- **Task 1:**
- **CIA TRIAD**

The **CIA Triad** is a widely used model that represents the core principles of information security. It stands for:

1. Confidentiality: Ensures that information is only accessible to those who have the proper authorization. This involves protecting data from unauthorized access or disclosure. Techniques used to ensure confidentiality include encryption, access controls, and authentication mechanisms.

2. Integrity: Ensures that information is accurate, complete, and unaltered during storage, transmission, and processing. Integrity is typically maintained through methods such as hashing, digital signatures, and checksums, which help detect unauthorized changes or corruption.

3. Availability: Ensures that information and systems are accessible and functional when needed by authorized users. Availability is achieved by implementing redundancy, disaster recovery plans, regular maintenance, and protection against denial-of-service (DoS) attacks.

Together, the CIA Triad serves as the foundation for creating secure systems and mitigating risks related to data loss, breaches, and other security incidents.

- **TYPES OF CYBER ATTACKS**

1. Phishing:

- Definition: Phishing is a social engineering attack where attackers impersonate legitimate organizations or individuals to trick victims into disclosing sensitive information, such as usernames, passwords, credit card details, or other personal information.
- How it Works: Typically, attackers send fraudulent emails, messages, or links that appear to come from trusted sources like banks, email providers, or government agencies. These messages often contain malicious links or attachments designed to steal credentials or install malware.

- Types of Phishing:

- **Spear Phishing:** A targeted attack aimed at specific individuals or organizations, often with highly personalized messages.
- **Whaling:** A type of spear-phishing focused on high-profile targets, such as executives (the "big fish").
- **Vishing:** Phishing carried out via phone calls, where attackers impersonate trusted entities (e.g., IRS, bank) to trick victims.
- **Smishing:** Phishing via SMS/text messages.

2. Malware (Malicious Software):

- **Definition:** Malware refers to any software designed to harm, exploit, or otherwise compromise the functionality of computers, networks, or devices. It includes a wide range of malicious programs.

- **Types of Malware:**
  - **Viruses:** Self-replicating programs that attach themselves to clean files and spread to other files or systems.
  - **Worms:** Similar to viruses but can spread autonomously without human intervention (e.g., through networks).
  - **Trojan Horses:** Malicious software disguised as legitimate programs or files, which trick users into downloading and executing them.
  - **Ransomware:** A type of malware that locks or encrypts the victim's data and demands payment (ransom) to restore access.
  - **Spyware:** Software that secretly collects information about a user's activities without their knowledge or consent.
  - **Adware:** Software that displays unwanted ads or redirects browser traffic to malicious sites.
  - **Rootkits:** Tools used to hide the presence of malware or unauthorized activity from detection software.

3. Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Attacks

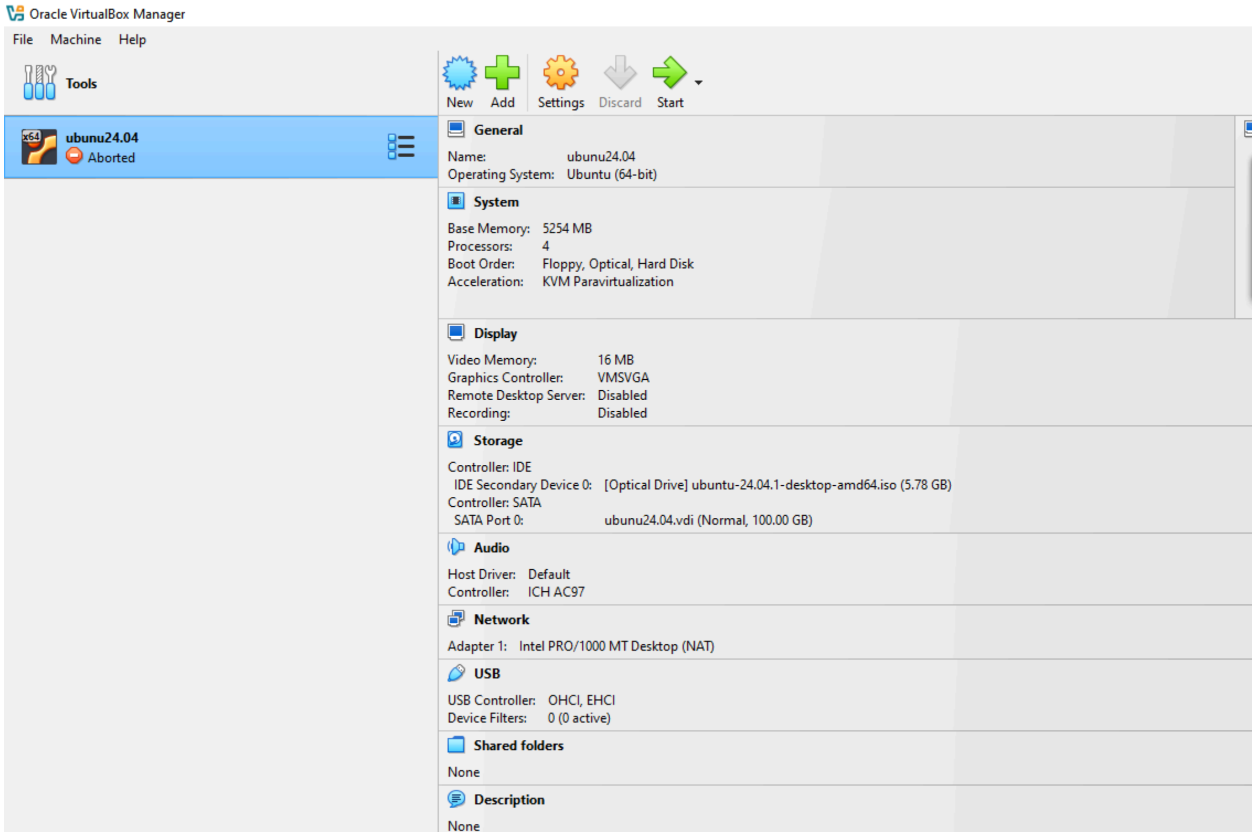
- **Definition:** A Denial-of-Service (DoS) attack aims to make a network service or website unavailable by overwhelming it with a flood of traffic or resource requests, preventing legitimate users from accessing it. When multiple systems are used to execute the attack, it's called a **Distributed Denial-of-Service (DDoS)** attack.
- **How it Works:** Attackers typically flood a target server or network with excessive traffic or resource requests, consuming system resources (e.g., bandwidth, CPU), causing the service to crash or become unavailable. DDoS attacks are often launched using botnets—large networks of compromised devices.

- **Types of DoS/DDoS Attacks:**
  - **Volume-based attacks:** These focus on consuming network bandwidth, such as ICMP floods, UDP floods, or DNS amplification attacks.

- **Protocol attacks:** These target network protocols and exploit vulnerabilities in protocol handling, such as SYN floods or Ping of Death.
