```
kali@kali:~$ sudo tcpdump -r password_cracking_filtered.pcap
reading from file password_cracking_filtered.pcap, link-type EN10MB (Ethernet)
08:51:20.800917 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074, w
in 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.800953 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr 25538253,nop,w
scale 4], length 0
08:51:20.801023 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074, w
in 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.801030 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr 25538253,nop,w
scale 4], length 0
08:51:20.801048 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074, w
in 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.801051 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr 25538253,nop,w
scale 4], length 0
...
```

Listing 127 - Using tcpdump to read packet capture

4.5.2 Filtering Traffic

The output is a bit overwhelming at first, so let's try to get a better understanding of the IP addresses and ports involved by using **awk** and **sort**.

First, we will use the **-n** option to skip DNS name lookups and **-r** to read from our packet capture file. Then, we can pipe the output into **awk**, printing the destination IP address and port (the third space-separated field) and pipe it again to **sort** and **uniq -c** to sort and count the number of times the field appears in the capture, respectively. Lastly we use **head** to only display the first 10 lines of the output:

```
kali@kali:~$ sudo tcpdump -n -r password_cracking_filtered.pcap | awk -F" " '{print $3
}' | sort | uniq -c | head
  12324 172.16.40.10.81
     18 208.68.234.99.32768
     18 208.68.234.99.32769
     18 208.68.234.99.32770
     18 208.68.234.99.32771
     18 208.68.234.99.32772
     18 208.68.234.99.32773
     18 208.68.234.99.32774
     18 208.68.234.99.32775
     18 208.68.234.99.32776
...
```

Listing 128 - Using tcpdump to read and filter the packet capture

We can see that 172.16.40.10 was the most common destination address followed by 208.68.234.99. Given that 172.16.40.10 was contacted on a low destination port (81) and 208.68.234.99 was contacted on high destination ports, we can rightly assume that the former is a server and the latter is a client.

We could also safely assume that the client address made many requests against the server, but in order to proceed without too many assumptions, we can use filters to inspect the traffic more closely.

In order to filter from the command line, we will use the source host (**`src host`**) and destination host (**`dst host`**) filters to output only source and destination traffic respectively. We can also filter by port number (**`-n port 81`**) to show both source and destination traffic against port 81. Let's try those filters now:

```
sudo tcpdump -n src host 172.16.40.10 -r password_cracking_filtered.pcap
...
08:51:20.801051 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr 25538253,nop,w
scale 4], length 0
08:51:20.802053 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [.], ack 89, win 905,
options [nop,nop,TS val 71430591 ecr 25538253], length 0
...
sudo tcpdump -n dst host 172.16.40.10 -r password_cracking_filtered.pcap
...
08:51:20.801048 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074, w
in 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.802026 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [.], ack 4166855390, w
in 115, options [nop,nop,TS val 25538253 ecr 71430591], length 0
...
sudo tcpdump -n port 81 -r password_cracking_filtered.pcap
...
08:51:20.800917 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [S], seq 1855084074, w
in 14600, options [mss 1460,sackOK,TS val 25538253 ecr 0,nop,wscale 7], length 0
08:51:20.800953 IP 172.16.40.10.81 > 208.68.234.99.60509: Flags [S.], seq 4166855389,
ack 1855084075, win 14480, options [mss 1460,sackOK,TS val 71430591 ecr 25538253,nop,w
scale 4], length 0
...
```

Listing 129 - Using tcpdump filters

We could continue to process this filtered output with various command-line utilities like awk and grep, but let's move along and actually inspect some packets in more detail to see what kind of details we can uncover.

To dump the captured traffic, we will use the **`-X`** option to print the packet data in both HEX and ASCII[108] format:

```
kali@kali:~$ sudo tcpdump -nX -r password_cracking_filtered.pcap
...
08:51:25.043062 IP 208.68.234.99.33313 > 172.16.40.10.81: Flags [P.], seq 1:140, ack 1
  0x0000:  4500 00bf 158c 4000 3906 9cea d044 ea63  E.....@.9....D.c
  0x0010:  ac10 280a 8221 0051 a726 a77c 6fd8 ee8a  ..(..!.Q.&.|o...
  0x0020:  8018 0073 1c76 0000 0101 080a 0185 b2f2  ...s.v..........
  0x0030:  0441 f5e3 4745 5420 2f2f 6164 6d69 6e20  .A..GET.//admin.
  0x0040:  4854 5450 2f31 2e31 0d0a 486f 7374 3a20  HTTP/1.1..Host:.
  0x0050:  6164 6d69 6e2e 6d65 6761 636f 7270 6f6e  admin.megacorpon
  0x0060:  652e 636f 6d3a 3831 0d0a 5573 6572 2d41  e.com:81..User-A
```

[108] (Wikipedia, 2019), https://en.wikipedia.org/wiki/ASCII

```
  0x0070:  6765 6e74 3a20 5465 6820 466f 7265 7374  gent:.Teh.Forest
  0x0080:  204c 6f62 7374 6572 0d0a 4175 7468 6f72  .Lobster..Author
  0x0090:  697a 6174 696f 6e3a 2042 6173 6963 2059  ization:.Basic.Y
  0x00a0:  5752 7461 5734 3662 6d46 7562 3352 6c59  WRtaW46bmFub3RlY
  0x00b0:  3268 7562 3278 765a 336b 780d 0a0d 0a    2hub2xvZ3kx....
...
```

Listing 130 - Using tcpdump to read the packet capture in hex/ascii output

We immediately notice that the traffic to 172.16.40.10 on port 81 looks like HTTP data. In fact, it seems like these HTTP requests contain Basic HTTP Authentication data, with the User agent "Teh Forest Lobster". This is a pretty clear sign that something strange is occurring.

In order to uncover the rest of the mystery, we will need to rely on advanced header filtering.

4.5.3 Advanced Header Filtering

At this point, to better inspect the requests and responses in the dump, we would like to filter out and display only the data packets. To do this, we will look for packets that have the *PSH* and *ACK* flags turned on. All packets sent and received after the initial 3-way handshake will have the *ACK* flag set. The *PSH* flag[109] is used to enforce immediate delivery of a packet and is commonly used in interactive *Application Layer* protocols to avoid buffering.

The following diagram depicts the TCP header and shows that the TCP flags are defined starting from the 14th byte.

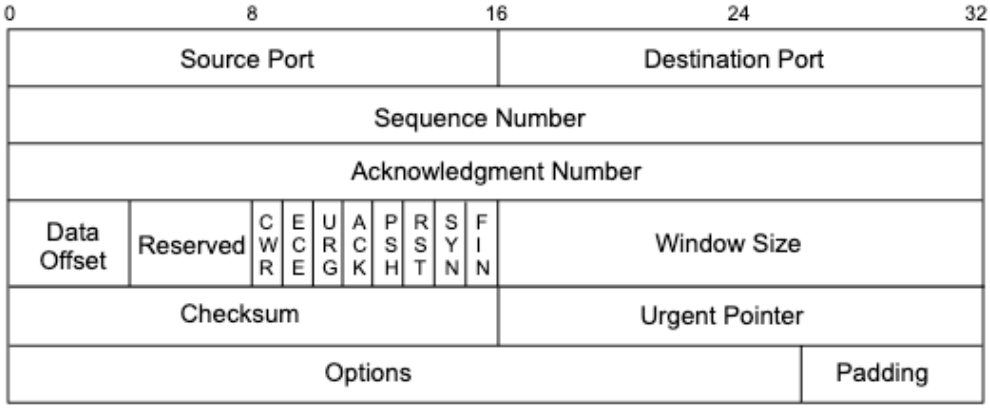


Figure 16: TCP packet displaying the flags in the 14th byte

Looking at Figure 16, we can see that *ACK* and *PSH* are represented by the fourth and fifth bits of the 14th byte, respectively:

```
CEUAPRSF
WCRCSSYI
REGKHTNN
00011000  = 24 in decimal
```

Listing 131 - Calculating the required bits

[109] (DARPA Inernet Program, 1981), https://tools.ietf.org/html/rfc793

Turning on only these bits would give us *00011000*, or decimal 24.

```
kali@kali:~$ echo "$((2#00011000))"
24
```

Listing 132 - Converting the binary bits to decimal in bash

We can pass this number to tcpdump with **'tcp[13] = 24'** as a display filter to indicate that we only want to see packets with the ACK and PSH bits set ("data packets") as represented by the fourth and fifth bits (**24**) of the 14th byte of the TCP header. Bear in mind, the tcpdump array index used for counting the bytes starts at zero, so the syntax should be (**tcp[13]**).

The combination of these two flags will hopefully show us only the HTTP requests and responses data. Here's the command we'll use to display packets that have the ACK or PSH flags set:

```
kali@kali:~$ sudo tcpdump -A -n 'tcp[13] = 24' -r password_cracking_filtered.pcap
06:51:20.802032 IP 208.68.234.99.60509 > 172.16.40.10.81: Flags [P.], seq 1855084075:1
E.....@.9....D.c..(
.].Qn.V+.]*....s1......
.....A..GET //admin HTTP/1.1
Host: admin.megacorpone.com:81
User-Agent: Teh Forest Lobster

...
E.....@.@.....(
.D.c.Q.^...E..?I...........
.A......HTTP/1.1 401 Authorization Required
Date: Mon, 22 Apr 2013 12:51:20 GMT
Server: Apache/2.2.20 (Ubuntu)
WWW-Authenticate: Basic realm="Password Protected Area"
Vary: Accept-Encoding
Content-Length: 488
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>401 Authorization Required</title>
</head><body>
<h1>Authorization Required</h1>
<p>This server could not verify that you
are authorized to access the document
requested.  Either you supplied the wrong
credentials (e.g., bad password), or your
browser doesn't understand how to supply
the credentials required.</p>
<hr>
<address>Apache/2.2.20 (Ubuntu) Server at admin.megacorpone.com Port 81</address>
</body></html>

...

08:51:25.044432 IP 172.16.40.10.81 > 208.68.234.99.33313:
E..s.m@.@..U..(
.D.c.Q.!o....&......^u.....
.A......HTTP/1.1 301 Moved Permanently
Date: Mon, 22 Apr 2013 12:51:25 GMT
```

```
Server: Apache/2.2.20 (Ubuntu)
Location: http://admin.megacorpone.com:81/admin/
Vary: Accept-Encoding
Content-Length: 333
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://admin.megacorpone.com:81/admin/">here</a>.</
p>
<hr>
<address>Apache/2.2.20 (Ubuntu) Server at admin.megacorpone.com Port 81</address>
</body></html>
```

Listing 133 - Using tcpdump with some advanced filtering

From here, our story becomes clearer. We see a significant amount of failed attempts to authenticate to the /admin directory, which resulted in HTTP 401 replies, while the last attempt to login seems to have succeeded, as the server replied with a HTTP 301 response. It seems someone gained access to one of megacorpone's servers!

4.5.3.1 Exercises

1. Use **tcpdump** to recreate the Wireshark exercise of capturing traffic on port 110.
2. Use the **-X** flag to view the content of the packet. If data is truncated, investigate how the **-s** flag might help.
3. Find all 'SYN', 'ACK', and 'RST' packets in the password_cracking_filtered.pcap file.
4. An alternative syntax is available in tcpdump where you can use a more user-friendly filter to display only ACK and PSH packets. Explore this syntax in the tcpdump manual by searching for "tcpflags". Come up with an equivalent display filter using this syntax to filter ACK and PSH packets.

4.6 Wrapping Up

In this module, we demonstrated some practical tools that are found in every pentester's toolkit including *Netcat*, *Socat*, *PowerShell*, *Wireshark*, and *Tcpdump*. These tools can assist in many ways during a penetration test, especially when a target is lacking in specialized tools or when we need to transfer small tools to expand our foothold on the target network.

5. Bash Scripting

The GNU Bourne-Again Shell (Bash)[110] is a powerful work environment and scripting engine. A competent security professional skillfully leverages Bash scripting to streamline and automate many Linux tasks and procedures. In this module, we will introduce Bash scripting and explore several practical scenarios.

5.1 Intro to Bash Scripting

A Bash script is a plain-text file that contains a series of commands that are executed as if they had been typed at a terminal prompt. Generally speaking, Bash scripts have an optional extension of .sh (for ease of identification), begin with **#!/bin/bash** and must have executable permissions set before they can be executed. Let's begin with a simple "Hello World" Bash script:

```
kali@kali:~$ cat ./hello-world.sh
#!/bin/bash
# Hello World Bash Script
echo "Hello World!"
```

Listing 134 - Creating a simple 'Hello World' Bash script

This script has several components worth explaining:

- Line 1: *#!* is commonly known as the *shebang*,[111] and is ignored by the Bash interpreter. The second part, */bin/bash*, is the absolute path[112] to the interpreter, which is used to run the script. This is what makes this a "Bash script" as opposed to another type of shell script, like a "C Shell script", for example.
- Line 2: *#* is used to add a comment, so all text that follows it is ignored.
- Line 3: *echo "Hello World!"* uses the *echo* Linux command utility to print a given string to the terminal, which in this case is "Hello World!".

Next, let's make the script executable and run it:

```
kali@kali:~$ chmod +x hello-world.sh

kali@kali:~$ ./hello-world.sh
Hello World!
```

Listing 135 - Running a simple 'Hello World' Bash script

The **chmod** command, along with the **+x** option is used to make the script executable, and **./hello-world.sh** is used to actually run it. The **./** notation may seem confusing but this is simply a path notation indicating that this script is in the current directory. Whenever we type a command, Bash tries to find it in a series of directories stored in a variable[113] called *PATH*. Since our home directory

[110] (GNU, 2017), http://www.gnu.org/software/bash/

[111] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Shebang_(Unix)

[112] (The Linux Information Project, 2005), http://www.linfo.org/absolute_pathname.html

[113] (O'Reilly Media, Inc., 1998), https://www.oreilly.com/library/view/learning-the-bash/1565923472/ch04s02.html.

is not included in that variable, we must use the relative path[114] to our Bash script in order for Bash to "find it" and run it.

Now that we have created our first Bash script, let's explore Bash in a bit more detail.

5.2 Variables

Variables are named places to temporarily store data. We can set (or "declare") a variable, which assigns a value to it, or read a variable, which will "expand" or "resolve" it to its stored value.

We can declare variable values in a number of ways. The easiest method is to set the value directly with a simple *name=value* declaration. Notice that there are no spaces before or after the "=" sign:

```
kali@kali:~$ first_name=Good
```

Listing 136 - Declaring a simple variable

Declaring a variable is pointless unless we can reference it. To do this, we precede the variable with the "$" character. Whenever Bash encounters this syntax in a command, it replaces the variable name with its value ("expands" the variable) before execution:

```
kali@kali:~$ first_name=Good

kali@kali:~$ last_name=Hacker

kali@kali:~$ echo $first_name $last_name
Good Hacker
```

Listing 137 - Declaring and displaying our own variables

Variable names may be uppercase, lowercase, or a mixture of both. However, Bash is case-sensitive so we must be consistent when declaring and expanding variables. In addition, it's good practice to use descriptive variable names, which make our scripts much easier to read and maintain.

Be advised that Bash interprets certain characters in specific ways. For example, this declaration demonstrates an improper multi-value variable declaration:

```
kali@kali:~$ greeting=Hello World
bash: World: command not found
```

Listing 138 - Attempting to assign a complex value to a variable

This was not necessarily what we expected. To fix this, we can use either single quotes (**'**) or double quotes (**"**) to enclose our text. However, Bash treats single and double quotes differently. When encountering single quotes, Bash interprets every enclosed character literally. When enclosed in double quotes, all characters are viewed literally except "*$*", "`", and "\" meaning variables will be expanded in an initial substitution pass on the enclosed text.

A simple example will help clarify this:

```
kali@kali:~$ greeting='Hello World'
```

[114] (The Linux Foundation, 2016), https://www.linux.com/blog/absolute-path-vs-relative-path-linuxunix

```
kali@kali:~$ echo $greeting
Hello World

kali@kali:~$ greeting2="New $greeting"

kali@kali:~$ echo $greeting2
New Hello World
```

Listing 139 - Using single and double quotes to illustrate complex variable assignment using a string

In this example, the single-quote-enclosed declaration of *greeting* preserved the value of our text exactly and did not interpret the space as a command delimiter. However, in the double-quote-enclosed declaration of *greeting2*, Bash expanded *$greeting* to its value ("Hello World"), honoring the special meaning of the "$" character.

We can also set the value of the variable to the result of a command or program. This is known as *command substitution*,[115] which allows us to take the output of a command or program (what would normally be printed to the screen) and have it saved as the value of a variable.

To do this, place the variable name in parentheses "**()**", preceded by a "**$**" character:

```
kali@kali:~$ user=$(whoami)

kali@kali:~$ echo $user
kali
```

Listing 140 - Illustrating the use of command substitution and variables

In Listing 140, we assigned the output of the **whoami** command to the *user* variable. We then displayed its value. An alternative syntax for command substitution using the backtick, or grave, character (`) is shown below:

```
kali@kali:~$ user2=`whoami`

kali@kali:~$ echo $user2
kali
```

Listing 141 - An alternative syntax for command substitution

The backtick method is older and typically discouraged as there are differences in how the two methods of command substitution behave.[116] It is also important to note that command substitution happens in a subshell and changes to variables in the subshell will not alter variables from the master process. This is demonstrated in the following example:

```
kali@kali:~$ cat ./subshell.sh
#!/bin/bash -x

var1=value1
echo $var1

var2=value2
echo $var2
```

[115] (GNU, 2019), https://www.gnu.org/software/bash/manual/html_node/Command-Substitution.html

[116] (BashFAQ, 2016), http://mywiki.wooledge.org/BashFAQ/082

```
$(var1=newvar1)
echo $var1

`var2=newvar2`
echo $var2

kali@kali:~$ ./subshell.sh
+ var1=value1
+ echo value1
value1
+ var2=value2
+ echo value2
value2
++ var1=newvar1
+ echo value1
value1
++ var2=newvar2
+ echo value2
value2
kali@kali:~$
```

Listing 142 - Command substitution in a subshell

In this example, first note that we changed the shebang, adding in the -x flag. This instructed Bash to print additional debug output, so we could more easily see the commands that were executed and their results. As we view this output, notice that commands preceded with a single "+" character were executed in the current shell and commands preceded with a double "++" were executed in a subshell.

This allows us to clearly see that the second declarations of *var1* and *var2* happened inside a subshell and did not change the values in the current shell as the initial declarations did.

5.2.1 Arguments

Not all Bash scripts require arguments.[117] However, it is extremely important to understand how they are interpreted by Bash and how to use them. We have already executed Linux commands with arguments. For example, when we run the command **`ls -l /var/log`**, both **`-l`** and **`/var/log`** are arguments to the **`ls`** command.

Bash scripts are no different; we can supply command-line arguments and use them in our scripts:

```
kali@kali:~$ cat ./arg.sh
#!/bin/bash

echo "The first two arguments are $1 and $2"

kali@kali:~$ chmod +x ./arg.sh

kali@kali:~$ ./arg.sh hello there
The first two arguments are hello and there
```

Listing 143 - Illustrating the use of arguments in Bash

[117] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Parameter_(computer_programming)

In Listing 143, we created a simple Bash script, set executable permissions on it, and then ran it with two arguments. The *$1* and *$2* variables represent the first and second arguments passed to the script. Let's explore a few special Bash variables:

Variable Name	Description
$0	The name of the Bash script
$1 - $9	The first 9 arguments to the Bash script
$#	Number of arguments passed to the Bash script
$@	All arguments passed to the Bash script
$?	The exit status of the most recently run process
$$	The process ID of the current script
$USER	The username of the user running the script
$HOSTNAME	The hostname of the machine
$RANDOM	A random number
$LINENO	The current line number in the script

Table 4 - Special Bash variables

Some of these special variables can be very useful when debugging a script. For example, we might be able to obtain the exit status of a command to determine whether it was successfully executed or not.

5.2.2 Reading User Input

Command-line arguments are a form of user input, but we can also capture interactive user input while a script is running with the **read** command. In this example, we will use **read** to capture user input and assign it to a variable:

```
kali@kali:~$ cat ./input.sh
#!/bin/bash

echo "Hello there, would you like to learn how to hack: Y/N?"

read answer

echo "Your answer was $answer"

kali@kali:~$ chmod +x ./input.sh

kali@kali:~$ ./input.sh
Hello there, would you like to learn how to hack: Y/N?
Y
Your answer was Y
```

Listing 144 - Collecting user input using read

We can alter the behavior of the **read** command with various command line options. Two of the most commonly used options include **-p**, which allows us to specify a prompt, and **-s**, which makes the user input silent. The latter is ideal for capturing user credentials:

```
kali@kali:~$ cat ./input2.sh
#!/bin/bash
# Prompt the user for credentials
```

```
read -p 'Username: ' username
read -sp 'Password: ' password

echo "Thanks, your creds are as follows: " $username " and " $password

kali@kali:~$ chmod +x ./input2.sh

kali@kali:~$ ./input2.sh
Username: kali
Password:
Thanks, your creds are as follows:  kali  and  nothing2see!
```

Listing 145 - Prompting user for input and silently reading it using read

5.3 If, Else, Elif Statements

Conditional statements allow us to perform different actions based on different conditions. The most common conditional Bash statements include *if*, *else*, and *elif*.

The *if* statement is relatively simple–it checks to see if a condition is true–but it requires a very specific syntax. Pay careful attention to this syntax, especially the use of required spaces:

```
if [ <some test> ]
then
  <perform an action>
fi
```

Listing 146 - General syntax for the if statement

In this listing, if "some test" evaluates as true, the script will "perform an action", or any commands between *then* and *fi*. Let's look at an actual example:

```
kali@kali:~$ cat ./if.sh
#!/bin/bash
# if statement example

read -p "What is your age: " age

if [ $age -lt 16 ]
then
  echo "You might need parental permission to take this course!"
fi

kali@kali:~$ chmod +x ./if.sh

kali@kali:~$ ./if.sh
What is your age: 15
You might need parental permission to take this course!
```

Listing 147 - Using the if statement in Bash

In this example, we used an *if* statement to check the age entered by a user. If the entered age was less than (**-lt**) 16, the script would output a warning message.

The square brackets ("[" and "]") in the *if* statement above are actually a reference to the *test* command. This simply means we can use all of the operators that are allowed by the *test* command. Some of the most common operators include:

Operator	Description: Expression True if…
!EXPRESSION	The EXPRESSION is false.
-n STRING	STRING length is greater than zero
-z STRING	The length of STRING is zero (empty)
STRING1 != STRING2	STRING1 is not equal to STRING2
STRING1 = STRING2	STRING1 is equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is equal to INTEGER2
INTEGER1 -ne INTEGER2	INTEGER1 is not equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is less than INTEGER2
INTEGER1 -ge INTEGER2	INTEGER1 is greater than or equal to INTEGER 2
INTEGER1 -le INTEGER2	INTEGER1 is less than or equal to INTEGER 2
-d FILE	FILE exists and is a directory
-e FILE	FILE exists
-r FILE	FILE exists and has read permission
-s FILE	FILE exists and it is not empty
-w FILE	FILE exists and has write permission
-x FILE	FILE exists and has execute permission

Table 5 - Common test command operators

With the above in mind, our previous example using *if* can be rewritten without square brackets as follows:

```
kali@kali:~$ cat ./if2.sh
#!/bin/bash
# if statement example 2

read -p "What is your age: " age

if test $age -lt 16
then
  echo "You might need parental permission to take this course!"
fi

kali@kali:~$ chmod +x ./if2.sh

kali@kali:~$ ./if2.sh
What is your age: 15
You might need parental permission to take this course!
```

Listing 148 - Using the test command in an if statement

Even though this example is functionally equivalent to the example using square brackets, using square brackets makes the code slightly easier to read.

We can also perform a certain set of actions if a statement is true and another set if it is false. To do this, we can use the *else* statement, which has the following syntax:

```
if [ <some test> ]
then
  <perform action>
else
```

```
  <perform another action>
fi
```

Listing 149 - General syntax for the else statement

Let's extend our previous "age" example to include the *else* statement:

```
kali@kali:~$ cat ./else.sh
#!/bin/bash
# else statement example

read -p "What is your age: " age

if [ $age -lt 16 ]
then
  echo "You might need parental permission to take this course!"
else
  echo "Welcome to the course!"
fi

kali@kali:~$ chmod +x ./else.sh

kali@kali:~$ ./else.sh
What is your age: 21
Welcome to the course!
```

Listing 150 - Using the else statement in Bash

Notice that the *else* statement was executed when the entered age was greater than (or more specifically "not less than") sixteen.

The *if* and *else* statements only allow two code execution branches. We can add additional branches with the *elif* statement which uses the following pattern:

```
if [ <some test> ]
then
  <perform action>
elif [ <some test> ]
then
  <perform different action>
else
  <perform yet another different action>
fi
```

Listing 151 - The elif syntax in Bash

Let's again extend our "age" example to include the *elif* statement:

```
kali@kali:~$ cat ./elif.sh
#!/bin/bash
# elif example

read -p "What is your age: " age

if [ $age -lt 16 ]
then
  echo "You might need parental permission to take this course!"
elif [ $age -gt 60 ]
then
```

```
  echo "Hats off to you, respect!"
else
  echo "Welcome to the course!"
fi

kali@kali:~$ chmod +x ./elif.sh

kali@kali:~$ ./elif.sh
What is your age: 65
Hats off to you, respect!
```

Listing 152 - Using the elif statement in Bash

In this example, the code execution flow was slightly more complex. In order of operation, the *then* branch executes if the entered age is less than sixteen, the *elif* branch is entered (and the "Hats off.." message displayed) if the age is greater than sixty, and the else branch executes only if the age is greater than sixteen but less than sixty.

5.4 Boolean Logical Operations

Boolean logical operators,[118] like *AND* (&&) and *OR* (||) are somewhat mysterious because Bash uses them in a variety of ways.

One common use is in *command lists*, which are chains of commands whose flow is controlled by operators. The "|" (pipe) symbol is a commonly-used operator in a command list and passes the output of one command to the input of another. Similarly, boolean logical operators execute commands based on whether a previous command succeeded (or returned True or *0*) or failed (returned False or non-zero).

Let's take a look at the *AND* (&&) boolean operator first, which executes a command only if the previous command succeeds (or returns True or *0*):

```
kali@kali:~$ user2=kali

kali@kali:~$ grep $user2 /etc/passwd && echo "$user2 found!"
kali:x:1000:1000:,,,:/home/kali:/bin/bash
kali found!

kali@kali:~$ user2=bob

kali@kali:~$ grep $user2 /etc/passwd && echo "$user2 found!"
```

Listing 153 - Using the AND (&&) boolean operator in a command list

In this example, we first assigned the username we are searching for to the *user2* variable. Next, we use the **grep** command to check if a certain user is listed in the /etc/passwd file, and if it is, **grep** returns *True* and the **echo** command is executed. However, when we try searching for a user that we know does not exist in the /etc/passwd file, our **echo** command is not executed.

[118] (MIT), https://libguides.mit.edu/c.php?g=175963&p=1158594

When used in a command list, the *OR* (||) operator is the opposite of *AND* (&&); it executes the next command only if the previous command failed (returned False or non-zero):

```
kali@kali:~$ echo $user2
bob

kali@kali:~$ grep $user2 /etc/passwd && echo "$user2 found!" || echo "$user2 not found
!"
bob not found!
```

Listing 154 - Using the OR (||) boolean operator in a command list

In the above example, we took our previous command a step further and added the *OR* (||) operator followed by a second **echo** command. Now, when **grep** does not find a matching line and returns *False*, the second **echo** command after the *OR* (||) operator is executed instead.

These operators can also be used in a *test* to compare variables or the results of other tests. When used this way, *AND* (&&) combines two simple conditions, and if they are *both* true, the combined result is success (or True or *0*).

Consider this example:

```
kali@kali:~$ cat ./and.sh
#/bin/bash
# and example

if [ $USER == 'kali' ] && [ $HOSTNAME == 'kali' ]
then
  echo "Multiple statements are true!"
else
  echo "Not much to see here..."
fi

kali@kali:~$ chmod +x ./and.sh

kali@kali:~$ ./and.sh
Multiple statements are true!

kali@kali:~$ echo $USER && echo $HOSTNAME
kali
kali
```

Listing 155 - Using the and (&&) boolean operator to test multiple conditions in Bash

In this example, we used *AND* (&&) to test multiple conditions and since both variable comparisons were true, the whole *if* line succeeded, so the *then* branch executed.

When used in a *test*, the *OR* (||) boolean operator is used to test one or more conditions, but *only one* of them has to be true to count as success.

Let's take a look at an example:

```
kali@kali:~$ cat ./or.sh
#!/bin/bash
# or example

if [ $USER == 'kali' ] || [ $HOSTNAME == 'pwn' ]
```

```
then
  echo "One condition is true, this line is printed"
else
  echo "You are out of luck!"
fi

kali@kali:~$ chmod +x ./or.sh

kali@kali:~$ ./or.sh
One condition is true, this line is printed

kali@kali:~$ echo $USER && echo $HOSTNAME
kali
kali
```

Listing 156 - Using the or (||) boolean operator to test multiple conditions in Bash

In this example, we used OR (||) to test multiple conditions and since one of the variable comparisons was true, the whole *if* line succeeded, so the *then* branch executed.

