Creating Synthetic Attacks with Evolutionary Algorithms for Proactive Defense of Industrial Control Systems

Nathaniel J. Haynes
Naval Postgraduate School
nathaniel.haynes@nps.edu

Thuy D. Nguyen
Naval Postgraduate School
tdnguyen@nps.edu

Neil C. Rowe
Naval Postgraduate School
ncrowe@nps.edu

Abstract

Industrial control systems (ICS) play an important role in critical infrastructure. Cybersecurity defenders can use honeypots (decoy systems) to capture and study malicious ICS traffic. A problem with existing ICS honeypots is their low interactivity, causing intruders to quickly abandon the attack attempts. This research aims to improve ICS honeypots by feeding them realistic artificially generated packets and examining their behavior to proactively identify functional gaps in defenses. Our synthetic attack generator (SAGO) uses an evolutionary algorithm on known attack traffic to create new variants of Log4j exploits (CVE-2021-44228) and Industroyer2 malware. We tested over 5,200 and 256 unique Log4j and IEC 104 variations respectively, with success rates up to 70 percent for Log4j and 40 percent for IEC 104. We identified improvements to our honeypot's interactivity based on its responses to these attacks. Our technique can aid defenders in hardening perimeter protection against new attack variants.

Keywords: synthetic attack, evolutionary algorithm, industrial control system, security testing, honeypot

1. 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 serious attacks to include effects on Ukrainian infrastructure and U.S. gas pipelines, and technical advisories on the CrashOverride, Shamoon, and Havex malware campaigns among others have been published (Cyber Security and Infrastructure Security Agency [CISA], 2021a).

Originally ICSs managed only physical processes through operational technology (OT), but as the Internet grew, ICSs became integrated with information technology. Currently 85% of the U.S. critical infrastructure is commercialized, which means the pursuit of safe throughput and availability can be prioritized over confidentiality and integrity (Stouffer et al., 2015). 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 methods are needed to test and harden ICSs.

Security of live ICSs is difficult to test. One solution is to emulate ICSs in virtual environments, which removes the risk of harming actual services. ICS honeypots could also offer rich data for analysis. At our school, previous research explored electrical-grid ICS honeypots and so far, saw attackers favoring the Hypertext Transfer Protocol (HTTP) much more than ICS protocols (Dougherty, 2020; Washofsky, 2021). To get more data about attacks on ICSs, some free and commercial vulnerability databases and open-source repositories of network traces are available (National Vulnerability Database [NVD], n.d.; The MITRE Corporation, 2021). However, they are limited in re-creating exploits for testing. Commercial products like Metasploit Pro and Immunity CANVAS can do penetration testing, but they have few ICS-related attacks. Most open-source ICS attack tools, like those in GitHub, are unmaintained. Even large public repositories of collected malware samples have sparse instances of ICS malware (VirusShare, n.d.).

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 Java logging library used in many systems including ICSs (CISA, 2021b); we saw

The views expressed in this material are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

URI: https://hdl.handle.net/10125/102842
978-0-9981331-6-4

it being used against the user interface of our previous ICS honeypot. However, patches to this vulnerability are installed slowly by ICS administrators due to potential losses in availability and incompatibility with older systems (Stouffer et al., 2015). Alternative cybersecurity solutions like honeypots may help protect these vulnerable ICS systems.

Also during this research, several Ukrainian power grid ICSs that used the IEC 104 protocol were attacked with the Industroyer2 malware. We have had experience with this protocol and decided to study the behavior of this malware as well.

We first collected real Log4j and IEC 104 network traffic to analyze the characteristics of actual attacks. We then made a honeypot susceptible to Log4j and created a synthetic attack generator (SAGO). We rated the output of the generator and compiled statistics on its degree of success. We also studied the traffic produced by the malware Industroyer2 and demonstrated a way to extend our attack-generation methods for IEC 104.

2. Threat Models and Related Work

2.1. The HTTP Protocol and Log4j

We focused on the Hypertext Transfer Protocol (HTTP) and the IEC 104 protocol, the IP/TCP extension of IEC 60870-5-104 standards. Although not an ICS protocol, HTTP is often used by ICS systems to provide a Web-based user interface for controlling industrial processes.