- **Application-layer attacks:** These focus on overloading application resources, such as HTTP floods, which are designed to overwhelm web servers.

Each of these attacks can lead to severe consequences for individuals and organizations, including data loss, financial theft, downtime, and damage to reputation. Defensive measures include using firewalls, anti-malware software, encryption, and employee education to prevent phishing and other social engineering attacks.

- ## Task 2



General
Name: ubunu24.04
Operating System: Ubuntu (64-bit)

System
Base Memory: 5254 MB
Processors: 4
Boot Order: Floppy, Optical, Hard Disk
Acceleration: KVM Paravirtualization

Display
Video Memory: 16 MB
Graphics Controller: VMSVGA
Remote Desktop Server: Disabled
Recording: Disabled

Storage
Controller: IDE
IDE Secondary Device 0: [Optical Drive] ubuntu-24.04.1-desktop-amd64.iso (5.78 GB)
Controller: SATA
SATA Port 0: ubunu24.04.vdi (Normal, 100.00 GB)

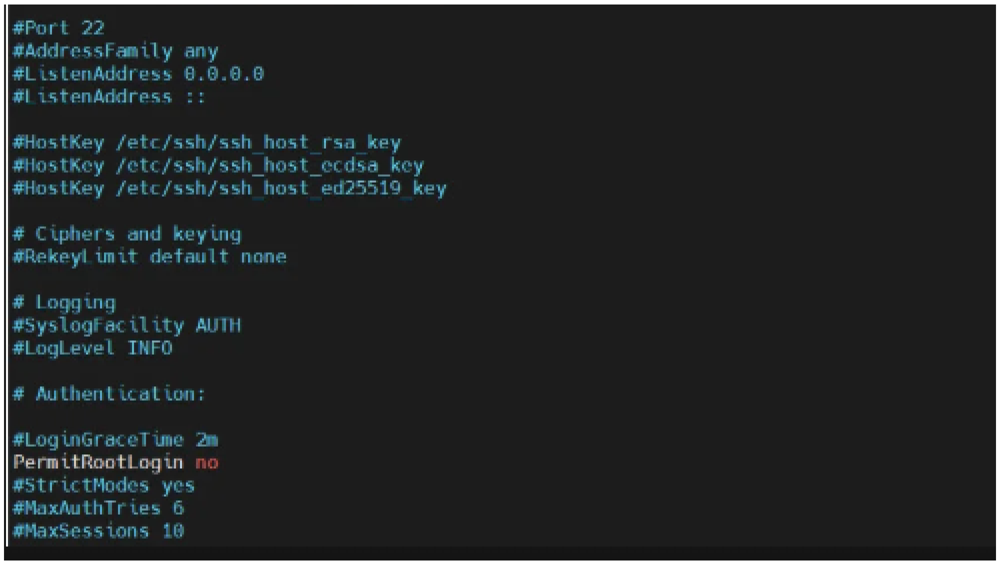
Audio
Host Driver: Default
Controller: ICH AC97