5.5 Loops

In computer programming, *loops*[119] help us with repetitive tasks that we need to run until a certain criteria is met. Iteration is particularly useful for penetration testers, so we recommend paying very close attention to this section.

In Bash, the two most predominant loop commands are *for*[120] and *while*.[121] We will take a look at both.

5.5.1 For Loops

For loops are very practical and work very well in Bash one-liners.[122] This type of loop is used to perform a given set of commands for each of the items in a list. Let's briefly look at its general syntax:

```
for var-name in <list>
do
  <action to perform>
done
```

Listing 157 - General syntax of the for loop

The *for* loop will take each item in the *list* (in order), assign that item as the value of the variable *var-name*, perform the given action between *do* and *done*, and then go back to the top, grab the next item in the list, and repeat the steps until the list is exhausted.

[119] (Whatis.com, 2005), http://whatis.techtarget.com/definition/loop

[120] (The Linux Foundation, 2010), https://www.linux.com/learn/essentials-bash-scripting-using-loops

[121] (Bash Guide for Beginners, 2008), http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_09_02.html

[122] (Bash One-Liners, 2019), http://www.bashoneliners.com/

Let's take a look at a more practical example that will quickly print the first 10 IP addresses in the 10.11.1.0/24 subnet:[123]

```
kali@kali:~$ for ip in $(seq 1 10); do echo 10.11.1.$ip; done
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
10.11.1.10
```

Listing 158 - An example using for loops in Bash

In this Bash one-liner (Listing 158), we used the **seq** command to print a sequence of numbers, in this case the numbers one through ten. Each number is then assigned to the *ip* variable, and then each IP address is displayed to the screen as the *for* loop runs multiple times, exiting at the end of the sequence.

Another way of re-writing the previous *for* loop involves *brace expansion*[124] using ranges.[125] Brace expansion using ranges is written giving the first and last values of the range and can be a sequence of numbers or characters. This is known as a "sequence expression":

```
kali@kali:~$ for i in {1..10}; do echo 10.11.1.$i;done
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
10.11.1.10
```

Listing 159 - Brace expansion using ranges in Bash

There is a lot of potential for this type of loop. Displaying IP addresses to the screen may not seem very useful, but we can use the same loop to run a port scan[126] using **nmap**[127] (which we discuss in detail in another module). We can also attempt to use the **ping** command to see if any of the IP addresses respond to ICMP echo requests,[128] etc.

[123] (Cisco, 2016), https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13788-3.html

[124] (GNU, 2019), https://www.gnu.org/software/bash/manual/html_node/Brace-Expansion.html

[125] (Bash Hackers Wiki, 2014), http://wiki.bash-hackers.org/syntax/expansion/brace

[126] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Port_scanner

[127] (Nmap, 2019), http://nmap.org/

[128] (Firewall.cx, 2018), http://www.firewall.cx/networking-topics/protocols/icmp-protocol/152-icmp-echo-ping.html

5.5.2 While Loops

While loops are also fairly common and execute code while an expression is true. *While* loops have a simple format and, like *if*, use the square brackets ([]) for the test:

```
while [ <some test> ]
do
  <perform an action>
done
```

Listing 160 - General syntax of the while loop

Let's re-create the previous example with a *while* loop:

```
kali@kali:~$ cat ./while.sh
#!/bin/bash
# while loop example

counter=1

while [ $counter -lt 10 ]
do
  echo "10.11.1.$counter"
  ((counter++))
done

kali@kali:~$ chmod +x ./while.sh

kali@kali:~$ ./while.sh
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
```

Listing 161 - Using a while loop in Bash

This is not the output we expected. This is a common mistake called an "*off by one*"[129] error. In the example above, we used *-lt* (less than) instead of *-le* (less than *or equal to*), so our counter only got to nine, not ten as originally intended.

The *((counter++))* line uses the double-parenthesis *(( ))*[130] construct to perform arithmetic expansion and evaluation at the same time. In this particular case, we use it to increase our *counter* variable by one. Let's re-write the *while* loop and try the example again:

```
kali@kali:~$ cat ./while2.sh
#!/bin/bash
# while loop example 2
```

[129] (Stack Overflow, 2019), https://stackoverflow.com/questions/2939869/what-is-exactly-the-off-by-one-errors-in-the-while-loop

[130] (Advanced Bash-Scripting Guide, 2014), http://tldp.org/LDP/abs/html/dblparens.html

```
counter=1

while [ $counter -le 10 ]
do
  echo "10.11.1.$counter"
  ((counter++))
done

kali@kali:~$ chmod +x ./while2.sh

kali@kali:~$ ./while2.sh
10.11.1.1
10.11.1.2
10.11.1.3
10.11.1.4
10.11.1.5
10.11.1.6
10.11.1.7
10.11.1.8
10.11.1.9
10.11.1.10
```

Listing 162 - An example using a while loop in Bash

Good. Our *while* loop is looking much better now.

5.6 Functions

In terms of Bash scripting, we can think of a function as a script within a script, which is useful when we need to execute the same code multiple times in a script. Rather than re-writing the same chunk of code over and over, we just write it once as a function and then call that function as needed.

Put another way, a function is a subroutine, or a code block that implements a set of operations–a "black box" that performs a specified task. Functions may be written in two different formats. The first format is more common to Bash scripts:

```
function function_name {
commands...
}
```

Listing 163 - One way of writing a function in Bash

The second format is more familiar to C programmers:

```
function_name () {
commands...
}
```

Listing 164 - Another way of writing a function in Bash

The formats are functionally identical and are a matter of personal preference. Let's look at a simple example:

```
kali@kali:~$ cat ./func.sh
#!/bin/bash
```

```
# function example

print_me () {
  echo "You have been printed!"
}

print_me

kali@kali:~$ chmod +x ./func.sh

kali@kali:~$ ./func.sh
You have been printed!
```

Listing 165 - Using a Bash function to print a message to the screen

Functions can also accept arguments:

```
kali@kali:~$ cat ./funcarg.sh
#!/bin/bash
# passing arguments to functions

pass_arg() {
  echo "Today's random number is: $1"
}

pass_arg $RANDOM

kali@kali:~$ chmod +x ./funcarg.sh

kali@kali:~$ ./funcarg.sh
Today's random number is: 25207
```

Listing 166 - Passing an argument to a function in Bash

In this case, we passed a random number, *$RANDOM*, into the function, which outputs it as *$1*, the functions first argument. Note that the function definition (**`pass_arg()`**) contains parentheses. In other programming languages, such as C, these would contain the expected arguments, but in Bash the parentheses serve only as decoration. They are never used. Also note that the function definition (the function itself) must appear in the script before it is called. Logically, we can't call something we have not defined.

Use a descriptive function name that describe the function's purpose.

In addition to passing arguments to Bash functions, we can of course return values from Bash functions as well. Bash functions do not actually allow you to return an arbitrary value in the traditional sense. Instead, a Bash function can *return* an exit status (zero for success, non-zero for failure) or some other arbitrary value that we can later access from the *$?* global variable (see Table 4). Alternatively, we can set a global variable inside the function or use command substitution to simulate a traditional return.

Let's look at a simple example that returns a random number into *$?*:

```
kali@kali:~$ cat funcrvalue.sh
#!/bin/bash
# function return value example

return_me() {
  echo "Oh hello there, I'm returning a random value!"
  return $RANDOM
}

return_me

echo "The previous function returned a value of $?"

kali@kali:~$ chmod +x ./funcrvalue.sh

kali@kali:~$ ./funcrvalue.sh
Oh hello there, I'm returning a random value!
The previous function returned a value of 198

kali@kali:~$ ./funcrvalue.sh
Oh hello there, I'm returning a random value!
The previous function returned a value of 313
```

Listing 167 - Returning a value from a function in Bash

Notice that a random number is returned every time we run the script, because we returned the special global variable $RANDOM (into *$?*). If we used the *return* statement without the $RANDOM argument, the exit status of the function (*0* in this case) would be returned instead.

Now that we have a basic understanding of variables and functions, we can dig deeper and discuss *variable scope*.[131]

The scope of a variable is simply the context in which it has meaning. By default, a variable has a *global* scope, meaning it can be accessed throughout the entire script. In contrast, a *local* variable can only be seen within the function, block of code, or subshell in which it is defined. We can "overlay" a global variable, giving it a local context, by preceding the declaration with the *local* keyword, leaving the global variable untouched. The general syntax is:

```
local name="Joe"
```

Listing 168 - Declaring a local variable

Let's see how *local* and *global* variables work in practice with a simple example:

```
kali@kali:~$ cat ./varscope.sh
#!/bin/bash
# var scope example

name1="John"
name2="Jason"

name_change() {
  local name1="Edward"
  echo "Inside of this function, name1 is $name1 and name2 is $name2"
```

[131] (Advanced Bash-Scripting Guide, 2014), http://tldp.org/LDP/abs/html/localvar.html

```
    name2="Lucas"
}

echo "Before the function call, name1 is $name1 and name2 is $name2"

name_change

echo "After the function call, name1 is $name1 and name2 is $name2"

kali@kali:~$ chmod +x ./varscope.sh

kali@kali:~$ ./varscope.sh
Before the function call, name1 is John and name2 is Jason
Inside of this function, name1 is Edward and name2 is Jason
After the function call, name1 is John and name2 is Lucas
```

Listing 169 - Illustrating variable scope in Bash

Let's highlight a few key points within Listing 169. First note that we declared two *global* variables, setting *name1* to *John* and *name2* to *Jason*.

Then, we defined a function and inside that function, declared a local variable called *name1*, setting the value to *Edward*. Since this was a local variable, the previous global assignment was not affected; *name1* will still be set to *John* outside this function.

Next, we set *name2* to Lucas, and since we did not use the *local* keyword, we are changing the global variable, and the assignment sticks both inside and outside of the function.

Based on this example, the following two points summarize variable scope:

- Changing the value of a local variable with the same name as a global one will not affect its global value.
- Changing the value of a global variable inside of a function – without having declared a local variable with the same name – will affect its global value.

5.7 Practical Examples

So far, we have covered the basics of Bash scripting. Let's put together all of the concepts we have discussed and walk through a few practical examples.

5.7.1 Practical Bash Usage – Example 1

In this example, we want to find all the subdomains listed on the main megacorpone.com web page and find their corresponding IP addresses. Doing this manually would be frustrating and time consuming, but with some basic Bash commands, we can turn this into an easy task. We'll start by downloading the index page with **wget**:

```
kali@kali:~$ wget www.megacorpone.com
URL transformed to HTTPS due to an HSTS policy
--2018-03-18 17:56:53--  http://www.megacorpone.com/
Resolving www.megacorpone.com (www.megacorpone.com)... 38.100.193.76
Connecting to www.megacorpone.com (www.megacorpone.com)|38.100.193.76|:80... connected
HTTP request sent, awaiting response... 200 OK
Length: 12520 (12K) [text/html]
```

```
Saving to: 'index.html'

index.html          100%[===================>]  12.23K  --.-KB/s    in 0s

2018-03-18 17:56:54 (2.56 MB/s) - 'index.html' saved [12520/12520]

kali@kali:~$ ls -l index.html
-rw-r--r-- 1 kali kali 12520 Mar 18 17:56 index.html
```

Listing 170 - Downloading the index.html page from megacorpone.com

Manually scanning the file, we see many lines we don't need. Let's start narrowing in on lines that we need, and strip out lines we don't. First, we can use **grep "href="** to extract all the lines in index.html that contain HTML links:

```
kali@kali:~$ grep "href=" index.html
...
<p><a href="http://beta.megacorpone.com/util/files/news.html">MegaCorp One CEO Joe She
er nominated for Nobel Physics, Medicine, and Literature prizes.</a></p>
                    <p><small>This is a fictitious company, brought to you by <a href=
"http://www.offensive-security.com/" target="_blank">Offensive Security</a>.</small></
p>
                        <a href="https://www.facebook.com/MegaCorp-One-393570024393695
/" target="_blank"><i class="fa fa-facebook"></i></a>
                        <a href="https://twitter.com/joe_sheer/"><i class="fa fa-twitt
er"></i></a>
                        <a href="https://www.linkedin.com/company/18268898/" target="_
blank"><i class="fa fa-linkedin"></i></a>
                        <a href="https://github.com/megacorpone" target="_blank"><i cl
ass="fa fa-github"></i></a>
...
```

Listing 171 - Identifying hyperlinks in the index.html file

In the excerpt above, the first line is a prime example of what we're looking for as it references a subdomain.

Let's use **grep** to grab lines that contain ".megacorpone", indicating the existence of a subdomain, and **grep -v** to strip away lines that contain the boring "www.megacorpone.com" domain we already know about:

```
kali@kali:~$ grep "href=" index.html | grep "\.megacorpone" | grep -v "www\.megacorpon
e\.com" | head
...
<li><a href="http://support.megacorpone.com/ticket/requests/index.html">Nanoprocessors
</a></li>
                    <li><a href="http://syslog.megacorpone.com/logs/sys/view.php">Perl
in VanHook Chemical Dispersal</a></li>
                    <li><a href="http://test.megacorpone.com/demo/index.php">What are
the ethics behind MegaCorp One?</a></li>
...
```

Listing 172 - Using grep to narrow our search

This output looks closer to what we need. By reducing our data in a logical way and making sequentially smaller reductions with each pass, we are in the midst of the most common cycle in data handling.

It looks like each line contains a link, and a subdomain, but we need to get rid of the extra HTML around our links. There are always multiple approaches to any task performed in Bash, but we'll use a little-known one for this. We will use the **-F** option of **awk** to set a multi-character delimiter, unlike **cut**, which is simple and handy but only allows single-character delimiters. We will set our delimiter to *http://* and tell **awk** we want the second field (**'{print $2}'**), or everything after that delimiter:

```
kali@kali:~$ grep "href=" index.html | grep "\.megacorpone" | grep -v "www\.megacorpon
e\.com" | awk -F "http://" '{print $2}'
admin.megacorpone.com/admin/index.html">Cell Regeneration</a></li>
intranet.megacorpone.com/pear/">Immune Systems Supplements</a></li>
mail.megacorpone.com/menu/">Micromachine Cyberisation Repair</a></li>
mail2.megacorpone.com/smtp/relay/">Nanomite Based Weaponry Systems</a></li>
siem.megacorpone.com/home/">Nanoprobe Based Entity Assimilation</a></li>
support.megacorpone.com/ticket/requests/index.html">Nanoprocessors</a></li>
...
```

Listing 173 - Using awk with a unique delimiter search

The beginning of each line in our output shows that we're on the right track. Now, we can use **cut** to set the delimiter to *"/"* (with **-d**) and print the first field (with **-f 1**), leaving us with only the full subdomain name:

```
kali@kali:~$ grep "href=" index.html | grep "\.megacorpone" | grep -v "www\.megacorpon
e\.com" | awk -F "http://" '{print $2}' | cut -d "/" -f 1
admin.megacorpone.com
intranet.megacorpone.com
mail.megacorpone.com
mail2.megacorpone.com
siem.megacorpone.com
support.megacorpone.com
...
```

Listing 174 - Cutting the domain names

This looks great! However, any Bash guru will take one look at our work and scoff. That's deserved because we've used basic tools in a clumsy way, even though our reductions were rather sound. Bash and its associated commands and built-ins are extremely powerful, and there's always more to learn.

If we were to criticize our work in an effort to improve (which we should always do) we might see some simple ways to improve. First, we began our search with **"href="** and later searched for **http://**. These are both essentially links, but this requires that every line has both strings. If a line only had **http://**, but not **"href="**, we would have missed a line. Redundancy should be avoided, especially when working with large files. In addition, we don't really care about links, necessarily, we are looking for subdomains ending in ".megacorpone.com" regardless of whether or not the reference is contained in a link.

Another thing to consider is that we spent a lot of time and energy whittling away at the results to find and carve out the subdomain names. This was clumsy, prone to error, and pointless when a well-formed regular expression search could handle this easily. We've mentioned the power of regular expressions before, but let's look at a practical example now that we've taken the hard route to this problem's solution.

In this example, we will use a simple regular expression to carve ".megacorpone.com" subdomains out of our file:

```
kali@kali:~$ grep -o '[^/]*\.megacorpone\.com' index.html | sort -u > list.txt

kali@kali:~$ cat list.txt
admin.megacorpone.com
beta.megacorpone.com
intranet.megacorpone.com
mail2.megacorpone.com
mail.megacorpone.com
siem.megacorpone.com
support.megacorpone.com
...
```

Listing 175 - A more elegant solution with regular expressions

This solution is quite compact, but introduces some new techniques. First, notice the **grep -o** option, which only returns the string defined in our regular expression. If we form our expression carefully, this single command will handle much of our previous data carving. The expression itself looks complex but is fairly straightforward.

The string we are searching for (**'[^/]*\.megacorpone\.com'**) is wrapped in single-quotes, which, as we mentioned, will not allow variable expansions and will treat all enclosed characters literally.

The first block in the expression (**[^/]***) is a negated (**^**) set (**[ ]**), which searches for any number of characters (*) not including a forward-slash. Notice that the periods are escaped with a backslash (**\.**) to reinforce that we are looking for a literal period. Next, the string must end with ".megacorpone.com". When grep finds a matching string, it will carve it from the line and return it.

For later use, we could include other characters in a negated list by including them in a comma-delimited list. This block, (**[^/,"]***), would exclude both forward-slash and double-quote characters, for example.

This is a lot of new material and can seem overwhelming, but this is a great reusable regular expression that finds any string ending with ".megacorpone.com" found after a forward-slash, and dutifully carves out URL-referenced subdomains. We can later reuse this regular expression to carve any number of strings from a file.

We could have done more with this regular expression, but it's a great start and a good example of why regular expressions are so valuable.

Now we have a nice, clean list of domain names linked from the front page of megacorpone.com. Next, we will use **host** to discover the corresponding IP address of each domain name in our text file. We can use a Bash one-liner loop for this:

```
kali@kali:~$ for url in $(cat list.txt); do host $url; done
admin.megacorpone.com has address 38.100.193.83
beta.megacorpone.com has address 38.100.193.88
intranet.megacorpone.com has address 38.100.193.87
mail2.megacorpone.com has address 38.100.193.73
```

Listing 176 - Looking for IP addresses using the host command

The **host** command gives us all sorts of output and not all of it is relevant. We will extract the IP addresses by piping the output into a **grep** for "has address", then **cut** the results and **sort** them:

```
kali@kali:~$ for url in $(cat list.txt); do host $url; done | grep "has address" | cut
-d " " -f 4 | sort -u
173.246.47.170
38.100.193.66
38.100.193.67
38.100.193.73
38.100.193.76
38.100.193.77
38.100.193.79
38.100.193.83
38.100.193.84
38.100.193.87
38.100.193.88
38.100.193.89
```

Listing 177 - Extracting the IP addresses only

Nice! We've extracted the ".megacorpone.com" subdomains from the web page and obtained the corresponding IP addresses. As we've seen, there are a number of ways we can handle data with both Bash and related utilities, as well as with regular expressions, and this reinforces the fact that any time spent studying these topics in depth will save a great deal of time handling data or expediting processes in the future.

5.7.2 Practical Bash Usage – Example 2

In this example, let's assume we are in the middle of a penetration test and have unprivileged access to a Windows machine. As we continue to collect information, we realize it may be vulnerable to an exploit that we read about that began with the letters a, f, and d but we can't remember the full name of the exploit. In an attempt to escalate our privileges, we want to search for that specific exploit.

To do this, we will need to search https://www.exploit-db.com for "afd windows", download the exploits that match our search criteria, and inspect them until we find the proper one. We could do this manually through the web site, which wouldn't take too long, but if we take the time to write a Bash script, we could easily use it to search and automatically download exploits later.

Using what we now know about scripting, let's try to automate this task.

We'll start with the *SearchSploit*[132] utility on Kali Linux. SearchSploit is a command line search tool for Exploit-DB that allows us to take an offline copy of the Exploit Database with us wherever we go. We will pass "afd windows" as a search string, use **-w** to return the URL for https://www.exploit-db.com rather than the local path, and **-t** to search the exploit title:

```
kali@kali:~$ searchsploit afd windows -w -t
---------------------------------------------------------------------------------
 Exploit Title                                |  URL
---------------------------------------------------------------------------------
Microsoft Windows (x86) - 'afd.sys' Privil | https://www.exploit-db.com/exploits/40564
```

[132] (Offensive Security, 2019), https://www.exploit-db.com/searchsploit/

```
Microsoft Windows - 'AfdJoinLeaf' Privileg | https://www.exploit-db.com/exploits/21844
Microsoft Windows - 'afd.sys' Local Kernel | https://www.exploit-db.com/exploits/18755
Microsoft Windows 7 (x64) - 'afd.sys' Dang | https://www.exploit-db.com/exploits/39525
Microsoft Windows 7 (x86) - 'afd.sys' Dang | https://www.exploit-db.com/exploits/39446
Microsoft Windows 7 Kernel - Pool-Based Ou | https://www.exploit-db.com/exploits/42009
Microsoft Windows XP - 'afd.sys' Local Ker | https://www.exploit-db.com/exploits/17133
Microsoft Windows XP/2003 - 'afd.sys' Priv | https://www.exploit-db.com/exploits/6757
Microsoft Windows XP/2003 - 'afd.sys' Priv | https://www.exploit-db.com/exploits/18176
---------------------------------------------------------------------------------------
```

Listing 178 - Using searchsploit to search for an exploit

This is a good start, but we need to trim the results. For now, we're only interested in the exploit's URL, so let's **grep** for "http" and then **cut** what we need. We will use a "|" field delimiter and extract the second field:

```
kali@kali:~$ searchsploit afd windows -w -t | grep http | cut -f 2 -d "|"
 https://www.exploit-db.com/exploits/40564
 https://www.exploit-db.com/exploits/21844
 https://www.exploit-db.com/exploits/18755
 https://www.exploit-db.com/exploits/39525
 https://www.exploit-db.com/exploits/39446
 https://www.exploit-db.com/exploits/42009
 https://www.exploit-db.com/exploits/17133
 https://www.exploit-db.com/exploits/18176
 https://www.exploit-db.com/exploits/6757
```

Listing 179 - Extracting the URL from the output of searchsploit

That looks a little better. Now that we have the URL for each exploit, we can use a Bash loop to download the files and save them locally. However, before we do that, we notice that each page has a link to download the "raw" exploit code, which is really what we're after:

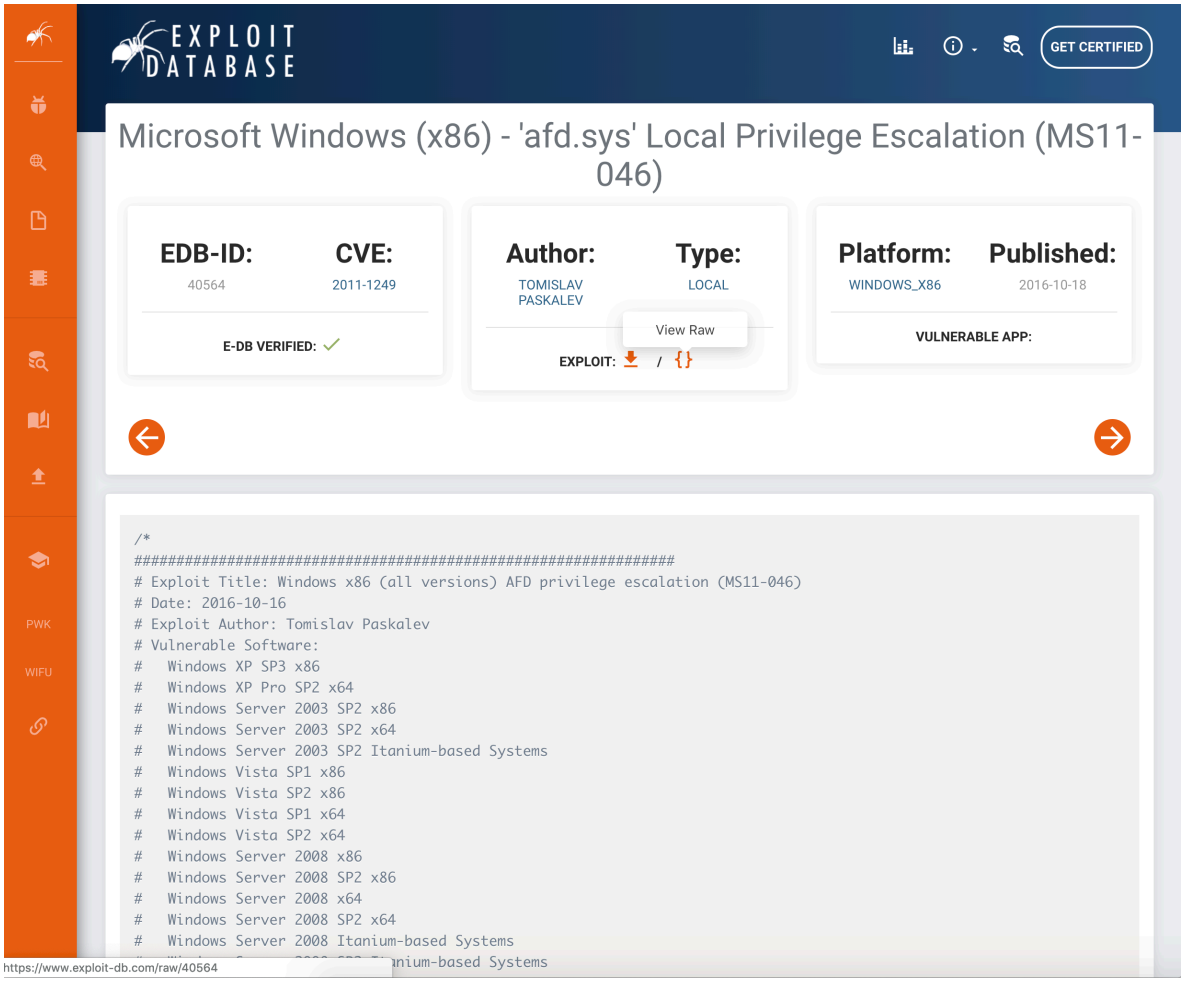


Figure 17: Exploit-DB raw download URL

We see that each original page (like */exploits/40564*) links to a raw exploit (like */raw/40564*). Armed with this information, we run sed (**`sed 's/exploits/raw/'`**) to modify the download URL and insert it into a Bash one-liner to download the raw code for the exploits:

```
kali@kali:~$ for e in $(searchsploit afd windows -w -t | grep http | cut -f 2 -d "|"); do exp_name=$(echo $e | cut -d "/" -f 5) && url=$(echo $e | sed 's/exploits/raw/') && wget -q --no-check-certificate $url -O $exp_name; done
```

Listing 180 - Downloading all exploits using some Bash-fu

Note the use of a for loop that iterates through the *SearchSploit* URLs we grabbed. Inside the loop, we set *exp_name* to the "name" of the exploit (using grep, cut, and command substitution), *url* to the raw download location (again with sed and command substitution). If that is successful (*&&*), we grab the exploit with *wget* (in quiet mode with no certificate check) saving it locally with the exploit name as the local file name.

Once we run it, we wait for the command prompt and verify that the exploits were indeed downloaded, using **`file`** to verify that the files are text:

```
kali@kali:~$ ls -l
total 124
```

```
-rw-r--r-- 1 root root  1363 Mar  7 19:52 17133
-rw-r--r-- 1 root root 12215 Mar  7 19:52 18176
-rw-r--r-- 1 root root  9698 Mar  7 19:52 18755
-rw-r--r-- 1 root root 11590 Mar  7 19:52 21844
-rw-r--r-- 1 root root 10575 Mar  7 19:52 39446
-rw-r--r-- 1 root root 14193 Mar  7 19:52 39525
-rw-r--r-- 1 root root 32674 Mar  7 19:52 40564
-rw-r--r-- 1 root root 12643 Mar  7 19:52 42009
-rw-r--r-- 1 root root   619 Mar  7 19:52 6757

kali@kali:~$ file 17133
17133: C source, ASCII text, with CRLF line terminators
```

Listing 181 - Verifying the files were indeed downloaded

We can inspect each exploit, and see that we did, in fact, grab the raw exploits:

```
kali@kali:~$ cat 17133
/*****************************************************************************
//
// Title: Microsoft Windows xp AFD.sys Local Kernel DoS Exploit
// Author: Lufeng Li of Neusoft Corporation
// Vendor: www.microsoft.com
// Vulnerable: Windows xp sp3
//
*****************************************************************************/

#include <stdio.h>
#include <Winsock2.h>

#pragma comment (lib, "ws2_32.lib")

BYTE buf[]={
0xac,0xfd,0xd3,0x00,0x01,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x20,0x00,0x00,0x00,0xe8,0xfd,0xd3,0x00,
0xb8,0xfd,0xd3,0x00,0xf8,0xfd,0xd3,0x00,0xc4,0xfd,
0xd3,0x00,0xcc,0xfd,0xd3,0x00};

int main( )
...
```

Listing 182 - Viewing a downloaded exploit

Even though we had success with a Bash one-liner, our code is not very clean and it's not particularly easy to re-use. Let's put everything together in a Bash script to solve these problems:

```
kali@kali:~$ cat dlsploits.sh
#!/bin/bash
# Bash script to search for a given exploit and download all matches.

for e in $(searchsploit afd windows -w -t | grep http | cut -f 2 -d "|")

do
  exp_name=$(echo $e | cut -d "/" -f 5)
  url=$(echo $e | sed 's/exploits/raw/')
  wget -q --no-check-certificate $url -O $exp_name
done
```

```
kali@kali:~$ chmod +x ./dlsploits.sh

kali@kali:~$ ./dlsploits.sh
```

Listing 183 - Using a Bash script to download the files

We can now manually inspect the exploits, find the ones we are interested in, try them on a test machine, and finally run the *proper* exploit on our target, since shotgunning random exploits at a live target is bad form and a recipe for total disaster.

