COMP 7003
Assignment 2
Report

Towa Quimbayo
A01086002
Feb 3rd, 2025

Purpose

This report aims to provide a comprehensive overview of Assignment 2, the packet capture and parsing program. This document details the list of program requirements, the implementation of its core functionalities, and the testing conducted across various operating systems. The program is designed to capture network packets in real-time, manually parsing key headers for ARP, and IPv4 (TCP, UDP, ICMP, and DNS) and display the extracted information in a correctly formatted manner.

Requirements

Task	Status
Live Packet Capture Using Scapy: Use Scapy's AsyncSniffer to capture live packets on a specific interface, optionally applying a BPF filter, and stop after capturing a set number of packets.	Fully implemented
Ethernet Header Parsing: Extract and display the Destination MAC, Source MAC, and EtherType from the raw hex data.	Fully implemented
ARP Header Parsing: When the EtherType indicates ARP (0x0806), extract and display all required ARP header fields in the correct format.	Fully implemented
IPv4 Header Parsing: When the EtherType indicates IPv4 (0x0800), extract and display the required IPv4 header fields in the correct format.	Fully implemented
UDP Header Parsing: For IPv4 packets where the Protocol field is UDP (17), extract and display the required UDP header fields in the correct format.	Fully implemented
ICMP Header Parsing: For IPv4 packets where the Protocol field is ICMP (1), extract and display the required ICMP header fields in the correct format.	Fully implemented
TCP Header Parsing: For IPv4 packets where the Protocol field is TCP (6), extract and display the required TCP header fields in the correct format.	Fully implemented
DNS Header Parsing: For IPv4 packets where the Protocol field is UDP (17), extract and display the required UDP and DNS header fields in the correct format.	Fully implemented
Testing with Local PCAP Files: The program supports reading packets from saved pcap files using Scapy's rdpcap() function.	Fully implemented

Platforms

This program has been tested on:

- Ubuntu 2024.04 LTS

Language

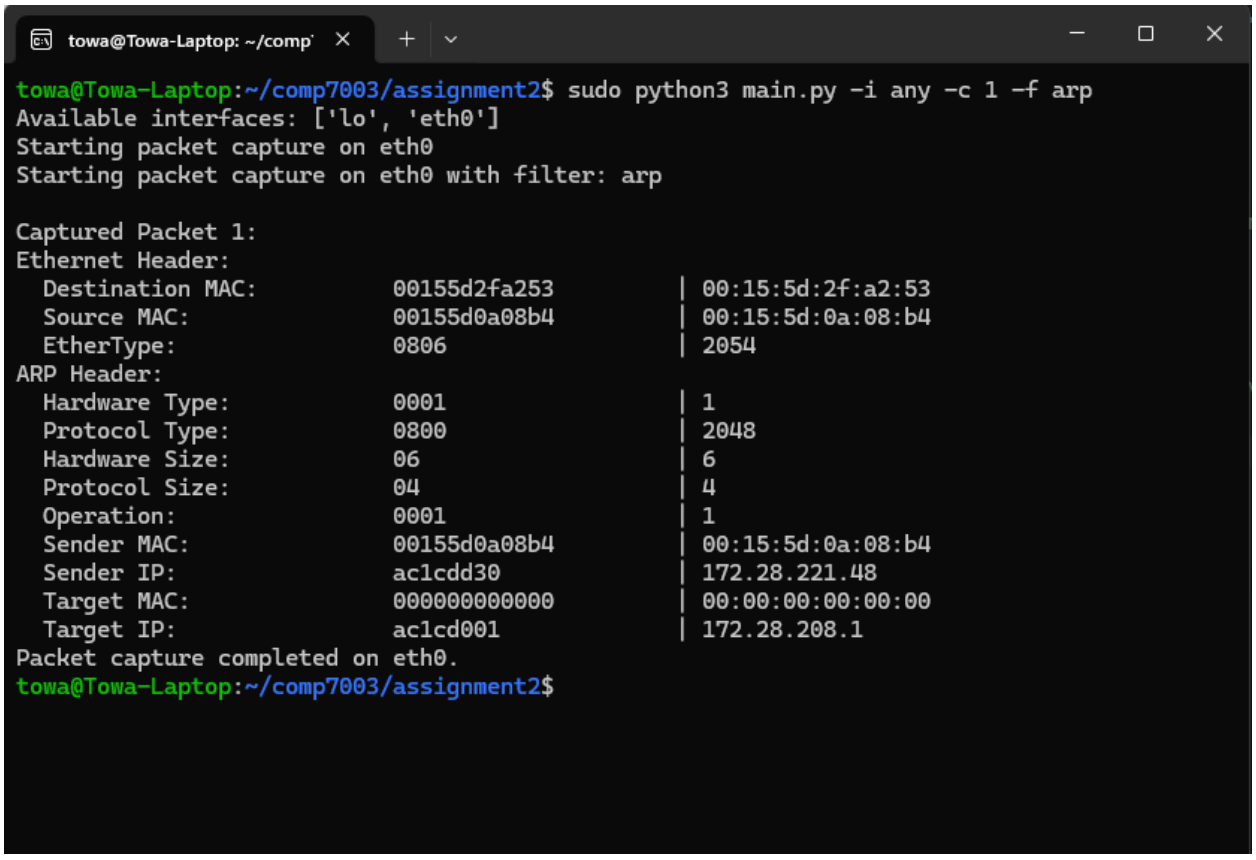
- Python 3.12.3
- Compiles with GCC 13.3.0 on Linux

