Assignment title: - Network Sniffer

Module Name: - ST4061 Programming and Algorithms 1

BSc. (hons) Ethical Hacking and Cyber Security,

Softwarica Collage of IT and E-commerce

Coventry University

Intake:

March 2023

Collage id: 230116

Coventry id: 14187632

Submitted by: -

Shreya Joshi

Batch 34

Submitted to: -

Suman Shrestha

ACKNOWLAGMENT

Before starting anything else, I would like to convey my gratitude to Softwarica Collage for providing me with a formidable chance to work on my project report, which is centered on developing a Networker Sniffer using Python. Suman Sir, my instructor, was immensely helpful and gave me valuable advice. I also want to express my gratitude to my friends who helped me improve my program and recommend ideas.

ABSTRACT

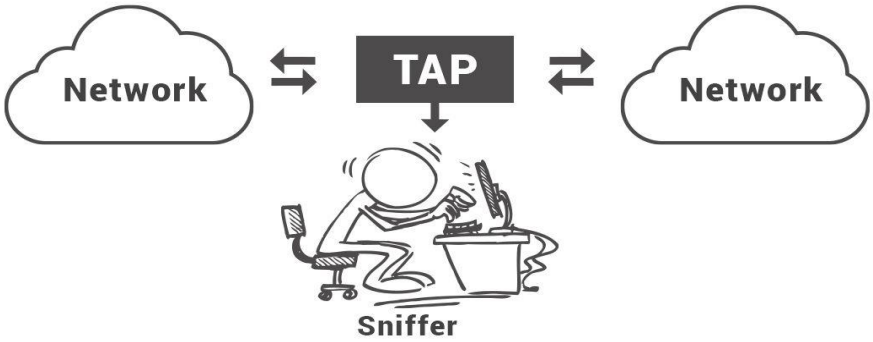
The goal of the network sniffer project is to create a tool that can analyze and monitor network traffic. It utilizes the programming language python and various libraries to analyze and intercept packets, which can provide valuable insight into how networks work. In its promiscuous mode, the tool can access the local network segment and collect packets from the source. It can then extract information such as the destination and origin IP addresses, protocols, timing, and payload. In addition, it has features that allow it to filter and display specific types of packets. The network sniffer project was developed with the help of python's versatile capabilities, allowing it to provide a variety of customizable solutions for network analysis, security, and troubleshooting.

TABLE OF CONTANT

INTRODUCTION

A network sniffer is a tool that can perform real-time analysis and interpretation of network traffic. It can examine and intercept packets to gain a deeper understanding of how communication works within a network. In addition to collecting information about the origin and destination of the packets, network sniffing can also extract other details such as the protocols and timing data.

In addition to serving various functions in the realm of security analysis and network administration, network sniffing also comes in handy when it comes to identifying and resolving network issues. This tool can be used by administrators, network sniffing also comes to identifying and revolving network issues. Administrators can use this tool to monitor and analyze the performance of the network bottlenecks and improve its efficiency. It can additionally detect and diagnose network problems, such as poor configuration or packet loss.



When it comes to preventing and detecting threats, network sniffing is especially important. Security personnel can use it to identify anomalous activities and breaches in the network. It can also reveal hidden vulnerabilities and provide valuable insight into how network work.

Before performing network sniffing, security personnel should first obtain the proper consent and authorization. Doing so will ensure that the activities are conducted in a responsible manner and the privacy rights of the users.

For security analysis, optimization, and troubleshooting purposes, network sniffing is a useful tool. Responsible use of this technique will let administrators gain profound insight int a computer network's security and functioning.

Note: A network sniffer is a type of tool sometimes referred to as a wireless sniffer, an Ethernet sniffer, a packet analyzer, or simply a snoop.

BACKGROUND

A network mapping process is a process that involves mapping the various components and services of a computer network. It helps identify and map out the devices and services that are part of the network. Administrators can use network mapping to plan for the expansion of their networks, identify and map out the various devices and services, and resolve connectivity issues. In addition, it helps them identify potential security threats. Tools for automating scanning and providing actionable intelligence are network mapping tools.

PROBLEM STATEMENT

The network sniffer software can be used in numerous ways in real-world scenarios. It can assist in identifying and preventing unauthorized access to the network, detecting data breaches and malware, and improving defenses against it. In addition, it can help in troubleshooting issues related to network performance.

Administrators can use the tool to analyze and monitor the usage patterns of their networks, identify bottlenecks, and plan the capacity of their networks. It can also help them comply with regulations by analyzing and capturing network traffic.

OBJECTIVE OF NETWORK SNIFFER

A network sniffer's intent is to gather and examine network packets in-order-to understand network behavior, solve problems, improve security, boost performance, and enable efficient network management. A network sniffer tries to give helpful insight that assists with improving the efficiency of networks, finding, and fixing issues, guaranteeing network security, and offering effective network management by collecting and evaluating network data.

