Security Operations & Incident Management

Lab: Filtering Network Traffic with tcpdump

1. Overview

The aim of this session is to familiarize you to the main concepts of filtering network traffic with tcpdump. This is important when you need to distinguish relevant traffic for further inspection.

By the end of this practical, you should be able to:

- Use relevant tcpdump command line options, with the use of a reference
- Dissect raw packets to determine protocol headers; IP header length, IP options, embedded protocol, IP datagram length, TCP header length, TCP options, fragmented traffic, etc.
- Write BPF filters to identify packets with relevant traffic
- Identify packets with IP options, TCP options.
- Identify signs of IDS insertion/evasion, by recognizing: Fragmentation overlap, Low Time To Live value, TCP Sequence Number overlap, Data on SYN, DF flag.

General instructions:

Please use one of the following links on tcpdump as reference for this practical, especially with regards to its command line options:

http://www.tcpdump.org/tcpdump_man.html

For a quick reference, you can find a tcpdump cheat sheet at:

http://packetlife.net/media/library/12/tcpdump.pdf

In order to run this practical, you need to open the virtual machine '**Ubuntu16045-base**' with VMWare Workstation. Use the following credentials to login:

Username: student
Password: student

Please open a terminal and type the following command, in order to change directory:

```
cd  Desktop/data/5-filtering-nw-traffic/
```

2. *Introduction to tcpdump*

**********Use the tcpdump binary file **INTRO** as input for this exercise. **********

Basic tcpdump command line options we will use:

-r	Read a tcpdump file
-n	Disable hostname resolution. This option is very important and will allow tcpdump to run a lot faster. Remember to include it every time you run tcpdump in the lab.
-x	Display the data of each packet in hex
-c <num>	Exit after reading <num> packets
-v	Display packets in verbose output. This will include fields, such as time to live, identification, total length and options in an IP packet.
-vv	Display packets in even more verbose output. Additional fields are included.
-vvv	Display packets in even more verbose output. This option will display most packet header fields.
-e	Display Link layer header, including MAC Addresses

In order to display the contents of the INTRO file in hex, you can run the following command in a terminal:

Tcpdump example: **`tcpdump -nxr INTRO`**

Exercise 1:

Specifically, answer the following questions pertaining to the hexadecimal output that has been displayed:

1) What records, if any, have **IP options**. How do you know this?
   Hint: You must use the tcpdump command line option to dump this record in hexadecimal.
   Hint: Use the IP header specification in Appendix A (or the TCP/IP Pocket Reference Guide) to find the location of the IP header length in the IP datagram.
   Hint: The IP header length is found in the 0 Byte offset (first Byte) of the IP header. Specifically, it is found in bits 4-7 (or low-order nibble) of the first Byte. Remember, each hex character that you see displayed in the hexadecimal output represents 4 bits.
   Hint: For an IP datagram with no options, the value found in the IP header length will be 5 (5 32-bit words). When there are IP options, the value found in the IP header length will be greater than 5 since the IP options are placed at the end of the IP header.
2) What is the **IP header length**, in decimal Bytes, of the first record?
   Hint: We already discovered where the IP header length was located in the hexadecimal output in the previous question. Remember that the value found in this field represents a 32-bit word, or 4 Bytes. So, you must multiply the value that you find in the IP header length by 4.
   Hint: The value found in the IP header is in hexadecimal. A hexadecimal value of "f" translates to a decimal 15.
3) What is the entire **IP datagram length**, in decimal Bytes, of the first record?
   Hint: You must use the tcpdump command line option to dump this record in hexadecimal.
   Hint: Use the layout of the IP header to find the location of the 16-bit total length in the IP datagram. After you dump the record in hex, you will see the IP datagram beginning with the IP header portion.
   Hint: The total length is located in the second and third Bytes offset of the IP datagram.

Starting at the beginning of the IP datagram, the 0 Byte offset will be represented by the first two hex characters, the 1 Byte offset by the next two hex characters, and then after the space, you will see the total datagram length. It is two Bytes long, making it the next four hex characters.

