Contents
Introduction ............................................................................................................................... 3
Lab 1 - Installing the Operating System (OS) ........................................................................ 3
Lab 2 – Working with TCPDump ............................................................................................ 4
Lab 3 - Installing Wireshark .................................................................................................. 7
Lab 4 - Working with Wireshark ............................................................................................ 7
   More With Wireshark ........................................................................................................ 9
       Normal Traffic (normal.pcapng) ................................................................................... 9
       Suspicious Traffic (suspicious.pcap) ............................................................................. 10
       Malicious Traffic (malicious.pcapng) ......................................................................... 11
Lab 5 - Packet Trace Exercise ................................................................................................ 13
Appendix A ............................................................................................................................ 14
Appendix B ............................................................................................................................ 17
Appendix C ............................................................................................................................ 22
Appendix D ............................................................................................................................ 23
Introduction

The current document provides an overview of the tools ‘tcpdump’ and ‘Wireshark’ and how to use them for network traffic analysis. The document follows the format of several lab exercises, including how to set up virtual machines for testing purposes. During these lab exercises, the installation process and the general usage of the network traffic analysis tools will be discussed. Screenshots of the tools in use are provided in several appendices to this document.

Lab 1 - Installing the Operating System (OS)

A topology of our network and pictures that provide proof of completion of this lab can be found in appendix A.

To properly perform network monitoring on our sandbox environment, a virtual machine running the FreeBSD OS version 9.1 was created. The VM runs on 2 processors with one core each and has 1GB of RAM available. It currently has one NIC because that is all that is needed for the moment, but since it’s a VM more NICS can be quickly installed if necessary. The other devices in our sandbox environment are:

- A VM running Windows 7
- A VM running Windows Server 2008
- A VM running Kali Linux

The Windows 7 and Windows Server 2008 VM’s have 1 virtual CPU and 4GB of memory allocated to them. The Kali Linux VM has 1 virtual processor and 2GB of memory allocated to it. All VMs in the sandbox are on the same virtual switch (NTS 350 Team 1) and the IP addressing scheme is as follows:

- FreeBSD: 192.240.60.10
- Windows Server 2008: 192.240.60.11
- Windows 7: 192.240.60.12
The virtual switch has the setting ‘promiscuous mode’ enabled so that it functions much like a hub; all traffic passing through the virtual switch is sent to all hosts. This allows our sensor to pick up on traffic that is not specifically addressed to its interface.

Lab 2 – Working with TCPDump

A topology of our network and pictures that provide proof of completion of this lab can be found in appendix B.

To capture network traffic in the sandbox environment, TCPdump was used on the sensor. Additionally, the network interface on the sensor was put in promiscuous mode so that it would capture all traffic, rather than traffic just addressed to the sensor. Since all VMs were placed on a separate network segment with no access to the Internet or other devices there were no security risks or legal and ethical concerns of capturing third party network traffic.

1. Running a simple ‘tcpdump -I em0’ command on the sensor showed that TCPdump was installed and functional on the sensor.
2. Next, a ping request was sent from the Windows 7 VM to the Windows Server 2008 VM. The sensor was able to pick up on this traffic.
3. Looking at the first captured echo request, the following information can be seen in the captured traffic:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>08:51:24.048783</td>
</tr>
<tr>
<td>Source IP</td>
<td>192.240.60.12</td>
</tr>
<tr>
<td>Source port</td>
<td>None (ICMP)</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Destination IP</td>
<td>192.240.60.11</td>
</tr>
<tr>
<td>Destination port</td>
<td>None (ICMP)</td>
</tr>
<tr>
<td>TCP flags</td>
<td>none</td>
</tr>
<tr>
<td>TCP sequence number</td>
<td>29</td>
</tr>
<tr>
<td>TCP last sequence number</td>
<td>none</td>
</tr>
<tr>
<td>TCP packet length</td>
<td>40</td>
</tr>
<tr>
<td>ACK flag (next expected)</td>
<td>none</td>
</tr>
<tr>
<td>Window size (host expects)</td>
<td>none</td>
</tr>
</tbody>
</table>

4. Tcpdump can also be used in combination with certain switches that customize its output. For instance:

a. Tcpdump will only display header information by default. It won’t display full packet content unless specified to write to an external file using the ‘-w’ switch. However, tcpdump can also be told to display link level header information by using the ‘-e’ switch.

b. The output of tcpdump can be limited to a set amount of packets by specifying the ‘-c’ switch following by a number.