SCOPE OF NETWORK SNIFFER

Network sniffers have a broad scope that includes capturing and analyzing network packets, monitoring traffic, troubleshooting issues, optimizing performance, enhancing security, and supporting compliance efforts.

LITERATURE REVIEW

Through studies on network sniffing, researchers have been able to gain a deeper understanding of how it works. For instance, one study analyzed the features and performance of Java, C, and python-based techniques. Another investigation explored the use of Scapy and python in network packet analysis.

Through studies on network sniffing, researchers have been able to gain a deeper understanding of how it works. They have also been able to integrate machine learning techniques into it to improve its capabilities. Some of the ethical and privacy considerations that engage in network sniffing have been brought to the attention of researchers.

Different programming languages' implementation of network sniffing have been compared, considering factors like the processing efficiency and packet capture rate. These studies, which are part of a large body of research, provide valuable information on various aspects of network sniffing.

METHODOLOGY

The approach taken by a network sniffer involves the capture of network packets within a designated local network section, followed by the examination of packet headers to extract crucial information like the source and destination IP addresses, protocols utilized, and timestamps. Moreover, the sniffer analyzes the packet contents to retrieve the transmitted data and enables the filtration of packets based on specific criteria. Additionally, it monitors and evaluates network performance indicators such as packet loss and latency, facilitating the identification of issues and the optimization of network resources. In terms of security analysis, network sniffers can detect suspicious network activities, unauthorized access attempts, and potential intrusions. To enhance understanding and facilitate decision-making, network sniffers often provide reporting and visualization features that present the captured data in a concise and comprehensible manner.

THE ALGRODHM

Step 1: Import the necessary modules from the Scapy library.

Step 2: Define the packet_callback function to extract and display packet information.

Step 3: Define the capture_packets function:

i. Use the sniff() function with the prn parameter set to the packet_callback function.
ii. Add a stop_filter parameter to specify the condition for stopping packet capture.
iii. Inside the packet_callback function, check if the packet has an IP layer and desired protocol (e.g., TCP) before displaying the packet information.

Step 4: Define the filter_packets function:

i. Use the sniff() function with the prn parameter set to the packet_callback function.
ii. Add an lfilter parameter to filter packets based on the desired protocol.
iii. Inside the packet_callback function, check if the packet has an IP layer and matches the specified protocol before displaying the packet information.

Step 5: Define the obtain_consent function to prompt the user for consent to capture packets.

Step 6: In the main program:

i. Check if the user gives consent using the obtain_consent function.
ii. If consent is given, call the capture_packets function to start capturing packets and display the information.
iii. If consent is not given, display a message indicating the lack of consent.

SOURSE CODE

```python
from scapy.all import *

# Packet capturing callback function
2 usages
def packet_callback(packet):
    # Extract packet information
    if packet.haslayer(IP):
        src_ip = packet[IP].src
        dest_ip = packet[IP].dst
        proto = packet[IP].proto
        payload = packet.payload

        # Display packet information
        print(f"Source IP: {src_ip} | Destination IP: {dest_ip} | Protocol: {proto} | Payload: {payload}")

# Capture network packets
1 usage
def capture_packets():
    sniff(prn=packet_callback, stop_filter=lambda packet: packet.haslayer(IP) and packet[IP].proto == 6, store=0)

# Filter packets and display specific types
def filter_packets(packet_type):
    sniff(prn=packet_callback, lfilter=lambda packet: packet.haslayer(IP) and packet[IP].proto == packet_type, store=0)

# Obtain proper consent and authorization
1 usage
def obtain_consent():
    consent = input("Do you consent to capture network packets? (yes/no): ")
    if consent.lower() == "yes":
        return True
    else:
        return False

# Main program
if __name__ == "__main__":
    if obtain_consent():
        capture_packets()
    else:
        print("Consent not given.NBSPExiting...")
```

OUTPUT:

```
Do you consent to capture network packets? (yes/no): yes
Source IP: 192.168.1.69 | Destination IP: 224.0.0.251 | Protocol: 17 | Payload: IP / UDP / DNS Ans "[b'bt=1C:D6:BE:78:C5:B1']"
Source IP: 18.157.122.248 | Destination IP: 192.168.1.82 | Protocol: 6 | Payload: IP / TCP 18.157.122.248:https > 192.168.1.82:65443 A
```

PROCEDURE

1.

```
from scapy.all import *
```

Scapy is a library in python that allows you to analyze and collect network data. The program's code defines various functions, such as extracting information, filtering specific types of packets, getting user consent, and displaying a message if the user refuses to give.