Hint: This value is in hexadecimal. You must convert it to the decimal system. E.g. 0 0 5 0 in base 16 corresponds to the decimal value of 80 ($0 * 16^3 + 0*16^2 + 5 * 16^1 + 0 * 16^0$=80).

4) Does the second record have **TCP options**? How do you know this?

Hint: You can use the tcpdump command line option to dump this record in hexadecimal.

Hint: You must locate the TCP header length to answer this question.

Hint: The TCP header length is found in the TCP header. The TCP header is found just after the IP header. So, figure out how long the IP header length is using the knowledge you have gained from the previous questions.

Hint: Once you have found where the IP header stops, you know that the TCP header starts immediately after that. The TCP header length is 12 Bytes offset from the beginning of the TCP header with counting starting at 0. Specifically, it is in the high-order (or leftmost) hex character in the 12th Byte offset.

Hint: The TCP header length is computed just like the IP header length. It is expressed in 32-bit words too. A value of more than 5 in this field means that there are TCP options.

5) What is the **source MAC address** of the last record?

Hint: You need to use a new command line option to display the MAC addresses in the Ethernet frame header.

Hint: The MAC addresses follow the time stamp and are the numbers with embedded colons(:). The source MAC address is the first MAC address that you see. The destination MAC address is the second MAC address displayed.

Hint: Use the –e option for tcpdump to display the MAC addresses of the Ethernet frame header.

6) If there are any **fragments** in the data, what is the decimal value of the fragment offset field found in the hex record (**not** the tcpdump translation of this value)?

Hint: You can use the tcpdump command line option to dump this record in hexadecimal.

Hint: Locate the 4th record, with the string "IP 127.0.0.1 > 127.0.0.1: tcp" in the standard output.

Hint: You must locate the 13-bit fragment offset in the IP header to answer this question.

Hint: The fragment offset is found in the 6th and 7th Bytes offset of the IP header.

It is in the low-order 13 bits. But, don't worry about that for now since all characters in this field except the rightmost are 0 anyway.

Hint: The value found in the fragment offset has to be multiplied by 8 in order to convert to Bytes. If you have found the correct field and multiplied by 8, then you can verify your answer by using the command line option –v and noting the offset value of the 4th record.

7) What is the **embedded protocol** found in each of the records?

Hint: You can use the tcpdump command line option to dump this record in hexadecimal.

Hint: Use the layout of the IP header to find the location of the 8-bit protocol field in the IP datagram.

Hint: The embedded protocol field is found in the 9th Byte offset of the IP header.

8) What are the **ICMP type** and **code**, in decimal, for an ICMP "ip reassembly exceeded error"?

Hint: You can use the tcpdump command line option to dump this record in hexadecimal.

Hint: You must figure out the IP header length in order to know where the IP header stops and the ICMP header begins.

Hint: Once you have discovered where the ICMP header starts, locate where the ICMP type and code are located in the header by looking at the ICMP header layout.

Hint: The ICMP type is in the 0 Byte offset of the ICMP header and the ICMP code is found in

the 1 Byte offset of the ICMP header. The ICMP type will need to be converted from hex to decimal.

3. *Writing BPF filters*

********** Use the tcpdump binary file **FILTERS** as input for this exercise. **********

A filter can be specified on the command line for Unix by placing the filter between single quotes; this is done so that whatever Unix shell you are using doesn't try to parse or interpret the filter.

As a guide for doing this first exercise, here is the correct format for the tcpdump command if you were to search for records from input file FILTERS that have the SYN flag set.

tcpdump example: tcpdump –r FILTERS –n 'tcp[13] & 0x02 != 0'

Use the TCP/IP header layouts found in the pocket reference guide, or in the final pages of this document to discover where the fields used in the exercises are located. Remember to include spaces on either side of the comparison operator (=, !=). Don't put a space between the !=.