Network
Adapter 1: Intel PRO/1000 MT Desktop (NAT)

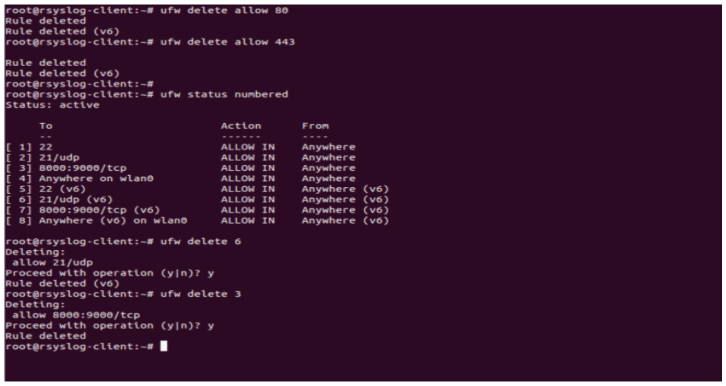
USB
USB Controller: OHCI, EHCI
Device Filters: 0 (0 active)

Shared folders
None

Description
None

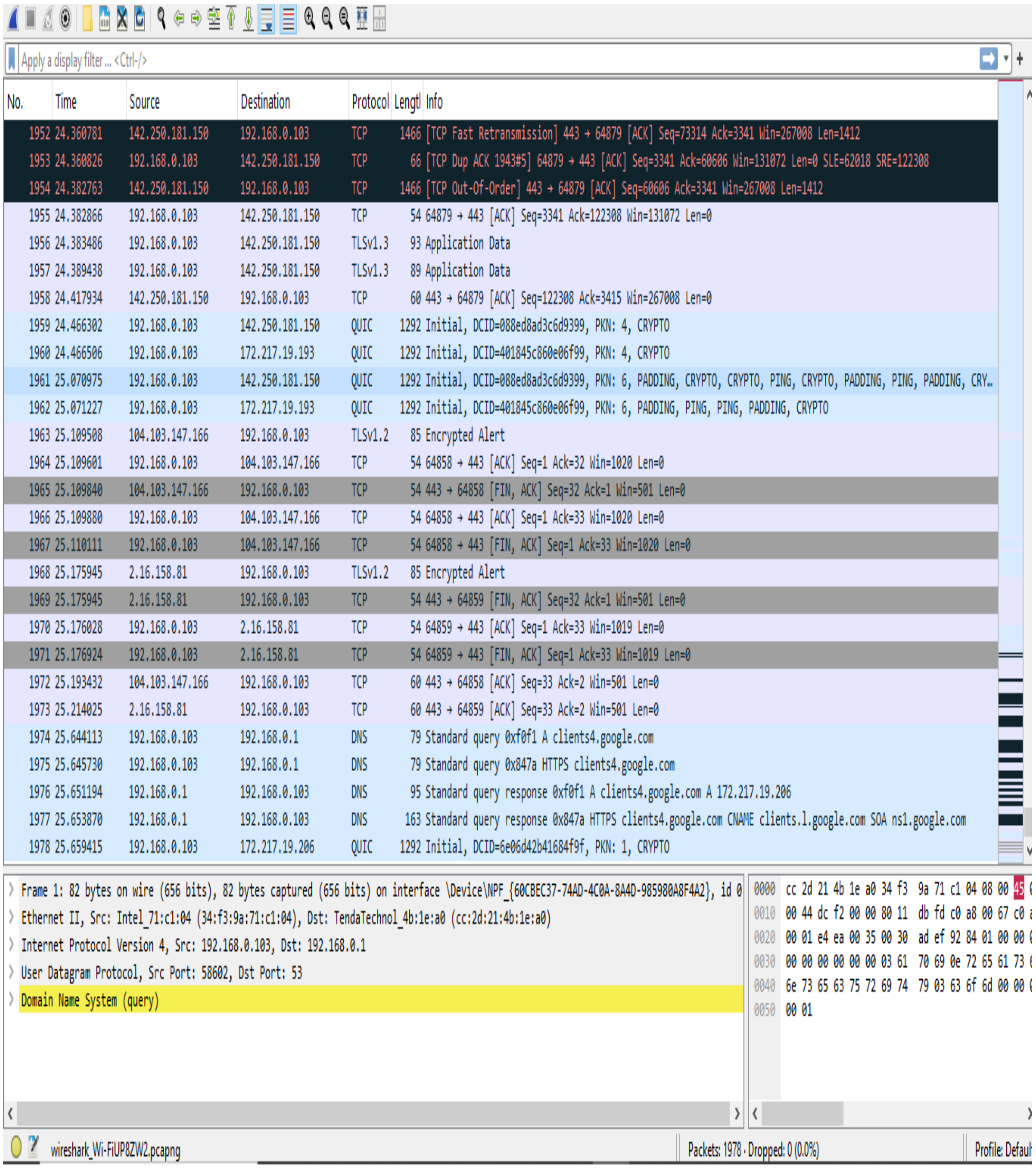


```
#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
PermitRootLogin no
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10
```



```
root@rsyslog-client:~# ufw delete allow 80
Rule deleted
Rule deleted (v6)
root@rsyslog-client:~# ufw delete allow 443

Rule deleted
Rule deleted (v6)
root@rsyslog-client:~#
root@rsyslog-client:~# ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 22                         ALLOW IN    Anywhere
[ 2] 21/udp                     ALLOW IN    Anywhere
[ 3] 8000:9000/tcp              ALLOW IN    Anywhere
[ 4] Anywhere on wlan0          ALLOW IN    Anywhere
[ 5] 22 (v6)                    ALLOW IN    Anywhere (v6)
[ 6] 21/udp (v6)                ALLOW IN    Anywhere (v6)
[ 7] 8000:9000/tcp (v6)         ALLOW IN    Anywhere (v6)
[ 8] Anywhere (v6) on wlan0     ALLOW IN    Anywhere (v6)

root@rsyslog-client:~# ufw delete 6
Deleting:
 allow 21/udp
Proceed with operation (y|n)? y
Rule deleted (v6)
root@rsyslog-client:~# ufw delete 3
Deleting:
 allow 8000:9000/tcp
Proceed with operation (y|n)? y
Rule deleted
root@rsyslog-client:~#
```

