MCS - Lab 1 - Pcap reader

Team members

- Valentin Minder
- Mélanie Huck
- James Nolan

Introduction

In this lab, we will build a command line tool that reads a pcap file and prints its data in a user friendly format. For this, we will use Erlang's unpacking capabilities.

We will test our code using Rebar's eunit tool.

API methods

Before coding, we discussed on what our program should do. We agreed on the following specification:

- Read `pcap` files
  - **Input**: a binary representation of a `pcap` file
  - **Output**: the global header as a map of key-values
  - **Output**: a binary representation of the `pcaps` packets
- Split a list of `pcap` packets
  - **Input**: a binary representation of a list of `pcap` packets
  - **Output**: A list of tuples, one for each `pcap` packet. Each tuple contains a map of the packet's header as key-values, and the decoded packet
  - *Note* : This method could be split in two smaller methods: one that split the binary in packets and one that read the parses the packets. To be discussed
- Read an `IP` datagram
  - **Input**: a binary representation of a a `pcap` packet
  - **Output**: IP headers as a map of key-values
  - **Output**: a binary representation of the `IP` packet's content
- Read an `ICMP` packet
  - **Input**: a binary representation of a an `ICMP` packet
  - **Output**: `ICMP` headers as a map of key-values
  - **Output**: a binary representation of the `ICMP` packet's content

Call hierarchy

We had to make a choice on how functions interact. Although each of them is assigned a specific task, the encapsulating nature of network layers inevitably make them depend on one another. For example, we had two options to implement the IP datagram decoder: 1. The function decoding the `IP` datagram calls the function to decode the underlying `ICMP` packet. 2. The function returns the binary payload and lets the user call the `ICMP` decoder if necessary.

We chose to combine those approaches. The function `read_global_header` returns a binary representation of the encapsulated data. All other function return a decoded version of their payload.

Checksum

We do not compute the `ICMP` and `IP` header checksums, leaving this as a potential future feature.

Tests

We built our program using a test driven approach. Our program pass all 13 tests. Our tests cover the following parts of our code:

```
Code Coverage:
pcap_client :   0%
pcapreader  :  87%
reader      : 100%

Total       : 43%
```

The client just uses the API to print the data as it wishes, so we focused on testing the `reader` api, because it contains the core features of our program.

Conclusion

We were surprised by how straight-forward it was to parse binary files with Erlang, compared to languages like Java.

Ressources

- The `pcap` file format
- The `TCP/IP` protocol
- The `ICMP` protcol