Exercise 1:

Use the example tcpdump commands to guide you in reading records from the input file **FILTERS**. Write a tcpdump filter to display those records with the **acknowledgement** flag set. The layout of the TCP flag Byte has been supplied below for assistance in figuring out the filter mask value.

```
2³      2²     2¹   2⁰    2³      2²     2¹   2⁰
l---------------------||-----------------------l
l CWR | ECE | URG | ACK| PSH | RST | SYN | FIN l 13th Byte offset of TCP
l---------------------||-----------------------l header
```

Hint: You must find a mask Byte that will zero out all bits except the ACK bit.
Hint: The filter format will be tcp[13] & 0x?? != 0, where ?? is the mask Byte.
Hint: The mask Byte will be the following in binary: 0001 0000 – convert that to hexadecimal.
Filter help: `tcp[13] & 0x10 != 0`
Record 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:

Read records from the input file using tcpdump and write a filter to display those records with either the **RST** or **ACK** flags 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.

```
2³      2²     2¹   2⁰    2³      2²     2¹   2⁰
l---------------------||-----------------------l
l CWR | ECE | URG | ACK| PSH | RST | SYN | FIN l 13th Byte offset of TCP
l---------------------||-----------------------l 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 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: `tcp[13] & 0x14 != 0`
Record 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.

Exercise 3:

Read records from the input using tcpdump and write a filter to display those records with exactly the **PUSH** and **FIN** flags set. What this means is that you must compare the TCP flag Byte with an exact value; do not use masking. For instance, if you were to check that both the ACK and RST values were set and no other flags were set, the filter would be tcp[13] = 0x14.

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

Hint: You must use a different format this time. You will not have to use mask bits. In this case, no other bits may be set so we can simply test if the TCP flag Byte equals a value.
Hint: The filter format will be tcp[13] = ??, where ?? is the value that will be either the decimal or hexadecimal representation of both the PUSH and FIN bits set simultaneously.
Hint: The value that will show that the PUSH and FIN bits are set is 0000 1001 – compute the decimal or the hexadecimal value of this binary number.
Filter help: `tcp[13] = 09`
Record 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 **FP** in the flags field and no other flags set in the tcpdump record.

Exercise 4:

Read records from the input file using tcpdump and write a filter to display those records with the **DF** flag set. You must find where the DF flag is located in the IP header. Then, count the number of Bytes displacement into the header where the Byte is found. Finally, mask out all bits in the Byte except the DF flag.