Web technologies like HTTP that handle user input are vulnerable to exploits like buffer overflows, cross-site scripting, and command injections (McClure et al., 2012). This research examined the recent Log4j HTTP-based attack targeting commonly used Java-based logging systems. This attack used command injections to execute code from a remote address. Rated a 10 out of 10 in severity by NIST's National Vulnerability Database, the Apache Log4j (also called Log4Shell) vulnerability CVE-2021-44228 quickly gained notoriety in the cybersecurity world (NVD, 2021).

Ten days after its disclosure, the attack had been observed 350 million times and had 1.4 times the activity volume as 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, 2021b). Despite the widespread proliferation of Java-based logging, no major compromises were reported (FortiGuard Labs, 2022). Nonetheless, CISA still recommends that organizations should continue testing and hardening their devices against the exploit (CISA, 2021b).

After Log4j was disclosed, many open-source projects studied the exploit but their usefulness to testing defenses against the exploit was limited. Metasploit has three Log4j modules that are functionally limited and do not offer any obfuscation. Another tool, Ox4Shell, can de-obfuscate and analyze Log4j payloads (Abeles & Vider, 2022), but it does not help test a system's robustness against such attacks.

Many OT systems are also vulnerable to Log4j exploits (Kovacs, 2022). ICS vendors such as Siemens (Siemens ProductCERT, 2021) have confirmed this problem. Log4j also enables attackers to use compromised IT systems to pivot to the control segment of an ICS network.

2.2. The IEC 104 Protocol

IEC 104 standards use Application Protocol Data Units, 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) (Matoušek, 2017). Each frame has a fixed-length header of Application Protocol Control Information (APCI) and a payload of the Application Service Data Unit (ASDU). Only I-format frames have ASDUs, which hold Information Objects, each with two components, an information-object address (IOA) and information elements. The information elements are the main data structures for passing information in the IEC 104 protocol. Each information element can only contain one data type, but each ASDU can hold multiple information elements. Example data types are single and double commands for controlling IEC devices, and short floating-point numbers for sensor values. We refer to I-format frames by the type of data it holds.

IEC 104 devices use I-frames to transfer data. However, this data is not protected because the IEC 104 protocol lacks encryption and authentication. Since ICS devices often have outdated software, CISA recommends continuous monitoring to defend against these threats (CISA, 2022). Still, the protocol's inherent vulnerabilities remain problematic for IEC 104 devices because attackers have several options for exploitation.

The popular penetration-testing tools Nmap and Metasploit have extensions specific to IEC 104. Nmap can perform IP enumeration across ICS devices that use IEC 104 (Timorin & Miller, n.d.). A rogue device can spoof an IEC 104 server and send unauthorized commands to ICS devices using Metasploit's IEC 104

Client Utility module, though only certain systems are vulnerable to this exploit (Metasploit, n.d.).

Using several tools (*Hping*, *Ettercap*, and *OpenMuc J60870*) to attack an emulated IEC 104 network, researchers found that unauthorized access and denial-of-service attacks were the least successful, while man-in-the-middle and traffic analysis attacks were more successful (Radoglou-Grammatikis et al., 2019). (Baiocco & Wolthusen, 2018) found that disrupting the time synchronization between two IEC 104 devices can cause a denial of service.

Another project analyzed IEC 104 attacks for creating intrusion-detection test datasets (Fundin, 2021). Eight out of the twelve attacks succeeded and while the datasets are publicly available, their Python code is not.

2.3. Automated Generation of Exploits

Automated testing reduces human dependency on finding vulnerabilities (Black et al., 2021). One project tested different open-source ICS software with "fuzzed" packets (Luo et al., 2020) and found many bugs. However, for large software applications, verifying correct behavior of every input through automated testing is unattainable. (Kuhn et al., 2009) have argued that exhaustive testing is unnecessary and only a few parameters typically contribute to faulty outcomes Instead, combinatorial testing uses input groups to reduce the testing space to a manageable size for software with many parameters. Such techniques can help find crashes and bugs, but do not help to determine whether the generated exploits succeed.