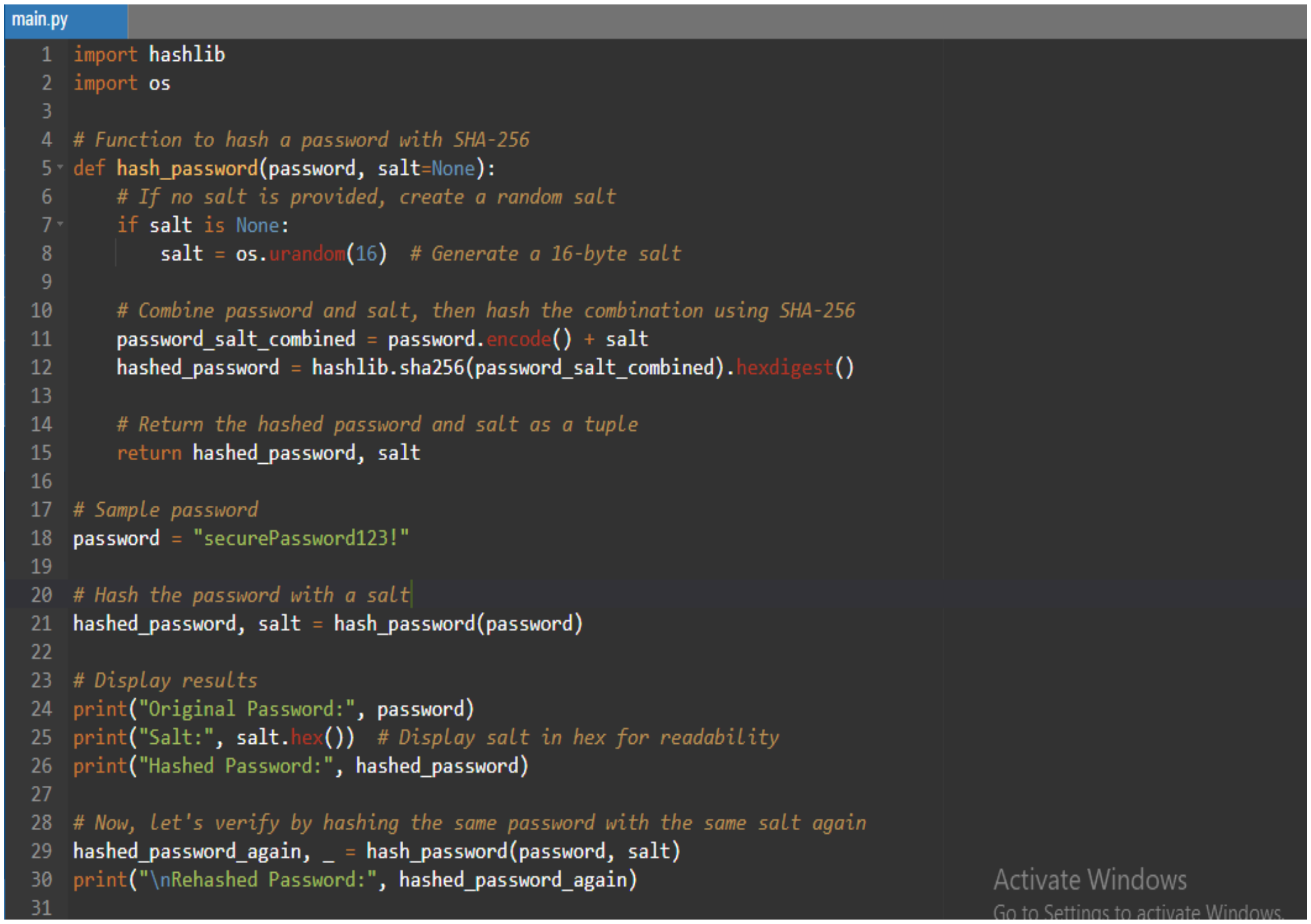
- **Task 3**



```
Apply a display filter ... <Ctrl-/>

No.   Time       Source           Destination      Protocol Length Info
1952 24.360781   142.250.181.150  192.168.0.103    TCP      1466 [TCP Fast Retransmission] 443 → 64879 [ACK] Seq=73314 Ack=3341 Win=267008 Len=1412
1953 24.360826   192.168.0.103    142.250.181.150  TCP        66 [TCP Dup ACK 1943#5] 64879 → 443 [ACK] Seq=3341 Ack=60606 Win=131072 Len=0 SLE=62018 SRE=122308
1954 24.382763   142.250.181.150  192.168.0.103    TCP      1466 [TCP Out-Of-Order] 443 → 64879 [ACK] Seq=60606 Ack=3341 Win=267008 Len=1412
1955 24.382866   192.168.0.103    142.250.181.150  TCP        54 64879 → 443 [ACK] Seq=3341 Ack=122308 Win=131072 Len=0
1956 24.383486   192.168.0.103    142.250.181.150  TLSv1.3    93 Application Data
1957 24.389438   192.168.0.103    142.250.181.150  TLSv1.3    89 Application Data
1958 24.417934   142.250.181.150  192.168.0.103    TCP        60 443 → 64879 [ACK] Seq=122308 Ack=3415 Win=267008 Len=0
1959 24.466302   192.168.0.103    142.250.181.150  QUIC     1292 Initial, DCID=088ed8ad3c6d9399, PKN: 4, CRYPTO
1960 24.466506   192.168.0.103    172.217.19.193   QUIC     1292 Initial, DCID=401845c860e06f99, PKN: 4, CRYPTO
1961 25.070975   192.168.0.103    142.250.181.150  QUIC     1292 Initial, DCID=088ed8ad3c6d9399, PKN: 6, PADDING, CRYPTO, CRYPTO, PING, CRYPTO, PADDING, PING, PADDING, CRY…
1962 25.071227   192.168.0.103    172.217.19.193   QUIC     1292 Initial, DCID=401845c860e06f99, PKN: 6, PADDING, PING, PING, PADDING, CRYPTO
1963 25.109508   104.103.147.166  192.168.0.103    TLSv1.2    85 Encrypted Alert
1964 25.109601   192.168.0.103    104.103.147.166  TCP        54 64858 → 443 [ACK] Seq=1 Ack=32 Win=1020 Len=0
1965 25.109840   104.103.147.166  192.168.0.103    TCP        54 443 → 64858 [FIN, ACK] Seq=32 Ack=1 Win=501 Len=0
1966 25.109880   192.168.0.103    104.103.147.166  TCP        54 64858 → 443 [ACK] Seq=1 Ack=33 Win=1020 Len=0
1967 25.110111   192.168.0.103    104.103.147.166  TCP        54 64858 → 443 [FIN, ACK] Seq=1 Ack=33 Win=1020 Len=0
