

CYBER SECURITY

DIGITAL ASSIGNMENT 1

SUBMITTED BY:

Siddharth Bhardwaj - 21BCE2001

Rashmi M T - 21BCE0174

INTRODUCTION

Overview of Packet Sniffing:

Packet sniffing is the practice of capturing data packets as they travel over a network, enabling the analysis of network traffic for monitoring and troubleshooting. In cybersecurity and network management, packet sniffing plays a critical role, providing insights into the structure and content of network traffic.

Packet sniffing is a critical process in network analysis that involves capturing data packets as they traverse a network. This practice serves as a fundamental tool for network monitoring, diagnosis, and security assessments. Packet sniffers, such as **Scapy** and **Wireshark**, are designed to intercept these packets, providing detailed insights into network traffic and enabling cybersecurity professionals to identify vulnerabilities, potential threats, or performance issues.

Importance and Applications of Packet Sniffing

Packet sniffing holds a vital role in cybersecurity and network management. It helps IT specialists:

- **Monitor Network Health:** Ensuring data packets flow smoothly and identifying issues like congestion or dropped packets.
- **Troubleshoot Problems:** Diagnosing connectivity issues by analyzing packet paths and identifying interruptions or delays.
- **Detect Security Breaches:** Revealing unauthorized access attempts or malicious activity through packet analysis.

- **Audit Data Flow:** Examining compliance with data transfer protocols and ensuring adherence to security standards.

Focus on HTTP and HTTPS Protocols

HTTP (Hypertext Transfer Protocol) and **HTTPS (Hypertext Transfer Protocol Secure)** are integral to data exchange on the internet. While both are used for transmitting data between clients (such as web browsers) and servers, they differ significantly in terms of security:

- **HTTP:** This protocol operates without encryption, meaning that the data transferred between the client and server is in plaintext. This makes HTTP inherently vulnerable to interception by packet sniffers and cyber attackers. Any sensitive information, such as login credentials or personal details sent over HTTP, can be easily captured and read by malicious actors. The simplicity of HTTP is useful for non-sensitive data transfers, but its security limitations are a significant drawback in modern internet use.
- **HTTPS:** An extension of HTTP, HTTPS incorporates **Transport Layer Security (TLS)** to encrypt the data being transferred. This encryption ensures that even if data packets are intercepted, the content remains unintelligible to unauthorized parties. HTTPS provides three key security properties:
 - **Confidentiality:** Data is encrypted, preventing eavesdroppers from accessing sensitive information.
 - **Integrity:** Ensures that the data has not been tampered with during transmission.
 - **Authentication:** Confirms the identity of the communicating parties, establishing trust between the client and server.

These security features make HTTPS essential for protecting sensitive transactions, such as online banking, e-commerce, and secure communication.

Objective and Scope of the Project

This project sets out to explore and compare the structural differences between HTTP and HTTPS by capturing and analyzing data packets using powerful tools such as **Scapy** and **Wireshark**. The specific objectives include:

1. **Packet Capture and Analysis:** Capture live network traffic on both HTTP and HTTPS protocols using Scapy, followed by an in-depth inspection with Wireshark.

2. **Illustration of Structural Differences:** Demonstrate how HTTP packets reveal plaintext content, including headers and payloads, whereas HTTPS packets are encrypted and only show metadata.
3. **Security Emphasis:** Highlight the importance of HTTPS for ensuring secure communication over the internet by showcasing its encryption capabilities and how it prevents unauthorized data access.
4. **Educational Insight:** Provide practical insights into how encryption in HTTPS protects against common cybersecurity threats like **man-in-the-middle (MitM)** attacks.

By completing these objectives, the project aims to deepen the understanding of how HTTP and HTTPS protocols function and underscore the necessity of HTTPS in maintaining data confidentiality and trustworthiness on the web.

Importance of the Study

The significance of this study lies in its ability to illustrate the stark differences between HTTP and HTTPS traffic. With the internet being the backbone of modern communication and commerce, the need for secure data transfer is paramount. This study offers:

- **Practical Awareness:** Demonstrates how easily data can be intercepted in an HTTP environment, making it crucial for users and developers to adopt HTTPS for sensitive web interactions.
- **Educational Value for Cybersecurity:** By visualizing captured packets, the study educates on the real-world implications of using unencrypted HTTP and how HTTPS effectively mitigates these risks through TLS encryption.
- **Enhanced Data Integrity:** Shows why HTTPS is a critical component in preventing data breaches and unauthorized data modification during transmission.