Compared to traditional software testing and probabilistic sampling (Choi et al., 2021), unsupervised learning algorithms have the advantage of generating new variations of tests. With predictions made from prior successful tests, unsupervised learning algorithms are more likely to continue finding successful tests than techniques like randomly fuzzing input which only relies on randomization to find new successful tests. Two approaches, evolutionary algorithms and generative adversarial networks, expand the traditional testing space. Evolutionary algorithms model a well-known biological process, are easy to implement, and are efficient when searching for new variations (Vikhar, 2016).

(Appelt et al., 2018) used an evolutionary algorithm to generate SQL-injection (SQLi) attacks for testing Web application firewalls. Their *mutation* operations were behavior changing, syntax repairing, and obfuscation. The new offspring, called a *generation*, trains a random-forest classifier. Testing then assigns a probability of detection to each offspring in the generation. Fitness selection picks the offspring with highest probabilities to mutate next.

Generative adversarial networks are another approach for automated test generation (Hong et al., 2020). Generative adversarial networks have tested autonomous vehicles image recognition and anomaly detection in intrusion-detection systems (Lin et al., 2021; Zhang et al., 2018). However, these networks are difficult to implement due to their complexity.

These test and attack generators try to craft new variants, starting from an exploit template or model. Instead of generating random attacks, examining attacks observed "in the wild" allows researchers to test new variants of popular attacks. For our evolutionary algorithm, we defined the exploit schema, attack features, and success criteria based on attacker behavior observed in different datasets. Recent research at NPS has collected several corpora of different attacks against ICS honeypots deployed in a commercial cloud environment (Washofsky, 2021).

3. Attacks on Our ICS Honeypot

3.1. Log4j Attacks

On December 10, 2021 when the Apache Log4j vulnerability was first announced, our honeypot experienced a large increase in HTTP requests with Log4j commands embedded in their headers. As the vulnerability fixes evolved, we saw different variations of the exploit. Due to the significance of the vulnerability and many ICS vendors reporting exposure (Kovacs, 2022), we decided to further study the related exploits.

Log4j configuration files contain important data about the system runtime environment which, if exploited, attackers can use to weaponize an attack. Log4j's *lookup* mechanism allows applications to insert values of configuration variables into log-destination strings. The syntax is "${variable}" where the variable is replaced with its current value in the configuration file. By observing the looked up variables, we can deduce the attacker's tactics. To enrich log details, Log4j can also refer to system and environment variables (The Apache Software Foundation, 2022). Lookups are triggered with the syntax "prefix:attribute" or "prefix:attribute:-default". As an example, the string "${docker:containerId}" logs the Docker container's identification. The symbol ":-" establishes a default value if the requested attribute cannot be mapped to the prefix. Lookups can be recursive which allows for more complex variable representations and mappings. To thwart intrusion-detection systems, attackers can recursively embed IOA lookups to create complex variations of Log4j

exploits that avoid known signatures (National Cyber Security Centrum, 2021).

Log4j exploits target the Java Naming and Directory Interface (JNDI), an interface for Java programs to retrieve objects from servers (Oracle, n.d.). JNDI resolves objects using various naming and directory services like the Lightweight Directory Access Protocol (LDAP) service. JNDI uses these services to query, resolve, and download objects from servers. It can also retrieve compiled Java files and execute them, a known vulnerability (Muñoz & Mirosh, 2016). Furthermore, Java applications with Log4j logging can use JNDI in the form of a JNDI lookup. This allows attackers to put JNDI command injections into protocol fields, like HTTP headers, that Java applications are likely to log.

Based on the Log4j exploit strings we collected on our honeypot, we explored exploits which use LDAP servers as the attack vector, the most frequently exploited service. 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 to another server that might have it. This way the JNDI lookup can request compiled Java classes from other servers like HTTP servers (Muñoz & Mirosh, 2016).

