeynet,” 2015, pp. 20–30. [Online]. doi: 10.14236/ewic/ICS2015.3

[25] N. Provos and others, “A Virtual Honeypot Framework.,” in *USENIX Security Symposium*, 2004, vol. 173, no. 2004, pp. 1–14.

[26] A. Mairh, D. Barik, K. Verma, and D. Jena, “Honeypot in Network Security: A Survey,” in *Proceedings of the 2011 International Conference on Communication, Computing & Security*, New York, NY, USA, 2011, pp. 600–605. [Online]. doi: 10.1145/1947940.1948065 [Online]. Available: https://doi.org/10.1145/1947940.1948065

[27] P. Sokol, J. Míšek, and M. Husák, “Honeypots and honeynets: issues of privacy,” *EURASIP Journal on Information Security*, vol. 2017, no. 1, p. 4, 2017. [Online]. doi: 10.1186/s13635-017-0057-4. [Online]. Available: https://doi.org/10.1186/s13635-017-0057-4

[28] I. Mokube and M. Adams, “Honeypots: concepts, approaches, and challenges,” in *Proceedings of the 45th annual southeast regional conference*, 2007, pp. 321–326.

[29] M. Andress, “The Honeynet Project,” *InfoWorld*, vol. 24, no. 20, p. 26, May 2002 [Online]. Available: https://www.infoworld.com/article/2677451/the-honeynet-project.html

[30] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, and R. Beyah, “Rethinking the Honeypot for Cyber-Physical Systems,” *IEEE Internet Computing*, vol. 20, no. 5, pp. 9–17, 2016.

[31] D. Buza, F. Juhász, G. Miru, M. Félegyházi, and T. Holczer, “CryPLH: Protecting Smart Energy Systems from Targeted Attacks with a PLC Honeypot,” 2014, pp. 181–192. [Online]. doi: 10.1007/978-3-319-10329-7_12.

[32] K. Wilhoit and S. Hilt, “The GasPot Experiment: Unexamined Perils in Using Gas-Tank-Monitoring Systems,” TrendMicro, White Paper, 2015 [Online]. Available: https://pdfs.semanticscholar.org/cfd2/41c388f3680251ab334bfeccc27a58030964.pdf?_ga=2.146996983.1553328232.1596817688-1345694954.1596817688

[33] H. Naruoka *et al.*, “ICS Honeypot System (CamouflageNet) Based on Attacker’s Human Factors,” *Procedia Manufacturing*, vol. 3, pp. 1074–1081, 2015. [Online]. doi: 10.1016/j.promfg.2015.07.175

[34] D. Antonioli, A. Agrawal, and N. O. Tippenhauer, “Towards High-Interaction Virtual ICS Honeypots-in-a-Box,” in *Proceedings of the 2Nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, New York, NY, USA, 2016, pp. 13–22. [Online]. doi: 10.1145/2994487.2994493 [Online]. Available: http://doi.acm.org/10.1145/2994487.2994493

[35] S. Litchfield, “HoneyPhy: A physics-aware CPS honeypot framework,” M.S. Electrical and Computer Engineering, Georgia Tech, 2017.

[36] K. Koltys and R. Gajewski, “SHaPe: A honeypot for electric power substation,” *Journal of Telecommunications And Information Technology*, vol. 2015, pp. 37–43, 2015.

[37] N. Cifranic, J. Romero-Mariona, B. Souza, and R. Hallman, “Decepti-SCADA: A Framework for Actively Defending Networked Critical Infrastructures,” 2020. [Online]. doi: 10.5220/0009343300690077

[38] IPCOMM GmbH, “ipRoute-IEC101/104: The IEC 60870-5-101/104 Router (Product Information Sheet).” [Online]. Available: https://www.ipcomm.de/product/ipRoute/en/ipRoute.pdf

[39] A. Paganini, “IndigoSCADA User Manual, rev. 334.” Enscada, 30-Apr-2019 [Online]. Available: http://www.enscada.com/a7khg9/IndigoSCADA_user_manual.pdf

[40] A. Paganini, “Personal Communication.” 11-Jul-2019.

[41] A. Paganini, “Personal Communication.” 15-Jul-2019.

[42] N. Cifranic, “Personal Communication.” 13-Aug-2019.

[43] D. P. Chassin, K. Schneider, and C. Gerkensmeyer, “GridLAB-D: An open-source power systems modeling and simulation environment,” in *2008 IEEE/PES Transmission and Distribution Conference and Exposition*, 2008, pp. 1–5. [Online]. doi: 10.1109/TDC.2008.4517260

[44] GridLAB-D Project, “Creating GLM Files - GridLAB-D Wiki.” [Online]. Available: http://gridlab-d.sourceforge.net/wiki/index.php/Creating_GLM_Files

[45] D. P. Chassin, J. C. Fuller, and N. Djilali, “GridLAB-D: An Agent-Based Simulation Framework for Smart Grids,” *Journal of Applied Mathematics*, vol. 2014, p. 492320, Jun. 2014. [Online]. doi: 10.1155/2014/492320

[46] Internet Assigned Numbers Authority, “Port 6267,” *Service Name and Transport Protocol Port Number Registry*. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=6267

[47] M. Sankur, R. Dobbe, E. Stewart, D. Callaway, and D. Arnold, “A Linearized Power Flow Model for Optimization in Unbalanced Distribution Systems,” 2016.