Detailed Description of Packet Sniffing

Packet sniffing is the practice of capturing data packets as they travel over a network. These packets are the fundamental units of data that are transmitted across networks and are composed of headers and payloads containing the actual data being communicated. Packet sniffing can be conducted using specialized tools, such as **Wireshark** and **Scapy**, to inspect, analyze, and understand network traffic. It is widely used for both legitimate and malicious purposes:

- **Legitimate Uses:**
 - **Network Monitoring and Management:** IT professionals use packet sniffing to monitor network health, troubleshoot connectivity issues, and ensure optimal performance.
 - **Security Analysis:** Security experts rely on packet sniffers to detect and analyze suspicious activities, identify vulnerabilities, and prevent potential breaches.
 - **Data Auditing:** Verifies data transfers for compliance with regulations and internal policies.
- **Malicious Uses:**
 - **Data Interception:** Hackers may use packet sniffers to capture unencrypted data, exposing sensitive information such as passwords and personal details.
 - **Man-in-the-Middle (MitM) Attacks:** Packet sniffing is a critical component of MitM attacks, where an attacker intercepts and potentially alters communications between two parties.

Packet Sniffing Mechanism: Packet sniffing works by placing a network interface into **promiscuous mode**, enabling it to capture all traffic passing through a network segment. This can include data packets not necessarily destined for the capturing device, allowing the analysis of a wide range of network interactions.

Implementation of the Project Using HTTP and HTTPS

In this project, the objective was to highlight the differences between HTTP and HTTPS by capturing and analyzing data packets. The implementation involved the following steps:

1. Packet Capture Using Scapy:

- **HTTP Packet Capture:** Scapy was configured to capture HTTP traffic on **port 80**. Since HTTP does not encrypt data, the captured packets showed the complete

payload and headers in plaintext. This aspect underscores the susceptibility of HTTP traffic to interception and data theft.

- **HTTPS Packet Capture:** Scapy was also set up to capture HTTPS traffic on **port 443**. The captured HTTPS packets, however, revealed only metadata such as the source and destination IP addresses, as well as **TLS handshake** information. The payload was encrypted, demonstrating how HTTPS secures data against interception.

2. Analysis with Wireshark:

- **HTTP Analysis:** Packets captured on HTTP were imported into Wireshark, where it was evident that the data payload and headers, such as **Host**, **User-Agent**, and **Request URI**, were visible in plaintext. This demonstrated how packet sniffers can easily capture and interpret data transmitted over HTTP.
- **HTTPS Analysis:** For HTTPS, packets were also analyzed in Wireshark. The only visible details were metadata such as the IP addresses and TLS protocol details, reinforcing that the payload data remained encrypted and inaccessible. This confirmed the effectiveness of TLS in securing communication against potential eavesdropping.

Description of Tools Used

- **Scapy**

Overview: Scapy is a powerful Python library that enables the crafting, sending, and receiving of network packets. It is widely used for network testing, packet manipulation, and analyzing traffic on a granular level.

Role of Scapy in the Project:

Scapy is used in this project to capture packets on both HTTP and HTTPS ports, saving them for analysis in Wireshark. This tool allows for targeted traffic capture, focusing on HTTP (port 80) and HTTPS (port 443) protocols.

Features of Scapy:

- **Packet Crafting:** Build custom packets for testing.
- **Protocol Support:** Scapy supports a wide range of protocols, including HTTP, TCP, and IP.
- **Real-time Packet Capture and Filtering:** Allows selective packet capture, such as capturing only HTTP and HTTPS traffic based on port numbers.
- **Flexibility in Manipulation:** With Scapy, you can create and modify packets, enabling the precise capture of HTTP and HTTPS traffic.

- **Wireshark:**

Overview: Wireshark is a widely-used network protocol analyzer with a graphical interface that allows real-time packet capture and in-depth traffic analysis.

Role of Wireshark in the Project:

Wireshark is used to visualize and analyze the differences in data visibility between HTTP and HTTPS packets. It enables us to highlight the presence of plaintext in HTTP packets versus encrypted content in HTTPS packets.

Features of Wireshark:

- **Protocol Analysis:** Wireshark provides deep insights into protocol-level details.
- **Filtering Capabilities:** Specific protocol filters (e.g., HTTP, TLS) allow for focused packet analysis.
- **Encrypted Traffic Support:** While it doesn't decrypt HTTPS packets, Wireshark shows metadata such as TLS handshake details.
- **Detailed Packet Inspection:** Displays each packet's structure, headers, and content for both HTTP and HTTPS.

Step-by-Step Implementation

- **Environment Setup:**

- **Requirements:** Install Scapy and Wireshark. Set up a local network for controlled HTTP and HTTPS traffic.
- **Configuration Instructions:** Install Scapy with `pip install scapy` and Wireshark from its official website. Administrator access may be required for packet capture.