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

3.2. IEC 104 Attacks

In April 2022, Ukrainian ICSs using IEC 104 were attacked. The malware used in this attack was a modified variant of those used in the CrashOverride campaign (Kapellmann et al., 2022). In 2016 the original malware, called Industroyer, targeted different ICS protocols, including IEC 104. The Industroyer malware was a Windows executable and, once installed on the victim system, established command-and-control connections, exploited the vulnerabilities of the chosen ICS protocol, and finally wiped the machine's data (Cherepanov, 2017). The IEC 104 part of Industroyer would try to end the original IEC 104 processes and manipulate the states of the discovered devices.

Industroyer was ineffective due to improper implementation of its ICS protocols (Slowik, 2019) that caused communications with IEC 104 devices to be rejected due to their failure to follow protocol standards. Industroyer2 appears to derive from the same codebase as Industroyer but only 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.

Industroyer2 sent single commands or double commands, based on the behavior of the victim ICS, to damage the IEC 104 devices. While running, Industroyer scanned every IOA of an IEC 104 device with general interrogation commands. Industroyer then iterated through the IOAs and turned them off and on repeatedly. Initial analyses indicated that the IOAs corresponded to ABB Distribution Recloser Relays, and that the attackers were trying to disrupt critical overcurrent protections (Kapellmann et al., 2022).

Our honeypot data did not include attack behavior like Industroyer2; the observed behavior was limited to scanning. Of the valid IEC 104 payloads, most attackers only sent general interrogation commands. IEC devices respond to general interrogation commands with every IOAs reachable on that IEC device. Furthermore, attackers did not perform follow-up actions after getting every device address, likely due to their limited understanding of IEC 104.

4. Generation of Synthetic Attacks

We used evolutionary algorithms to generate variations of existing attacks for testing our honeypots. To recreate the attacks, we sought exploitable attack patterns in the HTTP and ICS payloads of the honeypot's captured network traffic and sampled Industroyer2 packet captures. The HTTP and IEC 104 algorithms needed different syntax for setting machine-learning features.

Our process for generating and testing exploits (Figure 1) started with creating 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 the base exploit string "jndi:ldap://". We did not seed SAGO with our collected Log4j exploits because our dataset had many duplicates and lacked variety. Also, *crossover* operations can construct a child's features by combining selected features of two parents. Our

fitness function used the observed response by our honeypot to determine degree of success. We stored these success values in a random forest to predict if a new attack variant would succeed.

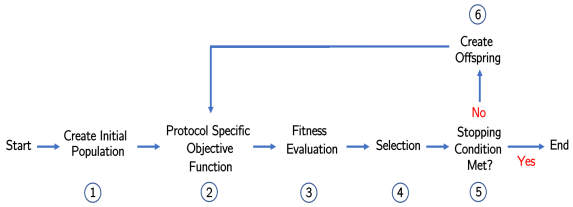

Figure 1. SAGO's process to create exploits

4.1. Log4j Attack Analysis

Our honeypot collected data from November 30, 2021 to January 17, 2022. Since the Log4j vulnerability was disclosed on December 10, 2021, we captured some of the first Log4j exploit variants. We observed 102 packets with Log4j exploits in HTTP request headers. The HTTP request methods were 98 GET commands and 4 POST commands. This activity originated from 30 unique sources and represented 0.1% of the honeypot's overall traffic.

To exploit vulnerable hosts, attackers embedded their Log4j exploits in the header fields they believed were the most likely to get logged. The user-agent header was the most popular and was used in 57 HTTP requests. Among these were 21 requests with the user-agent header as the only location with the Log4j exploit. Using only one header field for the exploit was common since it happened 48% of the times. On the other hand, 60% of the HTTP requests only had one variant of the exploit string in the headers, which means that the attackers wanted to increase their chances of the exploit getting logged.

In total, 205 exploits were found in the 102 HTTP header fields (Table 1). Most exploits tried to call an LDAP server and that accounted for 187 sample Log4j strings. The other 18 occurrences had DNS as their callback server. To obfuscate their exploit, attackers used lookups.

