Sabyasachi Gupta
326005513
CSCE 665 HW 1

Instructions to execute program,

1. First, put all the necessary program related files (main.cpp, functions.cpp, functions.h, libdef.h) in a single folder.
2. Execute the following code to compile the program.
   g++ main.cpp functions.cpp -lpcap -o analyzer -std=c++11
3. Next use the following code to execute the program.
   ./analyzer <pcap filename>
4. Uses C++11 standard for some APIs.

Currently it takes one file as input but processes for all the protocols HTTP, FTP and TELNET.

Design:

The packet analyzer program is made up pf multiple functions.

"libdef.h": This is a header file which consists of all the definitions and libraries which are necessary to run the program. It also consists of Ethernet, IP and TCP header structure.

"main.cpp": This is the main program file. It consists of multiple functions like packet_capture, process_sessions, print_header_info, print_payload etc.

"packet_capture": In this function we take the received packet from pcap file and strip it under Ethernet header, IP header, TCP header etc. And after stripping all the headers we store the rest of the payload to an unordered map or hash map with respective sequence number.

"process_sessions": After we are done parsing the data, next we reassemble the same using this function. We sort the created list with unique session id so that we can keep track of the conversation between two unique hosts. Next, we call the print_header_info and print_payload function to publish the data.

"print_header_info": We use this function to print the session details before we print the payload. Session detail consists of source and destination ip, protocol etc.

"print_payload": This function prints the remaining payload of the packet. It also calls upon telnet commands to segregate the telnet interactions.

I am handling packets so, I will be displaying the packet payload which is the meaningful data for respective application on client and server.

Initially I have started with storing sessions separately for each client and for each protocol and later realized that it makes better sense to combine all session data together.

Learning:

- I have got an in-depth knowledge on the working of these protocols by analyzing what fields are available in each protocol.
- On that line, the HTTP header ends with "\r\n\r\n" where control information ends and data for HTML starts.
- TELNET control information has sets of 3 Byte control words which start with FF and these control words are used initially to set up various options for further communication. I have tried to include as many command words possible.
- For FTP, all the control information takes place on that port and actual data transport takes place on a different port negotiated in the session for PASV mode.
- Many of the available packets not structured properly.

Debug Experience:

- All the Packet data like IP address, sequence number, acknowledgement number, source port, destination port etc. are in network byte order and hence for the display are to be converted to host format using ntoh functions.
- Wireshark helped a lot to trace all the protocol and analyze each session separately.
- Implementing hash map list was challenging as the storing each packet data and sorting data was error prone.
- First did a plain data dump of the given packets and then later applied filters using different header structure.
- Implementing libpcap filter was not yielding favorable results while fetching information.
- There are many packets with payload size zero.
- There are lot of code words in protocols especially for TELNET and hence few are displayed as ascii values.

Lessons:

- In networks miscalculation of a single bit can be costly, especially in case of header size calculation.
- TCP header variable length is present as 4 Bits in the TCP header.
- Implementation as well as analysis of the protocols was challenging.
- Did some part initially to parse headers for each of the protocols- HTTP method parse, TELNET CODE word parse etc. but later generalized the structure.
- C++ data structures and code snippets were very helpful to complete this homework.
- Had to change implementation multiple times because of the loose guidelines. In future what kind of parsing is needed should be mentioned.