Documents (Located in /report Folder)

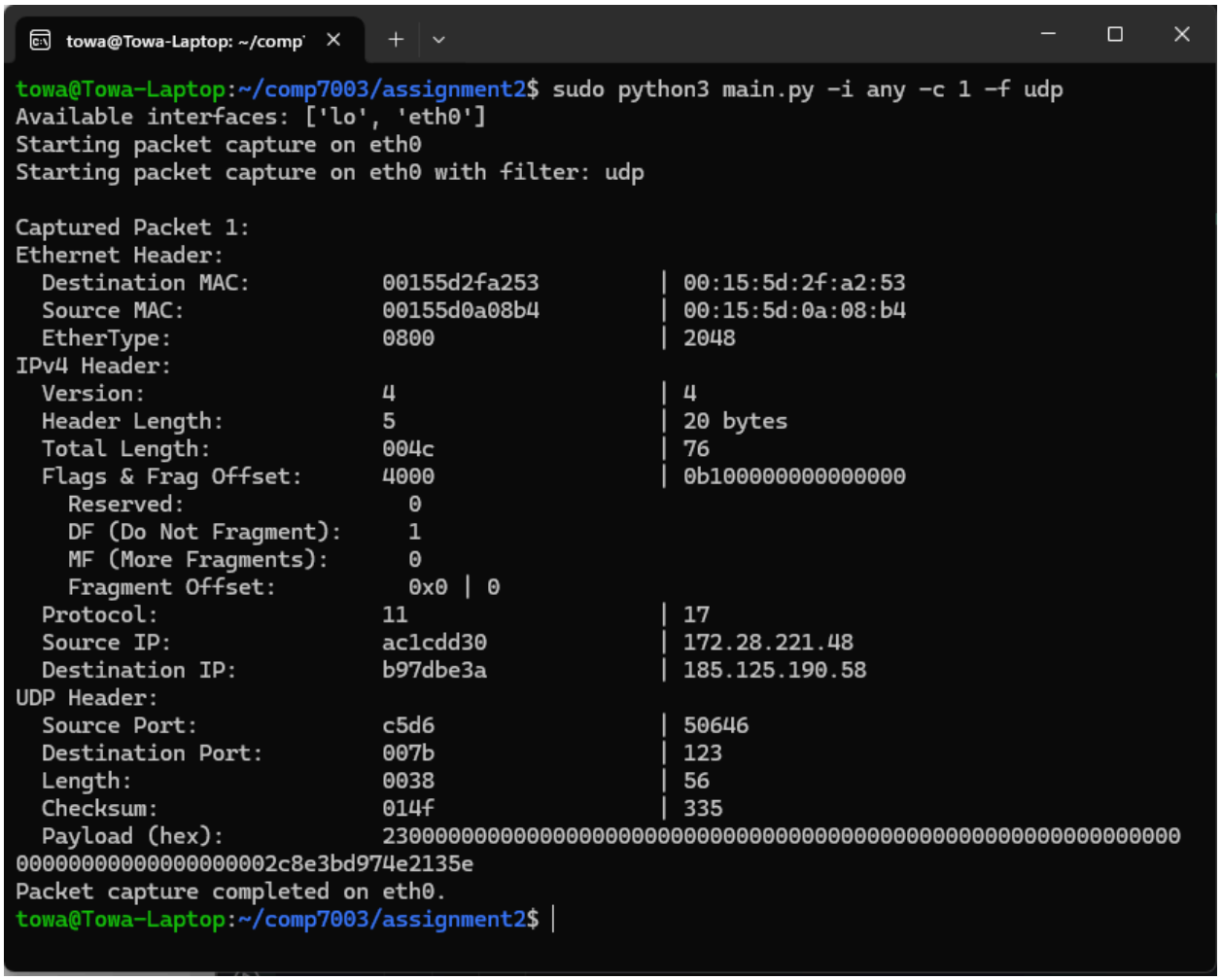
- Design
- Testing
- User Guide

Findings

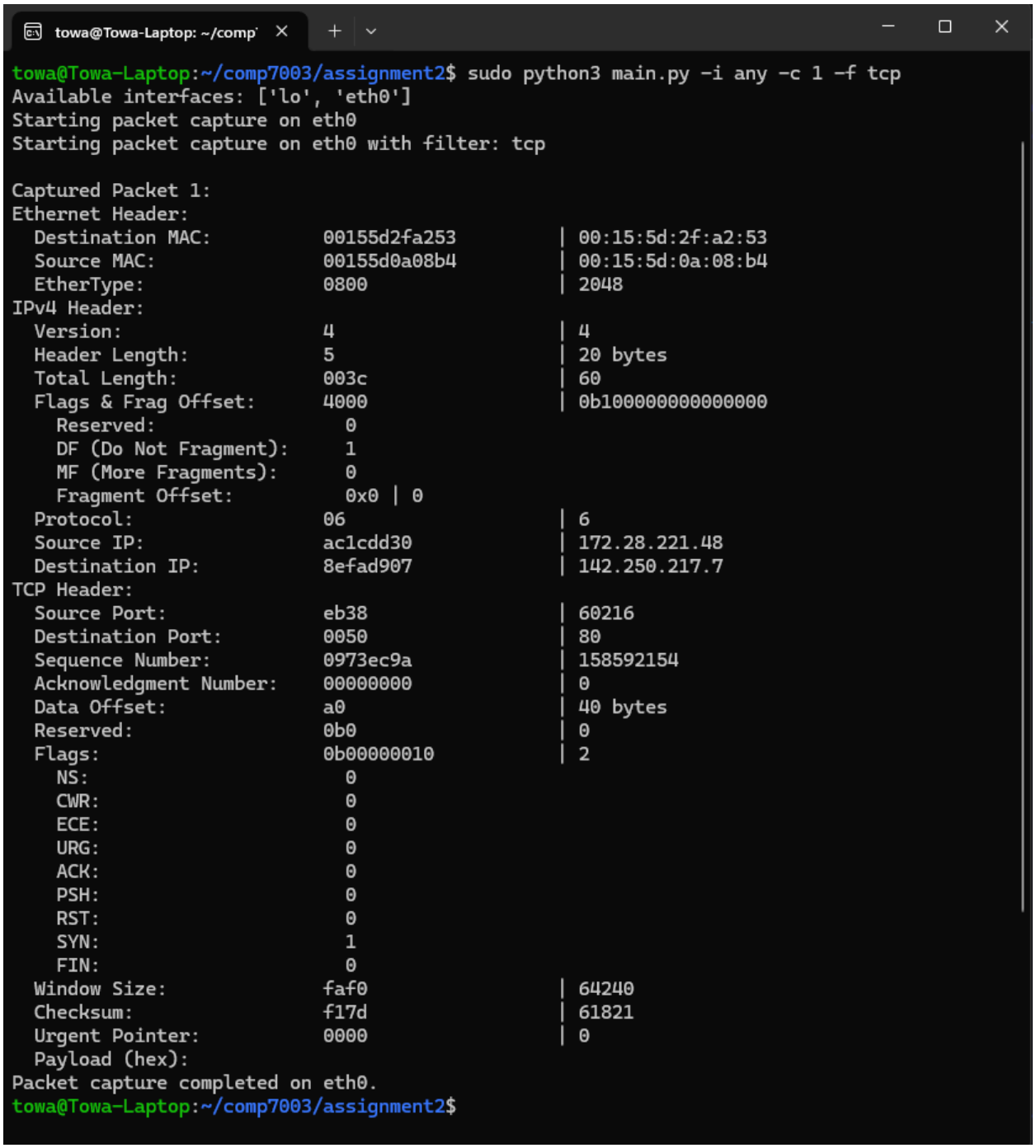
This is an example output after running the program for capturing 1 packet filtered by ARP. As we can see, it first scans for the available interfaces since we didn't specify them in the CLI arguments. Then it starts to capture the packet information including the Ethernet Header addresses and ARP Header addresses broken down into multiple segments. The EtherType shows a value of 0806 which indicates an ARP protocol.