Table 1. Embedded Log4j Exploits

Callback Server	Strings without Lookups	Strings with Lookups	Total
LDAP	104	83	187
DNS	2	16	18
Total	106	99	205

Four unique variations of the LDAP exploit used different combinations of lookups (lower case, environment variable, and empty string) to obfuscate the string. The DNS variations originated from two scanners: scanworld and securityscan. Scanworld simply used "jndi:dns://" while securityscan used "${::-j}ndi://dns://". This was identifiable because they included their name in a substring of the URI.

Although many attackers redundantly encoded their Log4j exploits into multiple HTTP header fields, they typically copied the same exploit in every header so if one exploit failed, all would fail.

4.2. Log4j Exploit Generation

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. Two constraints were to start and end with the property substitution symbols "${" and "{", 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 be parsed correctly by Log4j. An example exploit string is in Figure 2.

Though we could mutate every character in the exploit, we only explored transforming the substring "jndi:ldap://". This avoided complications with disrupting the IP address and instance identifier. Given the substring "${lower:j}ndi:ldap://", an example mutation would insert "$lower:" after the colon. The mutated string would then be "${lower:$lower:j}ndi:ldap://". In mutating 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.

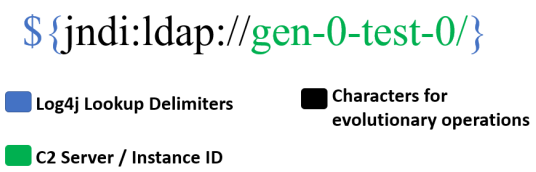

Figure 2. Log4j exploit schema

Besides mutations, we also used crossovers to find possible exploits. Our crossover operation

exchanged lookups between the Log4j exploit characters. For example, given two parent Log4j exploit strings, "${{env::-j}ndi:ldap://}" and "${jndi:${sys::-l}dap://}", with the "j" and "l" characters selected to cross, the result would be "${{env::-j}ndi:${sys::-l}dap://}". The number of lookups swapped between parents was controlled by the *number of crossings* hyperparameter while the location of lookups to be swapped was random.

Generated exploits were sent to the victim server one at a time. We then trained a random-forest classifier based on the exploits' features and the successes or failures as labels. For our research, successful exploits are those which cause Log4j to send unintended outbound requests to servers of our choosing. Failed exploits will get logged by Log4j but would not trigger any network traffic.

We sampled 75 percent of the generated Log4j exploits to train 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* hyperparameter.

Once trained, the classifier estimated the probability of each successful exploit and used the exploits with the highest probabilities to generate the next population.

4.3. IEC 104 Attack Analysis

Since our honeypot runs on a Linux operating system, we could not natively run the Industroyer2 malware to assess its effects on our simulated power grid. Instead we used three packet captures from (Hjelmvik, 2022) who executed Industroyer2 in their isolated environment. We used these samples because most of our honeypot's IEC 104 traffic was either malformed or sent out of order. The valid IEC 104 payloads only used general interrogation commands and no further commands were sent to the IOAs returned by our honeypot. This could indicate a lack of understanding of the IEC 104 protocol.

Compared to our collected IEC 104 data, the Industroyer2 traffic is more complex. Industoyer2 sets up the IEC 104 data transfer, interrogates to find all addressable IOAs, and then sends commands to each IOA depending on the data type each IOA supports. After probing all IOAs, Industroyer2 ends the connection.

4.4. IEC 104 Exploit Generation

Industroyer2 sent single and double commands to query a simulated industrial process. We chose to generate double commands because our honeypot does not process single commands. The specification for double commands is in Figure 3.

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 3. IEC 104 double command specification

Besides varying the double command bits, we kept the rest of the frame the same as Industroyer2. Since we only manipulated eight bits of the command field, fewer variants were possible for the IEC 104 attacks than for Log4j exploits. For our IEC 104 evolutionary operations, crossover operations randomly selected bits and swapped their corresponding values while mutation toggled the value of one random bit in the command, making the mutation-magnitude hyperparameter irrelevant.