c. The output of tcpdump can be redirected to a text file using the ‘>’ command, or the output redirector. This works the same as when using output redirector in any other Linux command and will write the output to a specified location (such as a text file) instead of STDOUT (usually the monitor).

d. Using the ‘-p’ switch in the tcpdump command will prevent placing the interface in promiscuous mode before starting the packet capture.
e. To capture all traffic for a specific port, the command ‘tcpdump port’ can be given, followed by a number for the port. If ‘port’ is preceded by either ‘src’ or ‘dst’ then tcpdump will only capture packets sent from or received by that port respectively.

f. To capture all traffic for a certain IP address, the command ‘tcpdump host’ can be given, followed by an IP address. Just like with port numbers, ‘host’ can be preceded by either ‘src’ or ‘dst’.

g. To pipe output of tcpdump to a pcap file, the switch ‘-w’ can be used, followed by the name of a file to write to. Tcpdump will now also write full packet content and not just headers.

h. To play back a previously captured saved data collection, the switch ‘-r’ can be used, followed by the name of the file to read from.

5. There is a wealth of information available on TCPDump. A great place to start is at www.tcpdump.org. Located there is version information, patches, bug reports, manual pages, and plenty of usage examples. Furthermore, here are a few other websites that offer great reference material for TCPDump:

   - http://media.packetlife.net/media/library/12/tcpdump.pdf
   - http://linux.about.com/library/cmd/blcmdl8_tcpdump.htm

6. The most important capability of tcpdump is its support of traffic filtering. Provide the syntax to capture just the data described in each of the following cases. Additionally document examples of each using data generated from your test network.

   a. IP traffic: Traffic can be limited to IP traffic by simply using the ‘ip’ operator
b. UDP traffic: Similar to IP traffic, a filter for UDP traffic can be set by using the ‘udp’ operator.

c. DNS traffic: Since DNS can work off TCP or UDP, this will be accomplished with help from the or[||] operator. For example ‘udp port 53 || tcp port 53’.

d. Traffic to or from a specific computer: Traffic can be filters for a specific computer by using the ‘host’ operator. For example ‘host [ip address || dns name]’

e. Traffic from but not to a specific computer: Use the host operator in conjunction with src. For example ‘src host [ip address || dns name]’

f. Traffic to or from a specific network: This is done with the ‘net’ operator followed by the network and subnet. For example ‘net 192.240.60.0/24’ or ‘net 192.240.60.0 mask 255.255.255.0’

g. IP broadcast traffic: Simply use ‘ip broadcast’

h. IP, ARP and RARP traffic, i.e., one filter that captures all at once: Simply use the ‘and[&&]’ operator between each protocol. For example ‘ip && arp && rarp’

Lab 3 - Installing Wireshark
A topology of our network and pictures that provide proof of completion of this lab can be found in appendix C. To analyze network traffic captured from the FreeBSD sensor, Wireshark running on Kali Linux is used in our sandbox environment. The choice for Kali Linux was made to provide a convenient method of moving pcap files from our FreeBSD system to our analyzer, namely through SSH. Also, Kali Linux comes with Wireshark installed by default.

Lab 4 - Working with Wireshark
This week all virtual machines running on UAT’s ESXi server cluster were unfortunately down, so no traffic from our test network could be captured. Instead, traffic from a virtual machine installed on a
different host was captured and examined. The VM OS was Backtrack Linux 5R3. Proof of completion of this lab can be found in appendix D.

