CPE464 – Program #1 – Packet Trace

Due: Monday April 13, 2015 at 11:59 pm.

This program is due Monday April 13, 2015 at 11:59 pm. The last possible late turn in date is Thursday April 16. You will lose 15% per late day. After April 16 you will receive a 0.

For this program you will write a packet sniffer program called trace. This program will output protocol header information for a number of header types. To write this program you will use the pcap library that allows you to sniff packets. For this assignment you will only be "sniffing" packets from trace files. Your program must run on unix2**.** (I highly recommend you install the needed software on your home Linux box and do your development at home. BUT your program MUST work on unix 2.)

pcap is an Applications Programmer Interface (API… a set of functions) that allows for the retrieval of packets from either a networking interface card (NIC) or a file. This library is used by both tcpdump and wireshark to retrieve packets. Because you need root privileges to retrieve packets from the NIC (and you do not have root on school machines!) you will write your program to retrieve packets from trace files. While I will provide you with a sample set of trace files, you can create new ones using wireshark on your home machine. For more information on pcap see: http://www.tcpdump.org/

Your program, called **trace**, will take one parameter. This is the file that contains the packets. This file will be in a format readable by the pcap library.

To run the program:

```
trace aTraceFile.pcap > someOutputFile.txt
```

(Notice the input trace file is a command line argument. The output goes to stdout. Redirect your output to a file and compare (using the Unix **diff** utility) your output with my output.)

Types of headers you need to process:

- Ethernet Header
- ARP Header both Request and Reply
- IP Header
- TCP Header
- ICMP Header for Echo request and Echo Reply
- UDP Header

Some functions of interest:

- Some pcap functions
  - pcap_open_offline(…)
  - pcap_next_ex(…)
  - pcap_close(…)
- inet_ntoa(…)
- ether_ntoa(…)
- htons(…), htonl(…), ntohs(…), and ntohl(…)
- memcpy(…)

Notes:

1) Install wireshark (www.wireshark.org) onto your home computer. This will allow you to look at the trace files and help you debug your program. Note – wireshark is installed on all unix based computers in the CSL.
2) Do not use any code off the web or from other students. You may look at the sample code provided at www.tcpdump.org but do not copy code off of this site. The work you turn in must be your entirely your own. You may use the code I handed out for the Internet Checksum.
3) Do not sniff packets on an open network or in a CSC lab. You should generate your packet traces on a closed network (home network) or in the Cisco lab. Packet sniffing in an open Cal Poly lab is against the Cal Poly RUP.
4) You must create your own header structures for Ethernet, IP, TCP, UDP, ARP if you need them. You cannot use structures for these headers (Ethernet, ip, tcp, udp) created by others. Part of this assignment is for you to learn more about the different headers.
5) You must break your code up into reasonable sized functions (~30 lines is good, 50 lines is getting long). You must at least create functions called ethernet(…), ip(…), tcp(…), arp(…), icmp(…). Each of these functions should process the appropriate PDU.
6) I have provided a sample set of traces. I will test your final program with additional traces. You need to create and turn in two additional test traces using wireshark.
7) You should not set a pcap filter. Instead for PDUs you cannot process, print out the message: "Unknown PDU
8) You must provide a Makefile. This Makefile should create the program called trace. It should also provide a clean target. This target should delete object files and the executable file.
9) Trace files and sample output can be found on blackboard.
10) You will need to understand the *pseudo-header* to correctly verify the TCP and UDP checksums.
11) Your output must match my output. Your program will be graded by diff'ing your programs output with my programs output. Do not get creative! Do not add additional headers, options, fields… anything. My output is always correct (even if it is wrong).

What to turn in:

1) Makefile that correctly compiles your program on Unix 2 without warnings. Note your Makefile MUST create a program called ***trace***.
2) All source code (include checksum.c and .h) needed to compile your program.
3) Two new tracefiles that you created.
4) Output of your program for these two new trace files.
5) Optional readme with comments for the TA

Grading: (You must complete the early parts to receive any credit for a later part.)

1) (10%) Correctly read from the file and output the Ethernet headers
2) (10%) Part 1 and output the ARP headers
3) (30%) Part 2 and output the IP headers
4) (40%) Part 3 and output the TCP headers (TCP Checksum must work for any part 3 credit)
5) (10%) Part 4 and output ICMP and UDP headers

Note: Style points, bad programming, late points, error handing points also will be considered.

I will provide trace files and output files on blackboard. Your program should produce output which diff's cleanly with my output. The whitespace in your output should be fairly close to my output including the spacing. For grading we will use the –ignore-all-space and the –B options when comparing your output to my sample output (so EXACT spacing is not required but it should look very close).

Note for TCP/UDP Port number output - use HTTP, Telnet, FTP, POP3 or SMTP otherwise use the port number. (see www.iana.org, look at port numbers for the correct values for these services.)

Header outputs:

The fields listed below are the required fields. To check output format, see the example tracefiles on blackboard. (The TRACE FILES are the standard I will hold you to… you MUST diff against them.)

- Ethernet Header Output:

  Dest MAC:
  Source MAC:
  Type: (Possible type: ARP, IP, Unknown)

- ARP Request/Reply Output

  Opcode: (possible Request or Reply)
  Sender MAC:
  Sender IP:
  Target MAC:
  Target IP:

- IP

  TOS:
  Time to live:
  Protocol: (possible ICMP, TCP, UDP, Unknown)
  Header checksum: (possible Correct, Incorrect)

Source IP:
Destination IP:

- ICMP
  Type: (possible Echo Request, Echo Reply, Unknown)

- TCP
  Source Port: (possible values see comment above)
  Destination Port: (possible values see comment above)
  Sequence number:
  Ack Number:
  SYN flag: (Yes or No)
  Reset flag: (Yes or No)
  FIN flag: (Yes or No)
  Window Size:
  Checksum: (possible Correct, Incorrect)

- UDP
  - Source Port: (possible values see below)
  - Destination Port: (possible values see below)

Thoughts:

- Test your program by diff'ing your output against my output. That is how I will test it.
- Declare your packet/frame buffer as an unsigned char array/pointer.
- Make sure you ntohs or ntohl when needed like on port numbers or length fields
- Note that the length field in the TCP pseudo header needs to be in network order
- The TCP header, specifically the checksum, takes the most time. You will receive zero (0) credit for the TCP header part of the program (part #4) without having the TCP checksum working correctly.