5.7.3 Practical Bash Usage – Example 3

As penetration testers, we are always trying to find efficiencies to minimize the time we spend analyzing data, especially the volumes of data we recover during various scans.

Let's assume we are tasked with scanning a class C subnet to identify web servers and determine whether or not they present an interesting attack surface. Port scanning is the process of inspecting TCP or UDP ports on a remote machine with the intention of detecting what services are running on the target and potentially what attack vectors exist. We will discuss port scanning in much more detail in another module, but for now, let's keep it general as this is a great example that shows how Bash scripting can automate a rather tedious task.

In order to accomplish our goal, we would first port scan the entire subnet to pinpoint potential open web services, then we could manually browse their web pages.

To begin, let's create a temporary folder to be used for this exercise:

```
kali@kali:~$ mkdir temp

kali@kali:~$ cd temp/
```

Listing 184 - Creating a temporary folder to be used for this exercise

Now that we've created the directory and have entered it with **cd**, let's move on to the more interesting part, a scan of the class C subnet. We will only focus on port 80 to keep the scope somewhat manageable and we will use **nmap** (which we discuss in a later module) as our port scanning tool:

```
kali@kali:~/temp$ sudo nmap -A -p80 --open 10.11.1.0/24 -oG nmap-scan_10.11.1.1-254
Starting Nmap 7.60 ( https://nmap.org ) at 2019-03-18 18:57 EDT
Nmap scan report for 10.11.1.8
Host is up (0.030s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.0.52 ((CentOS))
| http-methods:
|_  Potentially risky methods: TRACE
| http-robots.txt: 2 disallowed entries
|_/internal/  /tmp/
|_http-server-header: Apache/2.0.52 (CentOS)
|_http-title: Site doesn't have a title (text/html; charset=UTF-8).
MAC Address: 00:50:56:89:20:34 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open an
Device type: general purpose|WAP|firewall|proxy server|PBX
Running (JUST GUESSING): Linux 2.6.X (92%), ZoneAlarm embedded (90%), Cisco embedded
Aggressive OS guesses: Linux 2.6.18 (92%), Linux 2.6.9 (92%), Linux 2.6.9 - 2.6.27 (90
```

```
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

TRACEROUTE
HOP RTT      ADDRESS
1   30.19 ms 10.11.1.8

Nmap scan report for 10.11.1.10
Host is up (0.030s latency).

PORT   STATE SERVICE VERSION
80/tcp open  http    Microsoft IIS httpd 6.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/6.0
|_http-title: Under Construction
MAC Address: 00:50:56:89:06:D0 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open an
Device type: general purpose|WAP
Running (JUST GUESSING): Microsoft Windows 2003|XP|2000 (89%), Apple embedded (86%)
OS CPE: cpe:/o:microsoft:windows_server_2003::sp2 cpe:/o:microsoft:windows_xp::sp3 cpe
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

TRACEROUTE
HOP RTT      ADDRESS
1   30.41 ms 10.11.1.10
...
```

Listing 185 - Scanning the entire class C subnet to look for web servers

This is a pretty straightforward scan, with **-A** for aggressive scanning, **-p** to specify the port or port range, **--open** to only return machines with open ports, and **-oG** to save the scan results in greppable format. Again, don't fret if **nmap** is new to you. We will go into details later, but nmap certainly provided a decent amount of output to work with.

Let's **cat** the output file to familiarize ourselves with its format:

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254
# Nmap 7.60 scan initiated Sun Mar 18 18:57:48 2019 as: nmap -A -p80 --open -oG nmap-s
can_10.11.1.1-254 10.11.1.0/24
Host: 10.11.1.8 ()  Status: Up
Host: 10.11.1.8 ()  Ports: 80/open/tcp//http//Apache httpd 2.0.52 ((CentOS))/   Seq In
dex: 197  IP ID Seq: All zeros
Host: 10.11.1.10 () Status: Up
Host: 10.11.1.10 () Ports: 80/open/tcp//http//Microsoft IIS httpd 6.0/  Seq Index: 256
IP ID Seq: Incremental
Host: 10.11.1.13 () Status: Up
Host: 10.11.1.13 () Ports: 80/open/tcp//http//Microsoft IIS httpd 5.1/  Seq Index: 136
IP ID Seq: Incremental
...
```

Listing 186 - Becoming familiar with the resulting file from our nmap scan

Interestingly, it looks like each IP address is repeated twice, the first line displaying the machine status, and the second displaying the port number being scanned. Since we are only interested in unique IP addresses, some clean up is necessary. Let's **grep** for the lines containing port *80*:

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254 | grep 80
# Nmap 7.60 scan initiated Sun Mar 18 18:57:48 2019 as: nmap -A -p80 --open -oG nmap-s
can_10.11.1.1-254 10.11.1.0/24
Host: 10.11.1.8 ()  Ports: 80/open/tcp//http//Apache httpd 2.0.52 ((CentOS))/   Seq In
dex: 197  IP ID Seq: All zeros
Host: 10.11.1.10 () Ports: 80/open/tcp//http//Microsoft IIS httpd 6.0/  Seq Index: 256
IP ID Seq: Incremental
Host: 10.11.1.13 () Ports: 80/open/tcp//http//Microsoft IIS httpd 5.1/  Seq Index: 136
IP ID Seq: Incremental
...
```

Listing 187 - Searching the file for port 80 using the grep command

This is a great start but notice that the first line is irrelevant. To exclude it, we will **grep** again with **-v**, which is a "reverse-grep", showing only lines that do not match the search string. In this case, we don't want any lines that contain the case-sensitive keyword "Nmap":

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254 | grep 80 | grep -v "Nmap"
Host: 10.11.1.8 ()  Ports: 80/open/tcp//http//Apache httpd 2.0.52 ((CentOS))/   Seq In
dex: 197  IP ID Seq: All zeros
Host: 10.11.1.10 () Ports: 80/open/tcp//http//Microsoft IIS httpd 6.0/  Seq Index: 256
IP ID Seq: Incremental
Host: 10.11.1.13 () Ports: 80/open/tcp//http//Microsoft IIS httpd 5.1/  Seq Index: 136
IP ID Seq: Incremental
...
```

Listing 188 - Excluding any lines matching the Nmap keyword

Our output is looking much better. Let's try to extract just the IP addresses, as this is all we are really interested in. To do so, we'll use **awk** to print the second field, using Space as a delimiter:

```
kali@kali:~/temp$ cat nmap-scan_10.11.1.1-254 | grep 80 | grep -v "Nmap" | awk '{print
$2}'
10.11.1.8
10.11.1.10
10.11.1.13
10.11.1.14
10.11.1.22
10.11.1.24
10.11.1.31
10.11.1.39
10.11.1.49
10.11.1.50
10.11.1.71
...
```

Listing 189 - Using the awk command to print a list of IP addresses

Good. This looks like a clean IP address list. For the next step, we'll use a Bash one-liner to loop through the list of IPs above and run **cutycapt**,[133] which is a Qt WebKit web page rendering capture

[133] (CutyCapt, 2013), http://cutycapt.sourceforge.net/

utility. We will use **-url** to specify the target web site and **-out** to specify the name of the output file:

```
kali@kali:~/temp$ for ip in $(cat nmap-scan_10.11.1.1-254 | grep 80 | grep -v "Nmap" |
awk '{print $2}'); do cutycapt --url=$ip --out=$ip.png;done
```

Listing 190 - Using cutycapt to capture screenshots from all web servers

Once our loop is finished and we have a prompt, we can examine the list of output files that were created by our Bash one-liner with the **-1** option to **ls**, which lists one file per line, suppressing additional details:

```
kali@kali:~/temp$ ls -1 *.png
10.11.1.10.png
10.11.1.115.png
10.11.1.116.png
10.11.1.128.png
10.11.1.13.png
10.11.1.133.png
10.11.1.14.png
10.11.1.202.png
10.11.1.209.png
10.11.1.217.png
...
```

Listing 191 - Exploring the results from our Bash one-liner

Outstanding. We are getting closer to our goal. We could examine these files individually but the more attractive choice is to once again put our scripting knowledge to work and see if there is anything else we can automate. This will require not only Bash scripting skills but also basic HTML[134] knowledge:

```
kali@kali:~/temp$ cat ./pngtohtml.sh
#!/bin/bash
# Bash script to examine the scan results through HTML.

echo "<HTML><BODY><BR>" > web.html

ls -1 *.png | awk -F : '{ print $1":\n<BR><IMG SRC=\""$1""$2"\" width=600><BR>"}' >> w
eb.html

echo "</BODY></HTML>" >> web.html

kali@kali:~/temp$ chmod +x ./pngtohtml.sh

kali@kali:~/temp$ ./pngtohtml.sh

kali@kali:~/temp$ firefox web.html
```

Listing 192 - Creating a page to look at all the images from our scan results

This script builds an HTML file (*web.html*), starting with the most basic tags. Then, the **ls** and **awk** statements insert each *.PNG* file name into an HTML *IMG* tag and append this to our *web.html* file.

[134] (MDN Web Docs, 2019), https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started

Finally, we append HTML end tags into the file, make the script executable, and view it in our browser. The result is simple, but effective, giving us a view of each web server's main page:

Figure 18: Previewing scan results

Hopefully, these brief practical examples have given you an idea about some of the possibilities that Bash scripting has to offer. Learning to use Bash effectively will be essential when trying to quickly automate a large number of tasks during assessments.

5.7.3.1 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. Use the practical examples in this module to help you create a Bash script that extracts JavaScript files from the access_log.txt file (http://www.offensive-security.com/pwk-files/access_log.txt.gz). Make sure the file names DO NOT include the path, are unique, and are sorted.
4. Re-write the previous exercise in another language such as Python, Perl, or Ruby.

5.8 Wrapping Up

The GNU Bourne-Again Shell (Bash)[135] is a powerful work environment and scripting engine. A competent security professional skillfully leverages Bash scripting to streamline and automate many Linux tasks and procedures. In this module, we introduced Bash scripting and explored several practical scenarios.

[135] (GNU, 2017), http://www.gnu.org/software/bash/

6. Passive Information Gathering

Passive Information Gathering (also known as Open-source Intelligence or *OSINT*[136]) is the process of collecting openly available information about a target, generally without any direct interaction with that target.

There are a variety of resources and tools we can use to gather this information and the process is cyclical rather than linear. In other words, the "next step" of any stage of the process depends on what we find during the previous steps, creating "cycles" of processes. Since each tool or resource can generate any number of varied results, it can be hard to define a standardized process. The ultimate goal of passive information gathering is to obtain information that clarifies or expands an attack surface,[137] helps us conduct a successful phishing campaign, or supplements other penetration testing steps such as password guessing.

Due to the cyclical nature of this process, this module will unfold differently than previous modules. Instead of presenting linked scenarios, we will simply present various resources and tools, explain how they work, and arm you with the basic techniques required to build a passive information-gathering campaign.

Before we begin, we need to define passive information gathering. There are two different schools of thought on what constitutes "passive" in this context.

In the strictest interpretation, we *never* communicate with the target directly. For example, we could rely on third parties for information, but we wouldn't access any of the target's systems or servers.

Using this approach maintains a high level of secrecy about our actions and intentions, but can also be cumbersome and may limit our results.

In a looser interpretation, we might interact with the target, but only as a normal Internet user would. For example, if the target's website allows us to register for an account, we could do that. However, we would not test the website for vulnerabilities during this phase.

In this module, we will adopt this latter, less rigid interpretation for our approach.

Neither approach is necessarily "correct". We need to consider the scope and rules of engagement for our penetration test before deciding which to use. In addition, bear in mind this phase may not always be necessary and that even if this phase bears fruit (say in the form of a successful spearphishing campaign), the other phases may require equal or greater attention.

Before we begin discussing resources and tools, let's share a personal example of a penetration test that involved successful elements of a passive information gathering campaign.

A Note From the Author

Several years ago, we at Offensive Security were tasked with performing a penetration test for a small company. This company had virtually no Internet presence and very few externally exposed services, all of which proved to be secure. There was practically no attack surface to speak of. After

[136] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Open-source_intelligence

[137] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Attack_surface

a focused passive information gathering campaign that leveraged various Google search operators, connected bits of information "piped" into other online tools, and a bit of creative and logical thinking, we found a forum post made by one of the target's employees in a stamp-collecting forum:

```
Hi!
I'm looking for rare stamps form the 1950's - for sale or trade.
Please contact me at david@company-address.com
Cell: 999-999-9999
```

Listing 193 - A forum post

We used this information to launch a semi-sophisticated client-side attack. We quickly registered a stamps-related domain name and designed a landing page that displayed various rare stamps from the 1950's, which we found using Google Images. The domain name and design of the site definitely increased the perceived reliability of our stamp trading website.

Next, we embedded some nasty client-side attack exploit code in the site's web pages, and called "David" during the workday. During the call, we posed as a stamp collector that had inherited their Grandfather's huge stamp collection.

David was overjoyed to receive our call and visited the malicious website to see the "stamp collection" without hesitation. While browsing the site, the exploit code executed on his local machine and sent us a reverse shell.

This is a good example of how some innocuous passively-gathered information, such as an employee engaging in personal business with his corporate email, can lead to a foothold during a penetration test. Sometimes the smallest details can be the most important.

> *While "David" wasn't following best practices, it was the company's policy and lack of a security awareness program that set the stage for this breach. Because of this, we avoid casting blame on an individual in a written report. Our goal as penetration testers is to improve the security of our client's resources, not to target a single employee. Simply removing "David" wouldn't have solved the problem.*

Let's take a look at some of the most popular tools and techniques that can help us conduct a successful information gathering campaign. We will use MegaCorp One,[138] a fictional company created by Offensive Security, as the subject of our campaign.

6.1 Taking Notes

An information gathering campaign can generate a lot of data, and it's important that we manage that data well so that we can leverage it in further searches or use it in a later phase. There is no right or wrong way to take notes. However, we may find it easier to retrieve information later on if we keep detailed and well formatted notes.

[138] (Offensive Security, 2019), https://www.megacorpone.com/

6.2 Website Recon

If the client has a website, we can gather basic information by simply browsing the site. Small organizations may only have a single website, while large organizations might have many, including some that are not maintained. Let's check out MegaCorp One's website (*https://www.megacorpone.com/*) to learn more about our target.

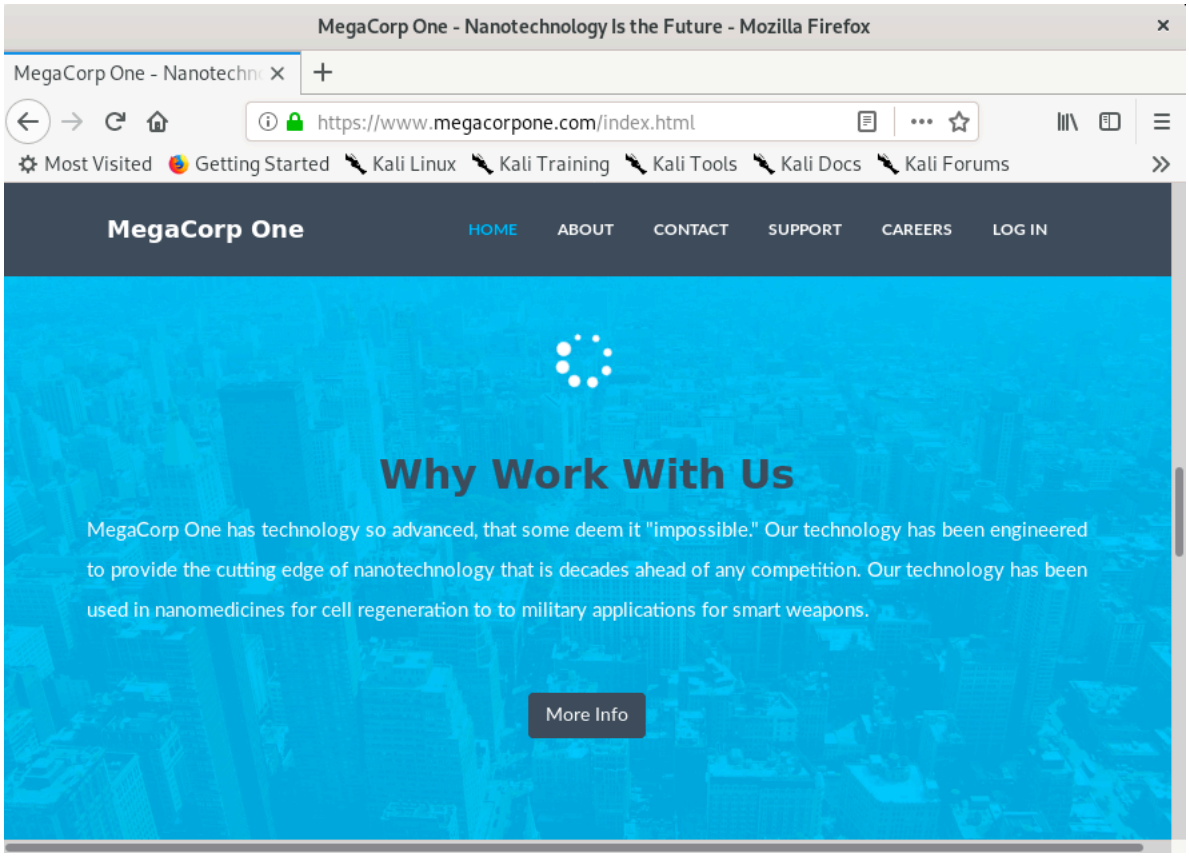


Figure 19: MegaCorp One

A quick review of their website reveals that they are a nanotech company.

The about page at *https://www.megacorpone.com/about.html* reveals email addresses and Twitter accounts of some of their employees:

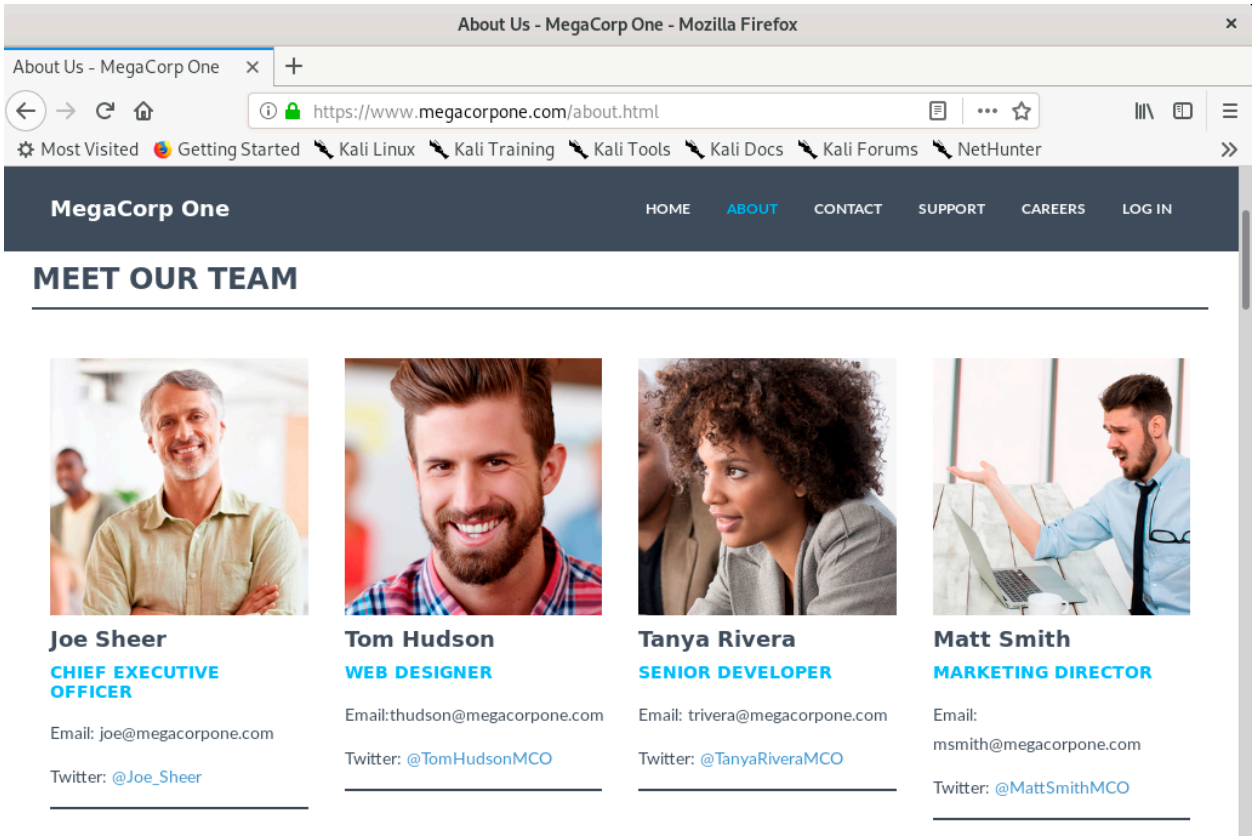


Figure 20: About Us - MegaCorp One

We could use these addresses in a social media information gathering campaign. We will discuss this in more detail in a later section.

It's also worth mentioning that the company's email address format follows a pattern of "first initial + last name". However, their CEO's email address simply uses his first name. This indicates that founders or long-time employees have a different email format than newer hires. This information could be useful in a later stage of the assessment.

Corporate social media accounts, found on the same page, are also worth recording for further research:

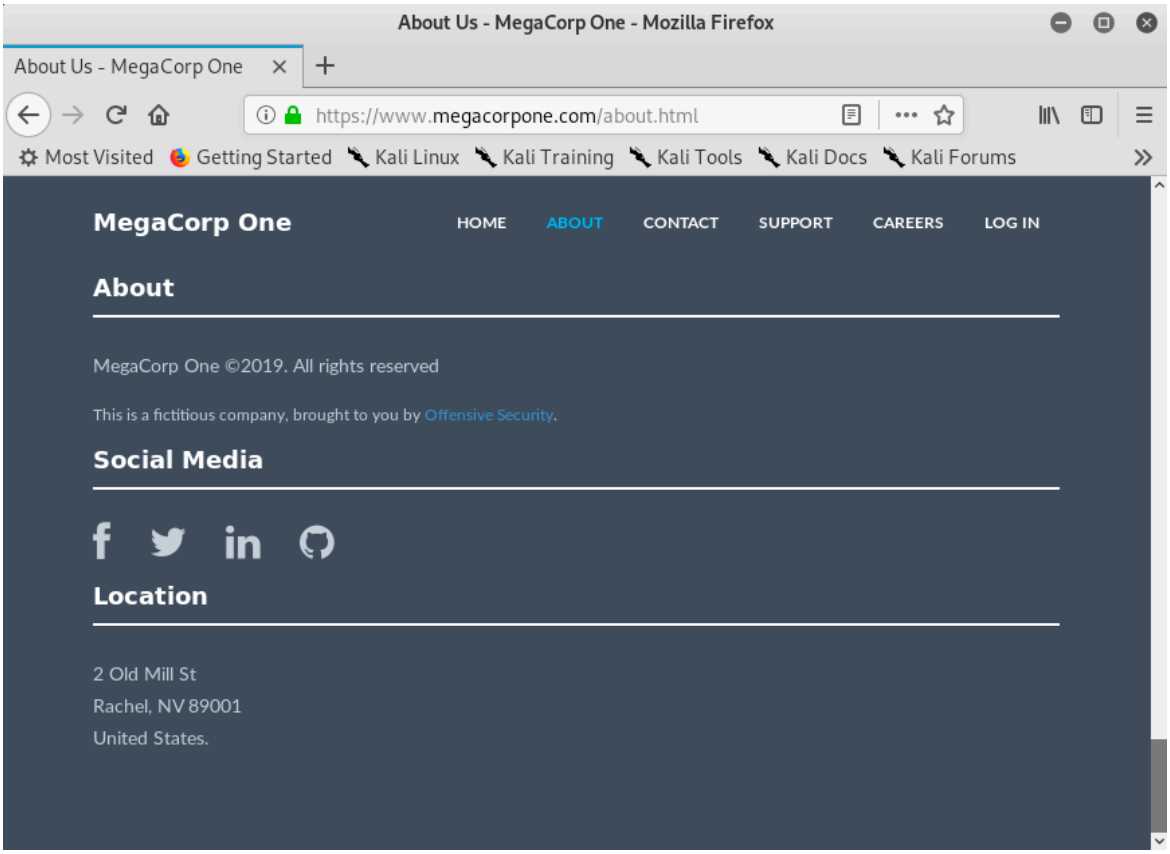


Figure 21: Social Media - MegaCorp One

Let's update our notes to keep track of each of these bits of information including the email address format and social media sites references.

6.3 Whois Enumeration

Whois[139] is a TCP service, tool, and a type of database that can provide information about a domain name, such as the *name server*[140] and *registrar*.[141] This information is often public since registrars charge a fee for private registration.

We can gather basic information about a domain name by executing a standard forward search by passing the domain name, megacorpone.com, into the **`whois`** client:

```
kali@kali:~$ whois megacorpone.com
   Domain Name: MEGACORPONE.COM
   Registry Domain ID: 1775445745_DOMAIN_COM-VRSN
   Registrar WHOIS Server: whois.gandi.net
   Registrar URL: http://www.gandi.net
   Updated Date: 2019-01-01T09:45:03Z
```

[139] (Wikipedia, 2019) https://en.wikipedia.org/wiki/WHOIS

[140] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Name_server

[141] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Domain_name_registrar

```
   Creation Date: 2013-01-22T23:01:00Z
   Registry Expiry Date: 2023-01-22T23:01:00Z
...
Registry Registrant ID:
Registrant Name: Alan Grofield
Registrant Organization: MegaCorpOne
Registrant Street: 2 Old Mill St
Registrant City: Rachel
Registrant State/Province: Nevada
Registrant Postal Code: 89001
Registrant Country: US
Registrant Phone: +1.9038836342
...
Registry Admin ID:
Admin Name: Alan Grofield
Admin Organization: MegaCorpOne
Admin Street: 2 Old Mill St
Admin City: Rachel
Admin State/Province: Nevada
Admin Postal Code: 89001
Admin Country: US
Admin Phone: +1.9038836342
...
Registry Tech ID:
Tech Name: Alan Grofield
Tech Organization: MegaCorpOne
Tech Street: 2 Old Mill St
Tech City: Rachel
Tech State/Province: Nevada
Tech Postal Code: 89001
Tech Country: US
Tech Phone: +1.9038836342
...
Name Server: NS1.MEGACORPONE.COM
Name Server: NS2.MEGACORPONE.COM
Name Server: NS3.MEGACORPONE.COM
...
```

Listing 194 - Using whois on megacorpone.com

Not all of this data is useful, but we did discover some valuable information. First, the output reveals that Alan Grofield registered the domain name. According to the Megacorp One Contact page, Alan is the “IT and Security Director”.

We also found the name servers for MegaCorp One. Name servers are a component of DNS, which we won’t be examining here, but we should add these servers to our notes.

In addition to this standard forward lookup, which gathers information about a DNS name, the whois client can also perform reverse lookups. Assuming we have an IP address, we can perform a reverse lookup to gather more information about it:

```
kali@kali:~$ whois 38.100.193.70
...
NetRange:       38.0.0.0 - 38.255.255.255
CIDR:           38.0.0.0/8
NetName:        COGENT-A
```