<table>
<thead>
<tr>
<th>Requested traffic</th>
<th>Filter / procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ping</td>
<td>To view only ping traffic, ‘icmp.type == 0</td>
</tr>
<tr>
<td>ARP request</td>
<td>To view only ARP requests, ‘arp.opcode == 1’ was applied as a display filter. ‘Opcode 1’ specifies ARP requests.</td>
</tr>
<tr>
<td>TCP frame</td>
<td>To view only TCP frames, ‘tcp’ was applied as a display filter.</td>
</tr>
<tr>
<td>UDP frame</td>
<td>To view only UDP frames, ‘udp’ was applied as a display filter.</td>
</tr>
<tr>
<td>DHCP IP address negotiation</td>
<td>To view only DHCP traffic ‘bootp’ was applied as a display filter.</td>
</tr>
<tr>
<td>DNS request</td>
<td>To view only DNS traffic ‘dns’ was applied as a display filter.</td>
</tr>
<tr>
<td>HTTP get</td>
<td>To view only HTTP GET requests ‘http.request’ was applied as a display filter.</td>
</tr>
<tr>
<td>Logging into an FTP site</td>
<td>To view only FTP traffic, ‘ftp’ was applied as a display filter.</td>
</tr>
<tr>
<td>Logging into a web site</td>
<td>To view only website logins, ‘http.authorization’ was applied as a display filter. These days there are very little website login pages that don’t use SSL so there were no results for this filter.</td>
</tr>
<tr>
<td>Downloading a picture (Jpeg)</td>
<td>To view only downloaded JPEG files, ‘image-jfif’ was applied as a display filter.</td>
</tr>
<tr>
<td>Downloading a binary file</td>
<td>To view only downloads of binary files (.exe in this case), ‘http contains .exe’ was applied as a display filter.</td>
</tr>
<tr>
<td>Extracting the file from the capture</td>
<td>Wireshark has a feature that can be used to extract downloaded files. To use this, go to ‘File - Export objects - HTTP’ and select the object that needs to be downloaded. However, the binary file that was downloaded during the packet capture was not listed among the HTTP objects, so another method of extraction was attempted.</td>
</tr>
<tr>
<td></td>
<td>1. The GET request for the file was located</td>
</tr>
<tr>
<td></td>
<td>2. ‘Follow TCP stream’ was selected</td>
</tr>
<tr>
<td></td>
<td>3. Only the server side of the conversation was selected by filtering out the client side</td>
</tr>
<tr>
<td></td>
<td>4. The TCP stream was saved to a file (file.exe).</td>
</tr>
<tr>
<td></td>
<td>5. File.exe was opened with Bless hex editor</td>
</tr>
<tr>
<td></td>
<td>6. Since file.exe doesn’t just contain the downloaded file but also some file headers, the start of the file code was located by searching the hex for a file signature (MZ / 4D 5A h).</td>
</tr>
<tr>
<td></td>
<td>7. Everything from the file header to the end of file was copied and pasted into a new file (file2.exe)</td>
</tr>
<tr>
<td></td>
<td>8. File2.exe should match the original file. However, the MD5 hashes were different. Inspection of the files showed that the original file was actually larger in size than the extracted file. This is surprising, since everything from the file header to the end of the file was extracted. Therefore, all the file code should have been included.</td>
</tr>
<tr>
<td></td>
<td>9. Replication of this procedure kept providing the same results.</td>
</tr>
</tbody>
</table>
Research indicates that this procedure for file extraction should have worked, but unfortunately it did not. No suggestions as to why it did not work can be provided.

10. Replication of this procedure with a file extraction tool such as ‘foremost’ also did not successfully reproduce the original file.

<table>
<thead>
<tr>
<th>A particular source IP address</th>
<th>To view only a specific source IP, ‘ip.src == 69.89.27.206’ was applied as a display filter. This displays only communication from pendrivelinux.com.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A particular destination IP address</td>
<td>To view only a specific source IP, ‘ip.dst == 69.89.27.206’ was applied as a display filter. This displays only communication to pendrivelinux.com.</td>
</tr>
<tr>
<td>A particular MAC address</td>
<td>To view only a specific MAC address, ‘eth.addr == 00:0c:29:33:96:9’ was applied as a display filter.</td>
</tr>
<tr>
<td>HTTP authentication</td>
<td>To view only HTTP authentication traffic, ‘http.authbasic’ was applied as a display filter. Just as with ‘http.authorization’ this yielded no results since no websites could be found that don’t use SSL on their login page.</td>
</tr>
</tbody>
</table>

**More With Wireshark**

**Normal Traffic (normal.pcapng)**

The traffic capture ‘normal.pcapng’ appears to be normal network traffic. Analysis of the pcap file did not show any suspicious or malicious traffic. The following analysis was applied to the captured network frames:

**Protocol Hierarchy**

Looking at the protocol hierarchy of the captured network traffic showed nothing suspicious. There were no frames marked as ‘data’, which is what typically happens when data is sent to a known port but the data doesn’t appear to correspond to the protocol for that port. The network traffic appeared to be primarily HTTP traffic.

**Visual Inspection:**

A visual inspection of the captured network traffic doesn’t immediately show suspicious activity.

Suspicious network traffic can sometimes be visually identified by ‘stripes’ in the consecutively captured frames, or by an unusually large amount of traffic coming from one host, possibly to different ports of another host. No such activity seemed apparent in this captured network traffic.

**Filters:**
1. ‘dns’ - this filter only displays DNS traffic. Inspection of the DNS traffic showed that no suspicious websites were requested.

2. ‘ftp’ - this filter displays only FTP traffic. FTP traffic would have been suspicious, but no FTP traffic was found in this capture file.