To check the success of the generated IEC 104 attack, ideally our exploits would cause the honeypot's IEC 104 server to change our simulated power grid's state. However, our honeypot's implementation only allowed read-only requests. Instead, we relied on the log produced by our honeypot to determine the success of the generated exploits. Some commands that used non-implemented or undefined bits caused our honeypot to end the connection without writing to its log. Hence, we could tell that a command succeeded if a log entry for the command was created. We considered the exploit succeeded 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. This is like the objective of Industroyer2 which used various commands to disrupt power-grid operation (Kapellmann et al., 2022).

5. Testing of Potential Exploits

We used two Debian Linux virtual machines on the DigitalOcean cloud platform for the attacker and victim systems (Figure 4). The victim machine was an

instance of the GridPot honeypot used in past NPS research (Washofsky, 2021). It ran T-Pot (Telekom Security, 2016), which manages honeypots running as Docker containers. Our honeypot was one of them. It uses GridLab-D to simulate a power distribution system (Chassin et al., 2008). GridPot also encapsulates the HTTP and IEC 104 honeypot Conpot.

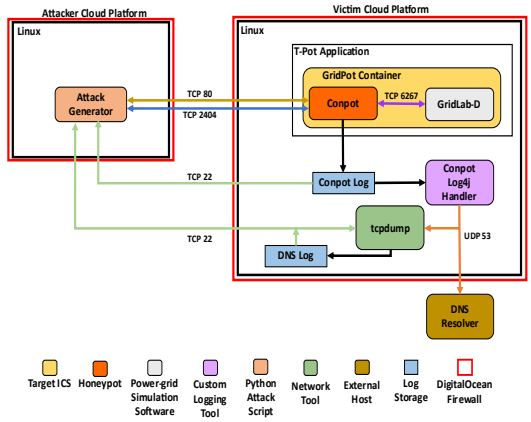

Figure 4. Experiment Design

We changed the honeypot from (Washofsky, 2021) to make it vulnerable to Log4j exploits. In our honeypot we sent HTTP logs produced by Conpot to a custom Java application, called the *Conpot Log4j Handler,* that logged HTTP user-agent strings with Log4j. If any user-agent string triggered the Log4j exploit, the victim machine tried to resolve the malicious server's IP address with DigitalOcean's DNS resolver. SAGO used Tcpdump's output to get feedback for its Log4j evolutionary algorithm. No changes to (Washofsky, 2021) were necessary for IEC 104. We used SSH to pull the IEC 104 server's log file and correlated successful attacks based on its entries.

5.1. Log4j Experiments

We ran Log4j exploits generated on our honeypot. Typically, developers use Log4j in their servers to log headers of HTTP requests. Since our implementation of T-Pot does not use Log4j, we had to simulate a vulnerable server. Since Conpot's HTTP server logs all the HTTP request headers, our solution was to send the Linux tail command on the Conpot log as input to the Conpot Log4j Handler.

On the victim machine, we installed a vulnerable version of Java and Log4j. Our custom Java application read from standard input, matched for a user-agent string, and if one were found, logged it using Log4j. A successful exploit sent to our honeypot server would trigger the JNDI lookup and result in a DNS query. SAGO used these queries to correlate success labels to the exploits.

SAGO initialized a population of artificial Logj4 exploits of count determined by the target *population size*. Each exploit was a base Log4j exploit string "jndi:ldap://" with one random lookup applied. SAGO then sent an SSH command to start Tcpdump on the victim machine with Tcpdump's standard terminal output directed to a DNS log file. Next, it 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 also sent another SSH command to stop Tcpdump and retrieve the DNS log. The retrieved DNS log recorded every successful exploit. SAGO could correlate success in the DNS log to the Log4j exploits using the instance identifiers (Figure 2). Our evolutionary algorithm for Log4j exploits used the hyperparameters in Table 2.

Table 2. Hyperparameter combinations for generating 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]

5.2. IEC 104 Experiments

We adapted SAGO to send double commands to the target IEC 104 server. Using Scapy, we simulated an Industroyer2 data setup using U-format *start* frames. We only sent one command per data transfer to determine if it was accepted by the honeypot. To end the connection, we sent a U-format *stop* frame. We used the hyperparameters in Table 3 for testing the IEC 104 attack creation.