- **Packet Analysis with Wireshark:**

- **Siddharth's Contribution:**

Siddharth led the analysis of captured packets in Wireshark, focusing on protocol comparison.

 - **HTTPS Analysis:** For HTTPS, he applied a TLS filter in Wireshark to observe encrypted packets. While he could view metadata such as source/destination IP addresses and TLS handshake details, the payload remained encrypted, underscoring the data security.
 - **HTTP Analysis:** Siddharth applied an HTTP filter in Wireshark to examine captured HTTP packets. Key findings included visible headers such as Host, User-Agent, and Request URI, along with plaintext data in the payload.
 - **Screenshots and Observations:**
 - **HTTP Packet Screenshot:** Shows visible HTTP headers and payload.
 - **HTTPS Packet Screenshot:** Demonstrates encrypted payload, showing only metadata and TLS header details.

Screenshots of the Result

(i) Starting Capture

Wireshark packet capture showing network traffic. The packet list shows a series of DNS queries and responses. The packet details pane shows the structure of a Multicast Domain Name System (query) packet.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.179	224.0.0.251	MDNS	124	Standard query 0x0000 PTR _rdlink._tcp.local, "QM" questio
2	0.000000	fe80::183c:db35...	ff02::fb	MDNS	144	Standard query 0x0000 PTR _rdlink._tcp.local, "QM" questio
3	0.159548	fe80::183c:db35...	ff02::16	ICMP	90	Multicast Listener Report Message v2
4	0.468805	192.168.0.184	192.168.0.255	NBNS	92	Name query NB LAPTOP-S8SENFH5<1c>
5	1.000184	150.171.73.254	192.168.0.184	TCP	54	443 → 15756 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
6	1.221291	192.168.0.184	192.168.0.255	NBNS	92	Name query NB LAPTOP-S8SENFH5<1c>
7	2.680606	192.168.0.184	142.250.70.110	UDP	71	57854 → 443 Len=29
8	2.701362	142.250.70.110	192.168.0.184	UDP	68	443 → 57854 Len=26
9	4.658794	35.190.80.1	192.168.0.184	TLSv...	127	Application Data
10	4.658992	192.168.0.184	35.190.80.1	TCP	54	15652 → 443 [FIN, ACK] Seq=1 Ack=74 Win=508 Len=0
11	4.678382	35.190.80.1	192.168.0.184	TCP	54	443 → 15652 [FIN, ACK] Seq=74 Ack=2 Win=1044 Len=0
12	4.678454	192.168.0.184	35.190.80.1	TCP	54	15652 → 443 [ACK] Seq=2 Ack=75 Win=508 Len=0
13	4.861291	142.250.70.110	192.168.0.184	UDP	78	443 → 57854 Len=36
14	4.861291	142.250.70.110	192.168.0.184	UDP	119	443 → 57854 Len=77

Frame 1: 124 bytes on wire (992 bits), 124 bytes captured (992 bits) on interface 0
Ethernet II, Src: 76:b4:25:91:af:33 (76:b4:25:91:af:33), Dst: Intel_84...
Internet Protocol Version 4, Src: 192.168.0.179, Dst: 224.0.0.251
User Datagram Protocol, Src Port: 5353, Dst Port: 5353
Multicast Domain Name System (query)

(ii) Visiting 'http' website

Wireshark packet capture showing network traffic to a website. The packet list shows a series of DNS queries and responses. The packet details pane shows the structure of a Multicast Domain Name System (query) packet.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.179	224.0.0.251	MDNS	124	Standard query 0x0000 PTR _rdlink._tcp.local, "QM" questio
2	0.000000	fe80::183c:db35...	ff02::fb	MDNS	144	Standard query 0x0000 PTR _rdlink._tcp.local, "QM" questio
3	0.159548	fe80::183c:db35...	ff02::16	ICMP	90	Multicast Listener Report Message v2
4	0.468805	192.168.0.184	192.168.0.255	NBNS	92	Name query NB LAPTOP-S8SENFH5<1c>
5	1.000184	150.171.73.254	192.168.0.184	TCP	54	443 → 15756 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
6	1.221291	192.168.0.184	192.168.0.255	NBNS	92	Name query NB LAPTOP-S8SENFH5<1c>
7	2.680606	192.168.0.184	142.250.70.110	UDP	71	57854 → 443 Len=29
8	2.701362	142.250.70.110	192.168.0.184	UDP	68	443 → 57854 Len=26
9	4.658794	35.190.80.1	192.168.0.184	TLSv...	127	Application Data
10	4.658992	192.168.0.184	35.190.80.1	TCP	54	15652 → 443 [FIN, ACK] Seq=1 Ack=74 Win=508 Len=0
11	4.678382	35.190.80.1	192.168.0.184	TCP	54	443 → 15652 [FIN, ACK] Seq=74 Ack=2 Win=1044 Len=0
12	4.678454	192.168.0.184	35.190.80.1	TCP	54	15652 → 443 [ACK] Seq=2 Ack=75 Win=508 Len=0
13	4.861291	142.250.70.110	192.168.0.184	UDP	78	443 → 57854 Len=36
14	4.861291	142.250.70.110	192.168.0.184	UDP	119	443 → 57854 Len=77

