Gazi University Faculty of Engineering

Project final report of the pack sniffer

Work made by:
Diego de Luis Ballesteros, ST NO: 23118080904, diegodeluis@gazi.edu.tr

Course:
Bilgisayar Ağları (BM402)

Academic year:
2023/2024

Index

Summary 3
Introduction 3
Material methods 4
Python version and libraries 4
First part: sniffer.py 5
Second part: info2.py and selenium1.py 5
How the code works 5
First part 5
Second part 6
How to run the code 7
Conclusion 8

Summary

This project develops a packet sniffer program to capture and analyze network packets. The program inspects Ethernet frames to identify IP and transport layer protocols (UDP, TCP, ICMP) using a Python script for real-time packet collection.

Key Points:

- Captures and categorizes network packets to understand communication protocols.
- Uses a Python script for real-time packet analysis.
- Analyzes UDP packets to reconstruct web browser displays (e.g., Google Chrome, Firefox).

Conclusions:

- Demonstrates effective network packet analysis.
- Provides insights into network protocols and user experience.

Recommendations:

- Expand protocol analysis capabilities.
- Enhance security features for threat detection.
- Integrate with other network monitoring tools for comprehensive analysis.

Introduction

This project involves the development of a sophisticated network monitoring program designed to capture and analyze network packets, commonly referred to as a packet sniffer. At its core, the program intercepts and examines each packet transmitted through the network access layer, specifically focusing on Ethernet frames. By doing so, it identifies and categorizes the IP protocols and transport layer protocols in use, such as UDP (User Datagram Protocol), TCP (Transmission Control Protocol), and ICMP (Internet Control Message Protocol).

The packet sniffing process is orchestrated by a Python script that utilizes a socket to collect packets in real-time. This script is the backbone of the project, enabling the detailed inspection and classification of network traffic. The ability to dissect each packet allows for a deeper understanding of the underlying communication protocols and the nature of the data being transmitted.

Beyond merely capturing and categorizing network packets, the project extends to a more advanced analysis phase. In the second part, the focus shifts to UDP packets specifically sourced from the internet. These packets are meticulously analyzed to extract valuable information. The goal is to use this information to reconstruct a visual representation of what would be displayed in a web browser such as Google Chrome, Firefox, or others. This capability is particularly useful for understanding and visualizing the data flow and the resulting user experience on the web.

Overall, this project not only showcases the practical implementation of network packet analysis using Python but also highlights the importance of understanding network protocols and their applications in real-world scenarios. By bridging the gap between raw network data and its visual representation, the project provides valuable insights into the behavior of internet traffic and the functioning of web browsers

Material methods

Now I will explain everything related to the technical computational aspects of the project

Python version and libraries

I used Python 3.8 for this project. The libraries I used were divided into two groups. For the first packet sniffer, an important library I used was socket, which collects the Ethernet frames from the network. For the second part, I used a couple of main libraries. I implemented code from Scapy, which helps create a simpler packet sniffer (used for UDP), and Selenium. The main function of Selenium is to automate the browser, but it can also convert some information from the browser into a screenshot.

First part: sniffer.py

```
import socket
import struct
import textwrap
```

Second part: info2.py and selenium1.py

```
from scapy.all import *
from threading import Timer
import os
```

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from datetime import datetime
import os
```

How the code works

First part

Sniffer.py

This code is a Python script designed to capture and analyze network traffic. Here's how it works:

1. **Socket Creation**:

```
# Create a socket to capture Ethernet frames
conn=socket.socket(socket.AF_PACKET,socket.SOCK_RAW,socket.ntohs(3))
```

2. **Traffic Capture**:

```
# Receive raw data and address
raw_data, addr = conn.recvfrom(65535)

# Unpack the Ethernet frame into its components
dest_mac, src_mac, eth_proto, data = ethernet_frame(raw_data)
```

3. **Ethernet Frame Unpacking**:

```
def ethernet_frame(data):
    """
    Function to unpack an Ethernet frame.

    Args:
        data (bytes): Ethernet frame data.

