Network Sniffer

Created by:

- Ahmed Halat (1006332951, halatahm)
- Mohamed Halat (1006322962, halatmoh)
- Armand Sarkezians (1006020574, sarkezi1)

for CSCD58 Winter 2022.

Running the application

First, make sure you read the readme's in ns-ui and ns-api. They will walk you through installing all of the dependencies and setting up the environment.

Then, to run the application follow these steps:

```
cd ns-ui
npm run start

cd ../ns-api
flask run

# Go to http://localhost:4200/ in your browser
```

Description and Goals

Our initial description of this project was simple. We wanted to make a simple Wireshark clone; a tool to read packets in from our device and parse them for data. We wanted to add a few features onto this, including a whois lookup, arp table lookup, and nmap scan. We also wanted to make the UI as simple and easy to use as possible.

Our goals as outline in the proposal were as follows:

- Create a simple UI that allows users to view packets
- Allow users to view the raw data of the packets
- Allow users to view the arp data of their device
- Allow users to view the whois data of all IP's on their arp table
- Allow users to view data regarding their network (achieved through nmap scan)
- Create dropdowns that allow our users to select which functionality they want to use

Contributions

Ahmed Halat

I mainly worked on getting the backend and socketIO server working. I also helped with writing the packet sniffing code in the backend and created the UI components in the frontend that render the formatted packet headers and RAW packet data in a human readable format.

Mohamed Halat

I worked on a few different parts of the project. I created the flask app that runs the backend and the socketIO server. I also created and designed most of the frontend. I also was responsible for data parsing for whois and arp data. The most challenging thing that I worked on was definitely setting up a socketIO server and getting it to work with the frontend and adding the ability to pause and resume the packet sniffing.

Armand Sarkezians

I worked primarily on the backend, but did some setup on the frontend to enable my teammates early on. I worked on the packet sniffing, arptables, and whois code, working with Ahmed to create sniffing capabilities using the socketIO library. I wrote all files in the folders `structures` and `helpers`, creating an object-oriented approach to organizing the different protocols we read from sniffing packets. On the frontend, I built the project initially and created several components for the landing page (header, title, dropdowns for each functionality, etc.). I also came up with the name (super creative I know).

Running the Sniffer

The runner is divided into 2 sections, a backend and a frontend. The backend is responsible for sniffing the network and the frontend is responsible for displaying the data, each in their own terminal.
Instructions for setting up and running each are available in the READMEs in their respective directories. The backend runs best on a Linux machine, while the frontend can be run on any machine.

Implementation and Documentation

Backend

We used a few different tools to implement this project. The backend is written in python and runs with Flask and SocketIO.

Sniffing

In order to perform packet sniffing on the backend, we used the Recvfrom function from the python socket library. This function allows us to receive data from a socket, and returns the data and the address of the sender.

The raw data from the socket is used to initialize the Ethernet Object we created. This object is parses the Ethernet header data and stores the rest of the raw object data.
We then initialize the IP Object and pass it the raw data from the Ethernet object. This object parses the IP header data and stores the rest of the raw object data. We then use the protocol number from the IP header to

determine which protocol to initialize next. We initialize either the TCP Object, UDP Object, or ICMP Object and pass it the raw data from the IP object. This object parses the protocol header data and stores the rest of the raw object data and is finally returned to the frontend through the SocketIO connection.

Sniffing is only turned on when there is a connection on the SocketIO server and the frontend sends a api request to the backend to start sniffing. This allows us to pause and resume sniffing without having to close the socket and open a new one. Sniffing is also paused when the frontend socket is disconnected to prevent the backend from running indefinitely.

Arptables, Whois and Nmap

The implementation for fetching ARPTables, whois and nmap information is very similar. We use the python subprocess library to run the respective commands and return the output.
In the case of ARPTables and Whois, we also use re to parse the output and return it in a more readable format. This required building out custom regex patterns for each command as seen in the Whois Object.

Frontend

Our frontend is built using Angular, a popular Typescript framework. The frontend UI is mainly designed and built by us, with a few components from PrimeNG, a popular UI library for Angular. We also used NGX-SocketIO to connect to the SocketIO server on the backend.

Analysis, Implementation and Lessons Learned

Through the use of Object Oriented Programming, we were able to create a modular and extensible codebase that is easy to maintain and expand upon. It also required that we think about the structure of the data we were working with and how to parse the required fields from the raw data into our frame/packet objects.

We also learned how to use SocketIO to create a real time connection between the frontend and backend. We took packets/second into consideration when deciding whether to use a polling or websocket connection. We decided to use a websocket connection as it is more efficient and allows for a more responsive UI. We also learned how to use the SocketIO library to pause and resume sniffing without having to close and open a new socket and how to take advantage of multiple threads to run the sniffing and SocketIO server concurrently.

We did also try to setup a REDIS server to save packets into a database and allow them to be loaded into the frontend on reload. However, the number of packets created significant slowdown in the server (when read/write) and we were unable to get this working in time for the deadline.

Overall, our implementation of an Object Oriented approach to packet sniffing and other network details allowed us to gain a deeper understanding of how the internet works and how to parse and display the data we receive from the network.

Conclusion

In conclusion, this project was super fun for all of us, we learned a lot during its creation and we hope you enjoy using/grading it as much as we enjoyed making it.

Thanks!