WIRETAP – TCPDUMP DATA ANALYSIS TOOL

In this project we implemented a software application that takes a tcpdump data file offline as input, analyzes it and outputs human readable statistics about the data file. We focused on data link layer Ethernet protocol.

The method we used was as follows:

- Open an input file using pcap_open_offline (). We make sure that this is a tcpdump data file and it can be opened
- As mentioned earlier, we only analyze data captured from Ethernet. We check this using pcap_datalink ()
- Read packet from the file using the pcap_loop (). This function is called for each packet using the session descriptor returned when we opened the input file. The pcap_loop () function loops until the maximum set number of packets has been read, it can also be set to loop forever until an error occurs, likely until end of data file. This is the option we took. The pcap_loop () function takes another argument, *pcap_handler processPacket* . *ProcessPacket( )* is the function we wrote to analyze, process and gather data for each packet.
- We use first argument of ProcessPacket ( ) function for total packet count, the next argument *struct pcap_pkthdr* to get the date, time and duration of packet capture and the other argument to cast and define pointer to the headers of different protocols, Ethernet, TCP, UDP.

For each packet processed the reference pointer is the data link layer protocol, Ethernet in our case. From the Ethernet header we get into its payload which contains packets from higher layers.

Ethernet_ hdr_pointer = packet_pointer

From the Ethernet header (fixed 14 bytes) we are able to get the source and target MAC of hosts
The Ethernet protocol type defines the protocol stack in the upper network layer; in our case is either IP or ARP. We do the casting to get the pointer to the different network protocol headers

IP_hdr_pointer = Ethernet header pointer + size_ of_ Ethernet_header
ARP_ hdr_pointer = Ethernet_header_pointer + size_of_Ethernet_header

From the network layer protocol headers we're able to get the source and destination IP addresses. The protocol field in IP header will determine the upper transport layer protocol, TCP or UDP

UDP_hdr_pointer = IP_hdr_pointer + size_of_IP_header
TCP_hdr_pointer = IP_hdr_pointer + size_of_IP_header

UDP header has a fixed length of 8 bytes (2 bytes for each of source and destination ports, 2 bytes packet length and 2 bytes for checksum. The UDP checksum is computed over a pseudo header of information from the IP header (source IP, Destination IP, Protocol, UDP length), UDP header and UDP payload using the 16-bit one's complement sum. We realized that the protocol is in host byte order so we had to convert it to network byte order before doing the sum. If the checksum field in the UDP header is cleared then checksumming is omitted for that packet.

The transport layer offers port multiplexing for the application layer protocols. For DHCP in our case - DHCP client/Server request, client use port 68 while server use port 67.

DHCP runs above UDP, so to get to DHCP header pointer and its payload

DHCP_hdr_pointer = UDP_hdr_pointer + size_of_UDP_header

Major Components of Project

Util.h header file contains all the header structure we used in the project. The UDP Checksum calculation method is also in this header.

process_tcp, process_ip, process_udp_process_dhcp, process_arp*:* these methods are the key methods of the project. They take the packets and process them according to their header structures.

Finding the DHCP Message Types: The DHCP packet has a fixed header size of 236 bytes that includes all fields up to "file name" after which the data portion starts. Now the first 4 bytes of data has some fixed values after which DHCP option starts. The first byte of options is Tag field and then Length field (variable as per Tag value) and then the option value as per Tag value.
For the Tag value of 53, which is message type, then length is 1 (1 byte) and so we need to skip 2 bytes to reach the option value.

Deliverables

Our routine outputs the following statistics:

- Start date, time and duration of the packet capture
- Total packets
- Unique Ethernet addresses, along with the total number of packets containing each address. Ethernet addresses are represented in *hexadecimal-colon* format.
- Unique IP addresses, along with the total number of packets containing each address. The IP addresses are represented in the standard a.b.c.d notation.
- Unique ARP participants, their associated MAC addresses, and IP addresses.
- For UDP, it lists the unique ports seen, either as source or as destination. Also determines
- For UDP, it determines the number of packets with a correct checksum, an incorrect checksum, and those that omit checksum calculations. Unique DHCP clients and servers, listed separately. It indicates the number of packets transmitted by each client or server.
- The number of DHCP packets that are DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, and DHCPACK.
- Average, minimum, and maximum packet sizes.
- unique ports in TCP traffic

This project enabled us to learn how to interception and capture packets transmitted or received over a LAN. Our focus centered on data link layer Ethernet protocol. We were also able to analyze network layer header and their payloads. Our analysis was on packets with IP and ARP headers at the network layer, TCP and UDP at the transport layer, and DHCP at the application layer. The knowledge, experience and skill gained from this project should enable us to extend analysis to other protocols