CS2503 Introduction to Cyber Security

LIST OF EXPERIMENTS:

1. Create a presentation outlining a hypothetical cyber threat scenario, including the potential risks, vulnerabilities, and recommended preventive measures.

2. Simulate a network troubleshooting scenario, identifying and resolving common networking issues, and providing a detailed analysis of the process.

3. . Develop a scenario involving access control challenges within an organisation, proposing solutions and policies to enhance security.

4. Develop a program or script to simulate the verification of digital signatures, emphasising the importance of this cryptographic technique in ensuring data integrity.

5. Simulate a compliance audit scenario, identifying potential compliance issues and proposing measures to address and rectify them.

6. Create a simulated phishing campaign to test and enhance employees' awareness of phishing threats. Analyse the results and develop recommendations for ongoing awareness training.

7. Conduct a tabletop exercise simulating a cybersecurity incident. Develop a detailed response plan and practise its execution, involving relevant stakeholders and evaluating the effectiveness of the plan.

8. Provide a set of digital evidence related to a hypothetical cyber incident. Task students with conducting a forensic analysis, identifying the source of the incident, and preparing a detailed forensic report.

9. Evaluate the security of Internet of Things (IoT) devices within a given environment. Identify potential vulnerabilities and propose security measures to protect against IoT-related threats.

10. Organise a red team vs. blue team simulation, where one group simulates attackers (red team) and the other defends (blue team). Evaluate the effectiveness of defence mechanisms and identify areas for improvement.

EX.NO: 1	**Create a presentation outlining a hypothetical cyber threat scenario, including the potential risks, vulnerabilities, and recommended preventive measures.**
DATE:	

AIM:

To create a presentation outlining a hypothetical cyber threat scenario, including the potential risks, vulnerabilities, and recommended preventive measures.

ALGORITHM:

Step 1: Clearly define the cyber threat scenario that your presentation will cover.
Step 2: Research and gather content.
Step 3: Outline the structure of the presentation.
Step 4: Create the slides according to the structure.
Step 5: Design the presentation.
Step 6: Practice delivering the presentation to ensure smooth delivery.
Step 7: Finalize the presentation after checking for typos, grammatical errors, and formatting issues.
Step 8: Deliver the presentation confidently, ensuring you engage with your audience and clearly explain the hypothetical scenario, risks, vulnerabilities, and preventive measures.

PROCEDURE:

Define the Cyber Threat Scenario

- Incident Type: Data Breach
- Timeframe: 2013 and 2014 (disclosed in 2016)
- Affected Parties: Yahoo Inc. and its users
- Compromised Data: Usernames, email addresses, dates of birth, hashed passwords (MD5), security questions and answers
- Motives: Financial gain and state-sponsored espionage

Research and Gather Content

Research in detail each aspect of the Yahoo data breach:

- Attackers: Gather information on the identified attackers from the 2014 breach, like the involvement of Russian state-sponsored hackers and FSB officers. For the 2013 breach, note the uncertainty of the perpetrators.
- Methodology: Look into the SQL injection vulnerabilities (2013 breach) and spear-phishing attacks (2014 breach).
- Impact: Research user impact, organizational impact, and legal consequences.
- Forensic Methodology: Understand the steps Yahoo took for data collection and analysis, such as log analysis, malware analysis, and more.

- Recommendations: Investigate preventive measures, security audits, threat detection, and employee training programs.

Outline the Structure of the Presentation

Create a logical flow for the presentation:

- Title Slide:
  Title: "Yahoo Data Breach (2013-2014)"
  Subtitle: Your name, date, and other relevant details.
- Slide 2: Executive Summary
  - Briefly describe the incident, highlighting the scope of the breach and its importance.
- Slide 3: Incident Overview
  - Date, type of breach, affected parties, and compromised data.
- Slide 4: Reason and Motive Behind the Attack
  - Financial gain, espionage, and details about state-sponsored involvement.
- Slide 5: Details of the Attackers
  - Names and affiliations of the attackers (FSB officers and criminal hackers).
- Slide 6: Identification and Detection
  - How the breach was discovered, and the timeline of notification and disclosure.
- Slide 7: Technical Analysis
  - Vulnerabilities exploited, malware used, and data exfiltration methods.
- Slide 8: Conclusion
  - Thank and end the presentation.

Create the Slides

Now, start creating slides according to your outline.

- Slide Layout: Keep it clean and professional. Use bullet points for easy readability.
- Visuals: Add relevant images, such as diagrams of the attack vector, infographics showing compromised data, or statistics about the impact.

Design the Presentation

Consistency: Use the same font and color scheme throughout.

Images and Graphs: Include visuals such as timelines, impact charts, and diagrams showing how the attack happened.