```
...
OrgName:        PSINet, Inc.
OrgId:          PSI
Address:        2450 N Street NW
City:           Washington
StateProv:      DC
PostalCode:     20037
Country:        US
RegDate:
Updated:        2015-06-04
...
```

Listing 195 - Whois reverse lookup

The results of the reverse lookup gives us information on who is hosting the IP address. This information could be useful later, and as with all the information we gather, we will add this to our notes.

6.3.1.1 Exercise

1. Use the whois tool in Kali to identify the name servers of MegaCorp One.

6.4 Google Hacking

The term "Google Hacking" was popularized by Johnny Long in 2001. Through several talks[142] and an extremely popular book (*Google Hacking for Penetration Testers*[143]), he outlined how search engines like Google could be used to uncover critical information, vulnerabilities, and misconfigured websites.

At the heart of this technique were clever search strings and *operators*[144] that allowed creative refinement of search queries, most of which work with a variety of search engines. The process is iterative, beginning with a broad search, which is narrowed down with operators to sift out irrelevant or uninteresting results.

Let's try a few of these operators and get a feel for how they work.

The *site* operator limits searches to a single domain. We can use this operator to get a rough idea of an organization's web presence:

[142] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Google_hacking

[143] (Johnny Long, Bill Gardner, Justin Brown, 2015), https://www.amazon.com/Google-Hacking-Penetration-Testers-Johnny/dp/0128029641/ref=dp_ob_image_bk

[144] (Google, 2019), https://support.google.com/websearch/answer/2466433?hl=en

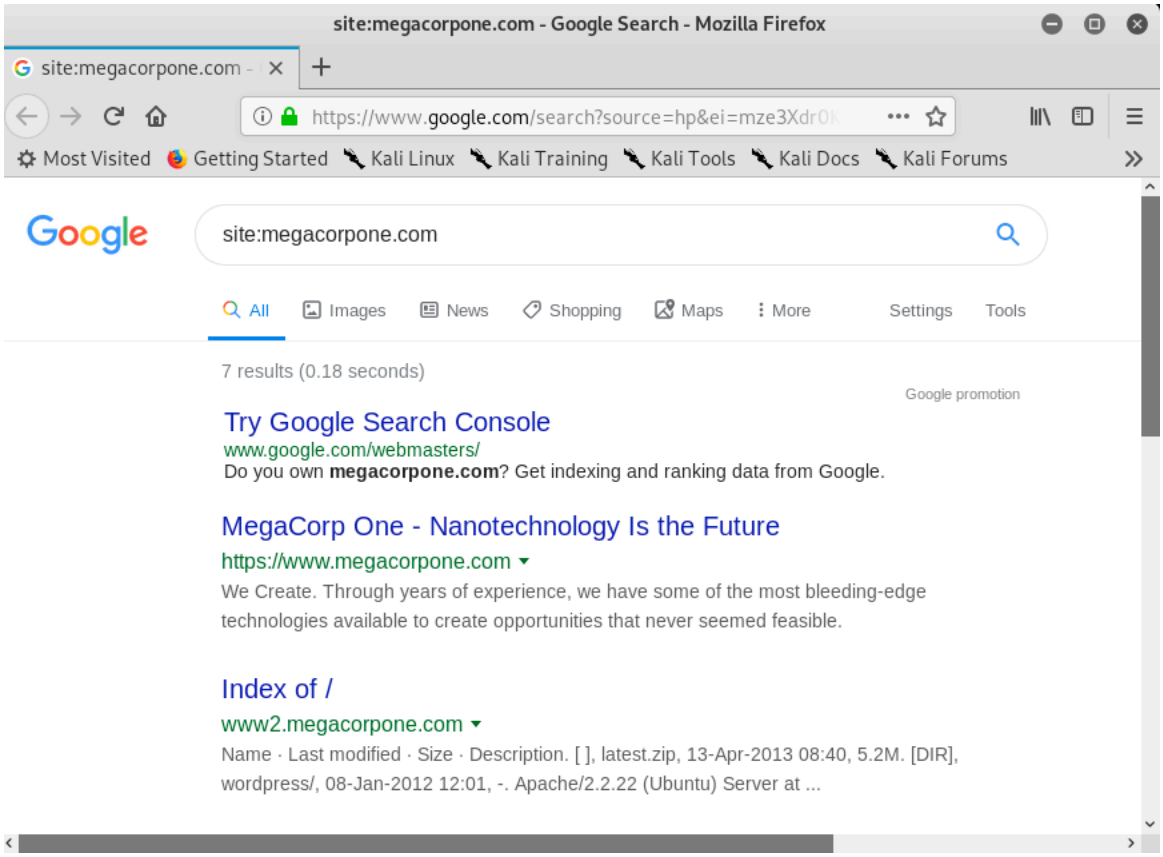


Figure 22: Searching with a Site Operator

We can then use further operators to narrow these results. For example, the *filetype* (or *ext*) operator limits search results to the specified file type.

In this example, we combine operators to locate PHP files (**`filetype:php`**) on www.megacorpone.com (**`site:megacorpone.com`**):

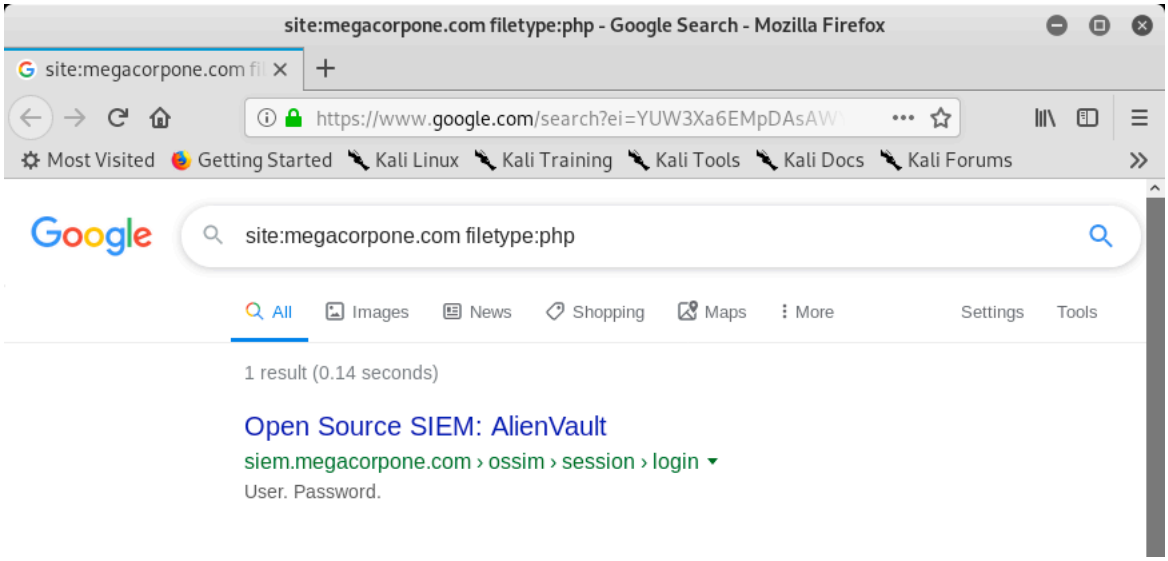


Figure 23: Searching with a Filetype Operator

We only get one result but it is an interesting one. Our query found the login page for an instance of AlienVault OSSIM,[145] a security information and event management (SIEM) platform. Security teams use SIEM tools to monitor applications and network traffic for malicious activities. Usually these tools are only available on internal networks. We should note this URL as it might prove useful if we can find user credentials to login during the active exploitation phase.

The *ext* operator could also be helpful to discern what programming languages might be used on a web site. Searches like *ext:jsp*, *ext:cfm*, *ext:pl* will find indexed Java Server Pages, Coldfusion, and Perl pages respectively.

We can also modify an operator using - to exclude particular items from a search, narrowing the results.

For example, to find interesting, non-HTML pages, we can use **`site:megacorpone.com`** to limit the search to megacorpone.com and subdomains, followed by **`-filetype:html`** to exclude HTML pages from the results:

[145] (AT&T, 2019), https://cybersecurity.att.com/products/ossim

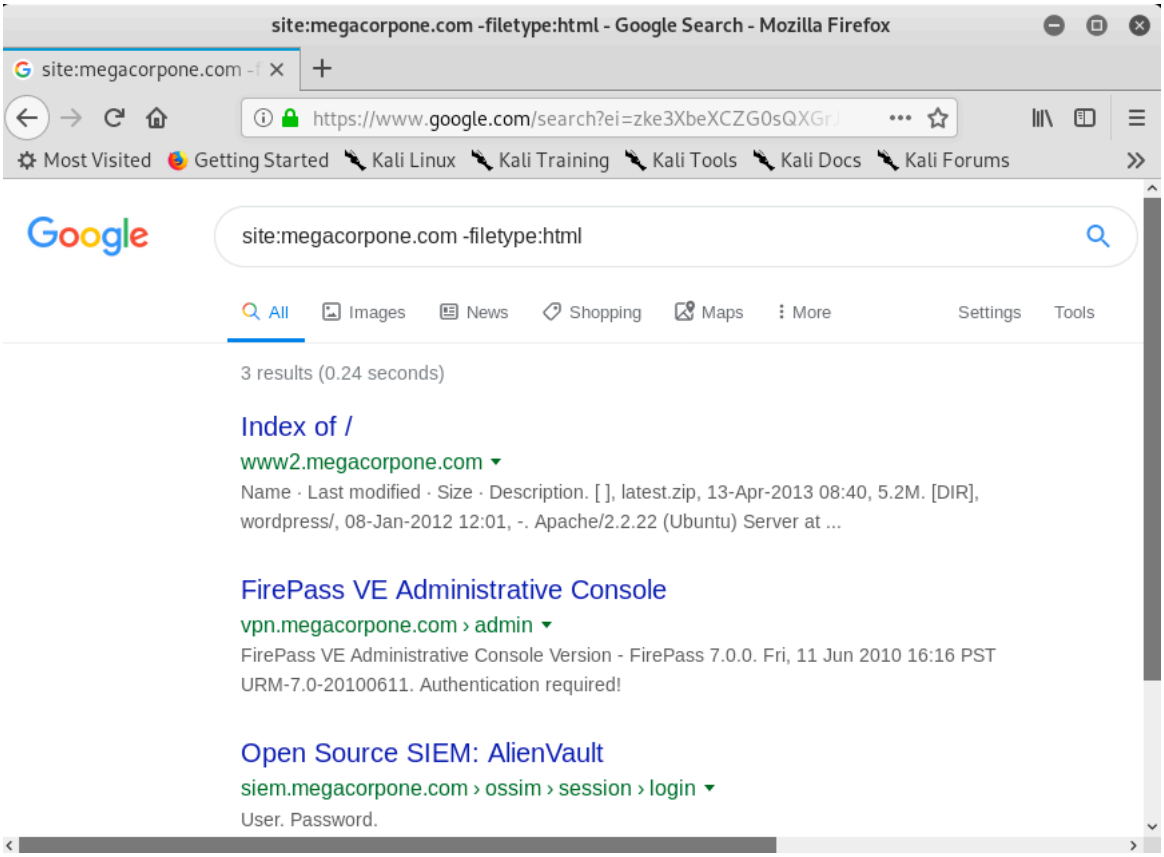


Figure 24: Searching with the Exclude Operator

In this case, we found several interesting pages, including an administrative console.

In another example, we can use a search for **`intitle:“index of” “parent directory”`** to find pages that contain “index of” in the title and the words “parent directory” on the page.

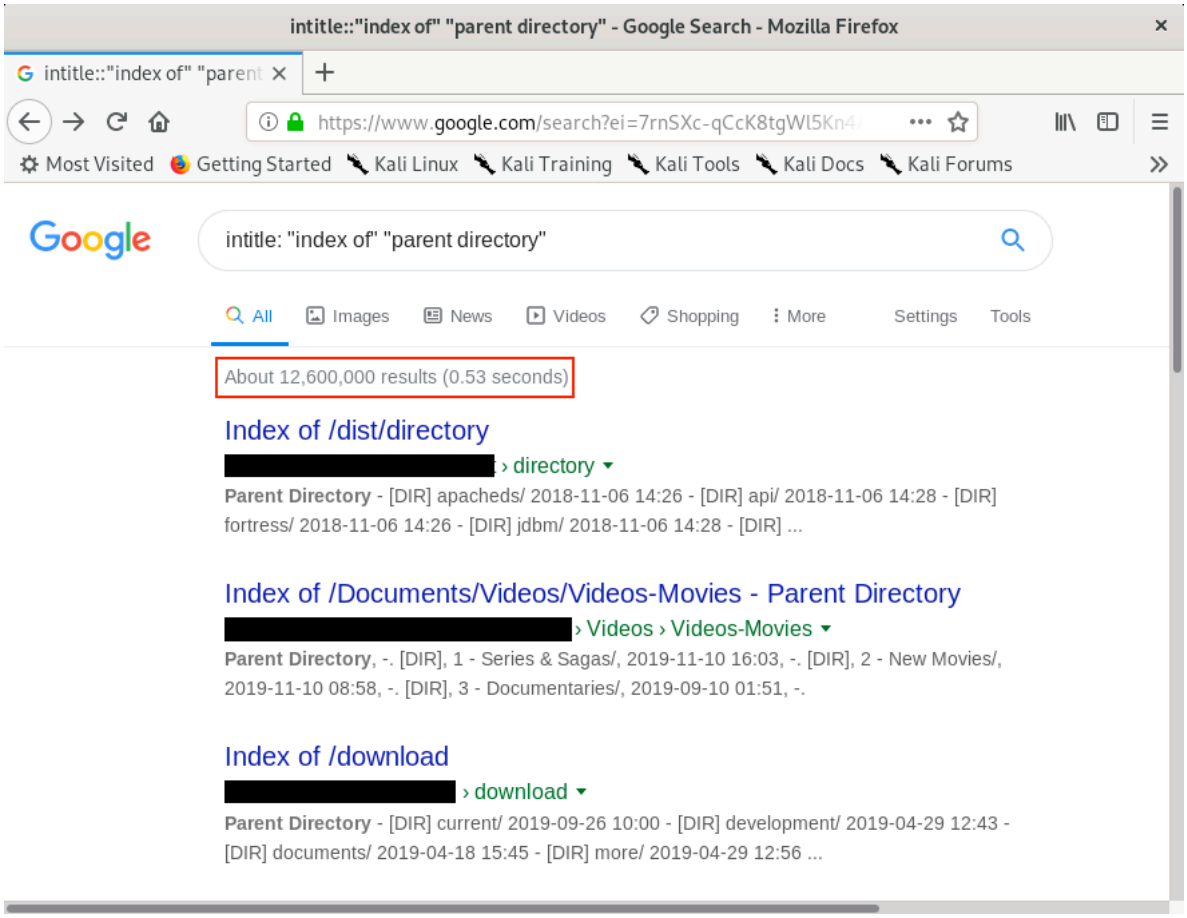


Figure 25: Using Google to Find Directory Listings

The output refers to *directory listing*[146] pages that list the file contents of the directories without index pages. Misconfigurations like this can reveal interesting files and sensitive information.

These basic examples only scratch the surface of what we can do with search operators. The Google Hacking Database (GHDB)[147] contains multitudes of creative searches that demonstrate the power of creative searching with combined operators:

[146] (MITRE, 2019), https://cwe.mitre.org/data/definitions/548.html

[147] (Offensive Security, 2019), https://www.exploit-db.com/google-hacking-database

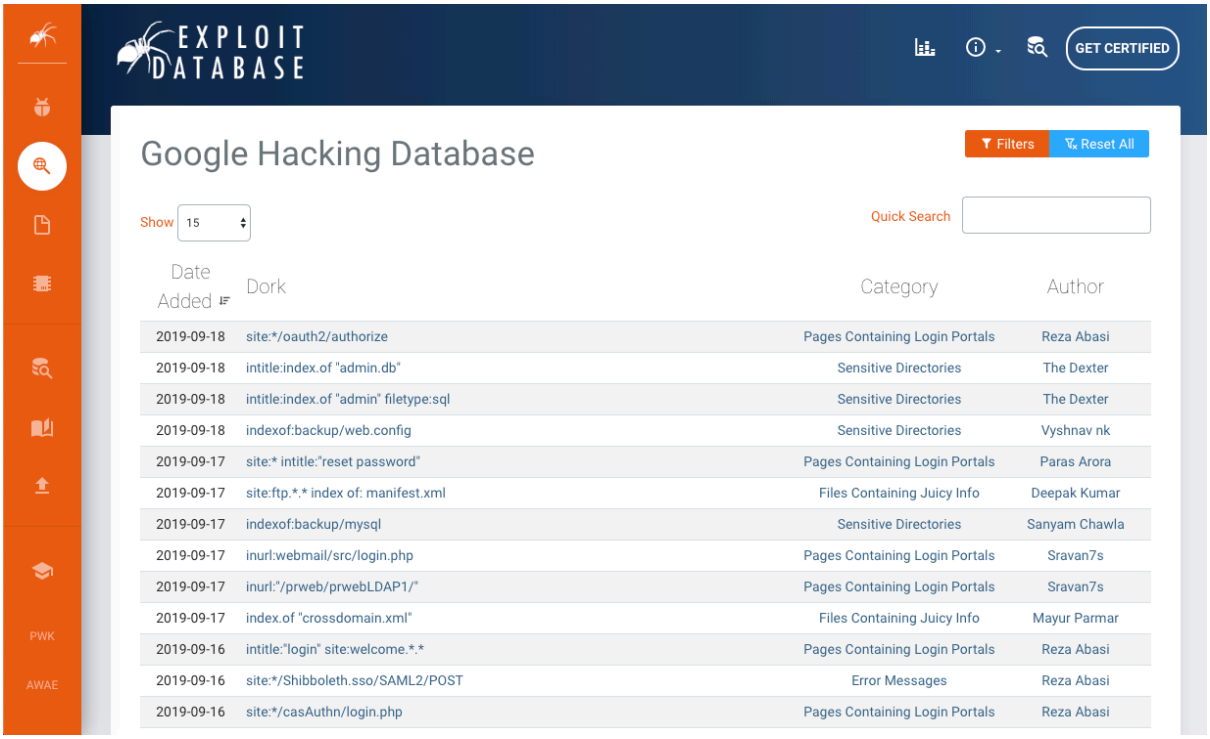


Figure 26: The Google Hacking Database (GHDB)

Mastery of these operators, combined with a keen sense of deduction, are key skills for effective search engine "hacking".

6.4.1.1 Exercises

1. Who is the VP of Legal for MegaCorp One and what is their email address?
2. Use Google dorks (either your own or any from the GHDB) to search www.megacorpone.com for interesting documents.
3. What other MegaCorp One employees can you identify that are not listed on www.megacorpone.com?

6.5 Netcraft

Netcraft[148] is an Internet services company based in England offering a free web portal that performs various information gathering functions. The use of services such as those offered by Netcraft is considered a passive technique since we never interact with our target directly.

Let's review some of Netcraft's capabilities. For example, we can use Netcraft's DNS search page (*https://searchdns.netcraft.com*) to gather information about the *megacorpone.com* domain:

[148] (Netcraft, 2019), https://www.netcraft.com/

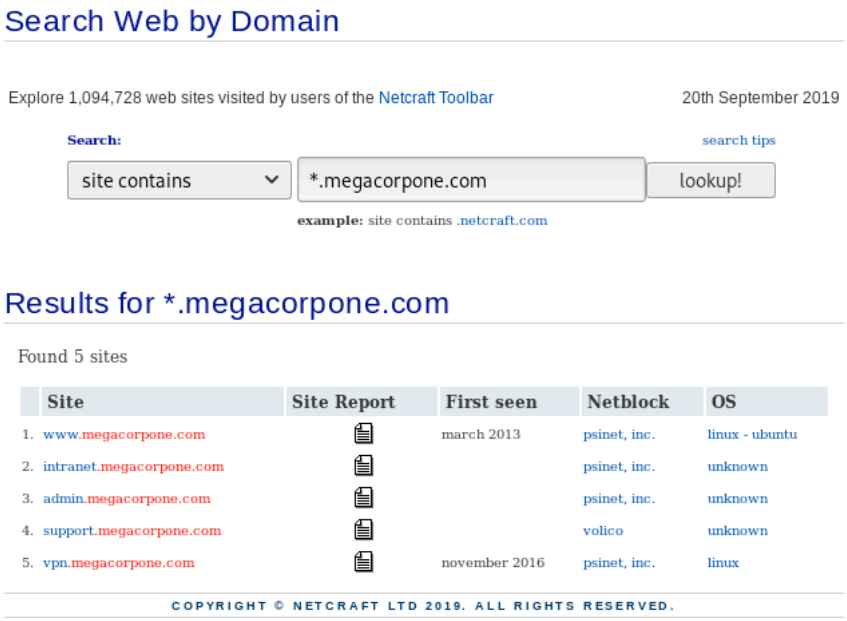


*Figure 27: Netcraft Results for *.megacorpone.com Search*

For each server found, we can view a “site report” that provides additional information and history about the server by clicking on the file icon next to each site URL.

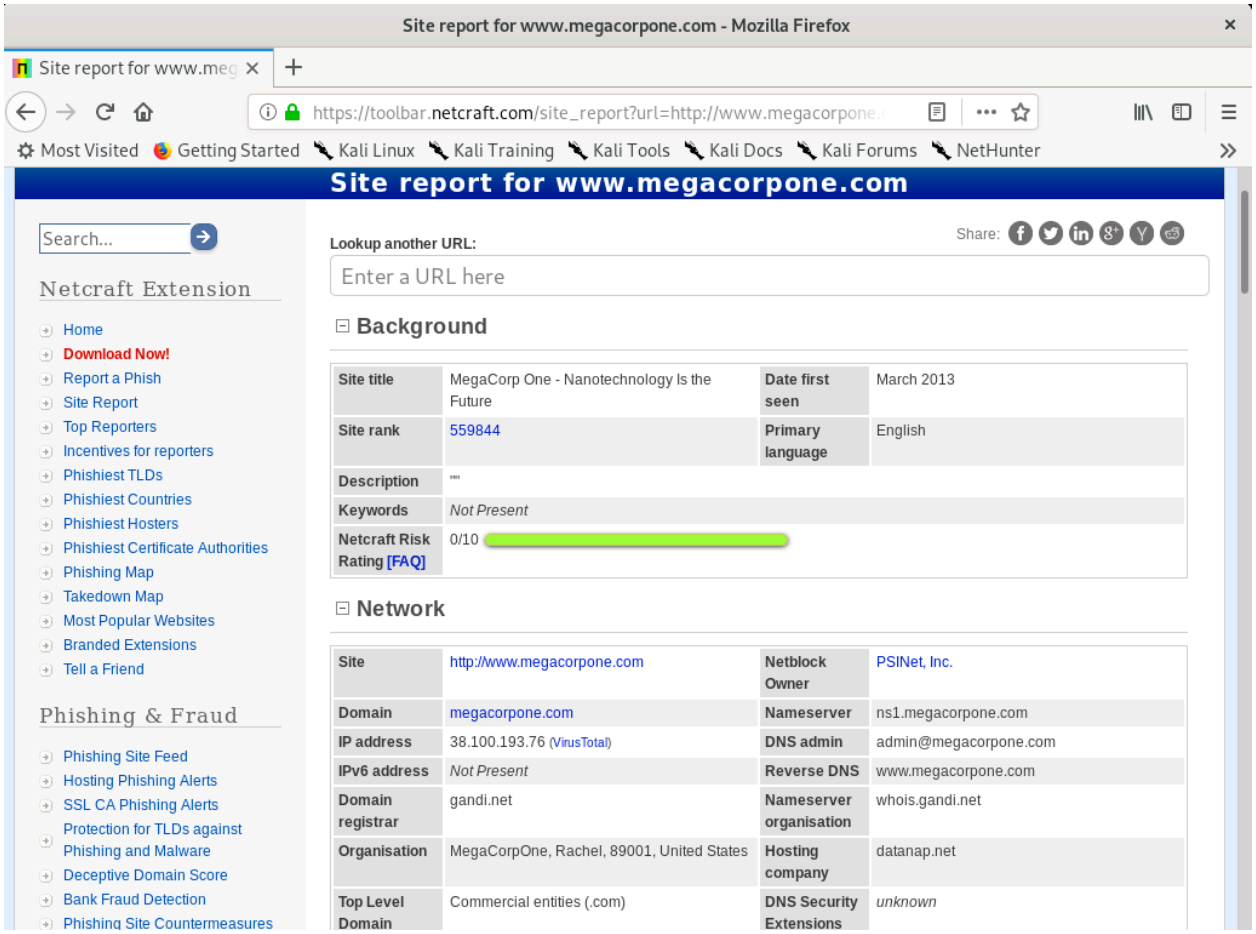


Figure 28: Netcraft Site Report for www.megacorpone.com

The start of the report covers registration information. However, if we scroll down, we discover various “site technology” entries:

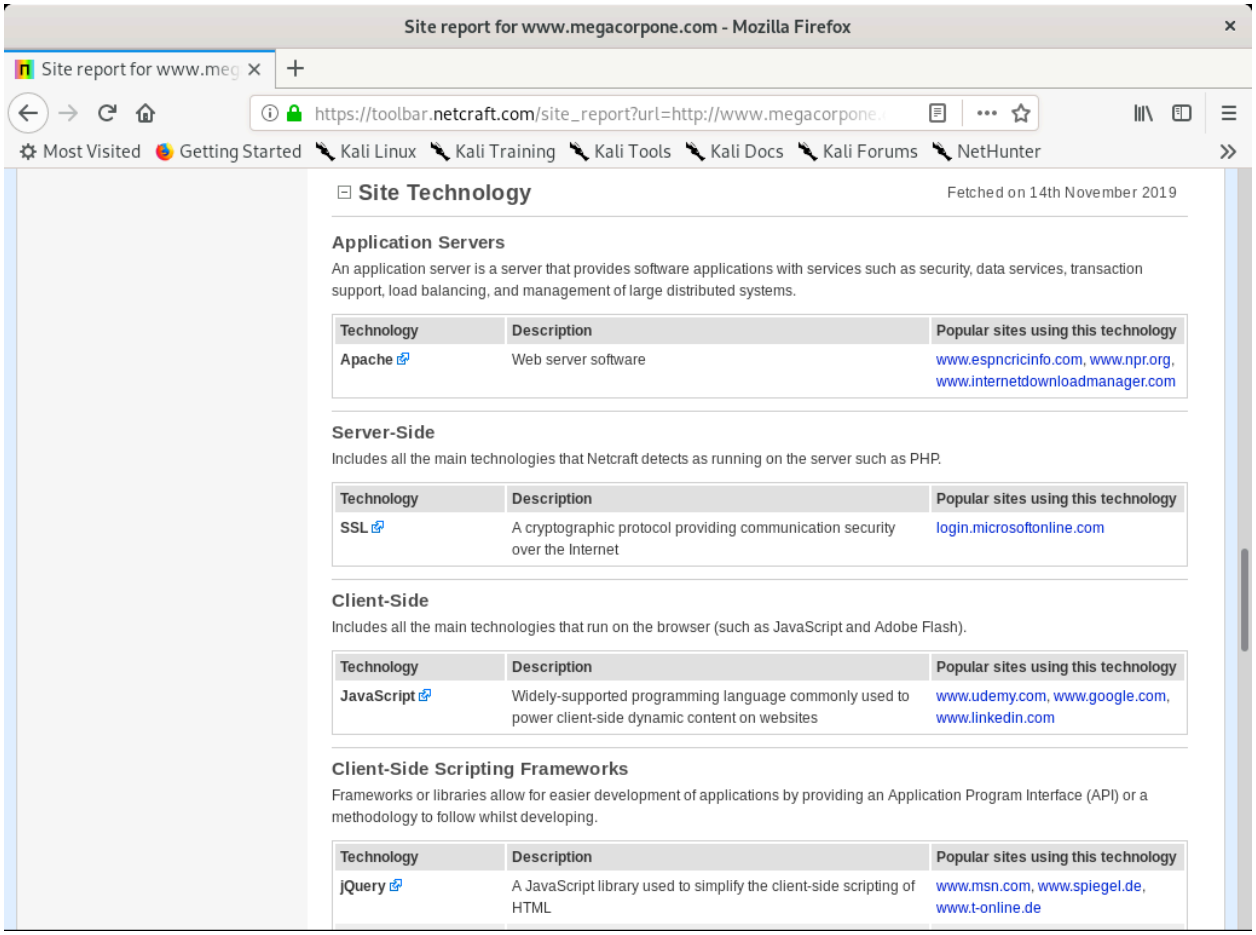


Figure 29: Site Technology for www.megacorpone.com

This list of subdomains and technologies will prove useful as we move on to active information gathering and exploitation. For now, we will add it to our notes.

6.5.1.1 Exercise

1. Use Netcraft to determine what application server is running on www.megacorpone.com.

6.6 Recon-ng

recon-ng[149] is a module-based framework for web-based information gathering. Recon-ng displays the results of a module to the terminal but it also stores them in a database. Much of the power of recon-ng lies in feeding the results of one module into another, allowing us to quickly expand the scope of our information gathering.

Let's use recon-ng to compile interesting data about MegaCorp One. To get started, let's simply run **`recon-ng`**:

[149] (Tim Tomes, 2019), https://github.com/lanmaster53/recon-ng

```
kali@kali:~$ recon-ng
[*] Version check disabled.

                                        /\
                                       / \\ /\
    Sponsored by...                  /\  /\/  \\V  \/\
                                    / \\/ // \\\\\ \\ \/\
                                   // // BLACK HILLS \/ \\
                                  www.blackhillsinfosec.com

                 ____   ____   ____   ____ _____ _  ____   ____  ____
                |____] | ___/ |____| |       |   | |____  |____ |
                |      |   \_ |    | |____   |   |  ____| |____ |____
                                  www.practisec.com

                      [recon-ng v5.0.0, Tim Tomes (@lanmaster53)]

[*] No modules enabled/installed.

[recon-ng][default] >
```

Listing 196 - Starting recon-ng

According to the output, we need to install various modules to use recon-ng.

We can add modules from the recon-ng "Marketplace".[150] We'll search the marketplace from the main prompt with **`marketplace search`**, providing a search string as an argument.

In this example, we will search for modules that contain the term **`github`**:

```
recon-ng][default] > marketplace search github
[*] Searching module index for 'github'...

