Name: Atmika Sharma

SBU ID: 111464371

--------------------------------------------------- **ASSIGNMENT 2** -------------------------------------------------

**ASSIGNMENT PROBLEM:**

This is a passive network monitoring application called mydump which uses libpcap. This program captures traffic from a network interface in promiscuous mode or reads the packets from a pcap trace file and prints the record for each packet in its standard output. The user can specify a BPF filter for capturing a subset of the traffic and/or a string pattern for capturing only packets with matching payloads.

The program conforms to the following specification:

*mydump [-i interface] [-r file] [-s string] expression*

------------------------------------------------------------------------------------------------------------------------

**FEATURES TO BE HANDLED:**

1. -i

-> Captures live from the network device <interface> (e.g., wlp2s0). If not specified, mydump automatically selects a default interface to listen on. Capture continues indefinitely until the user terminates the program.

2. -r

-> Reads packets from <file> in tcpdump format

3. -s

-> Keeps only packets that contain <string> in their payload (after any BPF filter is applied)

4. <expression>

-> A BPF filter that specifies which packets will be dumped. If no filter is given, all packets seen on the interface (or contained in the trace) are dumped. Otherwise, only packets matching <expression> are dumped.

-------------------------------------------------------------------------------------------------------------------------

**IMPLEMENTATION DETAILS:**

1) The program mydump starts with checking for command line options -i, -r and -s where -i checks what interface to read on, -r checks for the read file if available and -s checks for a string to be checked in the payload. A bpf filter is also supported in this program.

2) The program uses a while loop to check for the above specified command line options using the getopt() function and processes the arguments for the specified option.

3) For the -i option, the switch case uses the argument of this option to capture the device name to sniff on. If the -i option is not specified, this case assigns 'dev' the default device name, in case there is no read file specified.

4) For the -r option, the switch case uses the argument of this option to read a capture file (pcap) and assigns it to a handle for the packets to be sniffed.

5) For the -s option, the switch case uses the argument of this option to verify whether this string is present in the payload of the packet being currently sniffed.

6) After exiting the while loop, the program assigns the bpf filter to a variable to be used for pattern matching further in the program. It also checks for whether we have found a device or read file to sniff from. If not, the program exits.

7) If a device is available, we use pcap\_lookupnet() to look for a network number and mask associated with the available device.

8) We then print the information of the device that we have obtained. The program opens the capture device and assigns it to a handle. For this we use pcap\_open\_live() and the program exit if it is unable to do so.

9) We will also make sure we're capturing on an Ethernet device for which the value of pcap\_datalink(handle) has to be DLT\_EN10MB.

10) Up until now we had not handled the bpf filter from our command line. We now compile this filter expression using pcap\_compile() and then apply this filter it using pcap\_setfilter().

11) With the available information, we start processing the packets through sniffing using pcap\_loop() which loops through the packets obatined by applying the filters and options.

12) The callback function for when a capture device or read file is available to us is got\_packet().

13) We declare structs for our ethernet, ip, tcp and udp headers. In each packet, we access the header and payload information (eg. offsets etc.) through these stucts.

14) We find header offsets for ip, tcp and udp packets, compute payload offsets for all three protocols and also compute size for all three.

15) If we have a string to be checked, we call the string\_match() function which assigns all printable characters of our payload to a another unsigned character pointer. We use this to check for string matches in our payload.

16) We print our packet information within this condition for eg. gettime() to extract the timestamp, ether type etc.

17) If no string is to be checked we simply print our packet information for packets being sniffed.

18) For the purpose of printing we use the print\_payload() function.

