**PRACTICAL 10**

**Aim:-PACKET CAPTURING USING TCPDUMP**

**Objective:-**

* Observation of the functional difference between different packets and the analysis of the different headers and protocol
* Writing a filtering options and choosing different analysis options.
* To collect and analyze packets and frames from a network. The frames will be generated by your workstations using the ping and ssh commands

**Thoery:-**

Tcpdump command is also called as packet analyzer. tcpdump command will work on most flavors of unix operating system. tcpdump allows us to save the packets that are captured, so that we can use it for future analysis. The saved file can be viewed by the same tcpdump command. We can also use open source software like wireshark to read the tcpdump pcap files.

**PCAP FILE FORMAT**

The file has a global header containing some global information followed by zero or more records for each captured packet, looking like this:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Global Header | Packet Header | Packet Data | Packet Header | Packet Data | Packet Header | Packet Data | ... |

A captured packet in a capture file does not necessarily contain all the data in the packet as it appeared on the network; the capture file might contain at most the first *N* bytes of each packet, for some value of *N*. The value of *N*, in such a capture, is called the "snapshot length" or "snaplen" of the capture. *N* might be a value larger than the largest possible packet, to ensure that no packet in the capture is "sliced" short; a value of 65535 will typically be used in this case.

**Global Header**

* This header starts the libpcap file and will be followed by the first packet header:

typedef struct pcap\_hdr\_s

{

guint32 magic\_number; /\* magic number \*/

guint16 version\_major; /\* major version number \*/

guint16 version\_minor; /\* minor version number \*/

gint32 thiszone; /\* GMT to local correction \*/

guint32 sigfigs; /\* accuracy of timestamps \*/

guint32 snaplen; /\* max length of captured packets, in octets \*/

guint32 network; /\* data link type \*/

} pcap\_hdr\_t;

* + magic\_number: used to detect the file format itself and the byte ordering. The writing application writes 0xa1b2c3d4 with it's native byte ordering format into this field. The reading application will read either 0xa1b2c3d4 (identical) or 0xd4c3b2a1 (swapped). If the reading application reads the swapped 0xd4c3b2a1 value, it knows that all the following fields will have to be swapped too. For nanosecond-resolution files, the writing application writes 0xa1b23c4d, with the two nibbles of the two lower-order bytes swapped, and the reading application will read either 0xa1b23c4d (identical) or 0x4d3cb2a1 (swapped).
  + version\_major, version\_minor: the version number of this file format (current version is 2.4)
  + thiszone: the correction time in seconds between GMT (UTC) and the local timezone of the following packet header timestamps. Examples: If the timestamps are in GMT (UTC), thiszone is simply 0. If the timestamps are in Central European time (Amsterdam, Berlin, ...) which is GMT + 1:00, thiszone must be -3600. In practice, time stamps are always in GMT, so thiszone is always 0.
  + sigfigs: in theory, the accuracy of time stamps in the capture; in practice, all tools set it to 0
  + snaplen: the "snapshot length" for the capture (typically 65535 or even more, but might be limited by the user), see: *incl\_len* vs. *orig\_len*below
  + network: link-layer header type, specifying the type of headers at the beginning of the packet (e.g. 1 for Ethernet, see [tcpdump.org's link-layer header types page](http://www.tcpdump.org/linktypes.html) for details); this can be various types such as 802.11, 802.11 with various radio information, PPP, Token Ring, FDDI, etc.

**Record (Packet) Header**

* Each captured packet starts with (any byte alignment possible):
* typedef struct pcaprec\_hdr\_s {
* guint32 ts\_sec; /\* timestamp seconds \*/
* guint32 ts\_usec; /\* timestamp microseconds \*/
* guint32 incl\_len; /\* number of octets of packet saved in file \*/
* guint32 orig\_len; /\* actual length of packet \*/

} pcaprec\_hdr\_t;

* ts\_sec: the date and time when this packet was captured. This value is in seconds since January 1, 1970 00:00:00 GMT; this is also known as a UN\*X time\_t. You can use the ANSI C time() function from time.h to get this value, but you might use a more optimized way to get this timestamp value. If this timestamp isn't based on GMT (UTC), use thiszone from the global header for adjustments.
* ts\_usec: in regular pcap files, the microseconds when this packet was captured, as an offset to ts\_sec. In nanosecond-resolution files, this is, instead, the nanoseconds when the packet was captured, as an offset to ts\_sec. this value shouldn't reach 1 second (in regular pcap files 1 000 000; in nanosecond-resolution files, 1 000 000 000); in this case ts\_sec must be increased instead!
* incl\_len: the number of bytes of packet data actually captured and saved in the file. This value should never become larger than orig\_lenor the snaplen value of the global header.
* orig\_len: the length of the packet as it appeared on the network when it was captured. If incl\_len and orig\_len differ, the actually saved packet size was limited by snaplen.