+-----------------------------------------------------------------------------------------+
|                         Path                          | Version |     Status    | D | K |
+-----------------------------------------------------------------------------------------+
| recon/companies-multi/github_miner                    | 1.0     | not installed |   | * |
| recon/profiles-contacts/github_users                  | 1.0     | not installed |   | * |
| recon/profiles-profiles/profiler                      | 1.0     | not installed |   |   |
| recon/profiles-repositories/github_repos              | 1.0     | not installed |   | * |
| recon/repositories-profiles/github_commits            | 1.0     | not installed |   | * |
| recon/repositories-vulnerabilities/github_dorks       | 1.0     | not installed |   | * |
+-----------------------------------------------------------------------------------------+

D = Has dependencies. See info for details.
K = Requires keys. See info for details.
```

Listing 197 - Searching the Marketplace for GitHub modules

Notice that some of the modules are marked with an asterisk in the "K" column. These modules require credentials or API keys[151] for third-party providers. The recon-ng wiki[152] maintains a short

[150] (Tim Tomes, 2019), https://github.com/lanmaster53/recon-ng/wiki/Features#module-marketplace

[151] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Application_programming_interface_key

[152] (Tim Tomes, 2019), https://github.com/lanmaster53/recon-ng-marketplace/wiki/API-Keys

list of the keys used by its modules. Some of these keys are available to free accounts, while others require a subscription.

We can learn more about a module by using **marketplace info** followed by the module name. Since the GitHub modules require API keys, let's use this command to examine the recon/domains-hosts/google_site_web module:

```
[recon-ng][default] > marketplace info recon/domains-hosts/google_site_web

+-------------------------------------------------------------------------------+
| path          | recon/domains-hosts/google_site_web                           |
| name          | Google Hostname Enumerator                                    |
| author        | Tim Tomes (@lanmaster53)                                      |
| version       | 1.0                                                           |
| last_updated  | 2019-06-24                                                    |
| description   | Harvests hosts from Google.com by using the 'site' operator.  |
| required_keys | []                                                            |
| dependencies  | []                                                            |
| files         | []                                                            |
| status        | not installed                                                 |
+-------------------------------------------------------------------------------+

[recon-ng][default] >
```

Listing 198 - Getting information on a module

According to its description, this module searches Google with the "site" operator and it doesn't require an API key. Let's install the module with **marketplace install**:

```
[recon-ng][default] > marketplace install recon/domains-hosts/google_site_web
[*] Module installed: recon/domains-hosts/google_site_web
[*] Reloading modules...
[recon-ng][default] >
```

Listing 199 - Installing a module

After installing the module, we can load it with **module load** followed by its name. Then, we'll use **info** to display details about the module and required parameters:

```
[recon-ng][default] > modules load recon/domains-hosts/google_site_web

[recon-ng][default][google_site_web] > info

      Name: Google Hostname Enumerator
    Author: Tim Tomes (@lanmaster53)
   Version: 1.0

Description:
  Harvests hosts from Google.com by using the 'site' search operator. Updates the
  'hosts' table with the results.

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

```
Source Options:
  default        SELECT DISTINCT domain FROM domains WHERE domain IS NOT NULL
  <string>       string representing a single input
  <path>         path to a file containing a list of inputs
  query <sql>    database query returning one column of inputs

[recon-ng][default][google_site_web] >
```

Listing 200 - Using recon/domains-hosts/google_site_web

Notice that the output contains additional information about the module now that we've installed and loaded it. According to the output, the module requires the use of a *source*, which is the target we want to gather information about.

In this case, we will use **`options set SOURCE megacorpone.com`** to set our target domain:

```
[recon-ng][default][google_site_web] > options set SOURCE megacorpone.com
SOURCE => megacorpone.com
```

Listing 201 - Setting a source

Finally, we **`run`** the module:

```
[recon-ng][default][google_site_web] > run

---------------
MEGACORPONE.COM
---------------
[*] Searching Google for: site:megacorpone.com
[*] [host] www.megacorpone.com (<blank>)
[*] [host] vpn.megacorpone.com (<blank>)
[*] [host] www2.megacorpone.com (<blank>)
[*] [host] siem.megacorpone.com (<blank>)
[*] Searching Google for: site:megacorpone.com -site:www.megacorpone.com -site:vpn.meg
acorpone.com -site:www2.megacorpone.com -site:siem.megacorpone.com

-------
SUMMARY
-------
[*] 4 total (4 new) hosts found.
```

Listing 202 - Running a module

The results mirror what we found from the Netcraft DNS search. However, we haven't wasted our time here. Recon-ng stores results in a local database and these results will feed into other recon-ng modules.

We can use the **`show hosts`** command to view stored data:

```
[recon-ng][default][google_site_web] > back

[recon-ng][default] > show
Shows various framework items

Usage: show <companies|contacts|credentials|domains|hosts|leaks|locations|netblocks|po
rts|profiles|pushpins|repositories|vulnerabilities>
```

```
[recon-ng][default] > show hosts

+-----------------------------------------------------------------------------------+
| rowid |         host         | ip_address | region | country |      module     |
+-----------------------------------------------------------------------------------+
| 1     | www.megacorpone.com  |            |        |         | google_site_web |
| 2     | vpn.megacorpone.com  |            |        |         | google_site_web |
| 3     | www2.megacorpone.com |            |        |         | google_site_web |
| 4     | siem.megacorpone.com |            |        |         | google_site_web |
+-----------------------------------------------------------------------------------+

[*] 4 rows returned
[recon-ng][default] >
```

Listing 203 - Show hosts

We have four hosts in our database but no additional information on them. Perhaps another module can fill in the IP addresses.

Let's examine recon/hosts-hosts/resolve with **`marketplace info`**:

```
[recon-ng][default] > marketplace info recon/hosts-hosts/resolve

+-------------------------------------------------------------------------------+
| path          | recon/hosts-hosts/resolve                                     |
| name          | Hostname Resolver                                             |
| author        | Tim Tomes (@lanmaster53)                                      |
| version       | 1.0                                                           |
| last_updated  | 2019-06-24                                                    |
| description   | Resolves the IP address for a host. Updates the 'hosts' table |
| required_keys | []                                                            |
| dependencies  | []                                                            |
| files         | []                                                            |
| status        | installed                                                     |
+-------------------------------------------------------------------------------+

[recon-ng][default] >
```

Listing 204 - Module information for recon/hosts-hosts/resolve

The module description suits our needs so we will install it with **`marketplace install`**:

```
[recon-ng][default] > marketplace install recon/hosts-hosts/resolve
[*] Module installed: recon/hosts-hosts/resolve
[*] Reloading modules...
```

Listing 205 - Installing the resolve module

*An "Invalid command" error may indicate that we are at the wrong command level. If this happens, run **`back`** to return to the main recon-ng prompt and try the command again.*

Once the module is installed, we can use it with **modules load**, and run **info** to display information about the module and its options:

```
[recon-ng][default] > modules load recon/hosts-hosts/resolve

[recon-ng][default][resolve] > info

      Name: Hostname Resolver
    Author: Tim Tomes (@lanmaster53)
   Version: 1.0

Description:
  Resolves the IP address for a host. Updates the 'hosts' table with the results.

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

Source Options:
  default        SELECT DISTINCT host FROM hosts WHERE host IS NOT NULL AND ip_address
IS NULL
  <string>       string representing a single input
  <path>         path to a file containing a list of inputs
  query <sql>    database query returning one column of inputs

Comments:
  * Note: Nameserver must be in IP form.
```

Listing 206 - Installing and viewing recon/hosts-hosts/resolve

As is clear from the above output, this module will resolve the IP address for a host.

We need to provide the IP address we want to resolve as our source. We have four options we can set for the source: default, string, path, and query. Each option has a description alongside it as shown in Listing 206. For example, in the "google_site_web" recon-ng module, we used a string value.

However, we want to leverage the database this time. If we use the "default" value, recon-ng will look up the host information in its database for any records that have a host name but no IP address.

As shown in Listing 203, we have four hosts without IP addresses. If we select a "default" source, the module will run against all four hosts in our database automatically.

Let's try this out by leaving our source set to "default" and then run the module:

```
[recon-ng][default][resolve] > run
[*] www.megacorpone.com => 38.100.193.76
[*] vpn.megacorpone.com => 38.100.193.77
[*] www2.megacorpone.com => 38.100.193.79
[*] siem.megacorpone.com => 38.100.193.89
```

Listing 207 - Running recon/hosts-hosts/resolve

Nice. We now have IP addresses for the four domains.

If we **show hosts** again, we can verify the database has been updated with the results of both modules:

```
[recon-ng][default][resolve] > show hosts

+-------------------------------------------------------------------------------------+
| rowid |         host         |   ip_address   | region | country |      module     |
+-------------------------------------------------------------------------------------+
| 1     | www.megacorpone.com  | 38.100.193.76  |        |         | google_site_web |
| 2     | vpn.megacorpone.com  | 38.100.193.77  |        |         | google_site_web |
| 3     | www2.megacorpone.com | 38.100.193.79  |        |         | google_site_web |
| 4     | siem.megacorpone.com | 38.100.193.89  |        |         | google_site_web |
+-------------------------------------------------------------------------------------+

[*] 4 rows returned
```

Listing 208 - Show hosts after multiple modules

6.6.1.1 Exercise

(Reporting is not required for this exercise)

1. Use the recon/domains-hosts/google_site_web and recon/hosts-hosts/resolve modules to gather information on MegaCorp One.
2. Take some time to explore other recon-ng modules.

6.7 Open-Source Code

In the following sections, we will discuss various online tools and resources that can be used to passively search for information. One such source of interesting information are open-source projects and online code repositories, such as GitHub,[153] GitLab,[154] and SourceForge.[155]

Code stored online can provide a glimpse into the programming languages and frameworks used by an organization. In some rare occasions, developers have even accidentally committed sensitive data and credentials to public repos.

The search tools for some of these platforms will support the Google search operators that we discussed earlier in this module.

For example, GitHub's search[156] is very flexible. On GitHub, we will be able to search a user's or organization's repos, but we need an account if we want to search across all public repos.

In a previous module, we identified MegaCorp One's GitHub account.

Let's search that account's repos for interesting information. We can use **filename:users** to search for any files with the word "users" in the name:

[153] (GitHub, 2019), https://github.com/

[154] (GitLab, 2019), https://about.gitlab.com/

[155] (Slashdot Media, 2019), https://sourceforge.net/

[156] (GitHub, 2019), https://help.github.com/en/github/searching-for-information-on-github/searching-code

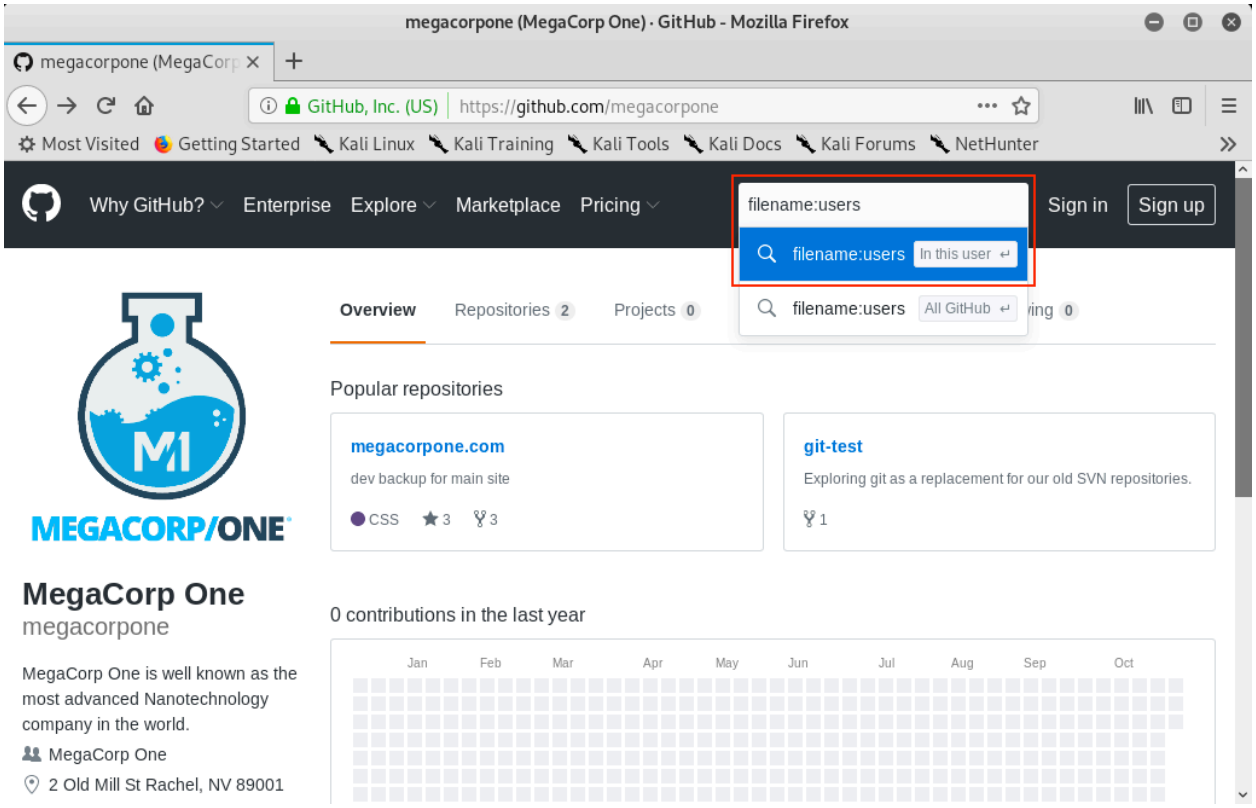


Figure 30: File Operator in GitHub Search

Our search only found one file - **xampp.users**. Even with a single result, we may have found something very interesting as *XAMPP*[157] is a web application development environment. Let's check the contents of the file.

[157] (Apache Friends, 2019), https://www.apachefriends.org/index.html

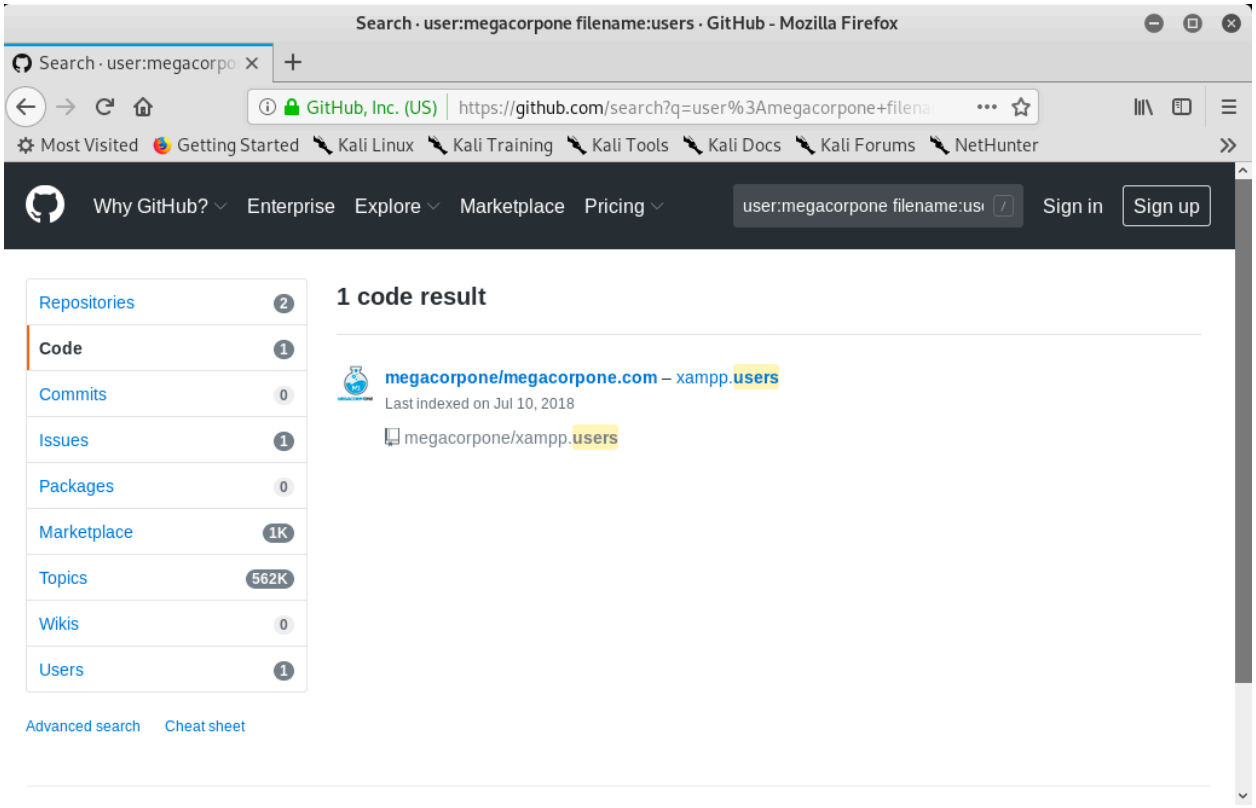


Figure 31: GitHub Search Results

This file appears to contain a username and password hash[158] that could be very useful when we begin our active attack phase. Let's add it to our notes.

[158] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Cryptographic_hash_function#Password_verification

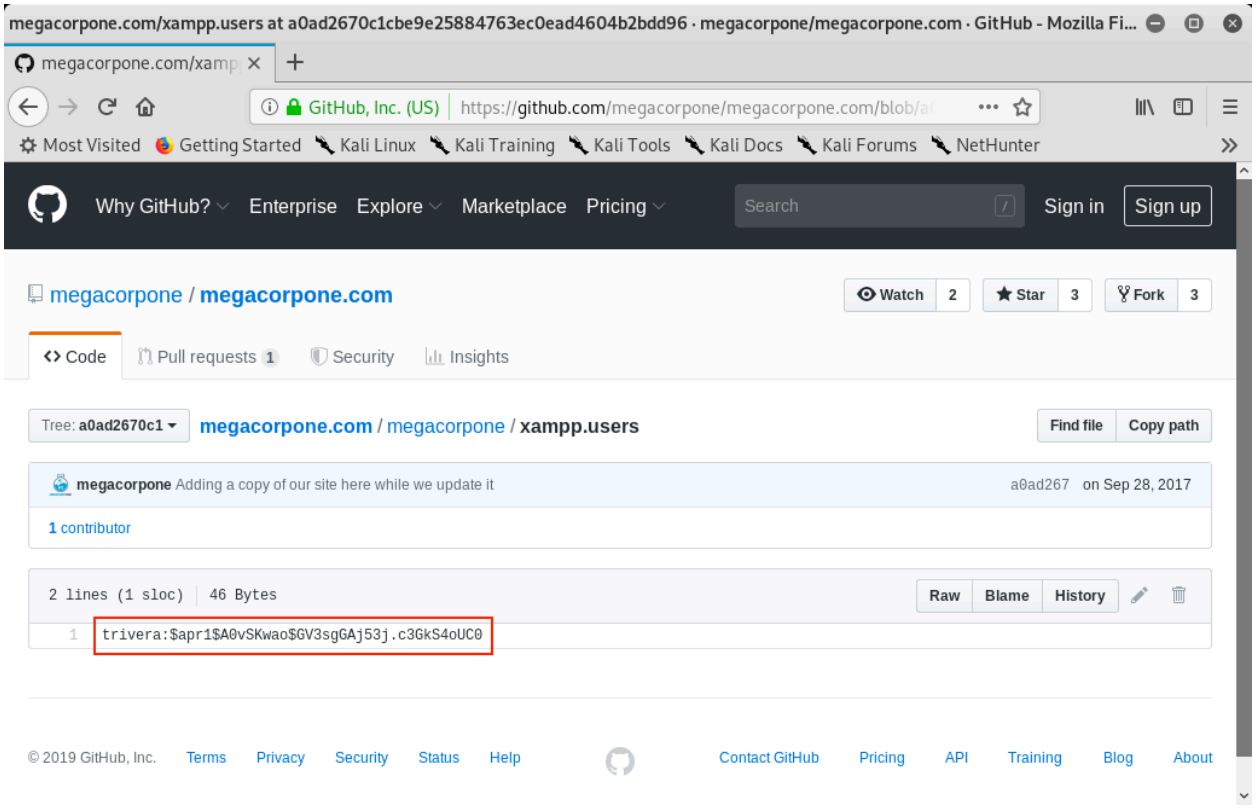


Figure 32: xampp.users File Content

This manual approach will work best on small repos. For larger repos, we can use several tools to help automate some of the searching, such as *Gitrob*[159] and *Gitleaks*.[160] Recon-ng also has several modules for searching GitHub. Most of these tools require an access token[161] to use the source code hosting provider's API.

For example, the following screenshot shows an example of Gitleaks finding an *AWS Client ID*[162] in a file:

[159] (Michael Henriksen, 2018), https://github.com/michenriksen/gitrob

[160] (Zachary Rice, 2019), https://github.com/zricethezav/gitleaks

[161] (GitHub, 2019), https://help.github.com/en/articles/creating-a-personal-access-token-for-the-command-line

[162] (Amazon Web Services, 2019), https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html#access-keys-and-secret-access-keys

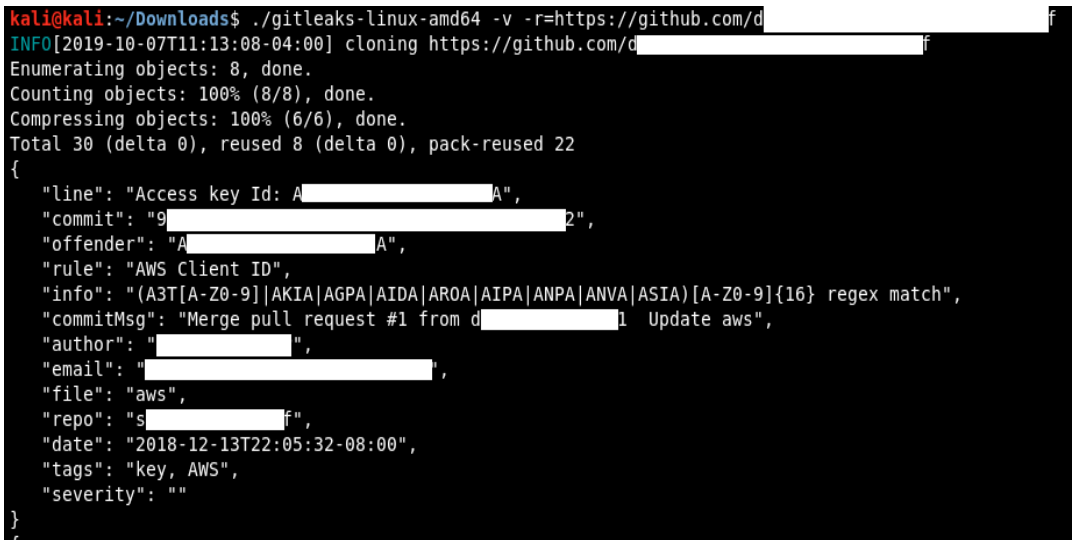


Figure 33: Example Gitleaks Output

Tools that search through source code for secrets, like Gitrob or Gitleaks, generally rely on regular expressions[163] or entropy[164] based detections to identify potentially useful information. Regular expressions are a predefined search pattern. They are particularly useful for searching through a body of text for variations of commonly used passwords. Entropy-based detection, on the other hand, attempts to find strings that are randomly generated. The idea here is that a long string of random characters and numbers is probably a password. Regardless of how a tool searches for secrets, no tool is perfect and they will miss things that a manual inspection might find.

6.7.1.1 Exercise

1. Search Megacorpone's GitHub repos for interesting or sensitive information.

6.8 Shodan

As we gather information on our target, it is important to remember that traditional websites are just one part of the Internet.

Shodan[165] is a search engine that crawls devices connected to the Internet including but not limited to the World Wide Web. This includes the servers that run websites but also devices like routers and IoT[166] devices.

[163] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Regular_expression

[164] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Password_strength#Random_passwords

[165] (Shodan, 2019), https://www.shodan.io/

[166] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Internet_of_things

To put it another way, Google and other search engines look for web server content, while Shodan searches for Internet-connected devices, interacts with them, and displays information about them.

Although Shodan is not required to complete any material in this module or the labs, it's worth exploring a bit. Before using Shodan we must register a free account, which provides limited access.

Let's start by using Shodan to search for **`hostname:megacorpone.com`**:

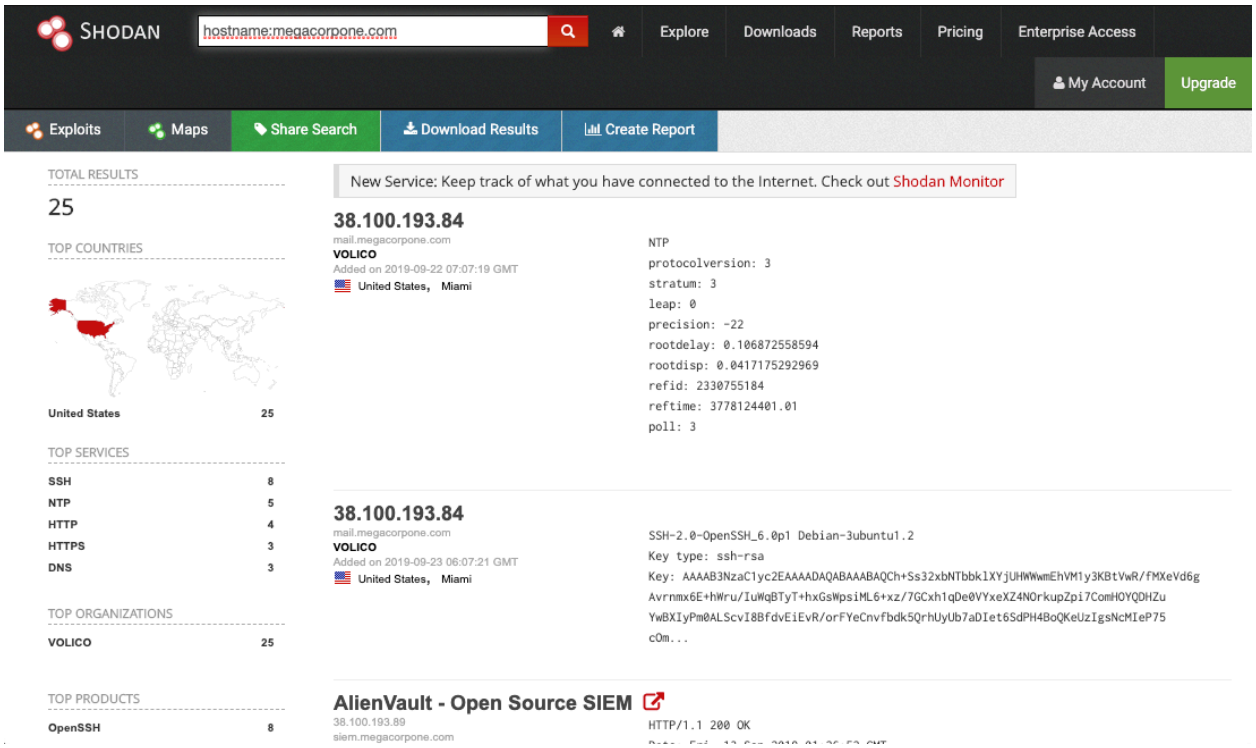


Figure 34: Searching MegaCorp One's domain with Shodan

In this case, Shodan lists the IPs, services, and banner information. All of this is gathered passively without interacting with the client's web site.

This information gives us a snapshot of our target's Internet footprint. For example, there are eight servers running SSH and we can drill down on this to refine our results by clicking on *SSH* under Top Services.

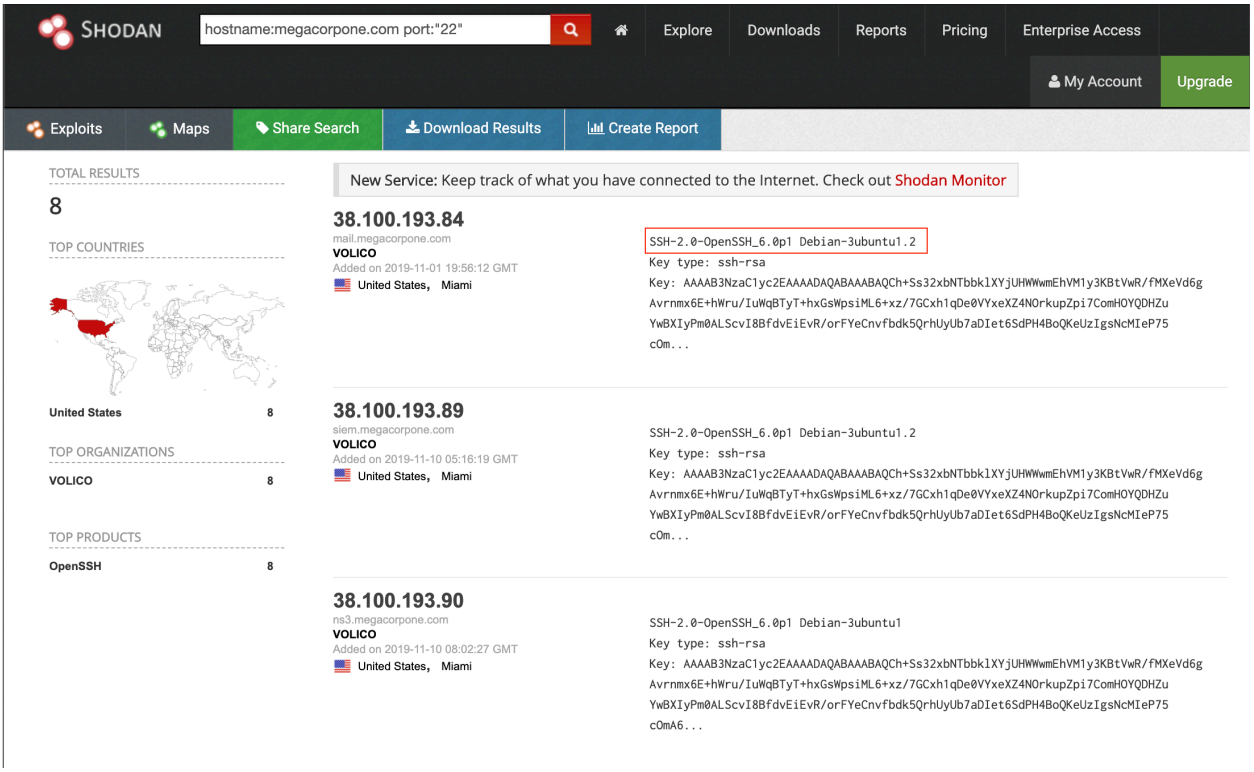


Figure 35: MegaCorp One servers running SSH

Based on Shodan's results, we know exactly which version of OpenSSH is running on each server. If we click on an IP address, we can retrieve a summary of the host.