3. ‘frame contains .exe’ - This filter displays only frames in which the string ‘.exe’ appears at some point. This would possibly reveal any transferred executables, which are not necessarily suspicious or malicious but they have a higher chance of being so. This captured network traffic did not display any frames containing ‘.exe’.

4. There are many other possible indicators of suspicious or malicious traffic and infinite filters that could be applied to reveal such traffic. Due to the constraint of finite time and the knowledge that this captured traffic in fact does not contain anything malicious or suspicious, no further filters were applied.

**Suspicious Traffic (suspicious.pcap)**

The traffic capture ‘suspicious.pcapng’ appears to contain suspicious network traffic. One particular host - 192.168.1.101 - tries to scan and connect to several ports and services on another host - 192.168.1.103. The following analysis was applied to the captured network frames:

**Protocol Hierarchy:**

The protocol hierarchy shows that the captured network traffic was primarily TCP traffic, although there was a considerable amount of SMB traffic as well. This in itself is not suspicious since SMB allows computers to connect to each other and share resources such as files and printers. However, because SMB allows this sharing of resources it is also often used as a gateway for remote exploitation.
**Visual Inspection:**
A visual inspection of the captured network traffic shows that right from the start, host 192.168.1.101 tries to connect to host 192.168.1.103 over SMB several times, failing each time due to the user id being incorrect (CHADWICK\Administrator). After several unsuccessful attempts to connect, host 192.168.1.101 then initiates a port scan of host 192.168.1.103. Almost every attempt is responded to with a [RST, ACK] packet, indicating that the port is not open. This generates a striped pattern in the network traffic that usually indicates that something suspicious is going.

**Filters:**
1. ‘smb’ - this filter only displays SMB traffic. This made it easier to determine that no successful SMB connection attempts had been made. A successful connection attempt after so many failed ones could indicate a successful exploit.
2. ‘smb.error.code==0x005b’ - this filter only displays frames in which a host replies with a ‘bad userid’ error code to an SMB connection attempt. This filter revealed that 13 unsuccessful SMB connection attempts had been made.
3. ‘tcp.flags == 0x0012’ - this filter only displays [SYN, ACK] frames. This revealed that out of the scanned ports, 192.168.1.103 had the following ports open: 135, 139, and 445.

**Malicious Traffic (malicious.pcapng)**
The traffic capture ‘malicious.pcapng’ appears to contain malicious network traffic. One host - 10.234.125.254 - tries multiple times to connect over FTP to host 10.121.70.151. The following analysis was applied to the captured frames:

**Protocol Hierarchy:**
The protocol hierarchy revealed that the captured frames are 100% TCP frames, and out of those TCP frames 35.67% were FTP frames. No other protocols were detected.
**Visual Inspection:**

A visual inspection of the captured traffic doesn’t immediately show that something might be wrong because in the default Wireshark profile, all FTP traffic is colored in some shade of gray. There is no ‘striped’ pattern that shows anything might be off. However, a closer inspection does show that there are many ‘530 Login Incorrect’ frames sent, indicating that a user is trying multiple times to log in with a wrong password.

Applying a coloring rule to all incorrect login attempts did reveal more of a striped pattern to indicate that something might be wrong.

**Filters:**

1. ‘ftp’ - this filter only displays frames in which the FTP protocol is used. This gets rid of all the TCP frames in which a connection is established between the hosts.

2. ‘ip.src==10.234.125.254 && ftp’ - this filter only displays frames using the FTP protocol and originating from the ‘attacker’. This revealed a list of frames in which every single frame represented a request to connect with a different username or password, clearly indicating a brute force attack.

3. ftp.response.arg == "Login incorrect." - this filter displays only the packets from the server back to the client saying ‘login incorrect’. At the bottom of the Wireshark window is indicated how many packets are displayed under the current display filter. 1410 incorrect login attempts were made.

4. ‘ip.src==10.121.70.151 && ftp’ - This filter only displays frames over the FTP protocol originating from the FTP server. Scrolling to the bottom of these frames revealed that the brute force attack was unsuccessful, or possibly that the end of the attack had not been captured. The last frame in the list under this filter was an ‘incorrect login’ message to the attacker.
Lab 5 - Packet Trace Exercise

The following is an analysis of the network traffic capture file ‘FTP Bruteforce.pcap’.

Here a brute force is being attempted by 10.234.125.254 on the FTP server 10.121.70.151. The packet capture shows that the attacker is continuously connecting to the FTP server and requesting to become admin. The server responds by telling the attacker “Password is required for admin”. The attacker then sends a password attempt. Finally, the FTP server responds saying “Login incorrect.” This process is repeated over and over but the attacker never successfully logs in as administrator.