[48] GridLAB-D Project, “Powerflow - GridLAB-D Wiki.” [Online]. Available: http://gridlab-d.sourceforge.net/wiki/index.php/Powerflow#Node

[49] V. Vyatkin, G. Zhabelova, N. Higgins, K. Schwarz, and N.-K. Nair, “Towards Intelligent Smart Grid Devices with IEC 61850 Interoperability and IEC 61499 Open Control Architecture,” in *Transmission and Distribution Conference and Exposition, 2010 IEEE PES*, 2010, pp. 1–8. [Online]. doi: 10.1109/TDC.2010.5484272

[50] T. Teodorowicz, “Comparison of SCADA protocols and implementation of IEC 104 and MQTT in MOSAIK,” Bachelor’s Thesis, University of Münster, Germany, 2017 [Online]. Available: https://www.uni-muenster.de/imperia/md/content/informatik/agremke/comparison_of_scada_protocols_and_implementation_of_iec_104_and_mqtt_in_mosaik.pdf. [Accessed: 22-Jul-2020]

[51] P. Matoušek, “Description and analysis of IEC 104 Protocol,” Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic, Technical Report FIT-TR-2017-12, Dec. 2017.

[52] PTC, Inc, “IEC 60870-5-104 Master Driver” [Online]. Available: https://www.kepware.com/getattachment/9d897da8-cbe7-4fb7-9f30-90dcecad8f31/iec-60870-5-104-master-manual.pdf

[53] C.-Y. Lin and S. Nadjm-Tehrani, “Understanding IEC-60870-5-104 Traffic Patterns in SCADA Networks,” in *CPSS ‘18: Proceedings of the 4th ACM Workshop on Cyber-Physical System Security*, 2018, pp. 51–60.

[54] Beckhoff Information Systems, “TwinCAT 2 Software Manual: TwinCAT PLC Lib: IEC 60870-5-104 Transport Interface” [Online]. Available: https://infosys.beckhoff.com/english.php?content=../content/1033/tcplclibiec870_5_104/html/tcplclibiec870_5_104_telegrammstructure.htm&id

[55] E. Magaña, I. Sesma, D. Morató, and M. Izal, “Remote access protocols for Desktop-as-a-Service solutions,” *PLOS ONE*, vol. 14, no. 1, pp. 1–28, 2019. [Online]. doi: 10.1371/journal.pone.0207512

[56] Internet Assigned Numbers Authority, “Port 3389,” *Service Name and Transport Protocol Port Number Registry*. [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=3389

[57] “Understanding the Remote Desktop Protocol (RDP).” [Online]. Available: https://support.microsoft.com/en-us/help/186607/understanding-the-remote-desktop-protocol-rdp

[58] openspecs-office, “[MS-RDPBCGR]: Connection Sequence,” *Microsoft Developer Network*. [Online]. Available: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpbcgr/023f1e69-cfe8-4ee6-9ee0-7e759fb4e4ee

[59] Ciprian Rusen, “You can’t enable the Guest account in Windows 10. Here’s why and how others are lying,” *Digital Citizen*, 09-Aug-2017. [Online]. Available: https://www.digitalcitizen.life/you-cant-enable-guest-account-windows-10-stop-trying

[60] D. Hyun, “Collecting cyberattack data for industrial control systems using honeypots,” M.S. Computer Science, Naval Postgraduate School, Monterey, CA, 2018 [Online]. Available: https://calhoun.nps.edu/handle/10945/58316

[61] Libiec61850 Project, “Documentation | libIEC61850 / lib60870-5.” [Online]. Available: https://libiec61850.com/libiec61850/documentation/

[62] H. Al-Alami, A. Hadi, and H. Al-Bahadili, “Vulnerability Scanning of IoT Devices in Jordan using Shodan,” 2017. [Online]. doi: 10.1109/IT-DREPS.2017.8277814.

[63] N. C. Rowe, T. D. Nguyen, M. M. Kendrick, Z. A. Rucker, D. Hyun, and J. C. Brown, “Creating Effective Industrial-Control-System Honeypots,” presented at the Hawaii International Conference on System Sciences, Wailea, HI, 2020 [Online]. Available: https://faculty.nps.edu/ncrowe/ics_honeypots_ncrowe.htm

[64] Wireshark Foundation, “Wireshark · About.” [Online]. Available: https://www.wireshark.org/about.html

[65] Scapy Project, “Scapy Documentation,” 07-Aug-2020. [Online]. Available: https://scapy.readthedocs.io/en/latest/

[66] MaxMind, Inc, “Developer Documentation,” *Developer Documentation*. [Online]. Available: https://dev.maxmind.com/#GeoIP

[67] Microsoft Corporation, “Windows Security Threat Protection: Guest Account Status,” Microsoft Documentation [Online]. Available: https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/accounts-guest-account-status

[68] M. Boddy, B. Jones, and M. Stockley, “RDP Exposed - The Threat That’s Already at Your Door,” Sophos, Inc, Sophos White Paper, 2019.

[69] Albert Lab, “How to Decrypt Remote Desktop Protocol(RDP) over TLS/SSL Traffic in Wireshark.” [Online]. Available: http://xinchunge.net/blog/?p=156

[70] S. Boslaugh, “Chapter 6: The t-test,” in *Statistics in a Nutshell*, O’Reilly Media, Incorporated, 2012 [Online]. Available: https://books.google.com/books?id=HZpoDjtKT0IC

[71] P. Dangeti, “Cosine Similarity,” in *Statistics for Machine Learning: Techniques for Exploring Supervised, Unsupervised, and Reinforcement Learning Models with Python and R*, Packt Publishing, 2017.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California