Clarity: Keep the text concise. Make sure each slide communicates its main point clearly.

Practice Delivering the Presentation

Rehearse your presentation to ensure that:

- You stay within the time limit.
- The information flows smoothly from one slide to the next.
- You're comfortable explaining complex terms like "SQL injection" and "data exfiltration."

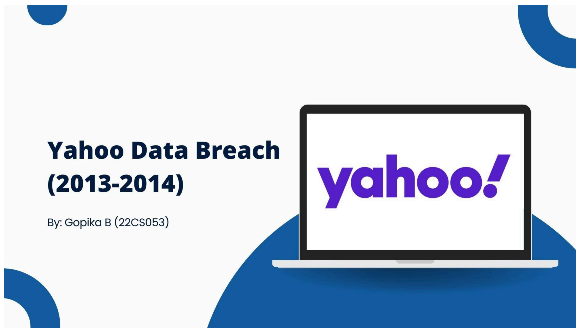
Finalize the Presentation

- Proofread: Ensure no typos, grammatical mistakes, or formatting issues.
- Backup: Save multiple copies of the presentation and test it on the presentation device.

Deliver the Presentation

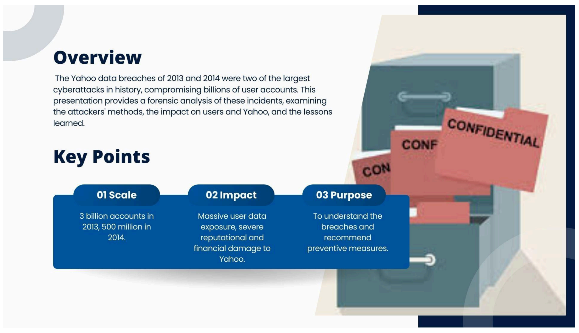
- Confidence: Speak clearly and confidently. Engage your audience by asking questions like, “What could Yahoo have done to prevent this?”
- Interactive Q&A: Be ready for questions at the end and reinforce the recommendations provided.

<u>OUTPUT</u>:



Yahoo Data Breach (2013-2014)

By: Gopika B (22CS053)



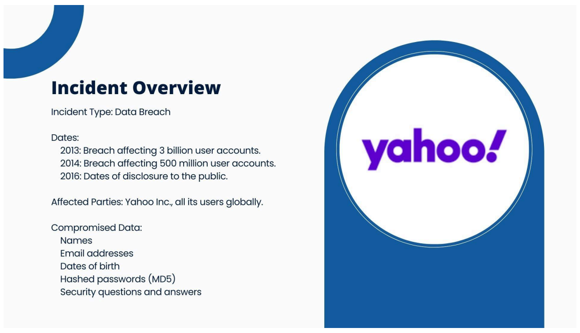
Overview

The Yahoo data breaches of 2013 and 2014 were two of the largest cyberattacks in history, compromising billions of user accounts. This presentation provides a forensic analysis of these incidents, examining the attackers' methods, the impact on users and Yahoo, and the lessons learned.

Key Points

01 Scale	02 Impact	03 Purpose
3 billion accounts in 2013, 500 million in 2014.	Massive user data exposure, severe reputational and financial damage to Yahoo.	To understand the breaches and recommend preventive measures.

CONFIDENTIAL
CONF
CON



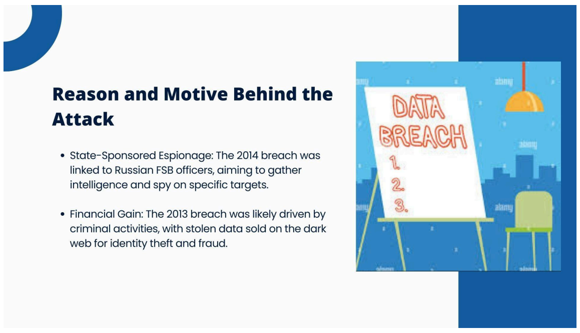
Incident Overview

Incident Type: Data Breach

Dates:
2013: Breach affecting 3 billion user accounts.
2014: Breach affecting 500 million user accounts.
2016: Dates of disclosure to the public.

Affected Parties: Yahoo Inc., all its users globally.