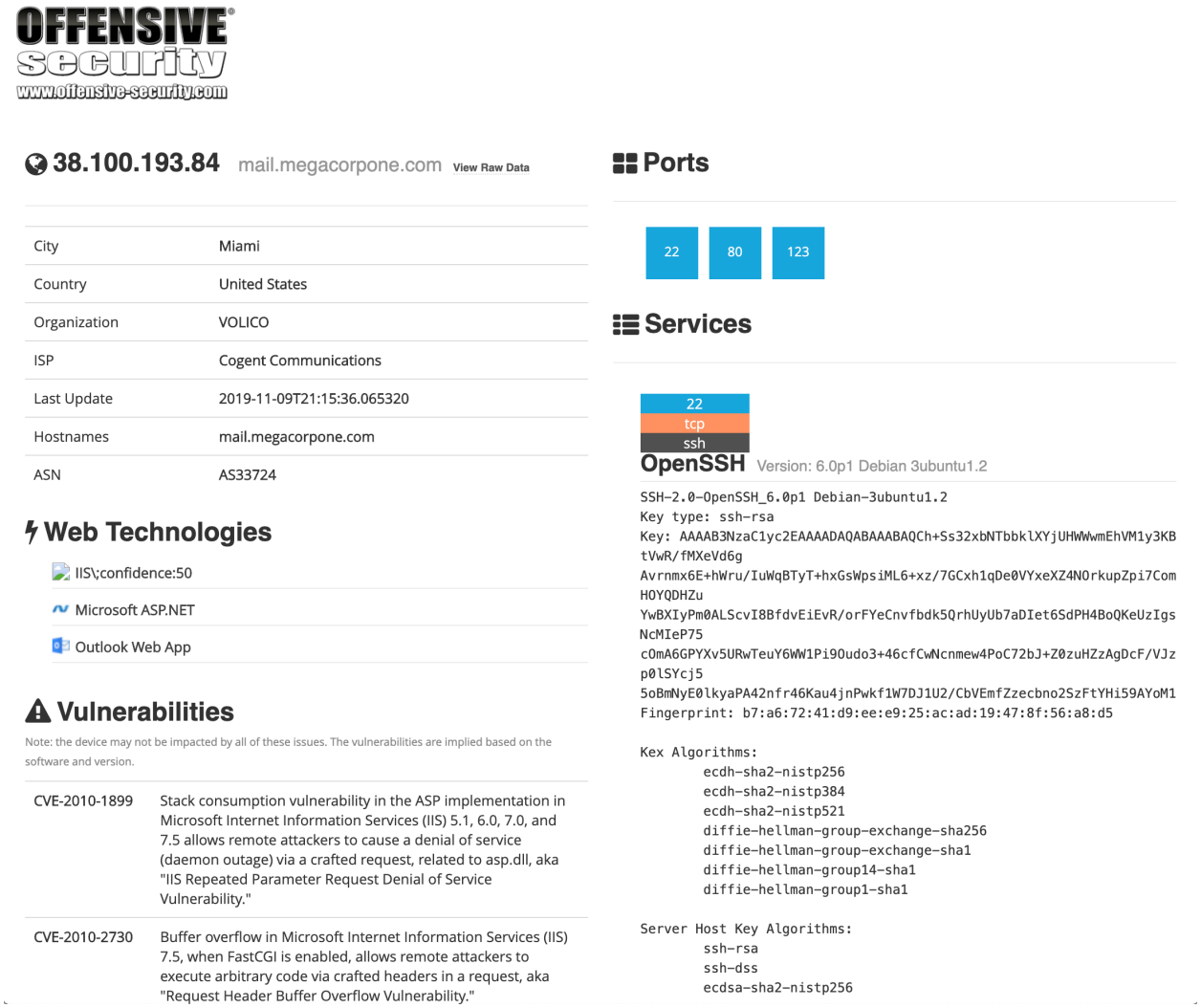


Figure 36: Shodan Host Summary

We can view the ports, services, and technologies used by the server on this page. Shodan will also reveal if there are any published vulnerabilities for any of the identified services or technologies. This information is invaluable when determining where to start when we move to active testing.

6.9 Security Headers Scanner

There are several other specialty websites that we can use to gather information about a website or domain's security posture. Some of these sites blur the line between passive and active information gathering, but the key point for our purposes is that a third-party is initiating any scans or checks.

One such site, *Security Headers*,[167] will analyze HTTP response headers and provide basic analysis of the target site's security posture. We can use this to get an idea of an organization's coding and security practices based on the results.

Let's scan www.megacorpone.com and check the results:

[167] (Scott Helme, 2019), https://securityheaders.com/

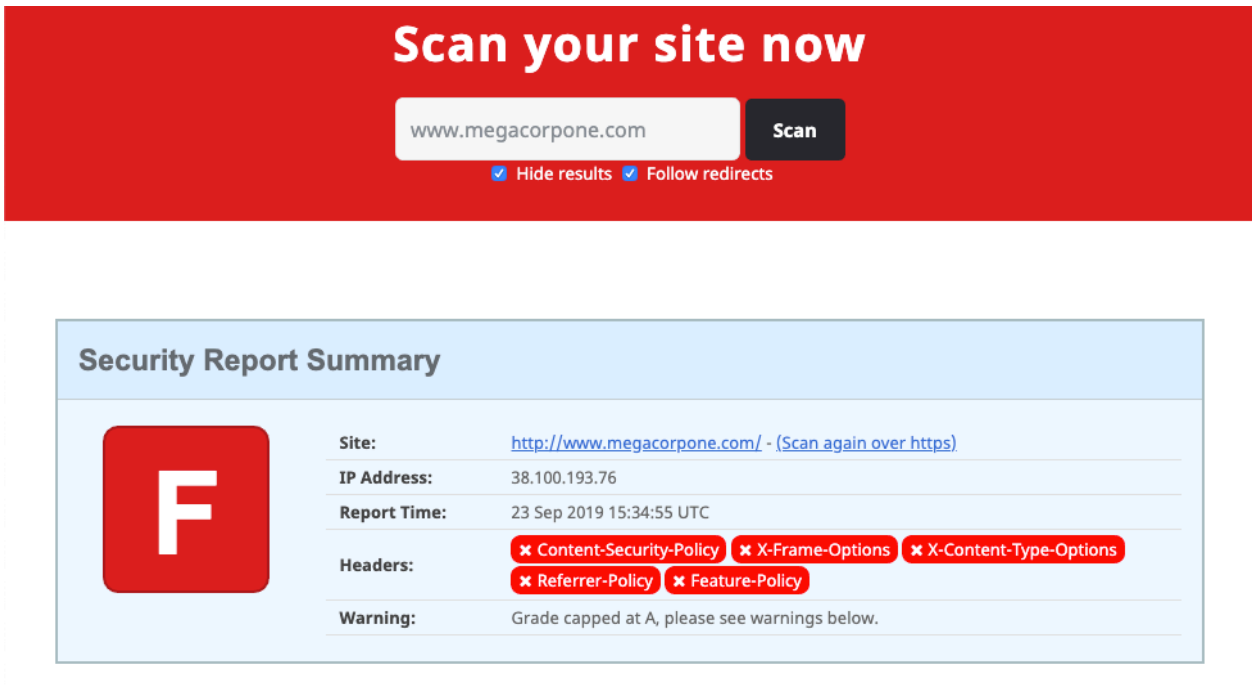


Figure 37: Scan results for www.megacorpone.com

The site is missing several defensive headers, such as *Content-Security-Policy*[168] and *X-Frame-Options*.[169] These missing headers are not necessarily vulnerabilities in and of themselves, but they could indicate web developers or server admins that are not familiar with *server hardening*.[170]

> *Server hardening, or secure configuration, is the overall process of securing a server via configuration. This includes things like disabling unneeded services, removing unused services or user accounts, rotating default passwords, setting appropriate server headers, and so forth. We don't need to know all the ins and outs of configuring every type of server, but understanding the concepts and what to look for can help when analyzing servers to determine how best to approach a potential target.*

6.10 SSL Server Test

Another scanning tool we can use is the *SSL Server Test* from Qualys SSL Labs.[171] This tool analyzes a server's SSL/TLS configuration and compares it against current best practices. It will

[168] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Content_Security_Policy

[169] (Mozilla, 2019, https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

[170] (NIST, 2019), https://csrc.nist.gov/publications/detail/sp/800-123/final

[171] (Qualys, 2019), https://www.ssllabs.com/ssltest/

also identify some SSL/TLS related vulnerabilities, such as Poodle[172] or Heartbleed.[173] Let's scan www.megacorpone.com and check the results:

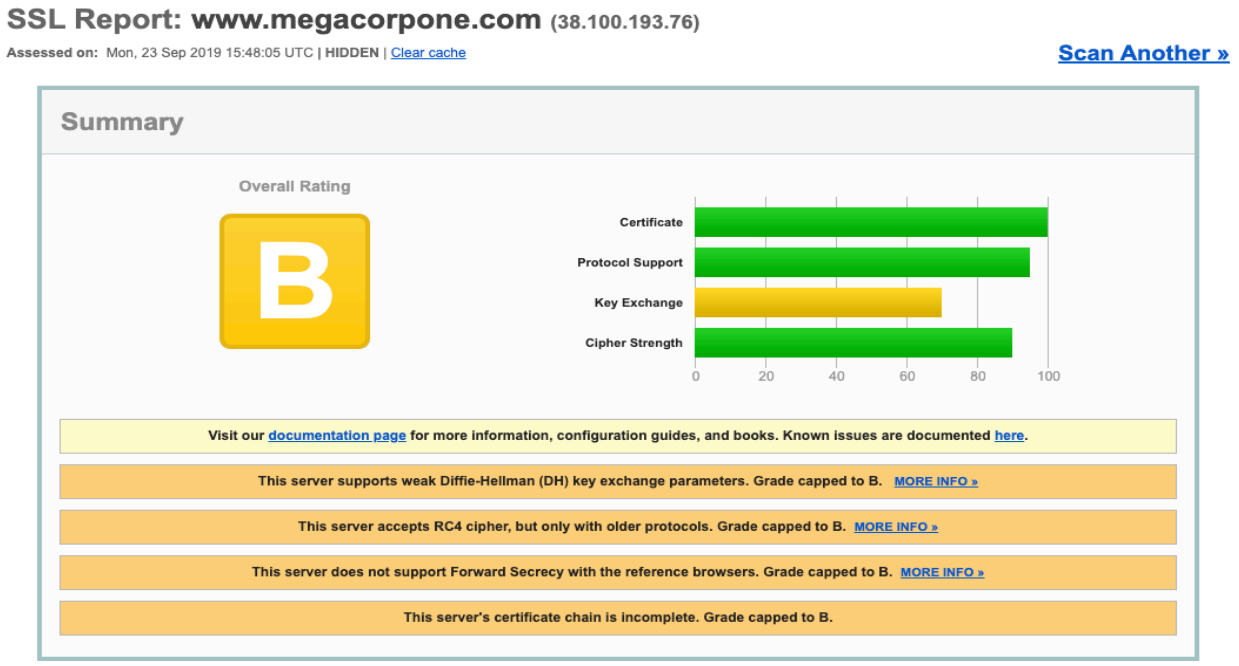


Figure 38: SSL Server Test results for www.megacorpone.com

The results are not as bad as the Security Headers check. However, the weak Diffie-Hellman key exchange, RC4 ciphers, and lack of Forward Secrecy suggest our target is not applying current best practices for SSL/TLS hardening. For example, disabling RC4 ciphers has been recommended for several years[174] due to multiple vulnerabilities. We can use these findings to get an insight on the security practices, or lack thereof, within the target organization.

6.11 Pastebin

Pastebin[175] is a website for storing and sharing text. The site doesn't require an account for basic usage. Many people use Pastebin because it is ubiquitous and simple to use. But since Pastebin is a public service, we can use it to search for sensitive information.

For example, we can use the website for basic searches, or use the API for more advanced uses. Let's search for megacorpone.com:

[172] (Wikipedia, 2019) https://en.wikipedia.org/wiki/POODLE

[173] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Heartbleed

[174] (Microsoft Security Response Center, 2013), https://msrc-blog.microsoft.com/2013/11/12/security-advisory-2868725-recommendation-to-disable-rc4/

[175] (Pastebin, 2019), https://pastebin.com/

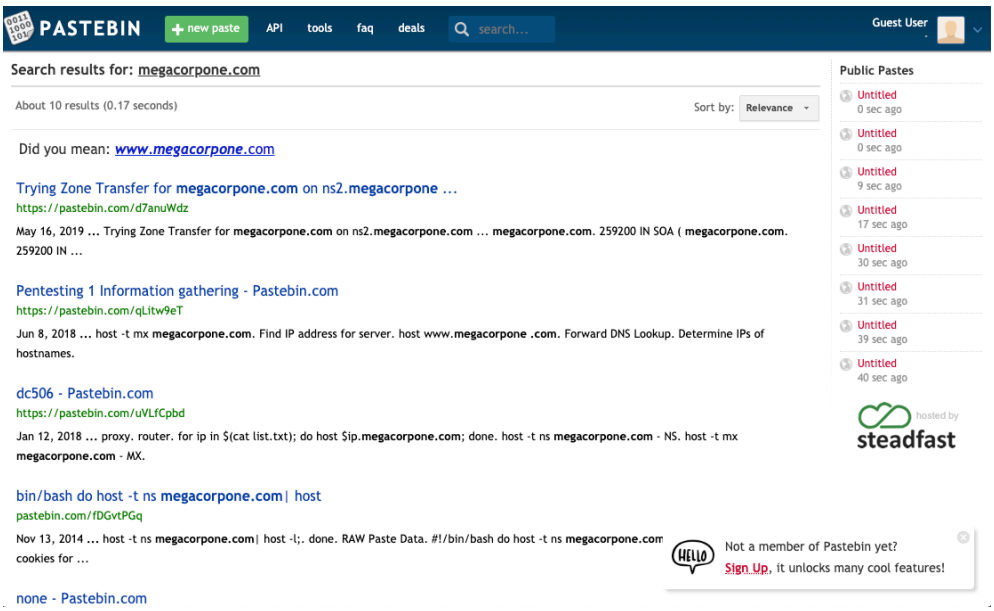


Figure 39: Searching Pastebin

There are only ten results, but some of them look very familiar. We might not find any new information here on MegaCorp One but we shouldn't overlook searching Pastebin on future information gathering efforts.

6.12 User Information Gathering

In addition to gathering information about our target organization's resources, we can also gather information about the organization's employees. Our purpose for gathering this information is to compile user or password lists, build pretexting for social engineering, augment phishing campaigns or client-side attacks, execute *credential stuffing*,[176] and much more. However, the rules of engagement vary for each penetration test. Some penetration tests may be limited to purely technical testing without any social engineering aspects. Other engagements may have few or no restrictions.

Some of the following methods will overlap with those already discussed in previous sections, but we'll go deeper into a few tools specific to user enumeration.

We do need to exercise some caution when we start gathering information on users. As we mentioned in our story about "David" in this module's introduction, our goal is to improve our client's security posture, not necessarily to get any one person fired. Similarly, we don't want to break any laws. A company can only authorize a test against their own systems. Employees' personal devices, third party email, and social media accounts usually fall outside this authorization.

[176] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Credential_stuffing

6.12.1 Email Harvesting

Let's begin our user information gathering with some basic email harvesting. In this case, we will use *theHarvester*,[177] which gathers emails, names, subdomains, IPs, and URLs from multiple public data sources.

For example, we can run theHarvester with **-d** to specify the target domain and **-b** to set the data source to search:

```
kali@kali:~$ theharvester -d megacorpone.com -b google
...
[-] Starting harvesting process for domain: megacorpone.com

[-] Searching in Google:
    Searching 0 results...
    Searching 100 results...
    Searching 200 results...
    Searching 300 results...
    Searching 400 results...
    Searching 500 results...

Harvesting results
No IP addresses found


[+] Emails found:
------------------
joe@megacorpone.com
mcarlow@megacorpone.com
first@megacorpone.com

[+] Hosts found in search engines:
------------------------------------

Total hosts: 13

[-] Resolving hostnames IPs...

Ns1.megacorpone.com:38.100.193.70
Siem.megacorpone.com:38.100.193.89
admin.megacorpone.com:38.100.193.83
beta.megacorpone.com:38.100.193.88
fs1.megacorpone.com:38.100.193.82
intranet.megacorpone.com:38.100.193.87
mail.megacorpone.com:38.100.193.84
mail2.megacorpone.com:38.100.193.73
ns1.megacorpone.com:38.100.193.70
ns2.megacorpone.com:38.100.193.80
url.megacorpone.com:empty
www.megacorpone.com:38.100.193.76
www2.megacorpone.com:38.100.193.79
```

[177] (Christian Martorella, 2019), https://github.com/laramies/theHarvester

Listing 209 - Running theHarvester on megacorpone.com

We found some email addresses, one of which, "first@megacorpone.com", appears to be new to us. We have also found some new subdomains of megacorpone.com. Let's add these to our notes as well.

This is a good reminder that information gathering is not always a neat, linear process. We may be looking for information on users and find something else about our target. This is one reason it's important to keep good notes.

6.12.1.1 Exercises

1. Use theHarvester to enumerate emails addresses for megacorpone.com.
2. Experiment with different data sources (-b). Which ones work best for you?

6.12.2 Password Dumps

Malicious hackers often dump breached credentials on Pastebin or other less reputable websites.[178] These password dumps can be extremely valuable for generating wordlists. For example, Kali Linux includes the "rockyou" wordlist generated from a data breach in 2009.[179]

Checking the email addresses we've found during user enumeration against password dumps can turn up passwords we could use in credential stuffing attacks.

6.13 Social Media Tools

Just about all organizations now maintain some sort of presence on *Social Media*.[180] The information a company posts can be very useful for us. We could, for example, use this information to identify potential employees and gain more information about the company and its operations. There are various ways to gather this public information with several tools we have already discussed, such as recon-ng and theHarvester. Let's explore a few additional tools.

6.13.1.1 Social-Searcher

Social-Searcher[181] is a search engine for social media sites. A free account will allow a limited number of searches per day. Social-searcher can be a quick alternative to setting up API keys on multiple more specialized services.

[178] (Troy Hunt, 2019), https://haveibeenpwned.com/PwnedWebsites

[179] (Wikipedia, 2019) https://en.wikipedia.org/wiki/RockYou#Data_breach

[180] (Wikipedia, 2019) https://en.wikipedia.org/wiki/Social_media

[181] (Social Searcher, 2019), https://www.social-searcher.com

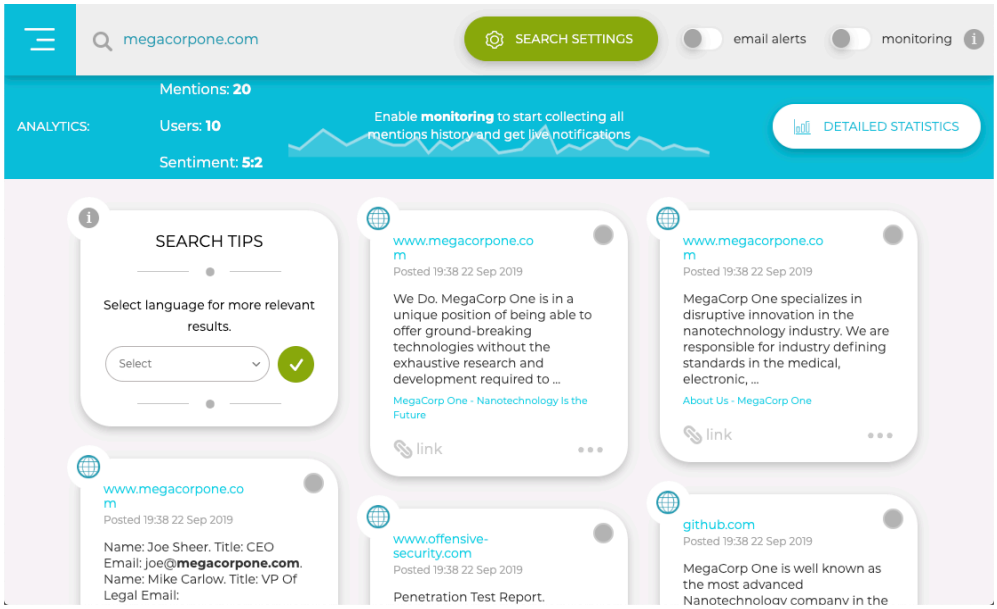


Figure 40: Using Social Searcher

The search results will include information posted by the target organization and what people are saying about it. Among other things, this can help us determine what sort of footprint and coverage an organization has on social media. Once we've done this, we may choose to move on to using site-specific tools.

6.13.2 Site-Specific Tools

There are two site-specific tools that we may want to familiarize ourselves with.

Twofi[182] scans a user's Twitter feed and generates a personalized wordlist used for password attacks against that user. While we will not run any attacks during passive information gathering, we can run this tool against any Twitter accounts we have identified to have a wordlist ready when needed. Twofi requires a valid Twitter API key.

linkedin2username[183] is a script for generating username lists based on LinkedIn data. It requires valid LinkedIn credentials and depends on a LinkedIn connection to individuals in the target organization. The script will output usernames in several different formats.

6.13.2.1 Exercise

1. Use any of the social media tools previously discussed to identify additional MegaCorp One employees.

[182] (Robin Wood, 2019), https://digi.ninja/projects/twofi.php

[183] (initstring, 2019), https://github.com/initstring/linkedin2username

6.14 Stack Overflow

Stack Overflow[184] is a website for developers to ask and answer coding related questions.

The site's value from an information gathering perspective is in looking at the types of questions a given user is asking or answering. If we can reasonably determine a user on Stack Overflow is also an employee of our target organization, we may be able to infer some things about the organization based on the employee's questions and answers.

For example, if we found a user that is always asking and answering questions about Python, it would be reasonable to assume they use that programming language on a daily basis, and it would likely be used at the organization where they are employed.

Even worse, if we find employees discussing sensitive information such as vulnerability remediation on these types of forums, we could discover unpatched vulnerabilities during this phase.

6.15 Information Gathering Frameworks

We will wrap up this module by briefly mentioning two additional tools that incorporate many of the techniques that we have discussed and add additional functionality. These tools are generally too heavy for the work we will do in the labs, but they are valuable during real-world assessments.

6.15.1 OSINT Framework

The *OSINT Framework*[185] includes information gathering tools and websites in one central location. Some tools listed in the framework cover more disciplines than information security.

[184] (Stack Exchange, 2019), https://stackoverflow.com/

[185] (Justin Nordine, 2019, https://osintframework.com/

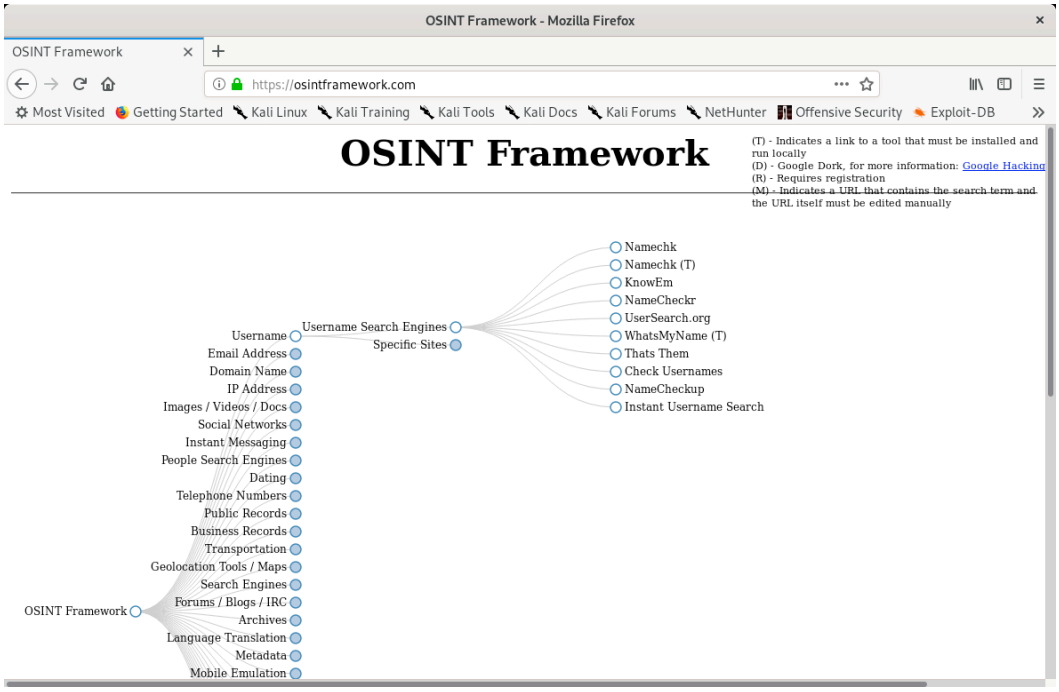


Figure 41: OSINT Framework

The OSINT framework is not meant to be a checklist, but reviewing the categories and available tools may spur ideas for additional information gathering opportunities.

6.15.2 Maltego

Maltego[186] is a very powerful data mining tool that offers an endless combination of search tools and strategies. The learning curve for it can be steep, and it is frankly overkill for this module, but its impressive capability warrants an introduction.

Maltego searches thousands of online data sources, and uses extremely clever "transforms" to convert one piece of information into another. For example, if we are performing a user information gathering campaign, we could submit an email address, and through various automated searches, "transform" that into an associated phone number or street address. During an organizational information gathering exercise, we could submit a domain name and "transform" that into a web server, then a list of email addresses, then a list of associated social media accounts, and then into a potential password list for that email account.

The combinations are endless, and the discovered information is presented in a scalable graph that allows easy zoom-and-pan navigation.

Maltego CE (the limited "community version" of Maltego) is included in Kali and requires a free registration to use. Commercial versions are also available and can handle larger datasets.

[186] (Paterva, 2019), https://www.paterva.com/buy/maltego-clients.php

Multiple vendors provide information that Maltgo can ingest and display. However, some providers also charge for access to their data. Maltego is not required to complete any material in the labs, but it can be an indispensable tool for large information gathering operations.

6.16 Wrapping Up

In this module, we explored the foundational aspects of the iterative process of passive information gathering. We covered a variety of techniques and tools to locate information about companies and their employees. This information can often prove to be invaluable in later stages of the engagement.

7. Active Information Gathering

In this module, we will move beyond passive information gathering and explore techniques that involve direct interaction with target services. We will take a look at some foundational techniques but bear in mind there are innumerable services that can be targeted in the field. This includes *Active Directory* for example, which we cover in more detail in a separate module. However, we will look at some of the more common active information gathering techniques in this module including port scanning and DNS, SMB, NFS, SMTP, and SNMP enumeration.

7.1 DNS Enumeration

The Domain Name System (DNS)[187] is one of the most critical systems on the Internet and is a distributed database responsible for translating user-friendly domain names into IP addresses.

This is facilitated by a hierarchical structure that is divided into several zones, starting with the top-level root zone. Let's take a closer look at the process and servers involved in resolving a hostname like www.megacorpone.com.

The process starts when a hostname is entered into a browser or other application. The browser passes the hostname to the operating system's DNS client and the operating system then forwards the request to the external DNS server it is configured to use. This first server in the chain is known as the *DNS recursor* and is responsible for interacting with the DNS infrastructure and returning the results to the DNS client. The DNS recursor contacts one of the servers in the DNS root zone. The root server then responds with the address of the server responsible for the zone containing the Top Level Domain (TLD),[188] in this case, the *.com* TLD.

Once the DNS recursor receives the address of the TLD DNS server, it queries it for the address of the authoritative nameserver for the megacorpone.com domain. The authoritative nameserver is the final step in the DNS lookup process and contains the DNS records in a local database known as the zone file. It typically hosts two zones for each domain, the forward lookup zone that is used to find the IP address of a specific hostname and the reverse lookup zone (if configured by the administrator), which is used to find the hostname of a specific IP address. Once the DNS recursor provides the DNS client with the IP address for www.megacorpone.com, the browser can contact the correct web server at its IP address and load the webpage.