----------------------------------------------------------------------------------------------------------------------------

CONTENTS OF MAKEFILE:

go: mydump.cpp

g++ -o mydump mydump.cpp -lpcap

clean:

rm mydump

----------------------------------------------------------------------------------------------------------------------------

CONTENTS OF TARBALL FILE:

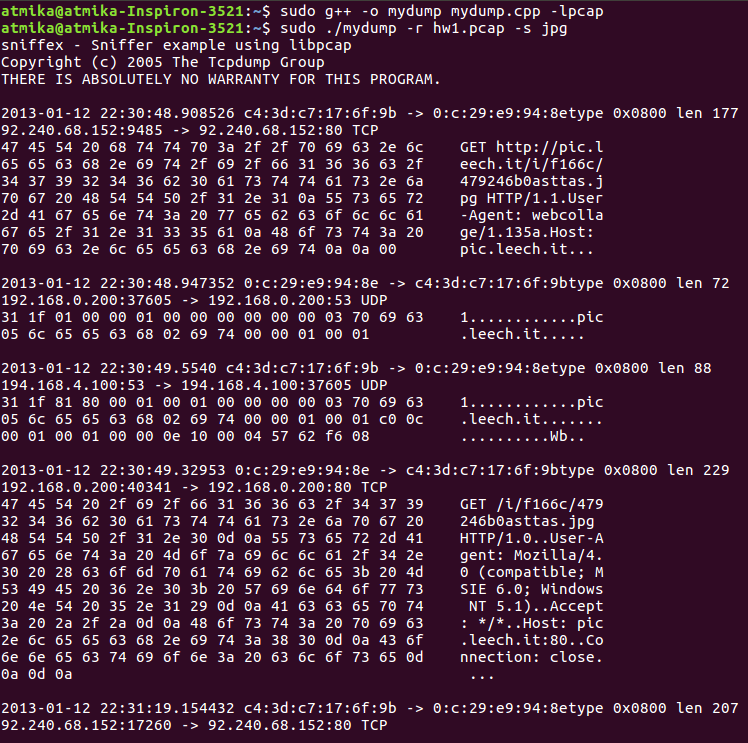
1. Makefile

2. Report

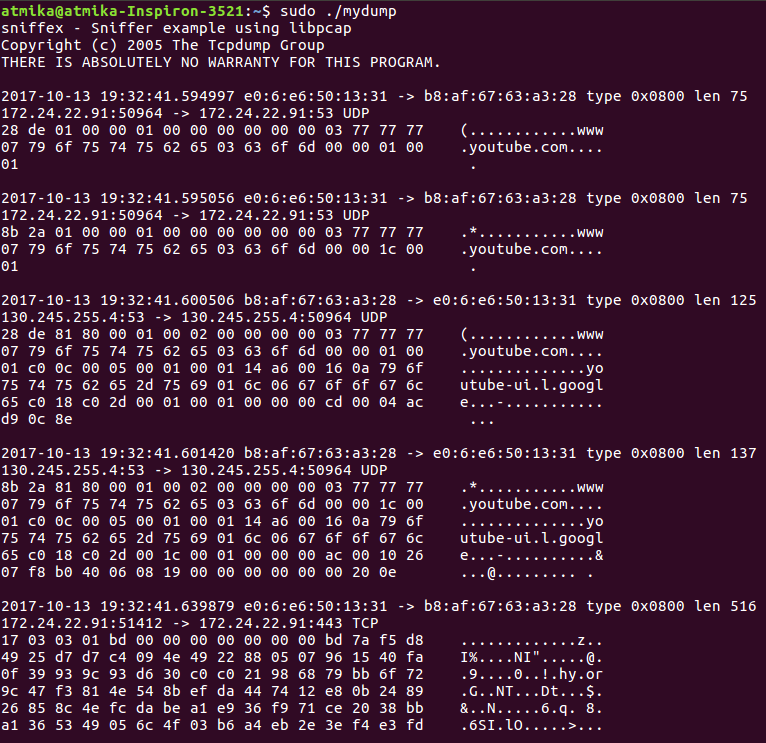
3. Source code

**OUTPUT:**

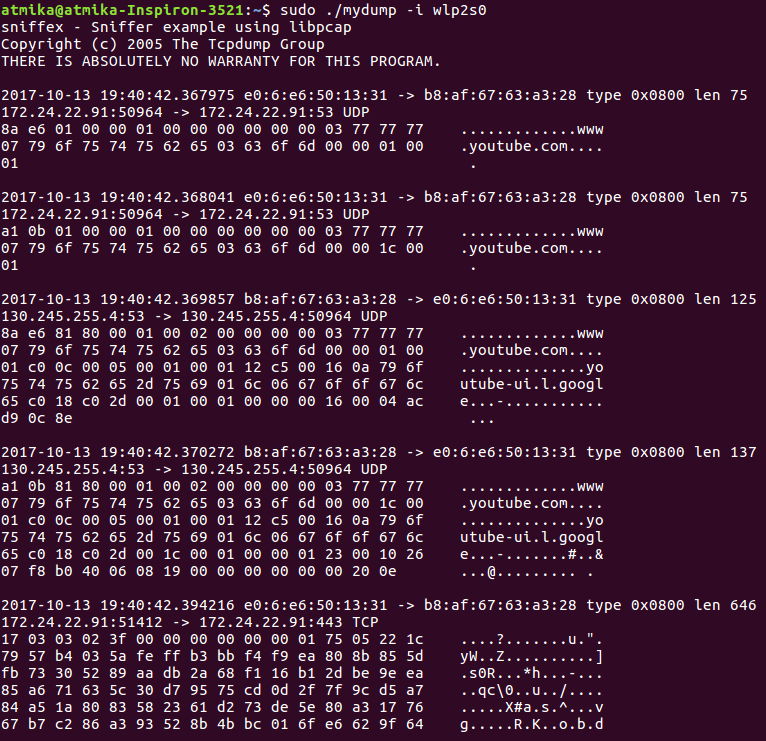
1. mydump -r <filename> -s <string>



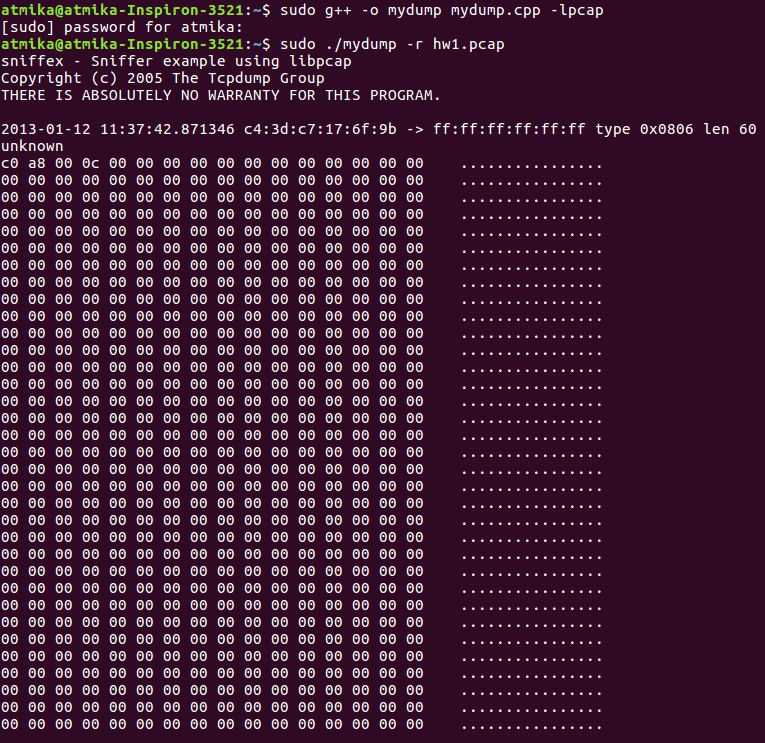
2. mydump



3. mydump -i <interface>



4. mydump -r <filename>



5. mydump -r <filename> tcp