1968 25.175945   2.16.158.81      192.168.0.103    TLSv1.2    85 Encrypted Alert
1969 25.175945   2.16.158.81      192.168.0.103    TCP        54 443 → 64859 [FIN, ACK] Seq=32 Ack=1 Win=501 Len=0
1970 25.176028   192.168.0.103    2.16.158.81      TCP        54 64859 → 443 [ACK] Seq=1 Ack=33 Win=1019 Len=0
1971 25.176924   192.168.0.103    2.16.158.81      TCP        54 64859 → 443 [FIN, ACK] Seq=1 Ack=33 Win=1019 Len=0
1972 25.193432   104.103.147.166  192.168.0.103    TCP        60 443 → 64858 [ACK] Seq=33 Ack=2 Win=501 Len=0
1973 25.214025   2.16.158.81      192.168.0.103    TCP        60 443 → 64859 [ACK] Seq=33 Ack=2 Win=501 Len=0
1974 25.644113   192.168.0.103    192.168.0.1      DNS        79 Standard query 0xf0f1 A clients4.google.com
1975 25.645730   192.168.0.103    192.168.0.1      DNS        79 Standard query 0x847a HTTPS clients4.google.com
1976 25.651194   192.168.0.1      192.168.0.103    DNS        95 Standard query response 0xf0f1 A clients4.google.com A 172.217.19.206
1977 25.653870   192.168.0.1      192.168.0.103    DNS       163 Standard query response 0x847a HTTPS clients4.google.com CNAME clients.l.google.com SOA ns1.google.com
1978 25.659415   192.168.0.103    172.217.19.206   QUIC     1292 Initial, DCID=6e06d42b41684f9f, PKN: 1, CRYPTO

> Frame 1: 82 bytes on wire (656 bits), 82 bytes captured (656 bits) on interface \Device\NPF_{60CBEC37-74AD-4C0A-8A4D-985980A8F4A2}, id 0
> Ethernet II, Src: Intel_71:c1:04 (34:f3:9a:71:c1:04), Dst: TendaTechnol_4b:1e:a0 (cc:2d:21:4b:1e:a0)
> Internet Protocol Version 4, Src: 192.168.0.103, Dst: 192.168.0.1
> User Datagram Protocol, Src Port: 58602, Dst Port: 53
> Domain Name System (query)

0000  cc 2d 21 4b 1e a0 34 f3  9a 71 c1 04 08 00 45
0010  00 44 dc f2 00 00 80 11  db fd c0 a8 00 67 c0
0020  00 01 e4 ea 00 35 00 30  ad ef 92 84 01 00 00
0030  00 00 00 00 00 00 03 61  70 69 0e 72 65 61 73
0040  6e 73 65 63 75 72 69 74  79 03 63 6f 6d 00 00
0050  00 01

wireshark_Wi-FiUP8ZW2.pcapng        Packets: 1978 · Dropped: 0 (0.0%)        Profile: Defaul
```

