Lab 3: Wiretap

Due October 30th at 11:00pm

Project goals

Programs such as tcpdump (`http://www.tcpdump.org/`) and Wireshark (`http://www.wireshark.org/`) allow the interception and analysis of packets being transmitted or received over a LAN. One prominent use of this information is in troubleshooting network configuration and reachability. In this project, we will provide you with data captured using these tools. Your task is to write the analysis routines similar to those provided by tcpdump and Wireshark.

Deliverables

Your submission should include the following files:

- `wiretap.c or wiretap.cpp`
- `Supporting files (*.c, *.cpp, *.h)`
- `Makefile`
- `README`
- `Write-up in a PDF format`

Your `README` file should contain a short header containing your name, username, and the assignment title. The `README` should additionally contain a short description of your code, tasks accomplished, and how to compile, execute, and interpret the output of your program.

Even though you are working in groups on this assignment, each student must submit their own `write-up`. You may not see your partner's written answers. The `write-up` PDF file should contain a short header containing your name, username, and the assignment title. It should also contain general information about the contents of each file that you submit. That is, a person not familiar with this assignment should have a clear understanding of the purpose of each submitted file after reading your `write-up`. Furthermore, the `write-up` should include a 'Credits:' section, which lists all the people (e.g., students in the class or not, friends, colleagues, etc.) and resources (e.g., web pages, manuals, books, code, etc.) that you consulted for this assignment. Finally, the `write-up` should contain team spirit scores for your partner(s).

Important notes

- You can use either C or C++.
- Your Wiretap program should always exit gracefully, even if it is presented with broken and/or IPv6 packets. Points will be taken off for program crashes, segmentation faults, and memory leaks.
- The code should be clean and well-commented.
- There should be no magic numbers in your code.
- Break the code into functions if you find yourself using more than 5 levels of indentation.
- The submitted code must compile and run on the CS `Burrow` machines (`silo.cs.indiana.edu`).
- Points will be taken off for failing to submit a working Makefile with your code.
- If the code does not run on `Burrow`, the group will have to demo the project on the machine of their choice during the office hours. This carries a 15-point penalty.
- If the code does not run on `Burrow` and none of the team members have a computer in their possession which is capable of running the code, the team will receive a failing grade for the project. No exceptions. The actual grade will depend on the submitted code and documentation.

Cheating and Plagiarism

Although it is okay to discuss assignments (verbally; no written exchanges) with other students, all solutions handed in must be your own. Your code must be a group effort between the two group members only (you are expected to write the code together). The `write-up` must be written individually; you may **not** read your project partner's (or any other student's) `write-up`.

Copying answers from friends or the Internet is prohibited. We will use different software packages to check for plagiarism. Furthermore, you must cite all sources used for problem solutions (i.e. websites, books, discussions with classmates, etc.) in the `write-up`. For more information please see Indiana University Code of Student Rights, Responsibilities, and Conduct: `http://www.indiana.edu/~code/`.

Submission Instructions

Submit a tarball with your code in a folder named *Project 3* via `OnCourse`. Submit `write-up` along with your code, but do not put it in the archive.

Submission example:

```
project3.tar
write-up.pdf
```

1 Project Specification

Your program, *Wiretap*, should take a file containing tcpdump data as its input and output the statistics detailed later in this document. Since this data is not in human-readable format, you will have to use the `Packet Capture Library`, `libpcap.a`, and the functions in its header file, `pcap.h` (found in /usr/include on the CS Linux machines) to read the data.

When compiling your program, include the `pcap` library by using `-lpcap` as the first argument to your GNU compiler. For example,

```
$ gcc -lpcap -o wiretap wiretap.c
```

will compile a C program with the pcap library support.

For C++, simply change the compiler from gcc to g++:

```
$ g++ -lpcap -o wiretap wiretap.cpp
```

1.1 Implementation details

The other steps you should follow are:

- Open an input file using function `pcap_open_offline()`.
- Check that the data you are provided has been captured from Ethernet using function `pcap_datalink()`.
- Read packets from the file using function `pcap_loop()`. Note that this function needs to be called only once. It takes 4 arguments. Of these, the second and the third arguments are of most interest to you. The second argument lets you specify how many packets to read from the file. The third argument, `pcap_handler callback`, is where most of the action happens. Here, `callback` is the function you write to process data from each packet.
- You can pass the callback function to the `pcap_loop()` function simply by giving its name as the appropriate argument to `pcap_loop()`. The callback function must be a void function that takes three arguments, of the types `u_char *, const struct pcap_pkthdr *, const u_char *`. The callback is called by `pcap_loop()` once for each packet. The second argument to the callback is the special libpcap header, which can be used to extract the entire packet length and the packet arrival time (see the `pcap_pkthdr` structure in /usr/include/pcap.h). The third argument contains the contents of a single packet (from the Ethernet packet header onward).
- Close the file using function `pcap_close()`.

