Program 1 (revision 1)

CPE 464 Fall 2017

Due by 23:59, September 29^{th}.
Handin closes 7:59, October 5^{th}.
This assignment is to be done individually.

Program Overview

For this program you will write a packet sniffer called `trace`. This program will output protocol header information for a number of different protocols. To write this program you will use the pcap library (libpcap) that allows you to sniff packets. Due to the campus network use policy, you will only be "sniffing" packets from trace files and using Wireshark to verify your program is producing the correct output.

Target Platform

This program must run correctly on unix4.

Wireshark

Begin this assignment by downloading and installing wireshark on your home PC. The URL for wireshark is in the references section of this writeup. Wireshark is also preinstalled on all the general purpose linux machines in the department. After you have installed wireshark use it to open up and examine the sample input files for the assignment. Look at the various headers for a few packets. Pay attention to how the headers are nested. It may be useful to have a header diagram with you when you are examining the packets. You should be comfortable using wireshark. It is an invaluable debugging aid for this course and networks in general.

Reading the Trace File

Pcap is an API that allows for the retrieval of packets from either a network interface (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 network (and you do not have root on unix4!) 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 additional ones using wireshark on your home machine.

Command Line Interface

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.

An example invocation of your program is as follows:

```
unix4>  trace-Linux-x86_64 aTraceFile
```

Detailed Specification

Your program must meet all requirements as specified in the course syllabus. Your program is responsible for parsing and printing the following headers to standard out:

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

You only need to print selected fields from the various headers. The exact format is specified by the example output. Make sure your output diffs cleanly (whitespace included) with the sample output provided. The detailed filed list is:

- 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 Request, Reply, Unknown)
```

- TCP

```
Source Port:        (possible values see below)
Destination Port:   (possible values see below)
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)
```

Your program should print out the port number is decimal form. However, for certain well-known TCP and UDP ports, your program must print ASCII labels. These labels are:

- HTTP
- Telnet
- FTP
- POP3
- SMTP

Consult /etc/services on `unix4` or the port assignment list available at http://www.iana.org/ to determine the appropriate port numbers for the services mentioned above.

Tips and Tricks

- You must create your own header structures if you need them. You cannot use structures created by others. Part of this assignment is for you to learn about the different headers and how encapsulation works.
- I have provided a sample set of traces. I will test your final program with additional traces. I recommend that you create additional test traces using wireshark on your home PC.
- You should not set a pcap filter. Instead for PDUs you cannot process, print out the message: "Unknown PDU"
- You will need to understand the pseudo-header to correctly verify the TCP and UDP checksums. It is explained in your course textbook and at various sites online.
- Your output must match my output. Your program will be graded by diffing your programs output with my programs output. Do not get creative! Do not add additional headers, options, fields, debugging output, or anything!

- One of the problems you will encounter moving between platforms is different native data alignments for primitive types. Compile and run the provided `alignment.c` program on different platforms to illustrate this. You must use `gcc` to compile the example program on Solaris. Understanding why `alignment.c` behaves the way it does will be very helpful for all the assignments in this course.
- I have provided an example makefile that illustrates how to properly compile a program that meets the course standards. The makefile also includes a "handin" target that will submit all necessary files.
- Both the TCP and IP headers are variable length. Make sure your code correctly accommodates all valid header lengths.

Network Use Policies

Do not sniff packets on an open network (*e.g.*, the campus wireless 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.

Provided files

The following files are available through the course website to help you with the program:

- A sample Makefile
- The smartalloc source and header file(s) for C and C++.
- The source and header file for an implementation of the Internet checksum. You will need this when verifying the checksum.
- Sample input and output files.

Collaboration

This is an individual assignment. Only collaboration as described in the syllabus for programming assignments is allowed. Refer to the syllabus for the details. Remember that all cheating and unauthorized collaboration will result in a F in this course and referral to the Judicial Affairs Office.

Base Functionality

The base functionality for this assignment is passing the `ArpTest` and `IP_bad_checksum` test cases that have been provided with this assignment.

What to turn in

Note that the example makefile includes a "make handin" target that should correctly submit all `p1` files. You will need to modify that target to work for future programs.

Submit via `handin` to the `464_p1` directory of the `bellardo` account (syntax: `handin bellardo 464_p1 <file1> <file2> ...`):

- your source and header file(s).

- A copy of the smartalloc source and header file.
- A copy of the Internet checksum source and header file.
- A makefile (called `Makefile`) that will build `trace-Linux-x86_64` on unix4 from your source when invoked with no target or with the appropriate "`trace-*-*`" target . The makefile must also remove all binary and object files when invoked with the "`clean`" target. Refer to the example makefile if you need more guidance on this. There is no constraint placed on the names of the object file(s), however I suggest giving them either a platform unique name or platform unique location, so you don't have problems when moving between different machines.
- A README file that contains:
  - Your name.
  - Any special instructions.
  - Any other thing you want me to know while I am grading it.

  The README file should be **plain text,** i.e, **not a Word document**, and should be named "README", all capitals with no extension.

Example Output

The example inputs are outputs for this assignment are available online.

Resources

The following is a list of resources you may or may not find useful while working on this assignment.

- Google - `http://www.google.com/`
- Man pages for libpcap. Specifically look at the functions `pcap_open_offline`, `pcap_next_ex`, and `pcap_close`.
- Man pages for the functions `inet_ntoa`, and `ether_ntoa`.
- Wireshark - `http://www.wireshark.org/`
- Online documentation for writing makefiles - `http://www.gnu.org/software/make/make.html`
- You can find detailed header layout information in your course textbook and through numerous online sources.
- Consult the wikipedia entry on endianness, and the documentation for the functions `htons`, `htonl`, `ntohs`, and `ntohl`.