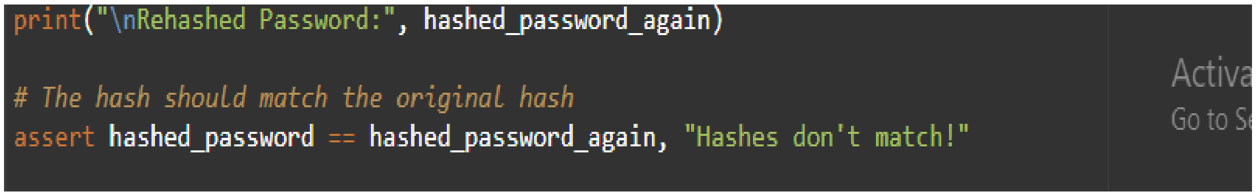
- ☐ Protocols observed : TCP, TLSv1.3, QUIC, DNS, TLSv1.2
- ☐ Source IP : 192.168.0.103
- ☐ Destination IP: 192.168.0.1
- ☐ Source Port: 58602
- ☐ Destination Port : 53

- **Task 4**

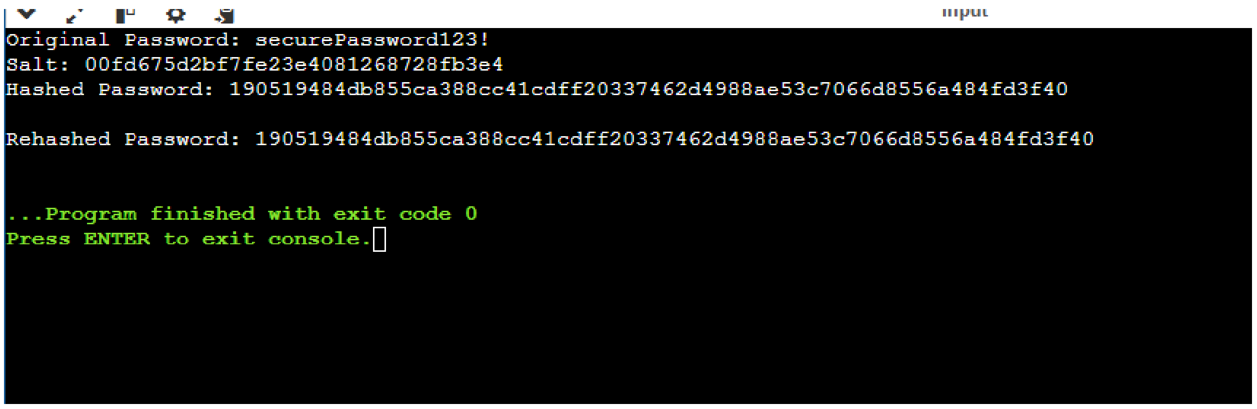


main.py

```python
import hashlib
import os

# Function to hash a password with SHA-256
def hash_password(password, salt=None):
    # If no salt is provided, create a random salt
    if salt is None:
        salt = os.urandom(16)  # Generate a 16-byte salt

    # Combine password and salt, then hash the combination using SHA-256
    password_salt_combined = password.encode() + salt
    hashed_password = hashlib.sha256(password_salt_combined).hexdigest()

    # Return the hashed password and salt as a tuple
    return hashed_password, salt

# Sample password
password = "securePassword123!"

# Hash the password with a salt
hashed_password, salt = hash_password(password)

# Display results
print("Original Password:", password)
print("Salt:", salt.hex())  # Display salt in hex for readability
print("Hashed Password:", hashed_password)

# Now, let's verify by hashing the same password with the same salt again
hashed_password_again, _ = hash_password(password, salt)
print("\nRehashed Password:", hashed_password_again)

```



```python
print("\nRehashed Password:", hashed_password_again)

# The hash should match the original hash
assert hashed_password == hashed_password_again, "Hashes don't match!"
```



```
Original Password: securePassword123!
Salt: 00fd675d2bf7fe23e4081268728fb3e4
Hashed Password: 190519484db855ca388cc41cdff20337462d4988ae53c7066d8556a484fd3f40

Rehashed Password: 190519484db855ca388cc41cdff20337462d4988ae53c7066d8556a484fd3f40


...Program finished with exit code 0
Press ENTER to exit console.
```

Password Security and Hashing:
Overview

Password security is critical in protecting user data, and hashing is a fundamental technique used in securing passwords. Hashing transforms the password into a fixed-length string of characters that is unique to the original password. This process ensures that the actual password is not stored directly, reducing the risk of password exposure in case of a data breach.

Salting is an additional technique where a random value (the salt) is added to the password before hashing. This ensures that even if two users have the same password, their stored hashes will be different due to the unique salt.

Python Script to Hash a Password Using SHA-256:

Below is a Python script that hashes a password using SHA-256. Additionally, it demonstrates how salting can strengthen the password security.