1.2 Specifics of Expected Wiretap Functionality

Your Wiretap should run on `Burrow` and must be written in C or C++. A user would invoke it as: "./wiretap [option1, ..., optionN]". Implement the following options:

- −−help. Example: "./wiretap −−help".
- −−open <capture file to open>. Example: "./wiretap −−open capture.pcap".

Details of each option are given below:

- **help:** When Wiretap is invoked with this option, it should show a short program description as well as display all options available to the user.
- **open:** When Wiretap is invoked with this option, it should open the specified file. If it is a valid `tcpdump` file, your program should parse it and display the output on the screen.

For example, the following command line options tell Wiretap to open `capture1.pcap` file located in the 'captures' directory.

```
$ ./wiretap --open captures/capture1.pcap
```

Your Wiretap program **must** support the options specified above exactly as they are written.

1.3 Output

The `callback` function should gather statistics from each packet to enable your program to print the following on standard out:

- Summary:
  - Start date and time, total duration, and total number of packets in the packet capture.
  - Average, minimum, and maximum packet sizes. Here, packet refers to everything beyond the tcpdump header.
- Link layer:
  - Unique Ethernet addresses found as both sources and destinations, along with the total number of packets containing each address (i.e. count the number of packets per unique address). Represent Ethernet addresses in hex-colon format.
- Network layer:
  - Unique Network layer protocols seen and total number of packets per protocol. Report protocols by protocol number, except for IP and ARP.
    - Note: The type field in an ethernet frame may specify either the packet length (particularly in packets based on older protocols) or the network layer protocol. You can differentiate the two by the value. All network layer protocols have a type of at least 0x0600 (0x0800 in practice). Your output should distinguish between the two possibilities.
  - Unique source and destination IP addresses, along with the total number of packets containing each address. Represent IPv4 addresses in the standard a.b.c.d notation. Ignore IPv6 addresses.
  - Unique ARP participants, their associated MAC and IP addresses, and total number of packets per each MAC and IP address. Ignore IPv6.
- Transport layer:

- Unique transport layer protocols seen and total number of packets per each protocol. Report protocols by protocol number, except for TCP, UDP, and ICMP.
- Unique source and destination TCP and UDP ports, along with the number of packets per unique port.
- For TCP, number of TCP packets containing specific flags.
- For TCP, number of TCP packets containing specific options.
- Number of packets containing unique ICMP types and codes.

The output produced by your program should be easy to read and understand. All specific formatting options are up to you, so long as you can justify your decisions. Sample output snippet:

```
$ ./wiretap --open sample.pcap
```

```
=== Summary ===

  Start date: 2010-11-05 11:33:12.140221
    Duration: 2 seconds
   # Packets: 5
    Smallest: 62 bytes
     Largest: 94 bytes
     Average: 75.2 bytes

=== Link layer ===

--- Source ethernet addresses ---

00:02:e6:fb:6f:02        1
00:d4:05:56:c8:10        4

--- Destination ethernet addresses ---

00:0f:1f:71:a4:4e        2
ff:ff:ff:ff:ff:ff        3

=== Network layer ===

--- Network layer protocols ---

  length = 0x0026        1
  length = 0x0052        1
  length = 0x0062        1
             ARP         1
              IP         1

--- Source IP addresses ---
     ...
     ...
     ...
```

1.4 Test files

Attached you will find two pcap files (along with the sample output) which can be used for testing purposes.

Summary of the attached files:

```
traceroute.pcap --  A traceroute (primarily UDP traffic and some ARP)
traceroute.txt --  sample output
wget.pcap --  Downloads of two websites (TCP traffic with some UDP and ARP)
wget.txt --  sample output
```

Note that your program should work on packet captures that have protocols your program does not understand (e.g., if presented with an ICMPv6 packet, your code should ignore the packet and continue analyzing other packets in the capture file).
Furthermore, additional test files will be used to grade your project. For example, even though you are not provided with a capture file containing a packet with the TCP URG flag set, your Wiretap program should be able to deal with such packets and display the correct output.

2 Miscellaneous

- Install Wireshark and experiment with it.
- Beej's Guide to Network Programming: `http://www.beej.us/guide/bgnet/`
- Getting started with pcap: `http://www.tcpdump.org/pcap.htm`
- Use the `ntohs` and `ntohl` functions as appropriate to read values that span multiple bytes. This ensures that the bytes are in proper byte order (Endianness) for use at this host.

3 Deliverables and grading

Submit your code and project files via `OnCourse` before the deadline.

4 Credits

This lab is © Minaxi Gupta, which we have modified for P-538.