```
2^3      2^2     2^1   2^0    2^3      2^2     2^1   2^0
l---------------------||-----------------------l
l RES | DF  | MF | 0F |  0F  |  0F |  0F  | 0F l 16th Byte offset of IP
l---------------------||-----------------------l header
```

Hint: You must find a mask Byte that will zero out all bits except the DF bit.
Hint: The filter format will be ip[6] & 0x?? != 0, where ?? is the mask Byte.
Hint: The mask Byte will be the following in binary: 0100 0000 – convert that to hexadecimal.
Filter help: `ip[6] & 0x40 != 0`
Record 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.

4. Examination of Datagram fields

Scenario: You are defending your network 10.10.x.x against a known archenemy (signature IP address of 1.1.1.1). You have chosen not to block their evilness since you want to see what kind of wily activity they are sending your way. In this exercise, you will examine traffic from source IP

address of 1.1.1.1 for signs of attempted elusive behavior against your Network Intrusion Detection System (NIDS). Your NIDS sits on a chokepoint to your network before any packet filtering. Additionally, the target IPs that the hacker has selected on your network are two hops away from the NIDS on a network segment that has a smaller MTU than the segment on which the NIDS resides.

Description: Read all records from a tcpdump file from source IP 1.1.1.1 and dump the output in hexadecimal for examination.

********** Use the tcpdump binary file **DG1** as input for this exercise. **********

Each of the records in the input file contains some kind of evidence of crafting that may be used to elude notice of a NIDS.
Categories or fields of crafting to examine for this exercise are:

- Fragmentation overlap
- Low time to live value
- TCP sequence number overlap
- Data on SYN
- Don't fragment flag

Exercise 1:

Description: Look at the third record. It has no "overt" signatures of crafting. However, what will keep this record from being delivered to the target host knowing that it resides on a network with a smaller **MTU** of 576 while the NIDS resides on a link with an MTU of 1500?

Hint: The tcpdump command is: `tcpdump –nxr DG1 –c 3`
Hint: This has to do with fragmentation.
Hint: What will prevent this record from being fragmented?

Exercise 2:

Description: Use the –S tcpdump command line option to display TCP sequence numbers in absolute numbers (not the relative numbers as shown by default). Look at the fourth and fifth records. Concentrate your inspection on the **TCP sequence numbers**. What appears to be wrong with them? What possible elusive behavior might this be attempting?

Hint: The tcpdump command is: tcpdump –Snxr DG1
Hint: You will see a beginning sequence number, separated by a colon and then an ending sequence number, followed by the number of Bytes exchanged delimited by parentheses such as **2930007158: 2930007159**(1).
Hint: Each TCP segment in a connection should have sequential Byte orders. In other words, the segment with the ending TCP sequence number of 2930007159 should be followed by a segment with a beginning TCP sequence number of 2930007159.

Exercise 3:

Description: Look at the sixth record. What is unusual about this **SYN** attempt?

Hint: The tcpdump command is: `tcpdump –nxr DG1 –c 6`
Hint: How many Bytes of data are normally sent on a SYN connection?
Hint: The data Bytes can be found after the TCP sequence number delimited by parentheses.

Exercise 4:

Description: Examine the seventh record for crafting. There is an additional piece of information included in the standard tcpdump output that you typically don't see. This is the **time to live** value. Knowing what you do about the topology of the target network from reading the introduction of this exercise, what will happen to this record after passing by the NIDS?

Hint: The tcpdump command is: `tcpdump -nvxr DG1 -c 7`

Exercise 5:

Description: Examine the final two records. What is wrong with the **fragments** that you see? What might the hostile attacker be attempting with this behavior?

Hint: The tcpdump command is: `tcpdump -nvxr DG1`

Hint: Examine the fragment ID's, the fragment offsets, and the fragment lengths.

Example: IP (tos 0x0, ttl 54, **id 1234**, **offset 0**, flags **[+],** proto TCP (6), **length 1480**)
1234 = fragment ID
0 = fragment offset
1480 = fragment length
[+] = More Fragments to follow (MF flag is set)

Appendix A – Protocol Headers

IP Header

0 15 16 31

4-bit version	4-bit header length	8-bit ToS	16-bit total length (in Bytes)				
16-bit IP identification number			R	DF	MF	13-bit fragment offset	
8-bit Time To Live (TTL)	8-bit protocol	16-bit header checksum					
32-bit source IP address							
32-bit destination IP address							
Options (if any)							
Data							

If protocol=1, Data contains ICMP.
If protocol=6, Data contains TCP
If protocol=17, Data contains UDP

TCP Header

0 15 16 31

16-bit source port number								16-bit destination port number
32-bit sequence number								
32-bit acknowledge number								
4-bit header length	reserved (6 bits)	URG	ACK	PSH	RST	SYN	FIN	16-bit window size
16-bit checksum								16-bit urgent pointer
Options (if any)								
Data (if any)								

UDP Header

0 15 16 31

16-bit source port number	16-bit destination port number
16-bit UDP length	16-bit UDP checksum
Data (if any)	

Ethernet Frame Header

Total length 64 to 1518 Bytes

6-Bytes destination MAC address	6-Bytes source MAC address	2-Bytes Ether Type	Data	4-Bytes CRC checksum

If Ether Type= 0x0800, Data contains IPv4 traffic.
If Ether Type= 0x0806, Data contains ARP traffic.