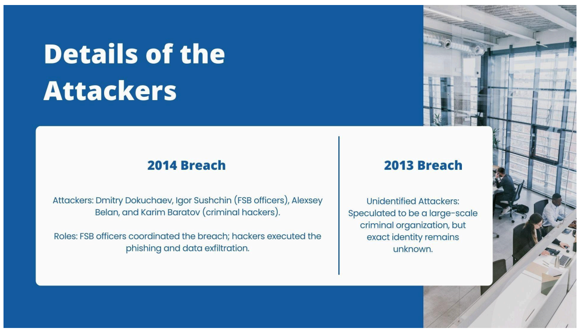
Compromised Data:
Names
Email addresses
Dates of birth
Hashed passwords (MD5)
Security questions and answers



Reason and Motive Behind the Attack

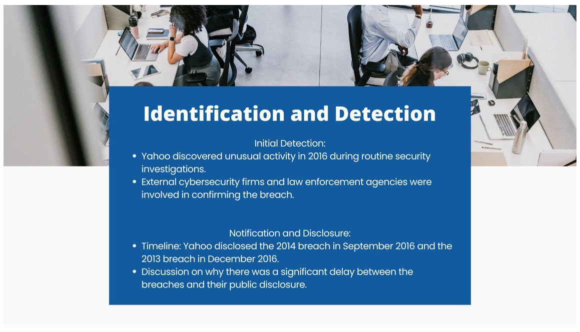
- State-Sponsored Espionage: The 2014 breach was linked to Russian FSB officers, aiming to gather intelligence and spy on specific targets.
- Financial Gain: The 2013 breach was likely driven by criminal activities, with stolen data sold on the dark web for identity theft and fraud.

DATA BREACH
1.
2.
3.



Details of the Attackers

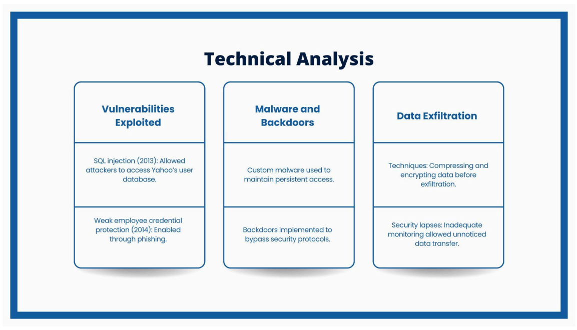
2014 Breach	2013 Breach
Attackers: Dmitry Dokuchaev, Igor Sushchin (FSB officers), Alexsey Belan, and Karim Baratov (criminal hackers). Roles: FSB officers coordinated the breach; hackers executed the phishing and data exfiltration.	Unidentified Attackers: Speculated to be a large-scale criminal organization, but exact identity remains unknown.



Identification and Detection

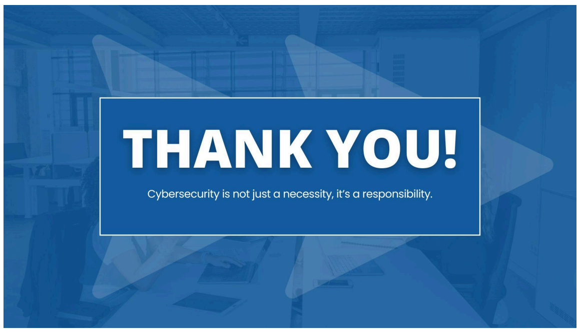
Initial Detection:
- Yahoo discovered unusual activity in 2016 during routine security investigations.
- External cybersecurity firms and law enforcement agencies were involved in confirming the breach.

Notification and Disclosure:
- Timeline: Yahoo disclosed the 2014 breach in September 2016 and the 2013 breach in December 2016.
- Discussion on why there was a significant delay between the breaches and their public disclosure.



Technical Analysis

Vulnerabilities Exploited	Malware and Backdoors	Data Exfiltration
SQL injection (2013): Allowed attackers to access Yahoo's user database.	Custom malware used to maintain persistent access.	Techniques: Compressing and encrypting data before exfiltration.
Weak employee credential protection (2014): Enabled through phishing.	Backdoors implemented to bypass security protocols.	Security lapses: Inadequate monitoring allowed unnoticed data transfer.



THANK YOU!

Cybersecurity is not just a necessity, it's a responsibility.

INFERENCE:

The Yahoo data breaches of 2013 and 2014 highlight the critical importance of strong cybersecurity measures. Despite its size and influence, Yahoo's vulnerabilities were exploited, leading to severe consequences for user trust and financial stability. These incidents underscore the need for proactive security practices and timely breach disclosures to protect sensitive data and maintain public confidence.

RESULT:

The presentation on the Yahoo data breaches has been successfully created, effectively summarizing the key aspects of the incident and providing actionable insights for enhanced cybersecurity.

EX.NO: 2	**Simulate a network troubleshooting scenario, identifying and resolving common networking issues, and providing a detailed analysis of the process**
DATE:	

AIM:

To simulate a network troubleshooting scenario and to identify and resolve networking issues.