This scenario is problematic because with the proper password dictionary the attacker would eventually successfully login as administrator. Once accomplished, they may be able to gain access to sensitive information and/or gain access to other devices on the network. To mitigate this, a login policy should be implemented to lock the account for a short period, perhaps 20 minutes, after a given number of attempts, perhaps 10. Using this policy would severely inhibit the attacker’s ability to brute force the server, and a lockout policy of 10 attempts would give legitimate users enough attempts to login without heavily increasing administrator overhead. Lastly, SFTP should be implemented instead of FTP if possible because it encrypts communications. This would thwart an eavesdropper’s ability to intercept communications.
Appendix A

Figure 1. Sandbox logical topology

Figure 2. Overview of FreeBSD VM
### Figure 3. Overview of Windows Server 2008 VM

<table>
<thead>
<tr>
<th>General</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS: Microsoft Windows Server ...</td>
<td>Consumed Host: 18 MHz</td>
</tr>
<tr>
<td>VM Version: 8</td>
<td>Consumed Host: 4067.00 MB</td>
</tr>
<tr>
<td>CPU: 1 vCPU</td>
<td>Active Guest: 122.00 MB</td>
</tr>
<tr>
<td>Memory: 4096 MB</td>
<td>Provisioned: 44.11 GB</td>
</tr>
<tr>
<td>Memory Overhead: 48.01 MB</td>
<td>Not-shared Storage: 27.39 GB</td>
</tr>
<tr>
<td>VMware Tools: Running (Current)</td>
<td>Used Storage: 27.39 GB</td>
</tr>
<tr>
<td>IP Addresses: 192.240.60.11</td>
<td></td>
</tr>
<tr>
<td>DNS Name: WIN08SRV</td>
<td>Storage</td>
</tr>
<tr>
<td>EVC Mode: N/A</td>
<td>Status:</td>
</tr>
<tr>
<td>State: Powered On</td>
<td>Drive Type:</td>
</tr>
<tr>
<td>Host: 192.168.10.30</td>
<td>NX3100-5 (Non-SSD)</td>
</tr>
<tr>
<td>Active Tasks:</td>
<td>Refresh Storage Use</td>
</tr>
<tr>
<td>Record/Replay Status:</td>
<td>N/A</td>
</tr>
<tr>
<td>vSphere HA Protection:</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Figure 4. Overview of Windows 7 VM

<table>
<thead>
<tr>
<th>General</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS: Microsoft Windows 7 (64-bit)</td>
<td>Consumed Host: 0 MHz</td>
</tr>
<tr>
<td>VM Version: 8</td>
<td>Consumed Host: 3116.00 MB</td>
</tr>
<tr>
<td>CPU: 1 vCPU</td>
<td>Active Guest: 122.00 MB</td>
</tr>
<tr>
<td>Memory: 4096 MB</td>
<td>Provisioned: 36.10 GB</td>
</tr>
<tr>
<td>Memory Overhead: 48.00 MB</td>
<td>Not-shared Storage: 17.76 GB</td>
</tr>
<tr>
<td>VMware Tools: Not running (Current)</td>
<td>Used Storage: 17.76 GB</td>
</tr>
<tr>
<td>IP Addresses:</td>
<td>Storage</td>
</tr>
<tr>
<td>DNS Name: N/A</td>
<td>Status:</td>
</tr>
<tr>
<td>EVC Mode: N/A</td>
<td>Drive Type:</td>
</tr>
<tr>
<td>State: Powered On</td>
<td>NX3100-5 (Non-SSD)</td>
</tr>
<tr>
<td>Host: 192.168.10.30</td>
<td>Refresh Storage Use</td>
</tr>
<tr>
<td>Active Tasks:</td>
<td>N/A</td>
</tr>
<tr>
<td>vSphere HA Protection:</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### General
- **Guest OS:** Other 2.6.x Linux (32-bit)
- **VM Version:** vmx-09
- **CPU:** 1 vCPU
- **Memory:** 2048 MB
- **Memory Overhead:** 30.82 MB
- **VMware Tools:** Running (Current)
- **IP Addresses:** 192.240.60.17
- **DNS Name:** kali
- **EVC Mode:** N/A
- **State:** Powered On
- **Host:** 192.168.10.30
- **Active Tasks:**

### Resources
- **Consumed Host:** 0 MHz
- **Consumed Host:** 472.00 MB
- **Active Guest:** 40.00 MB
- **Provisioned:** 32.09 GB
- **Not-shared Storage:** 32.09 GB
- **Used Storage:** 32.09 GB