Script:

~~~
```python
import hashlib
import os

# Function to hash a password with SHA-256
def hash_password(password, salt=None):
    # If no salt is provided, create a random salt
    if salt is None:
        salt = os.urandom(16)  # Generate a 16-byte salt

    # Combine password and salt, then hash the combination using SHA-256
    password_salt_combined = password.encode() + salt
    hashed_password = hashlib.sha256(password_salt_combined).hexdigest()
~~~

```
    # Return the hashed password and salt as a tuple
    return hashed_password, salt

# Sample password
password = "securePassword123!"

# Hash the password with a salt
hashed_password, salt = hash_password(password)

# Display results
print("Original Password:", password)
print("Salt:", salt.hex())  # Display salt in hex for readability
print("Hashed Password:", hashed_password)

# Now, let's verify by hashing the same password with the same salt again
hashed_password_again, _ = hash_password(password, salt)
print("\nRehashed Password:", hashed_password_again)

The hash should match the original hash
assert hashed_password == hashed_password_again, "Hashes don't match!"
```

Explanation of the Script:

1. Hashing the Password:
- The script uses Python's `hashlib` module to hash a password using SHA-256.
- The `os.urandom(16)` function is used to generate a random 16-byte salt.

2. **Salting:**
- A unique salt is added to the password before hashing, which ensures that the hash is unique even for users with the same password.

3. **output**:
- The script prints the original password, the generated salt (in hexadecimal), and the resulting hashed password.
- It re-hashes the same password with the same salt to show that the hashes will match, demonstrating the consistency of the hash function.

Why Hashing and Salting Are Essential in Cybersecurity:

1. **Protecting Stored Passwords**:
- Storing passwords in plaintext is highly insecure. If an attacker gains access to a database of passwords, they can easily retrieve them. Hashing ensures that even if a database is compromised, the original passwords are not directly exposed.

2. **Hashing Is One-Way:**
   - Hashing is a one-way operation, meaning that you cannot reverse the hash back into the original password. This ensures that even if an attacker gains access to the hashes, they cannot easily obtain the original passwords.

3. **Salting Makes Hashes Unique:**
   - Without salting, identical passwords will produce identical hashes, making it easier for attackers to perform attacks like rainbow table attacks (precomputed hashes for common passwords). Salting ensures that each password hash is unique, even if two users have the same password.

4. **Defense Against Precomputed Attacks:**
   - A salt makes precomputed attacks (like rainbow tables) infeasible because attackers would need to recompute hashes for each unique salt, which is time-consuming and computationally expensive.

5. **Increased Security in Case of a Breach:**
   - If an attacker obtains a database of salted hashes, the attacker cannot use the same hash for different users or passwords, making the password-cracking process significantly harder.

Conclusion:

Hashing and salting are critical techniques for securely storing passwords. Hashing ensures that passwords are not stored in plaintext, while salting adds an additional layer of security by ensuring that identical passwords do not produce identical hashes. These techniques are essential for defending against various types of cyberattacks, such as rainbow table and brute-force attacks, which aim to crack user passwords.

- ## Task 5:

Basic Threat Identification: Common Web Application Security Threats

Objective:
Understand and summarize five common web application security threats based on the OWASP Top 10.

Summary of Five Common Threats:

1. **Injection (SQL Injection):**

Description:

Injection attacks occur when malicious input is sent to a web application, tricking it into executing unintended commands or accessing unauthorized data.

Example:
An attacker enters `' OR '1'='1` in a login field, bypassing authentication to gain unauthorized access to user accounts.

Prevention:
- Use parameterized queries or prepared statements.
- Validate and sanitize user input.

2. Broken Authentication:
Description:
Occurs when authentication mechanisms are improperly implemented, allowing attackers to hijack user accounts or session tokens.

Example:
An attacker steals a user's session token through an insecure connection, enabling them to impersonate the user.

Prevention:
- Implement multi-factor authentication (MFA).
- Use secure session management practices (e.g., HTTPS, session timeouts).

3. Cross-Site Scripting (XSS):
Description:
XSS attacks involve injecting malicious scripts into a trusted website, targeting users who visit the site.

Example:
An attacker injects `<script>alert('Hacked');</script>` into a comment section, causing pop-ups or stealing session cookies from other users.

Prevention:
- Use proper output encoding.
- Implement a Content Security Policy (CSP).

4. Insecure Direct Object References (IDOR):
Description:
Occurs when an application exposes internal objects (files, database entries) without proper access control.

Example:
An attacker modifies a URL from `https://example.com/user/123` to `https://example.com/user/124` to access another user's private data.

Prevention:
- Enforce access control checks for all sensitive objects.
- Avoid exposing direct references to internal objects.

5. **Security Misconfiguration:**

Description:
This happens when servers, databases, or frameworks are misconfigured, leaving vulnerabilities open for attackers.

Example:
An application running with default admin credentials (`admin:admin`) allows attackers to gain unauthorized access.

Prevention:
- Disable unnecessary features and services.
- Regularly update and patch software.
- Implement secure configurations for all components.