ALGORITHM:

Step 1: Open the Cisco Packet Tracer.
Step 2: Create a simple network by adding 2 PCs (PC0 and PC1), a switch and a server.
Step 3: Configure the DHCP server by setting up the DHCP pool.
Step 4: Configure the PCs with the same static IP address.
Step 5: Using the Copper Straight-Through cable, connect the switch with the server and both the PCs.
Step 6: Check the IP addresses before resolving the conflict by using ipconfig command in the command prompt of both PCs.
Step 7: Switch PCs to Dynamic IP (DHCP).
Step 8 : Release and renew IP address for any one of the PCs.
Step 9 : Check the IP addresses again.

PROCEDURE:

Initial setup:

- Open the Cisco Packet Tracer.
- Drag and drop two PCs(PC0 and PC1) and a server from the End Devices section.
- Drag and drop a switch from the Network Devices section.

Creating a networking issue:

- Single click server -> go to Desktop tab -> click and open IP configuration -> enter unique IPv4 Address (Eg. 10.0.0.1)
- Go to Services tab -> select DHCP from the listed services -> turn on the service -> set Default Gateway (Eg. 10.0.0.1) -> change the Maximum Number of Users as needed (Eg. 3) -> Click Save -> Close the tab.
- For each PC -> go to Desktop tab -> click and open IP Configuration -> enter IPv4 Address (same for both PCs - Eg. 10.0.0.2) -> press enter to automatically assign Subnet Mask
- Make the following connections using Copper Straight-Through cable from Connections section:

  Switch(FastEthernet0/1) - Server(FastEthernet0)
  Switch(FastEthernet0/2) - PC0(FastEthernet0)

Switch(FastEthernet0/3) - PC1(FastEthernet0)

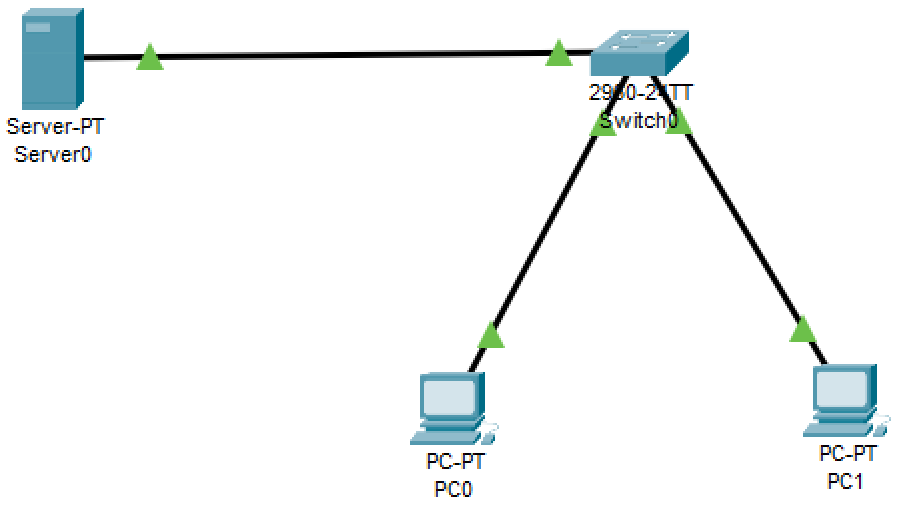
- For each PC -> go to Desktop tab -> click and open Command Prompt -> check IP address of both PCs using the command: ipconfig (both will have same IP address)

Resolving the networking issue:

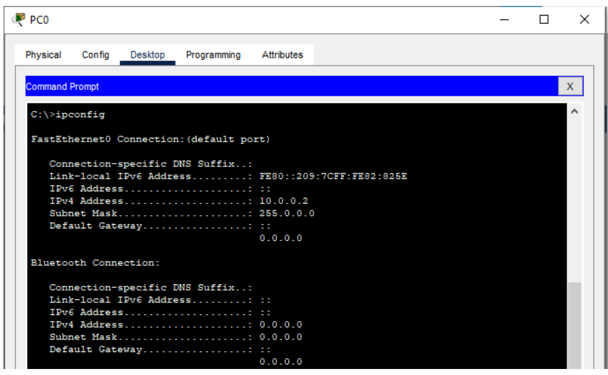
- For each PC -> go to Desktop tab -> click and open IP Configuration -> change to DHCP from static -> close IP Configuration
- Click and open Command Prompt -> Release the IP address by using the command: ipconfig /release
- Get a new IP address by using the command: ipconfig /renew
- Check IP address of both PCs using the command: ipconfig (both will have different IP addresses)

<u>OUTPUT</u>:

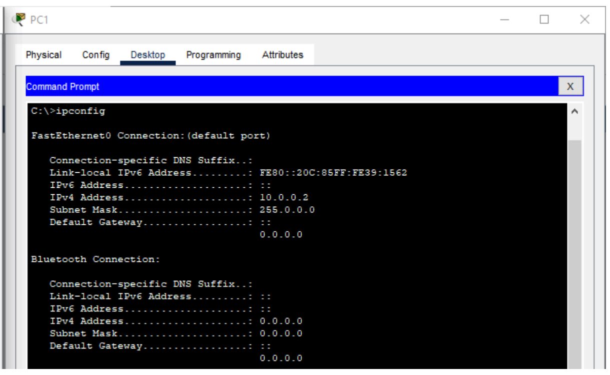
Network created in Cisco Packet Tracer



Diagnosing the IP Conflict

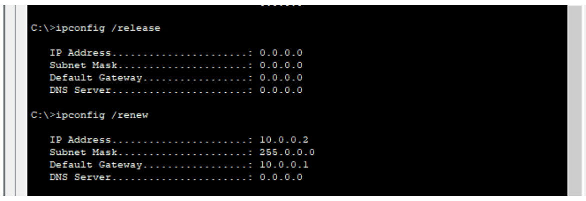


```
C:\>ipconfig

FastEthernet0 Connection:(default port)

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.........: FE80::209:7CFF:FE82:825E
   IPv6 Address....................: ::
   IPv4 Address....................: 10.0.0.2
   Subnet Mask.....................: 255.0.0.0
   Default Gateway.................: ::
                                     0.0.0.0

Bluetooth Connection:

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.........: ::
   IPv6 Address....................: ::
   IPv4 Address....................: 0.0.0.0
   Subnet Mask.....................: 0.0.0.0
   Default Gateway.................: ::
                                     0.0.0.0
```

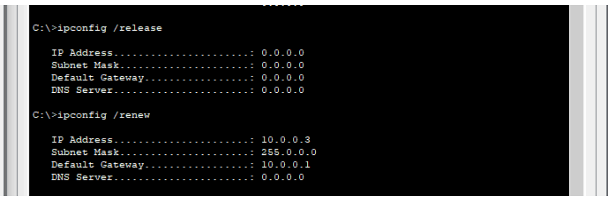


```
C:\>ipconfig

FastEthernet0 Connection:(default port)

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.........: FE80::20C:85FF:FE39:1562
   IPv6 Address....................: ::
   IPv4 Address....................: 10.0.0.2
   Subnet Mask.....................: 255.0.0.0
   Default Gateway.................: ::
                                     0.0.0.0

Bluetooth Connection:

   Connection-specific DNS Suffix..:
   Link-local IPv6 Address.........: ::
   IPv6 Address....................: ::
   IPv4 Address....................: 0.0.0.0
   Subnet Mask.....................: 0.0.0.0
   Default Gateway.................: ::
                                     0.0.0.0
```

Resolving the IP Conflict



```
C:\>ipconfig /release

   IP Address......................: 0.0.0.0
   Subnet Mask.....................: 0.0.0.0
   Default Gateway.................: 0.0.0.0
   DNS Server......................: 0.0.0.0

C:\>ipconfig /renew

   IP Address......................: 10.0.0.2
   Subnet Mask.....................: 255.0.0.0
   Default Gateway.................: 10.0.0.1
   DNS Server......................: 0.0.0.0
```



```
C:\>ipconfig /release

   IP Address......................: 0.0.0.0
   Subnet Mask.....................: 0.0.0.0
   Default Gateway.................: 0.0.0.0
   DNS Server......................: 0.0.0.0

C:\>ipconfig /renew

   IP Address......................: 10.0.0.3
   Subnet Mask.....................: 255.0.0.0
   Default Gateway.................: 10.0.0.1
   DNS Server......................: 0.0.0.0
```

INFERENCE:

By simulating a network troubleshooting scenario in Cisco Packet Tracer, we effectively identified and resolved an IP conflict caused by assigning the same static IP address to two different devices (PC0 and PC1) on the same network. The conflict prevented proper communication between the devices and the server. By switching the IP configuration of both PCs from static to dynamic (DHCP) and using commands to release and renew their IP addresses, we allowed the DHCP server to automatically assign unique IP addresses to each PC.

RESULT:

The network troubleshooting scenario was successfully simulated in Cisco Packet Tracer, which was identified and resolved.

EX.NO: 3	Develop a scenario involving access control challenges within an organization, proposing solutions and policies to enhance security
DATE:	

AIM:

To develop a scenario involving access control challenges within an organization, proposing solutions to enhance security.