2.

```
# Packet capturing callback function
2 usages
def packet_callback(packet):
    # Extract packet information
    if packet.haslayer(IP):
        src_ip = packet[IP].src
        dest_ip = packet[IP].dst
        proto = packet[IP].proto
        payload = packet.payload

        # Display packet information
        print(f"Source IP: {src_ip} | Destination IP: {dest_ip} | Protocol: {proto} | Payload: {payload}")
```

The provided code encapsulates a function that is used to process and capture network packets. It first checks if the packet is equipped with an IP layer, and it retrieves information related to its destination, protocol, and source IP. The extracted data is then displayed using a print statement. The code also displays the various details of the network packet, such as its payload content and origin IP address.

3.

```
# Capture network packets
1 usage
def capture_packets():
    sniff(prn=packet_callback, stop_filter=lambda packet: packet.haslayer(IP) and packet[IP].proto == 6, store=0)

# Filter packets and display specific types
def filter_packets(packet_type):
    sniff(prn=packet_callback, lfilter=lambda packet: packet.haslayer(IP) and packet[IP].proto == packet_type, store=0)
```

The code included in this package contains two functions that are related to network filtering and packet capture. The first one is called `capture_packets()` and is made use of the Scapy Library's `sniff()` function. The second function is called `packet_callback()` to extract and display information about the packets.

The packet also includes a stop filter that can prevent packet capture when the connection between an IP layer and the TCP protocol is encountered. The other function, `filter_packets()`,

takes advantage of the same function but with a different filter. It displays and extracts information about the packets based on their specific type.

The store parameter of the two functions is set to zero, which indicates that they should not be kept in the memory of the system. They allow network administrators to filter and capture packets according to their specific requirements.

4.

```
# Obtain proper consent and authorization
1 usage
def obtain_consent():
    consent = input("Do you consent to capture network packets? (yes/no): ")
    if consent.lower() == "yes":
        return True
    else:
        return False
```

The code snippet provides a function that asks the user to provide consent for the network packet capture process. It displays a question with input() and expects the user to either answer “yes” or “no.” The function compares the response with the letter “yes” or “no.” The function compares the response with the letter “yes” if it is converted to lowercase.

The function returns true if the answer is “yes” and false if it is not. Its purpose is to ensure that the user gives the necessary consent before starting the packet capture process.

5.

```
# Main program
if __name__ == "__main__":
    if obtain_consent():
        capture_packets()
    else:
        print("Consent not given. Exiting...")
```

The main program section of the code checks if the script is being executed as the main program. If so, it calls the `ontain_consent()` function to determine if the user has given consent for capturing network packets. If consent is obtained, the `capture_packets() function is executed to start capturing packets. On the other hand, if the user does not give consent, a message stating “Consent not given. Exiting…” is printed, and the program terminates. This section ensures that the network packet capturing process is executed only when the user provides proper consent.

Output:

The program's code snippet asks the user whether they want to be part of the process of capturing network messages. The user's response was a resounding "yes." The program then displays two network messages. The first shows the destination and origin IP addresses, as well as the protocol used, which is UDP 17. The second payload contains a DNS query for a Bluetooth device's information. The second network packet displays the origin and destination IP addresses, as well as the protocol used. It includes a TCP packet from the source to the destination.

FUTURE IMPROVEMENT

In the future, the given code may be improved by implementing error handling and exceptions management. The current state of the code assumes that users will always be asked to input “yes” and “no” before proceeding. But there is no indication that there is a procedure in place to oversee errors that are unexpected or incorrect.

One of the most crucial factors that can be considered when it comes to enhancing the code is implementing error handling techniques, such as try-except blocks. This method can help catch potential errors and provide the appropriate prompts or messages for correct input. It can also improve the program’s overall user experience.

CONCLUSION

The network sniffing program in python can be used to analyze and monitor network traffic. It can do so by leveraging the socket library, which allows it to collect various details about the network, such as the protocol used, the destination IP address, and the payload’s contents. It comes with customizable filters that can be used to identify packets and present relevant data. Security experts and network administrators can utilize this program to conduct security analyses and optimize the performance of their networks. The program can be further improved by adding support for different protocols, enhancing its filtering capabilities, and handling errors. It helps network administrators improve their understanding of their networks’ operations and control them properly.

REFERANCE

Https://www.lifewire.com/definition-of-sniffer-817996. (2004, June 27). *What is a network sniffer?* Lifewire. https://www.lifewire.com/definition-of-sniffer-817996

EC-Council. (2022, June 12). *What are sniffing attacks, and how can you protect yourself?* Cybersecurity Exchange. https://www.eccouncil.org/cybersecurity-exchange/ethical-hacking/what-are-sniffing-attacks/

Attention required! (n.d.). Attention Required! | Cloudflare. https://www.pagerduty.com/resources/learn/what-are-network-sniffers/

Network sniffer. (2023, March 14). EDUCBA. https://www.educba.com/network-sniffer/