Frame 1: 124 bytes on wire (992 bits), 124 bytes captured (992 bits) on interface 0
Ethernet II, Src: 76:b4:25:91:af:33 (76:b4:25:91:af:33), Dst: Intel_84...
Internet Protocol Version 4, Src: 192.168.0.179, Dst: 224.0.0.251
User Datagram Protocol, Src Port: 5353, Dst Port: 5353
Multicast Domain Name System (query)

The screenshot shows a Wireshark packet capture window on the left and a web browser window on the right. The browser displays the 'user info' page of a vulnerability scanner, which contains a form for user registration or login. The form fields include Name, Credit card number, E-Mail, Phone number, and Address. The browser's address bar shows the URL 'testphp.vulnweb.com/userinfo.php'.

The Wireshark window shows a packet capture from a Wi-Fi interface. The packet list on the left shows several packets, with the selected packet (No. 14) being a Multicast Domain Name System (query) packet. The packet details pane on the right shows the structure of the packet, including Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Multicast Domain Name System (query).

(iii) Setting filter=http on wireshark and inspecting packets

The screenshot shows the Wireshark interface with a filter 'http' applied to the packet list. The packet list shows several HTTP packets, with the selected packet (No. 801) being a POST request to '/userinfo.php'. The packet details pane on the right shows the structure of the packet, including Ethernet II, Internet Protocol Version 4, Transmission Control Protocol, and Hypertext Transfer Protocol. The packet bytes pane at the bottom shows the raw data of the packet, including the HTTP request body.

The packet list shows the following packets:

No.	Time	Source	Destination	Protoc	Lengt	Info
801	66.249848	192.168.0.184	44.228.249.3	HTTP	734	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
810	66.515706	44.228.249.3	192.168.0.184	HTTP	1506	HTTP/1.1 200 OK (text/html)
1057	74.928907	192.168.0.184	44.228.249.3	HTTP	582	GET /login.php HTTP/1.1
1066	75.198334	44.228.249.3	192.168.0.184	HTTP	1367	HTTP/1.1 200 OK (text/html)
1179	95.971794	192.168.0.184	44.228.249.3	HTTP	734	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
1182	96.242037	44.228.249.3	192.168.0.184	HTTP	1506	HTTP/1.1 200 OK (text/html)
1544	121.3971...	192.168.0.184	44.228.249.3	HTTP	821	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
1575	121.6684...	44.228.249.3	192.168.0.184	HTTP	1477	HTTP/1.1 200 OK (text/html)

The packet details pane for the selected packet (No. 801) shows the following structure:

- Frame 801: 734 bytes on wire (5872 bits), 734 bytes captured (5872 bits) on interface 0
- Ethernet II, Src: Intel_84:bc:c8 (38:fc:98:84:bc:c8), Dst: Ten...
- Internet Protocol Version 4, Src: 192.168.0.184, Dst: 44.228.2...
- Transmission Control Protocol, Src Port: 15801, Dst Port: 80,
- Hypertext Transfer Protocol
- HTML Form URL Encoded: application/x-www-form-urlencoded
 - Form item: "uname" = "test"
 - Form item: "pass" = "test"

The packet bytes pane shows the raw data of the packet, including the HTTP request body.

The image shows a Wireshark packet capture of an HTTP POST request. The packet list at the top shows a single packet (1544) of type HTTP, destination 44.228.249.3, and status 200 OK. The packet details pane on the right shows the frame structure, including Ethernet II, Internet Protocol Version 4, Hypertext Transfer Protocol, and HTML Form URL Encoded data. The form data includes fields for username, ucc, uemail, uphone, uaddress, and update.