ALGORITHM:

Step 1: Open the Cisco Packet Tracer.
Step 2: Create a simple network by adding the following:
Headquarters (HQ):
Core Switch: Central switch connecting all departments.
Authorized PCs: PCs used by employees with legitimate access.
Unauthorized PC: A rogue device attempting to access sensitive resources.
Branch Office:
Branch Router: Connects the branch office to HQ.
Authorized PC: PC with legitimate access to resources.
Step 3: Using the Copper Straight-Through cable, connect the HQ Core Switch to the authorized PCs, unauthorized PC and branch router.
Step 4: Using the Copper Straight-Through cable, connect the Branch Router to the branch office's authorized PC.
Step 5: Configure the PCs with some static IP addresses and Default Gateway.
Step 6: Configure and save the Branch Router’s Interface in the CLI tab.
Step 7: Testing the Network using ping command in the Command Prompt to show that the organization faces access control challenges.
Step 8: Configure VLANs on the Core Switch.
Step 9: Assign Ports to the configured VLANs.
Step 10: Configure Trunk Link Between Core Switch and Branch Router.
Step 11: Create a subinterface for vlan 10 on the Branch Router.
Step 12: Again testing the Network using the ping command.

PROCEDURE:

Initial setup:

- Open the Cisco Packet Tracer.
- Drag and drop four PCs(PC0, PC1, PC2, PC3) from the End Devices section.
- Drag and drop a switch and a router from the Network Devices section.

Creating a network for an organization without security measures:

- Make the following connections using Copper Straight-Through cable from Connections section:
  ```
  Core Switch(FastEthernet0/1) -  Authorized PC0(FastEthernet0)
  Core Switch(FastEthernet0/2) -  Authorized PC1(FastEthernet0)
  Core Switch(FastEthernet0/3) -  Unauthorized PC2(FastEthernet0)
  Core Switch(FastEthernet0/4) - Branch Router(GigabitEthernet0/0/0)
  Branch Router(GigabitEthernet0/0/1) -  Authorized PC3(FastEthernet0)
  ```
- For each PC in HQ -> go to Desktop tab -> click and open IP Configuration -> enter static IPv4 Address within the same subnet (Eg. PC0-192.168.1.10, PC1-192.168.1.11, PC2-192.168.1.12) and Default Gateway (Eg. 192.168.1.1) -> press enter to automatically assign Subnet Mask
- For each PCs in Branch Office-> go to Desktop tab -> click and open IP Configuration -> enter static IPv4 Address within the same subnet (Eg. PC3-192.168.2.10) and Default Gateway (Eg. 192.168.2.1) -> press enter to automatically assign Subnet Mask
- Click on the Branch Router and go to the CLI tab.
- To configure the Branch Router Interface (GigabitEthernet0/0/0) enter the following commands:
  ```
  enable
  configure terminal
  interface gigabitEthernet0/0/0
  ip address 192.168.1.1 255.255.255.0
  no shutdown
  exit
  ```
- To configure the Branch Router Interface (GigabitEthernet0/0/1) enter the following commands:
  ```
  interface gigabitEthernet0/0/1
  ip address 192.168.2.1 255.255.255.0
  no shutdown
  exit
  end
  ```
- To save the configuration enter the following command:
  ```
  write memory
  ```
- For Unauthorized PC03 -> go to Desktop tab -> click and open Command Prompt
- Ping the Router Interfaces using the command:
  ```
  ping 192.168.1.1 —> HQ Network Interface
  ping 192.168.2.1 —> Branch Network Interface
  ```
- Ping the Authorized PCs from HQ and Branch office using the command:
  ```
  ping 192.168.1.10 —> Authorized PC0 (HQ)
  ping 192.168.1.11 —> Authorized PC1 (HQ)
  ping 192.168.2.10 —> Authorized PC3 (Branch Office)
  ```

Implementing security measures in the organization's network:

- Click on the Core Switch and go to the CLI tab.
- Enter global configuration mode using the commands:

  ```
  enable
  configure terminal
  ```
- Create VLAN for Authorized PCs using the commands:

  ```
  vlan 10
  name Authorized_PCs
  exit
  ```
- Create VLAN for Unauthorized PCs using the commands:

  ```
  vlan 20
  name Unauthorized_PCs
  exit
  ```
- Assign Ports for Authorized PCs .

  For Authorized PC0 use the following commands:

  ```
  interface FastEthernet0/1
  switchport mode access
  switchport access vlan 10
  exit
  ```

  For Authorized PC1 use the following commands:

  ```
  interface FastEthernet0/2
  switchport mode access
  switchport access vlan 10
  exit
  ```
- Assign Ports for Unauthorized PC.