To improve the performance and reliability of DNS, DNS caching is used to store local copies of DNS records at various stages of the lookup process. It is for this reason that some modern applications, such as web browsers, keep a separate DNS cache. In addition, the local DNS client of the operating system also maintains its own DNS cache along with each of the DNS servers in the lookup process. Domain owners can also control how long a server or client caches a DNS record via the *Time To Live* (TTL) field of a DNS record.

[187] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Domain_Name_System

[188] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Top-level_domain

7.1.1 Interacting with a DNS Server

Each domain can use different types of DNS records. Some of the most common types of DNS records include:

- *NS* - Nameserver records contain the name of the authoritative servers hosting the DNS records for a domain.
- *A* - Also known as a host record, the "*a record*" contains the IP address of a hostname (such as www.megacorpone.com).
- *MX* - Mail Exchange records contain the names of the servers responsible for handling email for the domain. A domain can contain multiple MX records.
- *PTR* - Pointer Records are used in reverse lookup zones and are used to find the records associated with an IP address.
- *CNAME* - Canonical Name Records are used to create aliases for other host records.
- *TXT* - Text records can contain any arbitrary data and can be used for various purposes, such as domain ownership verification.

Due to the wealth of information contained within DNS, it is often a lucrative target for active information gathering.

To demonstrate this, we'll use the **host** command to find the IP address of www.megacorpone.com:

```
kali@kali:~$ host www.megacorpone.com
www.megacorpone.com has address 38.100.193.76
```

Listing 210 - Using host to find the A host record for www.megacorpone.com

By default, the host command looks for an *A* record, but we can also query other fields, such as MX or TXT records. To do this, we can use the -t option to specify the type of record we are looking for:

```
kali@kali:~$ host -t mx megacorpone.com
megacorpone.com mail is handled by 10 fb.mail.gandi.net.
megacorpone.com mail is handled by 50 mail.megacorpone.com.
megacorpone.com mail is handled by 60 mail2.megacorpone.com.
megacorpone.com mail is handled by 20 spool.mail.gandi.net.

kali@kali:~$ host -t txt megacorpone.com
megacorpone.com descriptive text "Try Harder"
```

Listing 211 - Using host to find the MX and TXT records for megacorpone.com

7.1.2 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 hostnames and IP addresses belonging to the same domain. For example, we know that the domain has a web server, with the hostname "www.megacorpone.com".

Let's run **host** against this hostname:

```
kali@kali:~$ host www.megacorpone.com
www.megacorpone.com has address 38.100.193.76
```

Listing 212 - Using host to look up a valid host

Now, let's see if megacorpone.com has a server with the hostname "idontexist". Notice the difference between the query outputs:

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

Listing 213 - Using host to look up an invalid host

In Listing 212, we queried a valid hostname and received an IP resolution response. By contrast, Listing 213 returned an error (*NXDOMAIN*[189]) that indicated that a public DNS record does not exist for that hostname. Now that we understand how to search for valid hostnames, we can automate our efforts.

7.1.3 Forward Lookup Brute Force

Brute force is a trial-and-error technique that seeks to find valid information, including directories on a webserver, username and password combinations, or in this case, valid DNS records. By using a wordlist that contains common hostnames, we can attempt to guess DNS records and check the response for valid hostnames.

In the examples so far, we used *forward lookups*, which request the IP address of a hostname, to query both a valid and an invalid hostname. If **`host`** successfully resolves a name to an IP, this could be an indication of a functional server.

We can automate the forward DNS lookup of common hostnames using the **`host`** command in a Bash one-liner.

First, let's build a list of possible hostnames:

```
kali@kali:~$ cat list.txt
www
ftp
mail
owa
proxy
router
```

Listing 214 - A small list of possible hostnames

Next, we can use a Bash one-liner to attempt to resolve each hostname:

```
kali@kali:~$ for ip in $(cat list.txt); do host $ip.megacorpone.com; done
www.megacorpone.com has address 38.100.193.76
Host ftp.megacorpone.com not found: 3(NXDOMAIN)
mail.megacorpone.com has address 38.100.193.84
Host owa.megacorpone.com not found: 3(NXDOMAIN)
Host proxy.megacorpone.com not found: 3(NXDOMAIN)
router.megacorpone.com has address 38.100.193.71
```

[189] (Internet Engineering Task Force, 2016), https://tools.ietf.org/html/rfc8020

Listing 215 - Using Bash to brute force forward DNS name lookups

With this simplified wordlist, we discovered entries for "www", "mail", and "router". The hostnames "ftp", "owa", and "proxy", however, were not found. Much more comprehensive wordlists are available as part of the SecLists project.[190] These wordlists can be installed to the /usr/share/seclists directory using the **sudo apt install seclists** command.

7.1.4 Reverse Lookup Brute Force

Our DNS forward brute force enumeration revealed a set of scattered IP addresses in the same approximate range (38.100.193.X). If the DNS administrator of megacorpone.com configured PTR[191] records for the domain, we could scan the approximate range with *reverse lookups* to request the hostname for each IP.

Let's use a loop to scan IP addresses 38.100.193.50 through 38.100.193.100. We will filter out invalid results by showing only entries that do not contain "not found" (with **grep -v**):

```
kali@kali:~$ for ip in $(seq  50 100); do host 38.100.193.$ip; done | grep -v "not fou
nd"
69.193.100.38.in-addr.arpa domain name pointer beta.megacorpone.com.
70.193.100.38.in-addr.arpa domain name pointer ns1.megacorpone.com.
72.193.100.38.in-addr.arpa domain name pointer admin.megacorpone.com.
73.193.100.38.in-addr.arpa domain name pointer mail2.megacorpone.com.
76.193.100.38.in-addr.arpa domain name pointer www.megacorpone.com.
77.193.100.38.in-addr.arpa domain name pointer vpn.megacorpone.com.
...
```

Listing 216 - Using Bash to brute force reverse DNS names

We have successfully managed to resolve a number of IP addresses to valid hosts using reverse DNS lookups. If we were performing an assessment, we could further extrapolate these results, and might scan for "mail1", "mail3", etc and reverse lookup positive results. The point is that these types of scans are often cyclical; we expand our search based on any information we receive at every round.

7.1.5 DNS Zone Transfers

A zone transfer is basically a database replication between related DNS servers in which the *zone file* is copied 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 only be allowed to authorized slave DNS servers but many administrators misconfigure their DNS servers, and in these cases, anyone asking for a copy of the DNS server zone will usually 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.

We have seen organizations whose DNS servers were misconfigured so badly that they did not separate their internal DNS namespace and external DNS

---

[190] (danielmiessler, 2019), https://github.com/danielmiessler/SecLists

[191] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Reverse_DNS_lookup

> *namespace into separate, unrelated zones. This allowed us to retrieve a complete map of the internal and external network structure. It is not uncommon for DNS servers to be misconfigured in this way.*[192]

A successful zone transfer does not directly result in a network breach, although it does facilitate the process.

The **host** command syntax for performing a zone transfer is as follows:

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

Listing 217 - Using host to perform a DNS zone transfer

From our earliest host command (Listing 210), we noticed that three DNS servers serve the megacorpone.com domain: *ns1*, *ns2*, and *ns3*. Let's try a zone transfer against each one.

We will use **host -l** (list zone) to attempt the zone transfers:

```
kali@kali:~$ host -l megacorpone.com ns1.megacorpone.com
Using domain server:
Name: ns1.megacorpone.com
Address: 38.100.193.70#53
Aliases:

Host megacorpone.com not found: 5(REFUSED)
; Transfer failed.
```

Listing 218 - The first zone transfer attempt fails

Unfortunately, it looks like the first nameserver, ns1, does not allow DNS zone transfers, so our attempt has failed.

Let's try to perform the same steps using the second nameserver, ns2:

```
kali@kali:~$ host -l megacorpone.com ns2.megacorpone.com
Using domain server:
Name: ns2.megacorpone.com
Address: 38.100.193.80#53
Aliases:

megacorpone.com name server ns1.megacorpone.com.
megacorpone.com name server ns2.megacorpone.com.
megacorpone.com name server ns3.megacorpone.com.
admin.megacorpone.com has address 38.100.193.83
beta.megacorpone.com has address 38.100.193.88
fs1.megacorpone.com has address 38.100.193.82
intranet.megacorpone.com has address 38.100.193.87
mail.megacorpone.com has address 38.100.193.84
mail2.megacorpone.com has address 38.100.193.73
ns1.megacorpone.com has address 38.100.193.70
...
```

Listing 219 - Using host to illustrate a DNS zone transfer

[192] (mandatoryprogrammer, 2019), https://github.com/mandatoryprogrammer/TLDR

This server allows zone transfers and provides a full dump of the zone file for the megacorpone.com domain, delivering a convenient list of IP addresses and corresponding DNS hostnames!

The megacorpone.com domain has very few DNS servers to check. However, some larger organizations might host many DNS servers, or we might want to attempt zone transfer requests against all the DNS servers in a given domain. Bash scripting can help with this task.

To attempt a zone transfer with the **host** command, we need two parameters: a nameserver address and a domain name. We can get the nameservers for a given domain with the following command:

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

Listing 220 - Using host to obtain DNS servers for a given domain name

Taking this a step further, we can write a Bash script to automate the process of identifying the relevant nameservers and attempting a zone transfer from each:

```
#!/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
```

Listing 221 - Our Bash DNS zone transfer script

Let's make the script executable and run it against megacorpone.com.

```
kali@kali:~$ chmod +x dns-axfr.sh

kali@kali:~$ ./dns-axfr.sh megacorpone.com
admin.megacorpone.com has address 38.100.193.83
beta.megacorpone.com has address 38.100.193.88
fs1.megacorpone.com has address 38.100.193.82
intranet.megacorpone.com has address 38.100.193.87
mail.megacorpone.com has address 38.100.193.84
mail2.megacorpone.com has address 38.100.193.73
ns1.megacorpone.com has address 38.100.193.70
ns2.megacorpone.com has address 38.100.193.80
ns3.megacorpone.com has address 38.100.193.90
```

```
router.megacorpone.com has address 38.100.193.71
...
```

Listing 222 - Running the DNS zone transfer Bash script

7.1.6 Relevant Tools in Kali Linux

There are several tools in Kali Linux that can automate DNS enumeration. Two notable examples are *DNSRecon* and *DNSenum*, which have useful options that we'll explore in the following sections.

7.1.6.1 DNSRecon

DNSRecon[193] is an advanced, modern DNS enumeration script written in Python. Running **dnsrecon** against megacorpone.com using the **-d** option to specify a domain name, and **-t** to specify the type of enumeration to perform (in this case a zone transfer), produces the following output:

```
kali@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 38.100.193.70
[*] Resolving NS Records
[*] NS Servers found:
[*]     NS ns1.megacorpone.com 38.100.193.70
[*]     NS ns2.megacorpone.com 38.100.193.80
[*]     NS ns3.megacorpone.com 38.100.193.90
[*] Removing any duplicate NS server IP Addresses...
[*]
[*] Trying NS server 38.100.193.80
[+] 38.100.193.80 Has port 53 TCP Open
[+] Zone Transfer was successful!!
[*]      NS ns1.megacorpone.com 38.100.193.70
[*]      NS ns2.megacorpone.com 38.100.193.80
[*]      NS ns3.megacorpone.com 38.100.193.90
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.178.215
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.178.217
[*]      MX @.megacorpone.com fb.mail.gandi.net 217.70.178.216
[*]      MX @.megacorpone.com spool.mail.gandi.net 217.70.178.1
[*]      A admin.megacorpone.com 38.100.193.83
[*]      A fs1.megacorpone.com 38.100.193.82
[*]      A www2.megacorpone.com 38.100.193.79
[*]      A test.megacorpone.com 38.100.193.67
[*]      A ns1.megacorpone.com 38.100.193.70
[*]      A ns2.megacorpone.com 38.100.193.80
[*]      A ns3.megacorpone.com 38.100.193.90
...
[*]
[*] Trying NS server 38.100.193.70
[+] 38.100.193.70 Has port 53 TCP Open
[-] Zone Transfer Failed!
[-] No answer or RRset not for qname
[*]
```

[193] (darkoperator, 2019), https://github.com/darkoperator/dnsrecon

```
[*] Trying NS server 38.100.193.90
[+] 38.100.193.90 Has port 53 TCP Open
[-] Zone Transfer Failed!
[-] No answer or RRset not for qname
```

Listing 223 - Using dnsrecon to perform a zone transfer

Based on the output above, we have managed to perform a successful DNS zone transfer against the megacorpone.com domain. The result is basically a full dump of the zone file for the domain.

Let's try to brute force additional hostnames using the list.txt file we created previously for forward lookups. That list looks like this:

```
kali@kali:~$ cat list.txt
www
ftp
mail
owa
proxy
router
```

Listing 224 - List to be used for subdomain brute forcing using dnsrecon

To begin the brute force attempt, we will use the **-d** option to specify a domain name, **-D** to specify a file name containing potential subdomain strings, and **-t** to specify the type of enumeration to perform (in this case **brt** for brute force):

```
kali@kali:~$ dnsrecon -d megacorpone.com -D ~/list.txt -t brt
[*] Performing host and subdomain brute force against megacorpone.com
[*]      A router.megacorpone.com 38.100.193.71
[*]      A www.megacorpone.com 38.100.193.76
[*]      A mail.megacorpone.com 38.100.193.84
[+] 3 Records Found
```

Listing 225 - Brute forcing hostnames using dnsrecon

Our brute force attempt has finished, and we have managed to resolve a few hostnames.

7.1.6.2 DNSenum

DNSEnum is another popular DNS enumeration tool. To show a different output, let's run **dnsenum** against the *zonetransfer.me* domain (which is owned by DigiNinja[194] and specifically allows zone transfers):

```
kali@kali:~$ dnsenum zonetransfer.me
dnsenum.pl VERSION:1.2.2
-----   zonetransfer.me   -----

Host's addresses:
__________________

zonetransfer.me                          7200     IN    A        217.147.180.162


Name Servers:
```

[194] (DigiNinja), https://digi.ninja/about.php

```
--------------

ns12.zoneedit.com                       3653     IN    A        209.62.64.46
ns16.zoneedit.com                       6975     IN    A        69.64.68.41


Mail (MX) Servers:
-------------------

ASPMX5.GOOGLEMAIL.COM                   293      IN    A        173.194.69.26
ASPMX.L.GOOGLE.COM                      293      IN    A        173.194.74.26
ALT1.ASPMX.L.GOOGLE.COM                 293      IN    A        173.194.66.26
ALT2.ASPMX.L.GOOGLE.COM                 293      IN    A        173.194.65.26
ASPMX2.GOOGLEMAIL.COM                   293      IN    A        173.194.78.26
ASPMX3.GOOGLEMAIL.COM                   293      IN    A        173.194.65.26
ASPMX4.GOOGLEMAIL.COM                   293      IN    A        173.194.70.26


Trying Zone Transfers and getting Bind Versions:
------------------------------------------------


Trying Zone Transfer for zonetransfer.me on ns12.zoneedit.com ...
zonetransfer.me                         7200     IN    SOA
zonetransfer.me                         7200     IN    NS
...
office.zonetransfer.me                  7200     IN    A        4.23.39.254
owa.zonetransfer.me                     7200     IN    A        207.46.197.32
info.zonetransfer.me                    7200     IN    TXT
asfdbbox.zonetransfer.me                7200     IN    A         127.0.0.1
canberra_office.zonetransfer.me         7200     IN    A        202.14.81.230
asfdbvolume.zonetransfer.me             7800     IN    AFSDB
email.zonetransfer.me                   2222     IN    NAPTR
dzc.zonetransfer.me                     7200     IN    TXT
robinwood.zonetransfer.me               302      IN    TXT
vpn.zonetransfer.me                     4000     IN    A        174.36.59.154
_sip._tcp.zonetransfer.me               14000    IN    SRV
dc_office.zonetransfer.me               7200     IN    A        143.228.181.132

ns16.zoneedit.com Bind Version: 8.4.X

brute force file not specified, bay.
```

Listing 226 - Using dnsenum to perform a zone transfer.

These enumeration tools are both capable and straightforward. Take some time to practice with each before continuing.

7.1.6.3 Exercises

1. Find the DNS servers for the megacorpone.com domain.
2. Write a small script to attempt a zone transfer from megacorpone.com using a higher-level scripting language such as Python, Perl, or Ruby.
3. Recreate the example above and use **`dnsrecon`** to attempt a zone transfer from megacorpone.com.

7.2 Port Scanning

Port scanning is the process of inspecting TCP or UDP ports on a remote machine with the intention of detecting what services are running on the target and what potential attack vectors may exist.

> *Please note that port scanning is not representative of traditional user activity and could be considered illegal in some jurisdictions. Therefore, it should not be performed outside the labs without direct, written permission from the target network owner.*

It is essential to understand the implications of port scanning, as well as the impact that specific port scans can have. Due to the amount of traffic some scans can generate, along with their intrusive nature, running port scans blindly can have adverse effects on target systems or the client network such as overloading servers and network links or triggering IDS. Running the wrong scan could result in downtime for the customer.

Using a proper port scanning methodology can significantly improve our efficiency as penetration testers while also limiting many of the risks. Depending on the scope of the engagement, instead of running a full port scan against the target network, we can start by only scanning for ports 80 and 443. With a list of possible web servers, we can run a full port scan against these servers in the background while performing other enumeration. Once the full port scan is complete, we can further narrow our scans to probe for more and more information with each subsequent scan. Port scanning should be viewed as a dynamic process that is unique to each engagement. The results of one scan determine the type and scope of the next scan.

7.2.1 TCP / UDP Scanning

We'll begin our exploration of port scanning with a simple TCP and UDP port scan using Netcat. It should be noted that Netcat is **not** a port scanner, but it can be used as such in a rudimentary way. Since it's already present on many systems, we can repurpose some of its functionality to mimic a basic port scan when we are not in need of a fully-featured port scanner. However, there are far better tools dedicated to port scanning that we will explore in detail as well.

7.2.1.1 TCP Scanning

The simplest TCP port scanning technique, usually called CONNECT scanning, relies on the three-way TCP handshake[195] mechanism. This mechanism is designed so that two hosts attempting to communicate can negotiate the parameters of the network TCP socket connection before transmitting any data. In basic terms, a host sends a TCP *SYN* packet to a server on a destination port. If the destination port is open, the server responds with a *SYN-ACK* packet and the client host sends an *ACK* packet to complete the handshake.

If the handshake completes successfully, the port is considered open.

[195] (Microsoft, 2010), http://support.microsoft.com/kb/172983

To illustrate this, we will run a TCP Netcat port scan on ports 3388-3390. The **-w** option specifies the connection timeout in seconds and **-z** is used to specify zero-I/O mode, which will send no data and is used for scanning:

```
kali@kali:~$ nc -nvv -w 1 -z 10.11.1.220 3388-3390
(UNKNOWN) [10.11.1.220] 3390 (?) : Connection refused
(UNKNOWN) [10.11.1.220] 3389 (?) open
(UNKNOWN) [10.11.1.220] 3388 (?) : Connection refused
 sent 0, rcvd 0
```

Listing 227 - Using nc to perform a TCP port scan

Based on this output, we can see that port 3389 is open while connections on ports 3388 and 3390 timed out. The screenshot below shows the Wireshark capture of this scan:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.11.0.126	10.11.1.220	TCP	74	54272 → 3390 [SYN] Seq=0 Win=292(
4	0.173465715	10.11.1.220	10.11.0.126	TCP	60	3390 → 54272 [RST, ACK] Seq=1 Acl
5	0.173993925	10.11.0.126	10.11.1.220	TCP	74	45692 → 3389 [SYN] Seq=0 Win=292(
6	0.257814845	10.11.1.220	10.11.0.126	TCP	74	3389 → 45692 [SYN, ACK] Seq=0 Acl
7	0.257843092	10.11.0.126	10.11.1.220	TCP	66	45692 → 3389 [ACK] Seq=1 Ack=1 W:
8	0.258157571	10.11.0.126	10.11.1.220	TCP	66	45692 → 3389 [FIN, ACK] Seq=1 Acl
9	0.258406541	10.11.0.126	10.11.1.220	TCP	74	34322 → 3388 [SYN] Seq=0 Win=292(
10	0.343710940	10.11.1.220	10.11.0.126	TCP	66	3389 → 45692 [ACK] Seq=1 Ack=2 W:
11	0.343750348	10.11.1.220	10.11.0.126	TCP	60	3388 → 34322 [RST, ACK] Seq=1 Acl
12	0.345885725	10.11.1.220	10.11.0.126	TCP	60	3389 → 45692 [RST, ACK] Seq=1 Acl

Figure 42: Wireshark capture of the Netcat port scan

In this capture (Figure 42) Netcat sent several TCP SYN packets to ports 3390, 3389, and 3388 on lines 1, 5 and 9 respectively. Due to a variety of factors, including timing issues, the packets may appear out of order in Wireshark. Notice that the server sent a TCP *SYN-ACK* packet from port 3389 on line 6, indicating that the port is open. The other ports did not reply with a similar SYN-ACK packet, so we can infer that they are not open. Finally, on line 8, Netcat closed down this connection by sending a *FIN-ACK* packet.

7.2.1.2 UDP Scanning

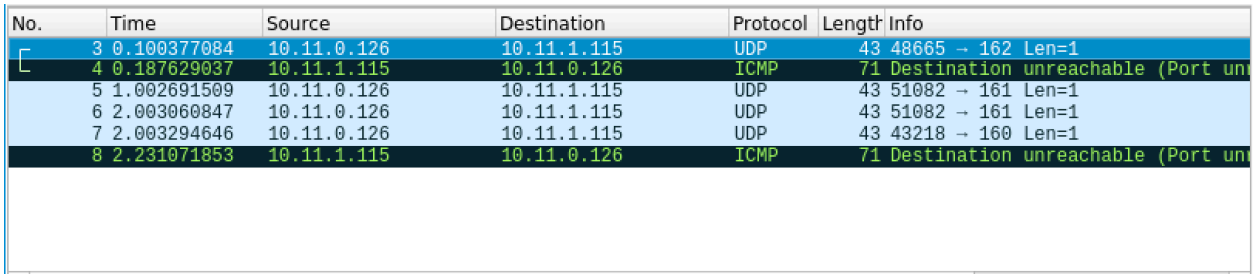
Since UDP is stateless and does not involve a three-way handshake, the mechanism behind UDP port scanning is different from TCP.

Let's run a UDP Netcat port scan against ports 160-162 on a different target. This is done using the only **nc** option we have not seen yet, **-u**, which indicates a UDP scan:

```
kali@kali:~$ nc -nv -u -z -w 1 10.11.1.115 160-162
(UNKNOWN) [10.11.1.115] 161 (snmp) open
```

Listing 228 - Using Netcat to perform a UDP port scan

From the Wireshark capture, we can see that the UDP scan uses a different mechanism than a TCP scan:



No.	Time	Source	Destination	Protocol	Length	Info
3	0.100377084	10.11.0.126	10.11.1.115	UDP	43	48665 → 162 Len=1
4	0.187629037	10.11.1.115	10.11.0.126	ICMP	71	Destination unreachable (Port un
5	1.002691509	10.11.0.126	10.11.1.115	UDP	43	51082 → 161 Len=1
6	2.003060847	10.11.0.126	10.11.1.115	UDP	43	51082 → 161 Len=1
7	2.003294646	10.11.0.126	10.11.1.115	UDP	43	43218 → 160 Len=1
8	2.231071853	10.11.1.115	10.11.0.126	ICMP	71	Destination unreachable (Port un

Figure 43: Wireshark capture of a UDP Netcat port scan

As seen in Figure 43, an empty UDP packet is sent to a specific port (packets 3, 5, 6, and 7). If the destination UDP port is open, the packet will be passed to the application layer and the response received will depend on how the application is programmed to respond to empty packets. In this example, the application sends no response. However, if the destination UDP port is closed, the target should respond with an ICMP port unreachable (as seen in packets 4 and 8), that is sent by the UDP/IP stack of the target machine.

Most UDP scanners tend to use the standard "ICMP port unreachable" message to infer the status of a target port. However, this method can be completely unreliable when the target port is filtered by a firewall. In fact, in these cases the scanner will report the target port as open because of the absence of the ICMP message.

7.2.1.3 Common Port Scanning Pitfalls

UDP scanning can be problematic for several reasons. First, UDP scanning is often unreliable, as firewalls and routers may drop ICMP packets. This can lead to false positives and ports showing as open when they are, in fact, closed. Second, many port scanners do not scan all available ports, and usually have a pre-set list of "interesting ports" that are scanned. This means open UDP ports can go unnoticed. Using a protocol-specific UDP port scanner may help in obtaining more accurate results. Finally, penetration testers often forget to scan for open UDP ports, instead focusing on the "more exciting" TCP ports. Although UDP scanning can be unreliable, there are plenty of attack vectors lurking behind open UDP ports.

7.2.2 Port Scanning with Nmap

Nmap[196] (written by Gordon Lyon, aka Fyodor) is one of the most popular, versatile, and robust port scanners available. It has been actively developed for over a decade and has numerous features beyond port scanning.

> *Some of the Nmap example scans we cover in this module are run using* ***sudo****. This is due to the fact that quite a few Nmap scanning options require access to raw sockets,[197] which in turn require root privileges. Raw sockets allow for surgical manipulation of TCP and UDP packets. Without access to raw sockets,*

[196] (Nmap, 2019), http://nmap.org/

[197] (Man7, 2017), http://man7.org/linux/man-pages/man7/raw.7.html

Nmap is limited as it falls back to crafting packets by using the standard Berkeley socket API.[198]

Let's explore some port scanning examples to get a better feel for Nmap and its options.

7.2.2.1 Accountability for Our Traffic

A default Nmap TCP scan will scan the 1000 most popular ports on a given machine. Before we start running scans blindly, let's examine the amount of traffic sent by this type of scan. We'll scan one of the lab machines while monitoring the amount of traffic sent to the target host using **iptables**.[199]

We will use several **iptables** options. First, we will use the **-I** option to insert a new rule into a given chain, which in this case includes both the **INPUT** (Inbound) and **OUTPUT** (Outbound) chains followed by the rule number. We will use **-s** to specify a source IP address, **-d** to specify a destination IP address, and **-j** to **ACCEPT** the traffic. Lastly, we will use the **-Z** option to zero the packet and byte counters in all chains.

Let's run those commands now:

```
kali@kali:~$ sudo iptables -I INPUT 1 -s 10.11.1.220 -j ACCEPT

kali@kali:~$ sudo iptables -I OUTPUT 1 -d 10.11.1.220 -j ACCEPT

kali@kali:~$ sudo iptables -Z
```

Listing 229 - Configuring our iptables rules for the scan

Now let's generate some traffic using **nmap**:

```
kali@kali:~$ nmap 10.11.1.220
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:20 EST
Nmap scan report for 10.11.1.220
Host is up (0.29s latency).
Not shown: 980 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
593/tcp   open  http-rpc-epmap
636/tcp   open  ldapssl
3268/tcp  open  globalcatLDAP
3269/tcp  open  globalcatLDAPssl
3389/tcp  open  ms-wbt-server
```

[198] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Berkeley_sockets#Socket_API_functions

[199] (netfilter, 2014), http://netfilter.org/projects/iptables/index.html

```
...

Nmap done: 1 IP address (1 host up) scanned in 46.29 seconds
```

Listing 230 - Scanning an IP for the 1000 most popular TCP ports

The scan completed and revealed a few open ports.

Now let's look at some **iptables** statistics to get an idea of how much traffic our scan generated. We will use the **-v** option to add some verbosity to our output, **-n** to enable numeric output, and **-L** to list the rules present in all chains:

```
kali@kali:~$ sudo iptables -vn -L
Chain INPUT (policy ACCEPT 1528 packets, 226K bytes)
 pkts bytes target     prot opt in     out     source               destination
 1263 51264 ACCEPT     all  --  *      *       10.11.1.220          0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 1323 packets, 191K bytes)
 pkts bytes target     prot opt in     out     source               destination
 1314 78300 ACCEPT     all  --  *      *       0.0.0.0/0            10.11.1.220
```

Listing 231 - Using iptables to monitor nmap traffic for a top 1000 port scan

According to this output, this default 1000-port scan has generated around 78 KB of traffic.

Let's use **iptables -Z** to zero the packet and byte counters in all chains again and run another **nmap** scan using **-p** to specify *ALL* TCP ports:

```
kali@kali:~$ sudo iptables -Z

kali@kali:~$ nmap -p 1-65535 10.11.1.220
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.220
Host is up (0.00067s latency).
Not shown: 65507 closed ports
PORT      STATE    SERVICE
21/tcp    open     ftp
53/tcp    open     domain
88/tcp    open     kerberos-sec
135/tcp   open     msrpc
139/tcp   open     netbios-ssn
389/tcp   open     ldap
445/tcp   open     microsoft-ds
464/tcp   open     kpasswd5
593/tcp   open     http-rpc-epmap
636/tcp   open     ldapssl
1291/tcp  filtered seagulllms
3268/tcp  open     globalcatLDAP
3269/tcp  open     globalcatLDAPssl
3389/tcp  open     ms-wbt-server
5722/tcp  open     msdfsr
9389/tcp  open     adws
12777/tcp filtered unknown
46056/tcp filtered unknown
```

```
47001/tcp open     winrm
...

Nmap done: 1 IP address (1 host up) scanned in 80.42 seconds

kali@kali:~$ sudo iptables -vn -L
Chain INPUT (policy ACCEPT 219K packets, 252M bytes)
 pkts bytes target     prot opt in     out     source               destination
66243 2659K ACCEPT     all  --  *      *       10.11.1.220          0.0.0.0/0

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 85792 packets, 11M bytes)
 pkts bytes target     prot opt in     out     source               destination
66768 4006K ACCEPT     all  --  *      *       0.0.0.0/0            10.11.1.220
```

Listing 232 - Using iptables to monitor nmap traffic for a port scan on ALL TCP ports

A similar local port scan explicitly probing all 65535 ports generated about 4 MB of traffic, a significantly higher amount. However, this full port scan has discovered new ports that were not found by the default TCP scan.