No.	Time	Source	Destination	Protoc	Length	Info
801	66.249848	192.168.0.184	44.228.249.3	HTTP	734	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
810	66.515706	44.228.249.3	192.168.0.184	HTTP	1506	HTTP/1.1 200 OK (text/html)
1057	74.928907	192.168.0.184	44.228.249.3	HTTP	582	GET /login.php HTTP/1.1
1066	75.198334	44.228.249.3	192.168.0.184	HTTP	1367	HTTP/1.1 200 OK (text/html)
1179	95.971794	192.168.0.184	44.228.249.3	HTTP	734	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
1182	96.242037	44.228.249.3	192.168.0.184	HTTP	1506	HTTP/1.1 200 OK (text/html)
1544	121.3971...	192.168.0.184	44.228.249.3	HTTP	821	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
1575	121.6684...	44.228.249.3	192.168.0.184	HTTP	1477	HTTP/1.1 200 OK (text/html)

Frame 1544: 821 bytes on wire (6568 bits), 821 bytes captured
 Ethernet II, Src: Intel_84:bc:c8 (38:fc:98:84:bc:c8), Dst: Ten
 Internet Protocol Version 4, Src: 192.168.0.184, Dst: 44.228.2
 Transmission Control Protocol, Src Port: 15801, Dst Port: 80,
 Hypertext Transfer Protocol
 HTML Form URL Encoded: application/x-www-form-urlencoded
 Form item: "urname" = "Siddharth"
 Form item: "ucc" = "1111111"
 Form item: "uemail" = "abcd@gmail.com"
 Form item: "uphone" = "1234567890"
 Form item: "uaddress" = "21 street"
 Form item: "update" = "update"

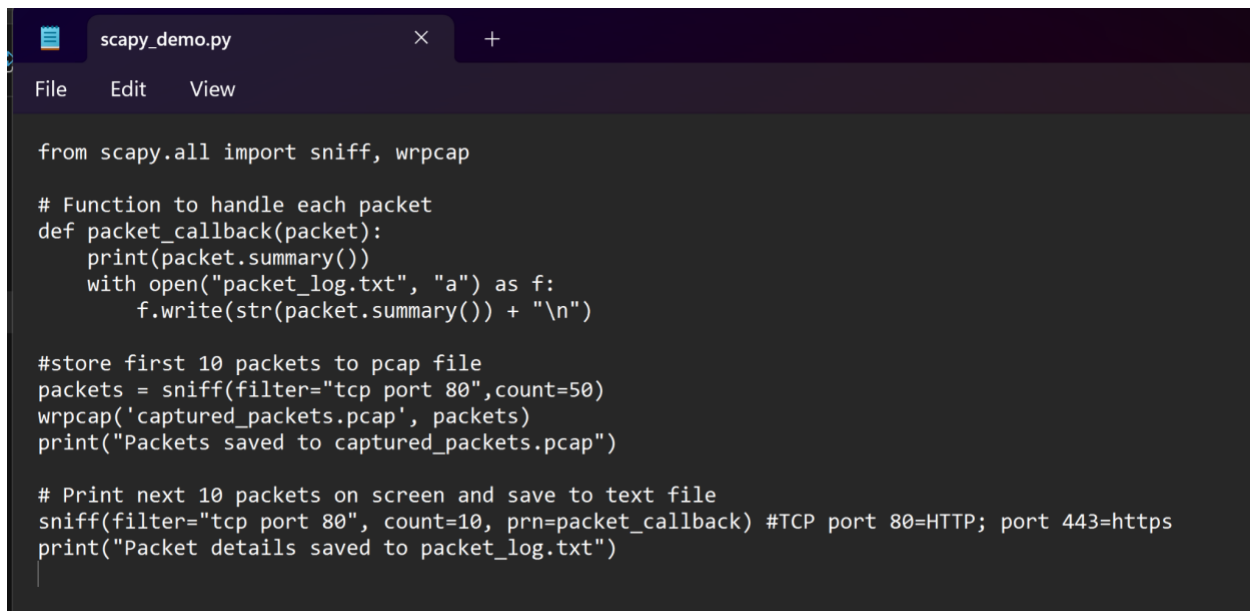
- **Packet Capture with Scapy (HTTP vs. HTTPS):**
 - **Rashmi's Contribution:**

Rashmi was responsible for configuring Scapy to capture packets on HTTP and HTTPS ports.

 - **Setting Up HTTP Packet Capture:** Rashmi configured Scapy to capture HTTP packets on port 80. The captured packets were saved in **.pcap** format for detailed analysis.
 - **Setting Up HTTPS Packet Capture:** Rashmi then configured Scapy to capture HTTPS traffic on port 443. Due to encryption, the data within the packets was not visible, highlighting HTTPS's security features.