    Returns:
        tuple: Tuple with Ethernet frame information.
    """
    dest_mac, src_mac, proto = struct.unpack('! 6s 6s H', data[:14])

    return get_mac_addr(dest_mac), get_mac_addr(src_mac), socket.htons(proto), data[14:]
```

4. **Protocol Analysis**:

```python
def ipv4_packet(data):
    version_header_length = data[0]

    #we translate to binary from hex
    version_header_length = bin(int(version_header_length, 16))[2:]

    #we translate to int to use the operator >>
    version = int(version_header_length) >> 4

    header_length = (int(version_header_length) & 15) * 4
    ttl, proto, src, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
    return version, header_length, ttl, proto, ipv4(src), ipv4(target), data[header_length:]


def ipv4(addr):
    return '.'.join(map(str, addr))


# Unpacks ICMP Packet
def icmp_packet(data):
    icmp_type, code, checksum = struct.unpack('! B B H', data[:4])
    return icmp_type, code, checksum, data[4:]
```

5. **Output Formatting**: The extracted information is then printed in a structured format, including Ethernet frame details, IPv4 packet details, and protocol-specific details like ICMP type, TCP flags, or UDP data.

Overall, this script serves as a basic network packet analyzer, capable of dissecting Ethernet frames and providing insights into the various protocols encapsulated within them.

Second part

Info2.py

This code is designed to sniff UDP traffic, specifically DNS requests, using Scapy, a Python library for packet manipulation. Here's how it works:

1. **UDP Sniffing Function**

   We will analyze DNS UDP packets and the url stored in the payload. It should be sent to a file called urls.txt.

```python
def udp_sniff(packet):
  # Check if packet has UDP layer and is DNS request
  if packet.haslayer(DNS) :
    src_ip = packet[IP].src
    dst_ip = packet[IP].dst
    src_port = packet[UDP].sport
    dst_port = packet[UDP].dport

    # Extract hostname from DNS layer
    try:
      hostname = packet[DNS].qd.qname.decode('utf-8')
    except UnicodeDecodeError:
      hostname = "(Decoding failed: unknown encoding)"


    print(f" source IP: {src_ip}")
    print(f" destination IP: {dst_ip}")
    print(f" source port: {src_port}")
    print(f" destination port: {dst_port}")
    print(f" Hostname: {hostname}")
    if (hostname.startswith("www") and not(hostname.endswith('tr'))) or hostname.startswith("es") or hostname.startswith("wiki"):
            hostname = "https://" + hostname
            if hostname.endswith('.'):
                hostname = hostname[:-1]
            with open("urls.txt", "a") as file:
                file.write(hostname + '\n')

  else:
    # Optional: Print a message for non-DNS packets
    pass
```

2. **Main Function**:

   The main function sets up packet sniffing on interface "enp0s3" to capture UDP traffic specifically for DNS requests. It executes this process asynchronously with a 12-second timeout.

selenium1.py

This code uses Selenium to take screenshots of web pages. The capture_screenshot function takes a URL saved by the other program, opens the page, and saves the screenshot. The main function iterates through a list of URLs and calls capture_screenshot for each one.

How to run the code

Requirements for the first program:

- Pip install socket

And then you should do: **sudo python sniffer.py**

For the second program, just place the folder “sniffer” on your main user home (cd $home), you go to sniffer folder (cd sniffer) and run the script (./start.sh). It will install all the dependencies and start the program. While the first script is running, you should enter in your browser and go to some webpages (make some traffic).

```
#!/bin/bash


pip install virtualenv
cd $home
sudo apt install python3.10-venv
python3 -m venv myenv
source myenv/bin/activate
pip install scapy
pip install selenium
cd sniffer

rm *.png
sudo rm urls.txt
sudo -E $HOME/myenv/bin/python $HOME/sniffer/info2.py


sudo chmod u+wrx urls.txt

$HOME/myenv/bin/python selenium1.py
```

Conclusion

The packet sniffer project successfully demonstrates the capability to capture and analyze network packets in real-time using a Python script. By inspecting Ethernet frames and identifying key IP and transport layer protocols, the project offers significant insights into network traffic. The advanced analysis of UDP packets to reconstruct web browser displays further underscores its utility in understanding user experience and data flow on the internet.

Key accomplishments include:
- Effective real-time packet capture and protocol analysis.
- Enhanced understanding of network communication protocols.
- Visualization of web traffic data to interpret user interactions.

Future recommendations include expanding the protocol analysis capabilities, enhancing security features to detect potential threats, and integrating the packet sniffer with other network monitoring tools for a more comprehensive analysis.