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 reveals 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 A or B network assessment.

In the next section, we'll explore some of Nmap's various scanning techniques.

7.2.2.2 Stealth / SYN Scanning

Nmap's preferred scanning technique is a SYN, or "stealth" scan.[200] There are many benefits to using a SYN scan and as such, it is the default scan technique used when no scan technique is specified in an **nmap** command and the user has the required raw sockets privileges.

SYN 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. At this point, the port scanner does not bother to send the final ACK to complete the three-way handshake.

```
kali@kali:~$ sudo nmap -sS 10.11.1.220
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.220
Host is up (1.3s latency).
Not shown: 980 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
```

[200] (Nmap, 2019), https://nmap.org/book/synscan.html

```
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
...
```

Listing 233 - Using nmap to perform a SYN scan

Because the three-way handshake is never completed, the information is not passed to the application layer and as a result, will not appear in any application logs. A SYN scan is also faster and more efficient because fewer packets are sent and received.

> *Please note that term "stealth" refers to the fact that, in the past, primitive firewalls would fail to log incomplete TCP connections. This is no longer the case with modern firewalls and even if the stealth moniker has stuck around, it could be misleading.*

7.2.2.3 TCP Connect Scanning

When a user running **nmap** does not have raw socket privileges, Nmap will default to the TCP connect scan[201] technique described earlier. Since a Nmap TCP connect scan makes use of the Berkeley sockets API to perform the three-way handshake, it does not require elevated privileges. However, because Nmap has to wait for the connection to complete before the API will return the status of the connection, a connect scan takes much longer to complete than a SYN scan.

There might be times when we need to specifically perform a connect scan with **nmap**, for example, when scanning via certain types of proxies. We use the **-sT** option to start a connect scan:

```
kali@kali:~$ nmap -sT 10.11.1.220
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:37 EST
Nmap scan report for 10.11.1.220
Host is up (1.3s latency).
Not shown: 980 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
88/tcp    open  kerberos-sec
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  microsoft-ds
464/tcp   open  kpasswd5
...
```

Listing 234 - Using nmap to perform a TCP connect scan

[201] (Nmap, 2019), https://nmap.org/book/scan-methods-connect-scan.html

7.2.2.4 UDP Scanning

When performing a UDP scan,[202] Nmap will use a combination of two different methods to determine if a port is open or closed. For most ports, it will use the standard "ICMP port unreachable" method described earlier by sending an empty packet to a given port. However, for common ports, such as port 161, which is used by SNMP, it will send a protocol-specific SNMP packet in an attempt to get a response from an application bound to that port. To perform a UDP scan, the **-sU** option is used and **sudo** is required to access raw sockets:

```
kali@kali:~$ sudo nmap -sU 10.11.1.115
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:46 EST
Nmap scan report for 10.11.1.115
Host is up (0.079s latency).
Not shown: 997 open|filtered ports
PORT    STATE SERVICE
111/udp open  rpcbind
137/udp open  netbios-ns
161/udp open  snmp

Nmap done: 1 IP address (1 host up) scanned in 22.49 seconds
```

Listing 235 - Using nmap to perform a UDP scan

The UDP scan (**-sU**) can also be used in conjunction with a TCP SYN scan (**-sS**) option to build a more complete picture of our target:

```
kali@kali:~$ sudo nmap -sS -sU 10.11.1.115
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 12:46 EST
Nmap scan report for 10.11.1.115
Host is up (0.15s latency).
Not shown: 997 open|filtered ports, 989 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
25/tcp    open  smtp
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
143/tcp   open  imap
199/tcp   open  smux
443/tcp   open  https
3306/tcp  open  mysql
32768/tcp open  filenet-tms
111/udp   open  rpcbind
137/udp   open  netbios-ns
161/udp   open  snmp

Nmap done: 1 IP address (1 host up) scanned in 64.74 seconds
```

Listing 236 - Using nmap to perform a combined UDP and SYN scan

In the next section, we'll explore techniques for handling larger networks or networks with traffic constraints.

[202] (Nmap, 2019), https://nmap.org/book/scan-methods-udp-scan.html

7.2.2.5 Network Sweeping

To deal with large volumes of hosts, or to otherwise try to conserve network traffic, we can attempt to probe targets using *Network Sweeping* techniques, in which we begin with broad scans, and use more specific scans against hosts of interest.

When performing a network sweep with Nmap using the **-sn** option, the host discovery process consists of more than just sending an ICMP echo request. Several other probes are used in addition to the ICMP request. Nmap also sends a TCP SYN packet to port 443, a TCP ACK packet to port 80, and an ICMP timestamp request to verify if a host is available or not.

```
kali@kali:~$ nmap -sn 10.11.1.1-254
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.5
Host is up (0.026s latency).
MAC Address: 00:50:56:89:70:15 (VMware)
Nmap scan report for 10.11.1.7
Host is up (0.026s latency).
MAC Address: 00:50:56:89:36:32 (VMware)
...
Nmap done: 254 IP addresses (44 hosts up) scanned in 6.14 seconds
```

Listing 237 - Using nmap to perform a network sweep

Searching for live machines using the **grep** command on a standard nmap output can be cumbersome. Instead, let's use Nmap's "greppable" output parameter, **-oG**, to save these results into a format that is easier to manage:

```
kali@kali:~$ nmap -v -sn 10.11.1.1-254 -oG ping-sweep.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:34 EST
Initiating ARP Ping Scan at 11:34
Scanning 254 hosts [1 port/host]
Completed ARP Ping Scan at 11:35, 4.71s elapsed (254 total hosts)
Initiating Parallel DNS resolution of 254 hosts. at 11:35
Completed Parallel DNS resolution of 254 hosts. at 11:35, 0.07s elapsed
Nmap scan report for 10.11.1.1 [host down]
Nmap scan report for 10.11.1.2 [host down]
Nmap scan report for 10.11.1.3 [host down]
Nmap scan report for 10.11.1.4 [host down]
Nmap scan report for 10.11.1.5
Host is up (0.026s latency).
MAC Address: 00:50:56:89:70:15 (VMware)
...

kali@kali:~$ grep Up ping-sweep.txt | cut -d " " -f 2
10.11.1.5
10.11.1.7
10.11.1.8
...
```

Listing 238 - Using nmap to perform a network sweep and then using grep to find live hosts

We can also sweep for specific TCP or UDP ports across the network, probing for common services and ports, in an attempt to locate systems that may be useful, or otherwise have known vulnerabilities. This scan tends to be more accurate than a ping sweep:

```
kali@kali:~$ nmap -p 80 10.11.1.1-254 -oG web-sweep.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:38 EST
Nmap scan report for 10.11.1.5
Host is up (0.036s latency).

PORT   STATE  SERVICE
80/tcp closed http
MAC Address: 00:50:56:89:70:15 (VMware)

Nmap scan report for 10.11.1.7
Host is up (0.029s latency).

PORT   STATE    SERVICE
80/tcp filtered http
MAC Address: 00:50:56:89:36:32 (VMware)

Nmap scan report for 10.11.1.8
Host is up (0.034s latency).

PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:50:56:89:20:34 (VMware)
...

kali@kali:~$ grep open web-sweep.txt | cut -d" " -f2
10.11.1.8
10.11.1.10
10.11.1.13
...
```

Listing 239 - Using nmap to scan for web servers using port 80

To save time and network resources, we can also scan multiple IPs, probing for a short list of common ports. For example, let's conduct a *TCP connect scan* for the top twenty TCP ports with the **--top-ports** option and enable OS version detection, script scanning, and traceroute with **-A**:

```
kali@kali:~$ nmap -sT -A --top-ports=20 10.11.1.1-254 -oG top-port-sweep.txt
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:40 EST
Nmap scan report for 10.11.1.5
Host is up (0.037s latency).

PORT     STATE  SERVICE        VERSION
21/tcp   closed ftp
22/tcp   closed ssh
23/tcp   closed telnet
25/tcp   closed smtp
53/tcp   closed domain
80/tcp   closed http
110/tcp  closed pop3
...
Host script results:
|_nbstat: NetBIOS name: ALICE, NetBIOS user: <unknown>, NetBIOS MAC: 00:50:56:89:70:15
| smb-os-discovery:
|   OS: Windows XP (Windows 2000 LAN Manager)
|   OS CPE: cpe:/o:microsoft:windows_xp::-
|   Computer name: alice
```

```
|   NetBIOS computer name: ALICE\x00
|   Domain name: thinc.local
|   Forest name: thinc.local
|   FQDN: alice.thinc.local
|_  System time: 2019-03-04T16:44:52+00:00
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
|_smb2-time: Protocol negotiation failed (SMB2)
...
```

Listing 240 - Using nmap to perform a top twenty port scan, saving the output in greppable format

The top twenty **nmap** ports are determined using the /usr/share/nmap/nmap-services file. The file uses a simple format of three whitespace-separated columns. The first is the name of the service, the second contains the port number and protocol, and the third, the "port frequency". Everything after the third column is ignored but is typically used for comments as can be seen by the use of the pound sign (#). The port frequency is based on how often the port was found open during research scans of the Internet:[203]

```
kali@kali:~$ cat /usr/share/nmap/nmap-services
...
finger    79/udp    0.000956
http    80/sctp    0.000000    # www-http | www | World Wide Web HTTP
http    80/tcp    0.484143    # World Wide Web HTTP
http    80/udp    0.035767    # World Wide Web HTTP
hosts2-ns    81/tcp    0.012056    # HOSTS2 Name Server
hosts2-ns    81/udp    0.001005    # HOSTS2 Name Server
...
```

Listing 241 - The nmap-services file showing the open frequency of TCP port 80

At this point, we could conduct a more exhaustive scan against individual machines that are service-rich or are otherwise interesting.

There are many different ways we can be creative with our scanning to conserve bandwidth or lower our profile, and most leverage interesting host discovery techniques,[204] which are worth further research.

7.2.2.6 OS Fingerprinting

Nmap has a built-in feature called *OS fingerprinting*,[205] which can be enabled with the **-O** option. This feature attempts to guess the target's operating system by inspecting returned packets. This is possible because operating systems often have slightly different implementations of the TCP/IP stack (such as varying default TTL values and TCP window sizes) and these slight variances create a fingerprint that Nmap can often identify.

[203] (Nmap, 2019), https://nmap.org/book/nmap-services.html

[204] (Nmap, 2019), https://nmap.org/book/man-host-discovery.html

[205] (Nmap, 2019), https://nmap.org/book/osdetect.html

Nmap will inspect the traffic received from the target machine and attempt to match the fingerprint to a known list.

For example, consider this simple **nmap** OS fingerprint scan.

```
kali@kali:~$ sudo nmap -O 10.11.1.220
...
Device type: general purpose
Running: Microsoft Windows 2008|7
OS CPE: cpe:/o:microsoft:windows_server_2008:r2 cpe:/o:microsoft:windows_7
OS details: Microsoft Windows 7 or Windows Server 2008 R2
Network Distance: 1 hop
...
```

Listing 242 - Using nmap for OS fingerprinting

The response suggests that the underlying operating system of this target is either Windows 7 or Windows 2008 R2.

> *Note that OS Fingerprinting is not always 100% accurate, but a best-guess attempt. Consider a more careful examination of the target to confirm an OS fingerprint scan.*

7.2.2.7 Banner Grabbing/Service Enumeration

We can also identify services running on specific ports by inspecting service banners (**-sV**) and running various OS and service enumeration scripts (**-A**) against the target:

```
kali@kali:~$ nmap -sV -sT -A 10.11.1.220
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
Nmap scan report for 10.11.1.220
Host is up (0.00043s latency).
Not shown: 979 closed ports
PORT      STATE SERVICE       VERSION
21/tcp    open  ftp           FileZilla ftpd 0.9.34 beta
| ftp-syst:
|_  SYST: UNIX emulated by FileZilla
53/tcp    open  domain        Microsoft DNS 6.1.7601 (1DB15D39) (Windows Server 2008 R
2 SP1)
| dns-nsid:
|_  bind.version: Microsoft DNS 6.1.7601 (1DB15D39)
88/tcp    open  kerberos-sec  Microsoft Windows Kerberos (server time: 2013-12-28 07:3
7:57Z)
135/tcp   open  msrpc         Microsoft Windows RPC

...
Nmap done: 1 IP address (1 host up) scanned in 55.67 seconds
```

Listing 243 - Using nmap for banner grabbing and/or service enumeration

Keep in mind that banners can be modified by system administrators. As such, these can be intentionally set to fake service names in order to mislead a potential attacker.

Banner grabbing has a significant impact on the amount of traffic used as well as the speed of the scan. We should always be mindful of the options we use with **nmap** and how they affect our scans.

7.2.2.8 Nmap Scripting Engine (NSE)

We can use the Nmap Scripting Engine (NSE)[206] to launch user-created scripts in order to automate various scanning tasks. These scripts perform a broad range of functions including DNS enumeration, brute force attacks, and even vulnerability identification. NSE scripts are located in the /usr/share/nmap/scripts directory.

For example, the *smb-os-discovery* script attempts to connect to the SMB service on a target system and determine its operating system:

```
kali@kali:~$ nmap 10.11.1.220 --script=smb-os-discovery
...
   OS: Windows Server 2008 R2 Standard 7601 Service Pack 1 (Windows Server 2008 R2 Sta
|   OS CPE: cpe:/o:microsoft:windows_server_2008::sp1
|   Computer name: master
|   NetBIOS computer name: MASTER\x00
|   Domain name: thinc.local
|   Forest name: thinc.local
|   FQDN: master.thinc.local
|_  System time: 2013-12-27T23:37:58-08:00

Nmap done: 1 IP address (1 host up) scanned in 5.85 seconds
```

Listing 244 - Using nmap's scripting engine (NSE) for OS fingerprinting

Another useful (and self-explanatory) NSE script is **dns-zone-transfer**:

```
kali@kali:~$ nmap --script=dns-zone-transfer -p 53 ns2.megacorpone.com
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:54 EST
Nmap scan report for ns2.megacorpone.com (38.100.193.80)
Host is up (0.010s latency).
Other addresses for ns2.megacorpone.com (not scanned):

PORT   STATE SERVICE
53/tcp open  domain
| dns-zone-transfer:
| megacorpone.com.           SOA  ns1.megacorpone.com. admin.megacorpone.com.
| megacorpone.com.           MX   10 fb.mail.gandi.net.
| megacorpone.com.           MX   20 spool.mail.gandi.net.
| megacorpone.com.           MX   50 mail.megacorpone.com.
| megacorpone.com.           MX   60 mail2.megacorpone.com.
| megacorpone.com.           NS   ns1.megacorpone.com.
...
```

Listing 245 - Using nmap to perform a DNS zone transfer

To view more information about a script, we can use the **--script-help** option, which displays a description of the script and a URL where we can find more in-depth information, such as the script arguments and usage examples.

[206] (Nmap, 2019), http://nmap.org/book/nse.html

```
kali@kali:~$ nmap --script-help dns-zone-transfer
Starting Nmap 7.70 ( https://nmap.org ) at 2019-05-06 11:02 MDT

dns-zone-transfer
Categories: intrusive discovery
https://nmap.org/nsedoc/scripts/dns-zone-transfer.html
  Requests a zone transfer (AXFR) from a DNS server.

  The script sends an AXFR query to a DNS server. The domain to query is
  determined by examining the name given on the command line, the DNS
  server's hostname, or it can be specified with the
  <code>dns-zone-transfer.domain</code> script argument. If the query is
  successful all domains and domain types are returned along with common
  type specific data (SOA/MX/NS/PTR/A).
...
```

Listing 246 - Using the --script-help option to view more information about a script

For times when Internet access is not available, much of this information can also be found in the NSE script file itself.

Take time to explore the various NSE scripts, as many of them are helpful and time-saving.

7.2.2.9 Exercises

1. Use Nmap to conduct a ping sweep of your target IP range and save the output to a file. Use grep to show machines that are online.
2. Scan the IP addresses you found in exercise 1 for open webserver ports. Use Nmap to find the webserver and operating system versions.
3. Use NSE scripts to scan the machines in the labs that are running the SMB service.
4. Use Wireshark to capture a Nmap connect and UDP scan and compare it against the Netcat port scans. Are they the same or different?
5. Use Wireshark to capture a Nmap SYN scan and compare it to a connect scan and identify the difference between them.

7.2.3 Masscan

Masscan[207] is arguably the fastest port scanner; it can scan the entire Internet in about 6 minutes, transmitting an astounding 10 million packets per second! While it was originally designed to scan the entire Internet, it can easily handle a class A or B subnet, which is a more suitable target range during a penetration test.

Masscan is not installed on Kali by default; it must be installed using **`apt install`**:

```
kali@kali:~$ sudo apt install masscan
...
The following NEW packages will be installed:
  masscan
0 upgraded, 1 newly installed, 0 to remove and 1469 not upgraded.
Need to get 184 kB of archives.
```

[207] (Offensive Security, 2019), https://tools.kali.org/information-gathering/masscan

```
After this operation, 401 kB of additional disk space will be used.
...
```

Listing 247 - Installing masscan on Kali Linux

Consider this demonstration that locates all machines on a large internal network with TCP port 80 open (using the **-p80** option). Since masscan implements a custom TCP/IP stack, it will require access to raw sockets and therefore requires **sudo**.

> *Please Note: This command is NOT to be tried in the PWK internal lab network as you will be scanning subnets you are not allowed to. This example is for illustration purposes only!*

```
kali@kali:~$ sudo masscan -p80 10.0.0.0/8
```

Listing 248 - Using masscan to look for all web servers within a class A subnet

To try masscan on a class C subnet in the PWK internal lab network, we can use the following example. We will add a few additional **masscan** options, including **--rate** to specify the desired rate of packet transmission, **-e** to specify the raw network interface to use, and **--router-ip** to specify the IP address for the appropriate gateway:

```
kali@kali:~$ sudo masscan -p80 10.11.1.0/24 --rate=1000 -e tap0 --router-ip 10.11.0.1

Starting masscan 1.0.3 (http://bit.ly/14GZzcT) at 2019-03-04 17:15:40 GMT
 -- forced options: -sS -Pn -n --randomize-hosts -v --send-eth
Initiating SYN Stealth Scan
Scanning 256 hosts [1 port/host]
Discovered open port 80/tcp on 10.11.1.14
Discovered open port 80/tcp on 10.11.1.39
Discovered open port 80/tcp on 10.11.1.219
Discovered open port 80/tcp on 10.11.1.227
Discovered open port 80/tcp on 10.11.1.10
Discovered open port 80/tcp on 10.11.1.50
Discovered open port 80/tcp on 10.11.1.234
...
```

Listing 249 - Using masscan with more advanced options

7.3 SMB Enumeration

The security track record of the Server Message Block (SMB)[208] protocol has been poor for many years due to its complex implementation and open nature. From unauthenticated SMB null sessions in Windows 2000 and XP, to a plethora of SMB bugs and vulnerabilities over the years, SMB has seen its fair share of action.[209]

That said, the SMB protocol has also been updated and improved in parallel with Windows Operating Systems releases.

[208] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Server_Message_Block

[209] (Mark A. Gamache, 2013), http://markgamache.blogspot.ca/2013/01/ntlm-challenge-response-is-100-broken.html

7.3.1 Scanning for the NetBIOS Service

The NetBIOS[210] service listens on TCP port 139 as well as several UDP ports. It should be noted that SMB (TCP port 445) and NetBIOS are two separate protocols. NetBIOS is an independent session layer protocol and service that allows computers on a local network to communicate with each other. While modern implementations of SMB can work without NetBIOS, *NetBIOS over TCP* (NBT)[211] is required for backward compatibility and is often enabled together. For this reason, the enumeration of these two services often goes hand-in-hand. These can be scanned with tools like **nmap**, using syntax similar to the following:

```
kali@kali:~$ nmap -v -p 139,445 -oG smb.txt 10.11.1.1-254
```

Listing 250 - Using nmap to scan for the NetBIOS service

There are other, more specialized tools for specifically identifying NetBIOS information, such as **nbtscan**, which is used in the following example. The **-r** option is used to specify the originating UDP port as 137, which is used to query the NetBIOS name service for valid NetBIOS names:

```
kali@kali:~$ sudo nbtscan -r 10.11.1.0/24
Doing NBT name scan for addresses from 10.11.1.0/24

IP address       NetBIOS Name     Server    User             MAC address
------------------------------------------------------------------------------
10.11.1.5        ALICE            <server>  ALICE            00:50:56:89:35:af
10.11.1.31       RALPH            <server>  HACKER           00:50:56:89:08:19
10.11.1.24       PAYDAY           <server>  PAYDAY           00:00:00:00:00:00
...
```

Listing 251 - Using nbtscan to collect additional NetBIOS information

7.3.2 Nmap SMB NSE Scripts

Nmap contains many useful NSE scripts that can be used to discover and enumerate SMB services. These scripts can be found in the /usr/share/nmap/scripts directory:

```
kali@kali:~$ ls -1 /usr/share/nmap/scripts/smb*
/usr/share/nmap/scripts/smb2-capabilities.nse
/usr/share/nmap/scripts/smb2-security-mode.nse
/usr/share/nmap/scripts/smb2-time.nse
/usr/share/nmap/scripts/smb2-vuln-uptime.nse
/usr/share/nmap/scripts/smb-brute.nse
/usr/share/nmap/scripts/smb-double-pulsar-backdoor.nse
/usr/share/nmap/scripts/smb-enum-domains.nse
/usr/share/nmap/scripts/smb-enum-groups.nse
/usr/share/nmap/scripts/smb-enum-processes.nse
/usr/share/nmap/scripts/smb-enum-sessions.nse
/usr/share/nmap/scripts/smb-enum-shares.nse
/usr/share/nmap/scripts/smb-enum-users.nse
/usr/share/nmap/scripts/smb-os-discovery.nse
...
```

Listing 252 - Finding various nmap SMB NSE scripts

[210] (Wikipedia, 2019), https://en.wikipedia.org/wiki/NetBIOS

[211] (Wikipedia, 2019), https://en.wikipedia.org/wiki/NetBIOS_over_TCP/IP

Here we find several interesting Nmap SMB NSE scripts that perform various tasks such as OS discovery and enumeration via SMB.

Let's try the **smb-os-discovery** module:

```
kali@kali:~$ nmap -v -p 139, 445 --script=smb-os-discovery 10.11.1.227
...
Nmap scan report for 10.11.1.227
Host is up (0.57s latency).
PORT    STATE SERVICE
139/tcp open  netbios-ssn

Host script results:
| smb-os-discovery:
|   OS: Windows 2000 (Windows 2000 LAN Manager)
|   OS CPE: cpe:/o:microsoft:windows_2000::-
|   Computer name: srv2
|   NetBIOS computer name: SRV2
|   Workgroup: WORKGROUP
...
```

Listing 253 - Using the nmap scripting engine to perform OS discovery

This particular script identified a potential match for the host operating system.

To check for known SMB protocol vulnerabilities, we can invoke one of the *smb-vuln* NSE scripts. We will take a look at **smb-vuln-ms08-067**, which uses the **--script-args** option to pass arguments to the NSE script.

> *Please Note: If we set the script parameter **unsafe=1**, the scripts that will run are almost (or totally) guaranteed to crash a vulnerable system. Needless to say, exercise extreme caution when enabling this argument, especially when scanning production systems.*

```
kali@kali:~$ nmap -v -p 139,445 --script=smb-vuln-ms08-067 --script-args=unsafe=1 10.1
1.1.5
Starting Nmap 7.70 ( https://nmap.org ) at 2019-03-04 11:27 EST
NSE: Loaded 1 scripts for scanning.
NSE: Script Pre-scanning.
...
Scanning 10.11.1.5 [2 ports]
...
Completed NSE at 00:04, 17.39s elapsed
Nmap scan report for 10.11.1.5
Host is up (0.17s latency).
PORT    STATE SERVICE
139/tcp open  netbios-ssn
445/tcp open  microsoft-ds
MAC Address: 00:50:56:AF:02:91 (VMware)

Host script results:
| smb-vuln-ms08-067:
|   VULNERABLE:
```

```
|   Microsoft Windows system vulnerable to remote code execution (MS08-067)
|     State: VULNERABLE
|     IDs:  CVE:CVE-2008-4250
|           The Server service in Microsoft Windows 2000 SP4, XP SP2 and SP3, Server 2
|           Vista Gold and SP1, Server 2008, and 7 Pre-Beta allows remote attackers to
|           code via a crafted RPC request that triggers the overflow during path cano
|
|     Disclosure date: 2008-10-23
|     References:
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4250
|_      https://technet.microsoft.com/en-us/library/security/ms08-067.aspx
...
```

Listing 254 - Determining whether a host is vulnerable to the MS08_067 vulnerability

In this case, Nmap identifies that the specific SMB service is missing at least one critical patch for the MS08-067[212] vulnerability.

7.3.2.1 Exercises

1. Use Nmap to make a list of the SMB servers in the lab that are running Windows.
2. Use NSE scripts to scan these systems for SMB vulnerabilities.
3. Use nbtscan and enum4linux against these systems to identify the types of data you can obtain from different versions of Windows.

7.4 NFS Enumeration

Network File System (NFS)[213] is a distributed file system protocol originally developed by Sun Microsystems in 1984. It allows a user on a client computer to access files over a computer network as if they were on locally-mounted storage.

NFS is often used with UNIX operating systems and is predominantly insecure in its implementation. It can be somewhat difficult to set up securely, so it's not uncommon to find NFS shares open to the world. This is quite convenient for us as penetration testers, as we might be able to leverage them to collect sensitive information, escalate our privileges, and so forth.

7.4.1 Scanning for NFS Shares

Both *Portmapper*[214] and *RPCbind*[215] run on TCP port 111. RPCbind maps RPC services to the ports on which they listen. RPC processes notify rpcbind when they start, registering the ports they are listening on and the RPC program numbers they expect to serve.

The client system then contacts rpcbind on the server with a particular RPC program number. The rpcbind service redirects the client to the proper port number (often TCP port 2049) so it can

[212] (Microsoft, 2008), http://technet.microsoft.com/en-us/security/bulletin/ms08-067

[213] (Wikipedia, 2019), https://en.wikipedia.org/wiki/Network_File_System

[214] (Wikipedia, 2017), https://en.wikipedia.org/wiki/Portmap

[215] (Red Hat, 2019), https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/storage_administration_guide/s2-nfs-methodology-portmap

communicate with the requested service. We can scan these ports with **nmap** using the following syntax:

```
kali@kali:~$ nmap -v -p 111 10.11.1.1-254
```

Listing 255 - Using nmap to identify hosts that have portmapper/rpcbind running

We can use NSE scripts like **rpcinfo** to find services that may have registered with rpcbind:

```
kali@kali:~$ nmap -sV -p 111 --script=rpcinfo 10.11.1.1-254
...
Nmap scan report for 10.11.1.72
Host is up (0.0055s latency).

PORT    STATE SERVICE VERSION
111/tcp open  rpcbind 2-4 (RPC #100000)
| rpcinfo:
|   program version   port/proto  service
|   100000  2,3,4        111/tcp  rpcbind
|   100000  2,3,4        111/udp  rpcbind
|   100003  2,3,4       2049/tcp  nfs
|   100003  2,3,4       2049/udp  nfs
|   100005  1,2,3      50255/udp  mountd
|   100005  1,2,3      56911/tcp  mountd
|   100021  1,3,4      40160/udp  nlockmgr
|   100021  1,3,4      57765/tcp  nlockmgr
|   100024  1          34959/udp  status
|   100024  1          46908/tcp  status
|   100227  2,3         2049/tcp  nfs_acl
|_  100227  2,3         2049/udp  nfs_acl
...
```

Listing 256 - Querying rpcbind in order to get registered services

7.4.2 Nmap NFS NSE Scripts

Once we find NFS running, we can collect additional information, enumerate NFS services, and discover additional services using NSE scripts found in the /usr/share/nmap/scripts directory:

```
kali@kali:~$ ls -1 /usr/share/nmap/scripts/nfs*
/usr/share/nmap/scripts/nfs-ls.nse
/usr/share/nmap/scripts/nfs-showmount.nse
/usr/share/nmap/scripts/nfs-statfs.nse
```

Listing 257 - Locating various NSE scripts for NFS

We can run all three of these scripts using the wildcard character (*) in the script name:

```
kali@kali:~$ nmap -p 111 --script nfs* 10.11.1.72
...
Nmap scan report for 10.11.1.72

PORT    STATE SERVICE
111/tcp open  rpcbind
| nfs-showmount:
|_  /home 10.11.0.0/255.255.0.0
```

Listing 258 - Running all NSE scripts for NFS