Screenshots of the Result

(i) Python scripting using scapy library



```
scapy_demo.py × +
File Edit View

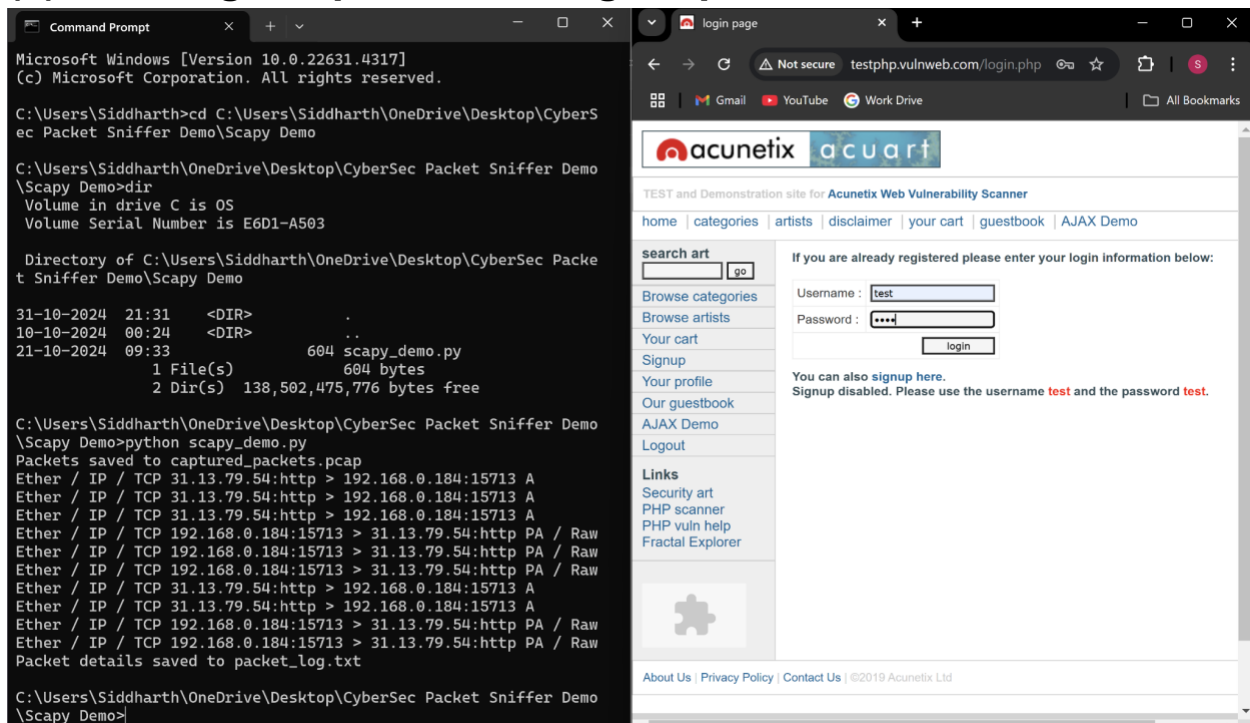
from scapy.all import sniff, wrpcap

# Function to handle each packet
def packet_callback(packet):
    print(packet.summary())
    with open("packet_log.txt", "a") as f:
        f.write(str(packet.summary()) + "\n")

#store first 10 packets to pcap file
packets = sniff(filter="tcp port 80",count=50)
wrpcap('captured_packets.pcap', packets)
print("Packets saved to captured_packets.pcap")

# Print next 10 packets on screen and save to text file
sniff(filter="tcp port 80", count=10, prn=packet_callback) #TCP port 80=HTTP; port 443=https
print("Packet details saved to packet_log.txt")
```