Table 3. Hyperparameter combinations for generating IEC 104 exploits

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]

After creating the first population, SAGO sent the exploits to the Conpot IEC 104 server. Using SSH, SAGO read the Conpot log and correlated 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. The fitness evaluation and selection steps of the IEC 104 attack generator were the same as the Log4j implementation.

6. Results and Discussions

6.1. Log4j Results

When testing SAGO over twenty generations, we noticed that the success rate flattened around ten generations (Figure. 5). Figure 6 shows successful exploits with different mutation rates.

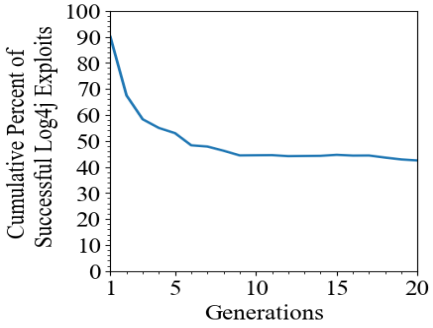

Figure 5. Successful Log4j Exploits over Generations with Population Size 20, Parent Population Size 10, Number of Crossings 6, Mutation Rate 0.50, and Mutation Magnitude 1

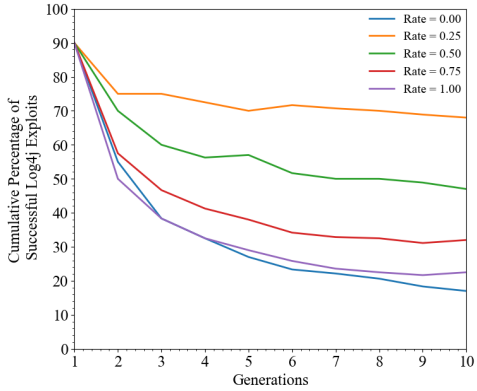

Figure 6. Log4j Successful Exploits When Varying Mutation Rate with Population Size 20, Parent Population Size 10, Number of Crossings 6, and Mutation Magnitude 1

Higher population sizes caused SAGO to quickly find the more successful attacks. Conversely parent population size and crossover did not significantly affect the cumulative success rate. However, the mutation rate did affect the creation of successful exploits. In Figure 6 the lower mutation rates of 0.25 and 0.50 performed the best. For mutation magnitude, higher values meant finding more exploits quickly, but in later generations such values caused too much variation to continually find new exploits.

We produced over 5,200 unique strings exploiting the Log4j vulnerability. More variations can be created using other malicious directory services besides LDAP like DNS and Remote Method Invocation (RMI). The unique strings created for these variants can be used to strengthen firewalls and intrusion-detection systems.

6.2. IEC 104 Results

After twenty generations, the cumulative percentage of successful attacks was about thirty percent (Figure 7). Also, the rate of discovering new attacks had not flattened at twenty generations like the Log4j attack rate. We saw similar results as Log4j for all hyperparameters except mutation rate (Figure 8).

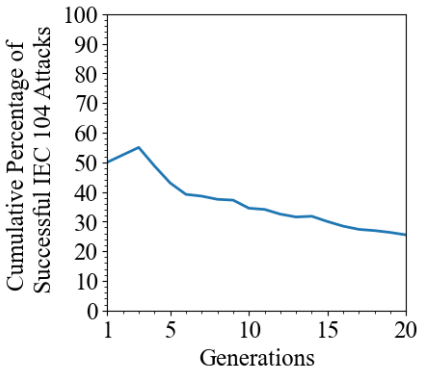

Figure 7. Successful IEC 104 Attacks Discovered Versus Generations with Population Size 20, Parent Population Size 10, Number of Crossings 4, and Mutation Rate 0.50

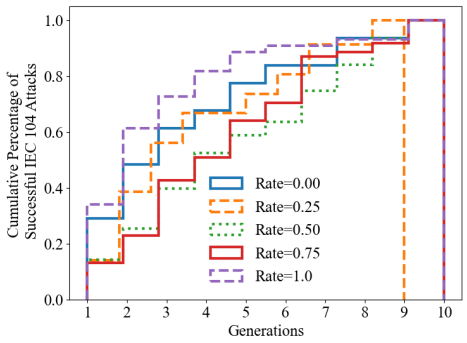

Figure 8. Cumulative IEC 104 Attack Success Rate Versus Generations with Population Size 20, Parent Population Size 10, and Number of Crossings 4

The extreme rates of zero and one still caused the worst performing outcomes, but generally the higher

mutation rates of 0.50 and 0.75 performed the best for all ten generations. At a mutation rate of 0.50 we saw our highest IEC 104 attack success at 40 percent. For IEC 104, SAGO found all 256 variations of the 8-bit double command.