#### Storage
- **NX3100-6 (NI...** Normal Non-SSD

#### Network
- **Type:** Standard port group

---

**Figure 5. Overview of Kali Linux VM**
Appendix B

Figure 6. Captured ICMP traffic from Windows 7 to Windows Server 2008

Figure 7. Having tcpdump display link level headers

Figure 8. Having tcpdump capture a specific number of packets
Figure 9. Having tcpdump output to a text file

```
root@GerbenTJ:/home/gerben/ctactivity # tcpdump -i eth0 -c 5 > capture.txt
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
5 packets captured
6 packets received by filter
8 packets dropped by kernel
```

Figure 10. Putting an interface out of promiscuous mode. Here we see tcpdump only pick up on broadcast traffic (ARP)

```
root@GerbenTJ:/home/gerben/ctactivity # tcpdump -i eth0 -p
```

Figure 11. Having tcpdump only pick up traffic going to or coming from port 80

```
root@GerbenTJ:/home/gerben/ctactivity # tcpdump -i eth0 -p
```

```
09:31:25.867466 ARP, Request who-has 192.234.60.11 tell 192.234.60.12, length 46
09:31:25.867719 ARP, Request who-has 192.234.60.11 tell 192.234.60.12, length 46
```
root@GerbenTJ:/home/gerbentj/classactivity # tcpdump -i eth0 host 192.240.60.11
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
09:40:03.213914 IP 192.240.60.12.49165 > 192.240.60.11.http: Flags [P.], seq 171
9799453:1717937971, ack 2045756805, win 62951, length 330
09:40:03.214721 IP 192.240.60.11.http > 192.240.60.12.49165: Flags [P.], seq 1:1
98, ack 338, win 64240, length 188
09:40:03.221119 IP 192.240.60.12.49165 > 192.240.60.11.http: Flags [P.], seq 338
:732, ack 189, win 64240, length 394
09:40:03.221428 IP 192.240.60.11.http > 192.240.60.12.49165: Flags [P.], seq 189
:777, ack 772, win 63866, length 188
09:40:03.425203 IP 192.240.60.12.49165 > 192.240.60.11.http: Flags [P.], ack 377,
win 64852, length 0
09:40:03.480858 ARP, Request who-has 192.240.60.11 (00:50:56:95:0c:40 (oui Unknown))
tell 192.240.60.12, length 46
09:40:03.4808747 ARP, Reply 192.240.60.11 is-at 00:50:56:95:0c:40 (oui Unknown),
length 46
7 packets captured
9 packets received by filter
0 packets dropped by kernel

Figure 12. Having tcpdump only pick up traffic coming from or going to 192.240.60.11

root@GerbenTJ:/home/gerbentj/classactivity # tcpdump -i eth0 -w capture.pcap host
192.240.60.11
Listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
16 packets captured
10 packets received by filter
0 packets dropped by kernel

Figure 13. Having tcpdump write captured packets to a pcap file

root@GerbenTJ:/home/gerbentj/classactivity # tcpdump -r capture.pcap
Reading from file capture.pcap, link-type EN10MB (Ethernet)
09:52:38.973655 IP 192.240.60.12.49169 > 192.240.60.11.http: Flags [P.], seq 123
4099435:1234094773, ack 625341936, win 32756, length 398
09:52:38.973595 IP 192.240.60.11.http > 192.240.60.12.49169: Flags [P.], seq 1:1
99, ack 338, win 253, length 188
09:52:38.973647 IP 192.240.60.12.49169 > 192.240.60.11.http: Flags [P.], seq 338
:732, ack 189, win 32709, length 394
09:52:38.988154 IP 192.240.60.11.http > 192.240.60.12.49169: Flags [P.], seq 189
:732, ack 338, win 252, length 188
09:52:39.177124 IP 192.240.60.12.49169 > 192.240.60.11.http: Flags [P.], ack 377,
win 32662, length 0