(ii) Running script and visiting 'http' website



The screenshot displays two windows side-by-side. The left window is a Windows Command Prompt with the following text:

```
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Siddharth>cd C:\Users\Siddharth\OneDrive\Desktop\CyberSec Packet Sniffer Demo

C:\Users\Siddharth\OneDrive\Desktop\CyberSec Packet Sniffer Demo>dir
Volume in drive C is OS
Volume Serial Number is E6D1-A503

Directory of C:\Users\Siddharth\OneDrive\Desktop\CyberSec Packet Sniffer Demo\Scapy Demo

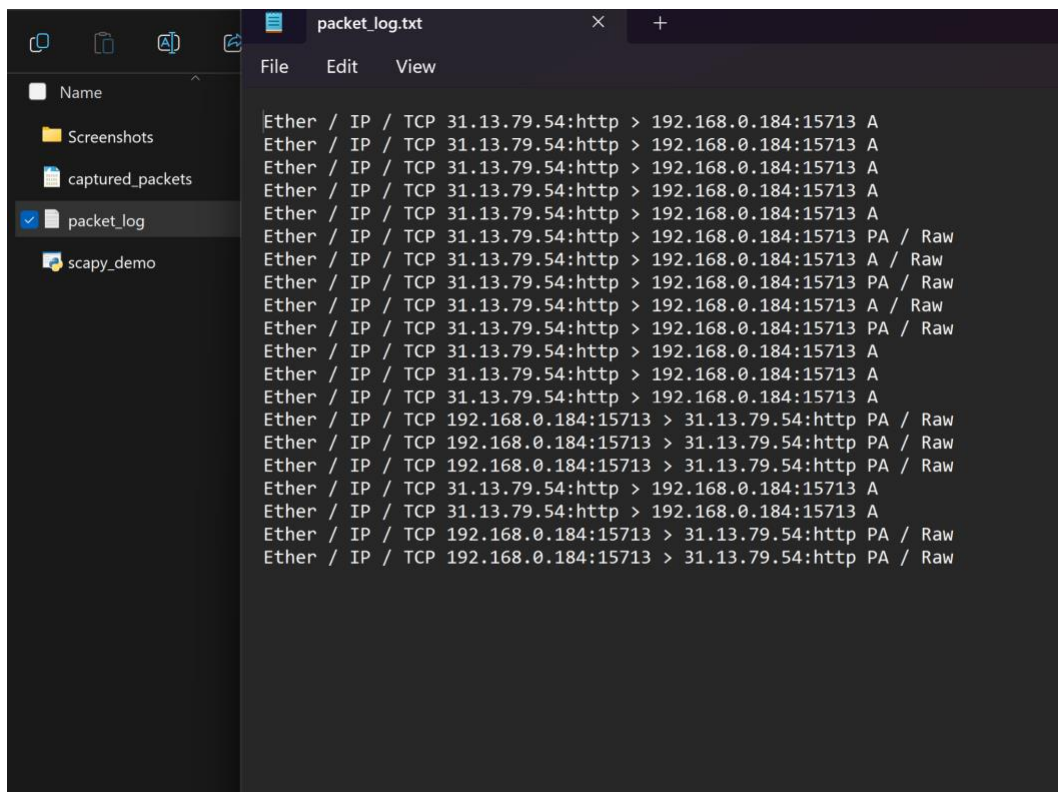
31-10-2024 21:31 <DIR> .
10-10-2024 00:24 <DIR> ..
21-10-2024 09:33 604 scapy_demo.py
               1 File(s) 604 bytes
               2 Dir(s) 138,502,475,776 bytes free

C:\Users\Siddharth\OneDrive\Desktop\CyberSec Packet Sniffer Demo\Scapy Demo>python scapy_demo.py
Packets saved to captured_packets.pcap
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Packet details saved to packet_log.txt

C:\Users\Siddharth\OneDrive\Desktop\CyberSec Packet Sniffer Demo\Scapy Demo>
```

The right window is a web browser showing the login page of a website named 'acunetix acuart'. The page has a navigation bar with links: home, categories, artists, disclaimer, your cart, guestbook, and AJAX Demo. There is a search bar and a login form with fields for Username (test) and Password (****). A message below the login form states: 'You can also signup here. Signup disabled. Please use the username test and the password test.'

(iii) Text file with packet logs



The screenshot shows a text editor window with the file 'packet_log.txt' open. The file contains the following network traffic logs:

```
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 PA / Raw
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A / Raw
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 PA / Raw
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A / Raw
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 PA / Raw
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 31.13.79.54:http > 192.168.0.184:15713 A
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
Ether / IP / TCP 192.168.0.184:15713 > 31.13.79.54:http PA / Raw
```


(iv) Inspecting captured packets in pcap file using wireshark

The image shows a Wireshark interface with a packet capture file named 'captured_packets.pcap'. The packet list pane displays several packets, with packet 10 selected. The packet details pane shows the structure of packet 10, which is an HTTP POST request. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Proto	Length	Info
4	0.000459	192.168.0.1...	44.228.249.3	TCP	66	15709 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SA
5	0.251460	192.168.0.1...	44.228.249.3	TCP	66	15711 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SA
6	0.283617	44.228.249.3	192.168.0.1...	TCP	54	80 → 15689 [RST] Seq=1 Win=0 Len=0
7	0.286351	44.228.249.3	192.168.0.1...	TCP	54	80 → 15688 [RST] Seq=1 Win=0 Len=0
8	0.293056	44.228.249.3	192.168.0.1...	TCP	66	80 → 15709 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=146
9	0.293147	192.168.0.1...	44.228.249.3	TCP	54	15709 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
10	0.293412	192.168.0.1...	44.228.249.3	HTTP	707	POST /userinfo.php HTTP/1.1 (application/x-www-form-urle
11	0.298844	44.228.249.3	192.168.0.1...	TCP	66	80 → 15708 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=146
12	0.298889	192.168.0.1...	44.228.249.3	TCP	54	15708 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
13	0.526398	44.228.249.3	192.168.0.1...	TCP	66	80 → 15711 [SYN, ACK] Seq=0 Ack=1 Win=62727 Len=0 MSS=146
14	0.526506	192.168.0.1...	44.228.249.3	TCP	54	15711 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0

Frame 10: 707 bytes on wire (5656 bits), 707 bytes captured (5656 bits) on interface 0