From the results of our IEC 104 attacks, we found a deficiency in the translation of the IEC 104 commands to GridLab-D's API in which our honeypot accepted the commands used by the exploits but did not change the underlying power grid's state as expected. With proper translation, our honeypot could change its power readings based on user input. Attackers then could send IEC 104 double commands and see the resulting power readings change on the Web interface of our honeypot. Such interactivity would make our honeypot more convincing and would prompt more attack behavior of those trying to disrupt the power grid.

7. Conclusion

Our synthetic 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 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 between 0 and 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.

SAGO can also be improved to support other hyperparameter values and evolutionary operations to create more diverse attack variants. Our approach applies to other industrial protocols that use commands with various bitstrings to change ICS states, for example, IEC 61850. Since bit manipulation is common for these protocols, evolutionary algorithms can be used to generate new attack variants for such ICS protocols and systems.

Continuous monitoring is a key proactive mitigation to defend against emergent exploits. Existing monitoring solutions can benefit from the synthetic attacks created by SAGO.

8. References

Abeles, D., & Vider, R. (2022). *Ox4Shell* (Version 1.1) [Computer software]. Oxeye Security LTD. https://github.com/ox-eye/Ox4Shell

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

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

Chassin, D. P., Schneider, K., & Gerkensmeyer, 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

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

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, December 23). Mitigating Log4Shell and other Log4j-related vulnerabilities. https://www.cisa.gov/uscert/ncas/alerts/aa21-356a

Cyber Security and Infrastructure Security Agency (2022, May 25). *APT cyber tools targeting ICS/SCADA devices*. https://www.cisa.gov/uscert/ncas/alerts/aa22-103a

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

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

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

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

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

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

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

McClure, S., Scambray, J., & Kurtz, G. (2012). Hacking exposed 7: Network security secrets & solutions. McGraw-Hill.

Metasploit (n.d.). IEC 104 client utility. Retrieved March 25, 2022, from https://github.com/rapid7/metasploit-framework/blob/master/modules/auxiliary/client/iec104/iec104.rb

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

Oracle (n.d.). *Lesson: Overview of JNDI*. Java Documentation. Retrieved September 1, 2022, from https://docs.oracle.com/javase/tutorial/jndi/overview/index.html

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

Reitz, K. (2022). *Requests* (Version 2.28.1) [Computer software]. Python Software Foundation. https://github.com/psf/requests

Sermersheim, J. (2006). Lightweight directory access protocol (LDAP): The protocol (RFC No. 4511). RFC Editor. https://doi.org/10.17487/RFC4511

Siemens ProductCERT (2021, December 13). *SSA-661247: Apache Log4j vulnerabilities (Log4Shell, CVE-2021-44228, CVE-2021-45046)—Impact to Siemens products*. https://cert-portal.siemens.com/productcert/pdf/ssa-661247.pdf

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

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

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

The Apache Software Foundation (2022, February 23). Lookups. Apache Logging Services. https://logging.apache.org/log4j/2.x/manual/lookups.html

The MITRE Corporation (2021, October 12). CVE. https://cve.mitre.org/

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/

Timorin, A., & Miller, D. (n.d.). Script IEC-identify. NMAP. Retrieved March 25, 2022, from https://nmap.org/nsedoc/scripts/iec-identify.html

Tsaraias, 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

Vikhar, P. A. (2016). Evolutionary algorithms: A critical review and its future prospects. 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), 261–265. https://doi.org/10.1109/ICGTSPICC.2016.7955308

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