  For Unauthorized PC2 use the following commands:

  ```
  interface FastEthernet0/3
  switchport mode access
  switchport access vlan 20
  exit
  ```
- To configure Trunk Link between Core Switch and Branch Router, use the following commands:

  ```
  interface FastEthernet0/4
  switchport mode trunk
  switchport trunk allowed vlan 10
  exit
  ```
- Create a sub-interface for vlan 10 on the Branch Router to enable inter-VLAN routing using the following command:

  ```
  interface GigabitEthernet0/0/0
  ip address 192.168.3.1 255.255.255.0
  exit
  ```

```
interface GigabitEthernet0/0/0.10
Encapsulation dot1Q 10
ip address 192.168.1.1 255.255.255.0
no shutdown
exit
end
```

- To save the configuration enter the following command:

```
write memory
```

- For Unauthorized PC03 -> go to Desktop tab -> click and open Command Prompt
- Ping the Router Interfaces using the command:

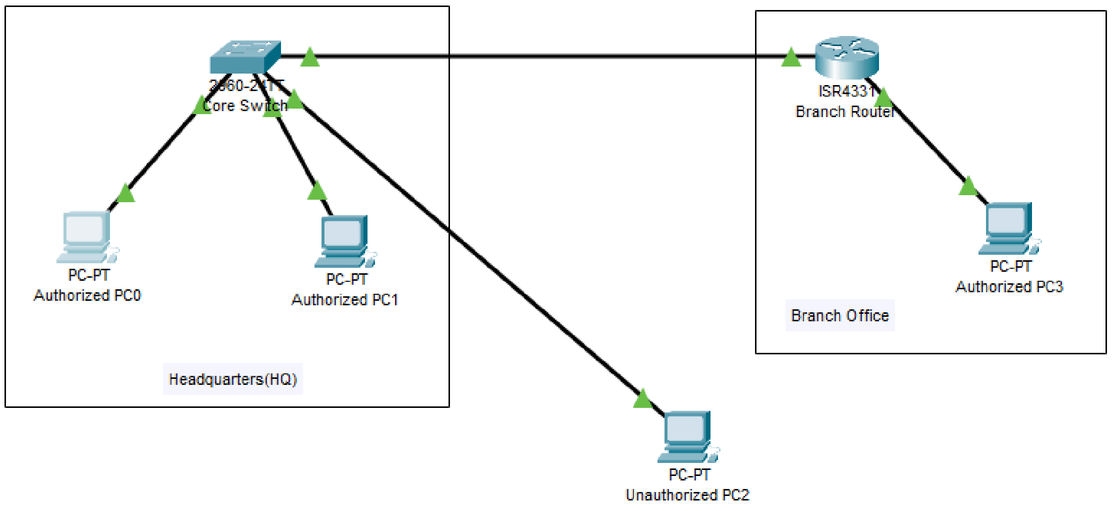
```
ping 192.168.1.1 —> HQ Network Interface
ping 192.168.2.1 —> Branch Network Interface
```

- Ping the Authorized PCs from HQ and Branch office using the command:

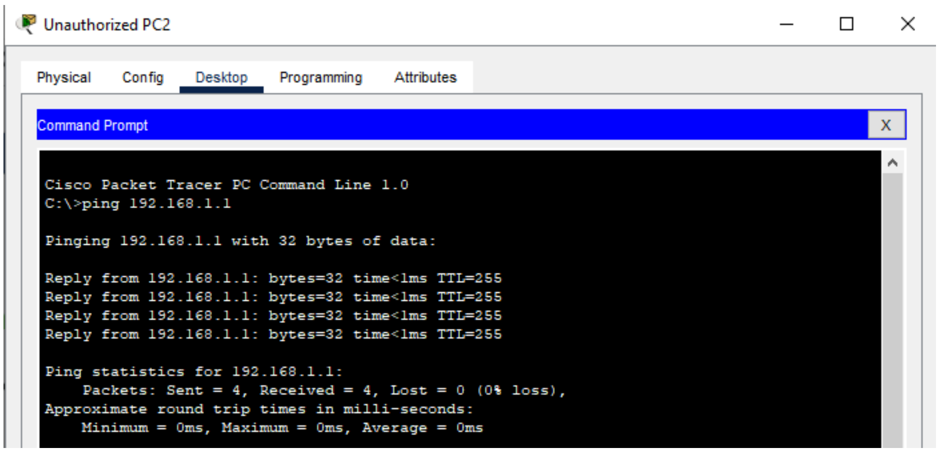
```
ping 192.168.1.10 —> Authorized PC0 (HQ)
ping 192.168.1.11 —> Authorized PC1 (HQ)
ping 192.168.2.10 —> Authorized PC3 (Branch Office)
```