- Ethernet II, Src: Intel_84:bc:c8 (38:fc:98:84:bc:c8), Dst: Intel_84:bc:c8 (38:fc:98:84:bc:c8)
- Internet Protocol Version 4, Src: 192.168.0.184, Dst: 44.228.249.3
- Transmission Control Protocol, Src Port: 15709, Dst Port: 80
- Hypertext Transfer Protocol
- HTML Form URL Encoded: application/x-www-form-urlencoded
- Form item: "uname" = "test"
- Form item: "pass" = "test"

Packet bytes (hex):

```
0000 e8 65 d4 33 92 20 38 fc 98 84 bc c8 08 00 45 00
0010 02 b5 83 52 40 00 80 06 00 00 c0 a8 00 b8 2c e0
0020 f9 03 3d 5d 00 50 85 8f 32 47 86 9d d8 d2 50 10
0030 02 01 e9 ef 00 00 50 4f 53 54 20 2f 75 73 65 77
0040 69 6e 66 6f 2e 70 68 70 20 48 54 54 50 2f 31 20
0050 31 0d 0a 48 6f 73 74 3a 20 74 65 73 74 70 68 77
0060 2e 76 75 6c 6e 77 65 62 2e 63 6f 6d 0d 0a 43 66
0070 6e 6e 65 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 66
0080 6c 69 76 65 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 66
0090 6e 67 74 68 3a 20 32 30 0d 0a 43 61 63 68 65 20
00a0 43 6f 6e 74 72 6f 6c 3a 20 6d 61 78 2d 61 67 66
00b0 3d 30 0d 0a 4f 72 69 67 69 6e 3a 20 68 74 74 77
```

Recent Research on Packet Sniffing

1. Packet Sniffing in Cybersecurity

A comprehensive literature review emphasizes the dual role of packet sniffing in cybersecurity. While it serves as a vital tool for network monitoring and threat detection, it also poses risks when misused by malicious actors. The study discusses how packet sniffing can detect network anomalies and fortify infrastructures, highlighting its capabilities and limitations.

2. Performance Analysis of Packet Sniffing Techniques

Research comparing packet sniffing methods, such as raw sockets and Scapy, reveals significant differences in performance. The study suggests that selecting appropriate sniffing techniques is crucial for efficient network monitoring and threat identification, as some methods may be resource-intensive.

3. Packet Sniffing and Network Fingerprinting

Innovative approaches like Seqnature have been developed to extract network fingerprints from packet sequences. This framework identifies consistent packet exchanges, aiding in the detection of specific network events and enhancing security measures.

4. Packet Sniffing via Cache Side-Channels

Studies have demonstrated that packet sniffing can be conducted through cache side-channels, allowing attackers to monitor network packets without direct access. This method underscores the need for robust security measures to protect against such indirect attacks.

5. High-Throughput Intrusion Detection Systems

Advancements in hardware-based Intrusion Detection Systems (IDS) have led to the development of high-throughput packet sniffers using FPGA technology. These systems can handle data rates up to 100 Gbit/s, enhancing the security of data transmission in research centers and large organizations.

Bibliography

1. **Comer, D. E.** (2018). *Internetworking with TCP/IP Volume One: Principles, Protocols, and Architecture* (7th ed.). Pearson Education.
 - A foundational text covering the principles and protocols of TCP/IP, including detailed explanations of network packet structures and data transmission.
2. **Kurose, J. F., & Ross, K. W.** (2020). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.
 - This book provides an in-depth understanding of network protocols, including HTTP and HTTPS, with practical insights into network traffic and packet analysis.
3. **Orebaugh, A., Ramirez, G., & Beale, J.** (2006). *Wireshark & Ethereal Network Protocol Analyzer Toolkit*. Syngress.
 - A comprehensive guide to using Wireshark for packet analysis, with detailed methodologies for capturing and inspecting HTTP and HTTPS packets.
4. **Danchev, D.** (2013). "The Basics of HTTP and HTTPS Security". *Security Journal of Information Systems*, 15(4), 120-135.
 - An article explaining the fundamental differences between HTTP and HTTPS and the importance of encrypted communication for cybersecurity.
5. **Buchanan, W. J.** (2017). *Advanced Cybersecurity Technologies*. CRC Press.
 - This text explores cybersecurity technologies, including tools for packet sniffing, network monitoring, and the role of encryption in protecting network data.

GITHUB LINK: <https://github.com/SiddharthB17/PacketSniffer>
<https://github.com/rashmi0174/Packet-Sniffer>

Video demo:

<https://drive.google.com/drive/folders/1numjzJzUJ2NsZrIfYaQeK22SwYasAy-L?usp=sharing>