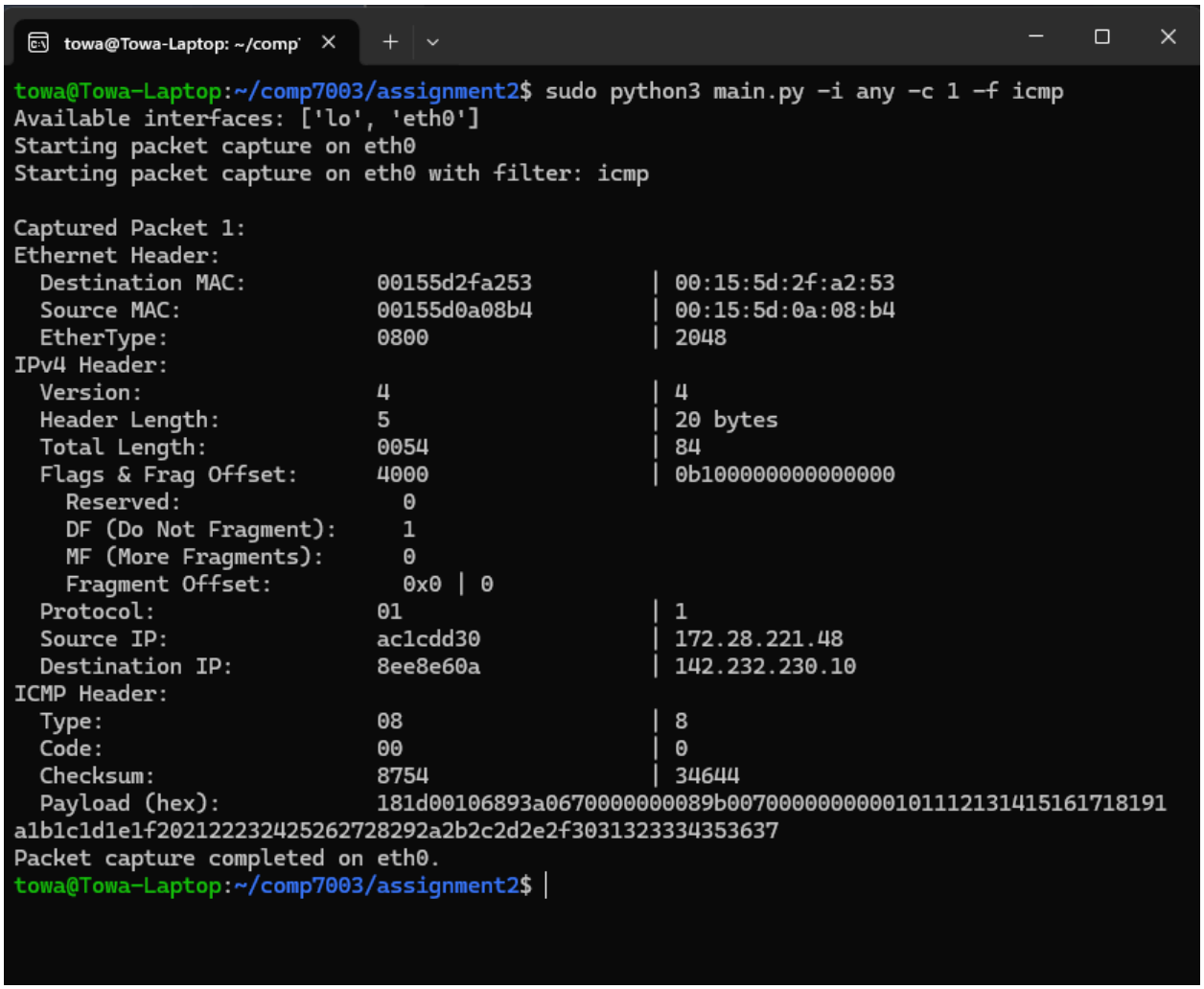
This second example is the output after running the program for capturing 1 packet filtered by UDP. As with the first output, it first scans for the available interfaces since we didn't specify them in the CLI arguments. Then it starts to capture the packet information including the Ethernet Header addresses, IPv4 Header addresses, and UDP Header addresses broken down into multiple segments. The EtherType shows a value of 0800 which indicates an IPv4 protocol and then the IPv4 header shows a protocol value of 17 in decimal which indicates UDP.