Figure 14. Having tcpdump read from a previously captured pcap file
Figure 15. Filtering for IP traffic

root@GerbenTJ:/home/gerbenj/week6 # tcpdump -i eth0 -c 4 -nn ip
tcpdump: verbose output suppressed, use -v or --verbose for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:21:22.466277 IP 192.240.60.12.49175 > 192.240.60.11.88: Flags [S], seq 288331
    ttl 128, win 0x192, options [mss 1468,nop,wscale 2,nop,nop,sackOK], length 0
22:21:22.466580 IP 192.240.60.11.88 > 192.240.60.12.49175: Flags [S], seq 28276
    ack 200331,3214, win 0x192, options [mss 1468,nop,wscale 0,nop,nop,sackOK], length 0
22:21:22.466855 IP 192.240.60.12.49175 > 192.240.60.11.88: Flags [s], ack 1, win 32050, length 32050
    ack 1, win 32050, length 339
4 packets received by filter
4 packets dropped by kernel

root@GerbenTJ:/home/gerbenj/week6 #

Figure 16. Filtering for UDP traffic

root@GerbenTJ:/home/gerbenj/week6 # tcpdump -i eth0 -c 4 -nn udp
tcpdump: verbose output suppressed, use -v or --verbose for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:23:43.427558 IP 192.240.60.12.137 > 192.240.60.255.137: NBT UDP PACKET(137):
    QUERY; REQUEST; BROADCAST
22:23:43.427558 IP 192.240.60.12.137 > 192.240.60.255.137: NBT UDP PACKET(137):
    QUERY; REQUEST; BROADCAST
22:23:44.191591 IP 192.240.60.12.137 > 192.240.60.255.137: NBT UDP PACKET(137):
    QUERY; REQUEST; BROADCAST
22:23:44.191591 IP 192.240.60.12.137 > 192.240.60.255.137: NBT UDP PACKET(137):
    QUERY; REQUEST; BROADCAST

Figure 17. Filtering for traffic to or from Specific Host

root@GerbenTJ:/home/gerbenj/week6 # tcpdump -i eth0 -c 4 -nn host 192.240.60.12
tcpdump: verbose output suppressed, use -v or --verbose for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:27:01.694158 ARP, Request who-has 192.240.60.12 tell 192.240.60.12, length 46
22:27:01.694491 IP 192.240.60.12 > 192.240.60.12: ICMP echo request, id 1, seq 5
    length 80
22:27:01.694520 IP 192.240.60.12 > 192.240.60.12: ICMP echo reply, id 1, seq 5
    length 80

Figure 18. Filtering for traffic from Specific Host

root@GerbenTJ:/home/gerbenj/week6 # tcpdump -i eth0 -c 4 -nn src host 192.240.60
    .12
tcpdump: verbose output suppressed, use -v or --verbose for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:25:25.699217 ARP, Request who-has 192.240.60.12 tell 192.240.60.12, length 46
22:25:26.699403 IP 192.240.60.12 > 192.240.60.12: ICMP echo request, id 1, seq 5
    length 80
22:25:27.713361 IP 192.240.60.12 > 192.240.60.12: ICMP echo request, id 1, seq 5
    length 80
Figure 19. Filtering for traffic to or from a Network

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:28:55.260860 ARP, Request who-has 192.240.60.11 tell 192.240.60.12, length 46
22:28:55.269994 ARP, Reply 192.240.60.11 is-at 00:50:56:95:0c:40, length 46
22:28:55.270056 IP 192.240.60.12.49177 > 192.240.60.11.100: Flags [S] vs [0], seq 160309
0933, win 8192, options [mss 1460,mscale 2,nop,nop,sackOK1], length 0
22:28:55.270267 ARP, Request who-has 192.240.60.12 tell 192.240.60.11, length 46
```

Figure 20. Filtering for IP Broadcast Traffic

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:35:17.789885 IP 192.240.60.12 > 192.240.60.11: ICMP echo request, id 8, seq 0,
2, length 40
22:35:17.790248 IP 192.240.60.11 > 192.240.60.12: ICMP echo reply, id 1, seq 82,
length 40
22:35:18.782778 IP 192.240.60.12 > 192.240.60.11: ICMP echo request, id 1, seq 83,
3, length 40
22:35:18.782931 IP 192.240.60.11 > 192.240.60.12: ICMP echo reply, id 1, seq 83,
length 40
```

Figure 21. Filtering for ARP, IP, and RARP
Appendix C

Figure 22. Sandbox environment topology

Figure 223. Specifications of Kali Linux VM
Appendix D

Figure 24. Display filter and results to see just network ping traffic

Figure 25. Display filter and results to see just ARP requests
Figure 26. Display filter and results to see just TCP frames

Figure 237. Display filter and results to see just UDP frames
**Figure 248.** Display filter and results to see just DHCP traffic