### **Packet Data**

* The actual packet data will immediately follow the packet header as a data blob of incl\_len bytes without a specific byte alignment.

## Libraries

It shouldn't be too hard to implement functions to read/write a libpcap file from scratch as it's a really simple file format. However, if you want to use a library for this purpose, or if you need to actually capture packets from a live network, the following libraries are available to do just this:

* [libpcap](http://www.tcpdump.org/): the origin of this file format (for UN\*X based systems)
* [WinPcap](http://www.winpcap.org/): Windows based version of libpcap

There are wrappers for various programming languages available (but you must have one of the above libs installed):

* [Net::Pcap](http://search.cpan.org/~kcarnut/Net-Pcap-0.05/Pcap.pm): Perl based libpcap wrapper
* [Jpcap](http://netresearch.ics.uci.edu/kfujii/jpcap/doc/links.html): JAVA based libpcap wrapper
* [python-libpcap](http://sourceforge.net/projects/pylibpcap/): Python based libpcap wrapper
* [Ruby/Pcap](http://www.goto.info.waseda.ac.jp/~fukusima/ruby/pcap-e.html): Ruby based libpcap wrapper
* You may add a libpcap wrapper for your Favorite programming language or using Google if it's still missing here ...

Note that if you write your own code, it will fail to read any capture files in the "next generation libpcap" format mentioned below. If you use libpcap, however, it will, when linked (at build time or run time) with a version of libpcap/WinPcap that can read those files, be able to read "next generation libpcap" files that don't use features not supported by the current libpcap API (such as packets from multiple interfaces with different link-layer types) as well as reading the current libpcap format. As such, you should use libpcap/WinPcap if you can, rather than writing your own code to read those files.

## Drawbacks

The libpcap format is very simple, one of the reasons that it has gained such a wide usage. Unfortunately it misses a few things which would be helpful:

* nanosecond time resolution
* user comments: "shows connection breakdown starting at packet 1432"
* interface information (like the network card manufacturer)
* packet drop count (and probably other counts as well)

**Windump Commands**

**List of Interfaces**

**tcpdump -D**

|  |
| --- |
|  |

**Packet Capturing in ASCIII VALUES**

**tcpdump –A**

|  |
| --- |
|  |

**No Packet Capturing Required**

**tcpdump –i interfacename –c no of packets**

|  |
| --- |
|  |

**Packet Capturing Write to Particular File**

**tcpdump -n -w filename.pcap**

|  |
| --- |
|  |

**Packet Reading from PCAP FILES**

**tcpdump -n –r filename.pcap**

|  |
| --- |
|  |

**Hexadecimal Format Packet Capturing**

**tcpdump -n -i interface -c no of packets -XX**

**Example:-tcpdump -n -i eth0 -c25 -XX**

|  |
| --- |
|  |

**Packet Capturing using Source host and Destination host**

**tcpdump –i interfacename –c no of packets src host ipaddress and dst host ipaddress**

|  |
| --- |
|  |

**Without Timestamp Packet Capturing**

**tcpdump -n -i interface -c no of packers -t**

|  |
| --- |
|  |

**Only Seconds in Packet Capturing**

**tcpdump -n -i interface -c no of packets -tt**

|  |
| --- |
|  |

**Timestamps in Packet Capturing**

**tcpdump -n -i interface -c no of packets -ttt**

|  |
| --- |
|  |

**Ascii Output for packet capturing**

**tcpdump -n -i interface -c no of packets -A**

|  |
| --- |
|  |

**VIVA QUESTIONS**

**Q 1 WHAT is the PURPOSE OF TCPDUMP??**

|  |
| --- |
| Ans: Tcpdump command is also called as packet analyzer. tcpdump command will work on most flavors of unix operating system. tcpdump allows us to save the packets that are captured, so that we can use it for future analysis. The saved file can be viewed by the same tcpdump command. We can also use open source software like wireshark to read the tcpdump pcap files. |

**Q2 What are the interface we can use during the packet capturing??**

Q3 How can we limit the packet capturing?

**Conclusion**