This third example is the output after running the program for capturing 1 packet filtered by TCP. As with the first output, it first scans for the available interfaces since we didn't specify them in the CLI arguments. Then it starts to capture the packet information including the Ethernet Header addresses, IPv4 Header addresses, and TCP Header addresses broken down into multiple segments. The EtherType shows a value of 0800 which indicates an IPv4 protocol and then the IPv4 header shows a protocol value of 6 in decimal which indicates TCP.



This fourth example is the output after running the program for capturing 1 packet filtered by ICMP. As with the first output, it first scans for the available interfaces since we didn't specify them in the CLI arguments. Then it starts to capture the packet information including the Ethernet Header addresses, IPv4 Header addresses, and ICMP Header addresses broken down into multiple segments. The EtherType shows a value of 0800 which indicates an IPv4 protocol and then the IPv4 header shows a protocol value of 1 in decimal which indicates ICMP.



This last example is the output after running the program for capturing 1 packet filtered by DNS through Port 53. As with the first output, it first scans for the available interfaces since we didn't specify them in the CLI arguments. Then it starts to capture the packet information including the Ethernet Header addresses, IPv4 Header addresses, UDP Header addresses and DNS Header addresses broken down into multiple segments. The EtherType shows a value of 0800 which indicates an IPv4 protocol and then the IPv4 header shows a protocol value of 17 in decimal which indicates UDP, and the DNS traffic was able to be successfully detected. (The screenshot below is from a Mac, previous screenshots were from a Windows)

```
1 -f "src port 53"
[Password:                                                                    ]
Available interfaces: ['lo0', 'gif0', 'stf0', 'anpi1', 'anpi0', 'en3', 'e
n4', 'en1', 'en2', 'bridge0', 'ap1', 'en0', 'awdl0', 'llw0', 'utun0', 'ut
un1', 'utun2', 'utun3', 'utun4', 'utun5', 'utun6', 'utun7']
Starting packet capture on en0
Starting packet capture on en0 with filter: src port 53

Captured Packet 1:
Ethernet Header:
  Destination MAC:          822b644d2756         | 82:2b:64:4d:27:56
  Source MAC:               d007ca53d6a0         | d0:07:ca:53:d6:a0
  EtherType:                0800                 | 2048
IPv4 Header:
  Version:                  4                    | 4
  Header Length:            5                    | 20 bytes
  Total Length:             0053                 | 83
  Flags & Frag Offset:      0000                 | 0b0
    Reserved:                 0
    DF (Do Not Fragment):     0
    MF (More Fragments):      0
    Fragment Offset:          0x0 | 0
  Protocol:                 11                   | 17
  Source IP:                8ee84cc8             | 142.232.76.200
  Destination IP:           0a4123c8             | 10.65.35.200
UDP Header:
  Source Port:              0035                 | 53
  Destination Port:         d099                 | 53401
  Length:                   003f                 | 63
  Checksum:                 e0b1                 | 57521
  Payload (hex):            0e8d8180000100010000000106676f6f676c6503636f6
d0000010001c00c000100010000000100048efb294e0000291000000000000000
Detected DNS traffic
DNS Header:
  Transaction ID:           0e8d
  Flags:                    0b1000000110000000 | 33152
  Questions:                1
  Answers:                  1
  Authority RRs:            0
  Additional RRs:           1
  Payload (hex):            06676f6f676c6503636f6d0000010001c00c000100010
000000100048efb294e0000291000000000000000
Packet capture completed on en0.
```