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>192.168.174.125</td>
<td>192.168.174.254</td>
<td>DHCP</td>
<td>342</td>
<td>Transaction ID 0x63de8b15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.174.125</td>
<td>192.168.174.254</td>
<td>DHCP</td>
<td>342</td>
<td>Transaction ID 0x63de8b15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.174.125</td>
<td>192.168.174.254</td>
<td>DHCP</td>
<td>342</td>
<td>Transaction ID 0x63de8b15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.174.125</td>
<td>192.168.174.254</td>
<td>DHCP</td>
<td>342</td>
<td>Transaction ID 0x63de8b15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192.168.174.125</td>
<td>192.168.174.254</td>
<td>DHCP</td>
<td>342</td>
<td>Transaction ID 0x63de8b15</td>
</tr>
</tbody>
</table>

**Figure 259.** Display filter and results to see just DNS traffic

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
</table>
| 190 | 17:37:11 | 192.168.174.125     | 192.168.174.2      | DNS      | 89     | Standard query response 0xbbe A safesurfing-cache.google
| 194 | 17:37:11 | 192.168.174.125     | 192.168.174.2      | DNS      | 77     | Standard query response 0x37a A ocsp.verisign.com
| 196 | 17:37:11 | 192.168.174.125     | 192.168.174.2      | DNS      | 86     | Standard query response 0x37a A CNAME ocsp.verisign.com
| 197 | 17:37:11 | 192.168.174.125     | 192.168.174.2      | DNS      | 133    | Standard query response 0x37a A CNAME ocsp.verisign.com
| 199 | 17:37:11 | 192.168.174.125     | 192.168.174.2      | DNS      | 92     | Standard query response 0x37a A CNAME ocsp.verisign.com

Fragment offset: 9
Time to live: 128
Protocol: UDP (17)
Header checksum: 0x5736 [correct]
Source: 192.168.174.2 (192.168.174.2)
[Source GeoIP: Unknown]
Figure 30. Display filter and results to see just HTTP GET requests

Figure 31. Display filter and results to see just FTP traffic
Figure 32. Display filter and results to see just website logins

Figure 263. Display filter and results to see just downloaded JPEG files
Figure 274. Display filter and results to see just downloaded binary (.exe) files.

Figure 285. Following TCP stream to see only frames relevant to the download.
Figure 296. Start of file transfer, as identified by the file header (MZ)

```
root@bt-:/Desktop# md5deep YUMI-0.0.9.9.exe
ba9ef21869fc2f675726db9b57b10bf /root/Desktop/YUMI-0.0.9.9.exe
root@bt-:/Desktop# md5deep file.exe
cc7b9987bf100d8c5c87c9e038e000d /root/Desktop/file.exe
root@bt-:/Desktop#
```

Figure 307. Comparison of MD5 hash values shows that files are not identical

Figure 318. Opening extracted file in Bless hex editor shows that more than just the file is extracted.
Figure 329. Everything between file header and end of file was copied and pasted into a new file (file2.exe). File should now be identical to original.

```
root@bt:~:/Desktop# file file2.exe
file2.exe: PE32 executable for MS Windows (GUI) Intel 80386 32-bit
root@bt:~:/Desktop# md5deep YUMI-0.0.9.9.exe
            ba9ef21809f5cf26d75276db9b57b10bf /root/Desktop/YUMI-0.0.9.9.exe
root@bt:~:/Desktop# md5deep file2.exe
           15eef6e6a031a5774672605579067 /root/Desktop/file2.exe
root@bt:~:/Desktop# ls -l
total 15720
-rw-r--r-- 1 root root  1876 2013-06-29 09:09 Carve_Script.sh
drwxr-xr-x 3 root root 4096 2013-05-22 10:56 CFR412
-rw-r--r-- 1 root root 1057702 2013-07-05 11:01 file2.exe
-rw-r--r-- 1 root root 1092466 2013-07-05 10:57 file.exe
-rw-r--r-- 1 root root  6193 2013-07-05 09:32 images.jpeg
drwxr-xr-x 2 root root 4096 2013-06-29 20:17 my_programs
drwxr-xr-x 5 root root 4096 2013-04-30 14:09 NTS330 final
-rw------- 1 root root 12715980 2013-07-05 09:39 NTS 350 unit 8
drwxr-xr-x 4 root root 4096 2013-07-02 19:34 NTS370
-rw-r--r-- 1 root root 1191053 2013-07-05 09:31 YUMI-0.0.9.9.exe
root@bt:~:/Desktop#
```

Figure 40. Linux recognizes extracted file as a Windows executable. However, extracted file is still not identical to original. Original file is actually larger than extracted file.
Figure 41. Display filter and results to see only traffic from a particular source IP address

Figure 42. Display filter and results to see only traffic from a particular destination IP address

Figure 333. Display filter and results to see only traffic from a particular MAC address
Figure 344. Display filter and results to see only HTTP authentication traffic