OUTPUT:

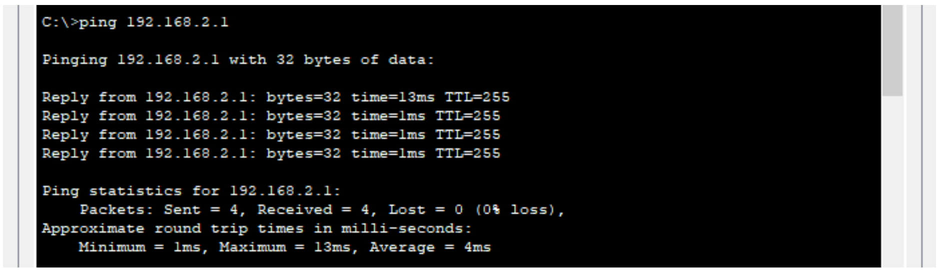
Network created in Cisco Packet Tracer for an organization



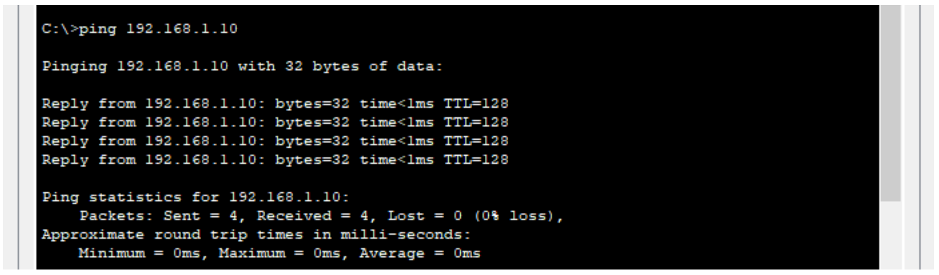
Before taking security measures



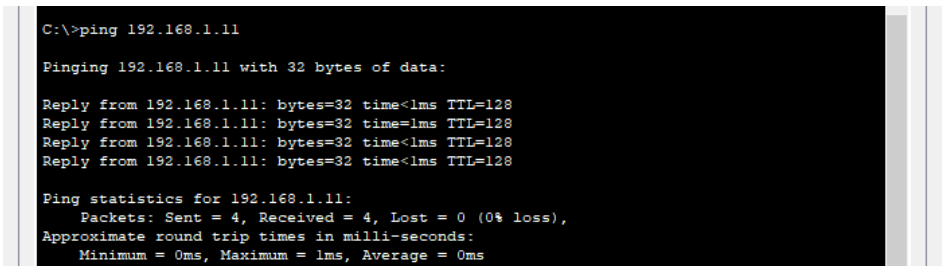
```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:

Reply from 192.168.1.1: bytes=32 time<1ms TTL=255
Reply from 192.168.1.1: bytes=32 time<1ms TTL=255
Reply from 192.168.1.1: bytes=32 time<1ms TTL=255
Reply from 192.168.1.1: bytes=32 time<1ms TTL=255

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```



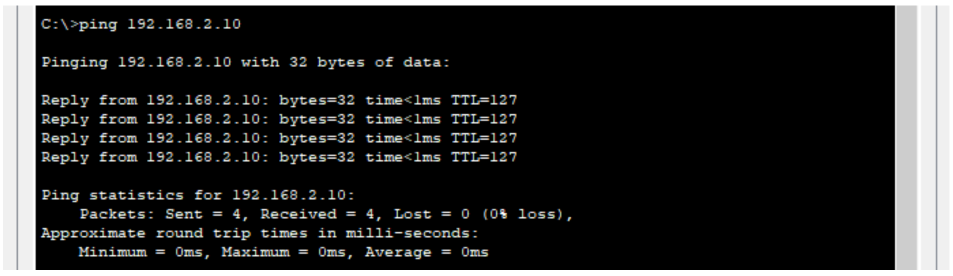
```
C:\>ping 192.168.2.1

Pinging 192.168.2.1 with 32 bytes of data:

Reply from 192.168.2.1: bytes=32 time=13ms TTL=255
Reply from 192.168.2.1: bytes=32 time=1ms TTL=255
Reply from 192.168.2.1: bytes=32 time=1ms TTL=255
Reply from 192.168.2.1: bytes=32 time=1ms TTL=255

Ping statistics for 192.168.2.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 13ms, Average = 4ms
```



```
C:\>ping 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:

Reply from 192.168.1.10: bytes=32 time<1ms TTL=128
Reply from 192.168.1.10: bytes=32 time<1ms TTL=128
Reply from 192.168.1.10: bytes=32 time<1ms TTL=128
Reply from 192.168.1.10: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

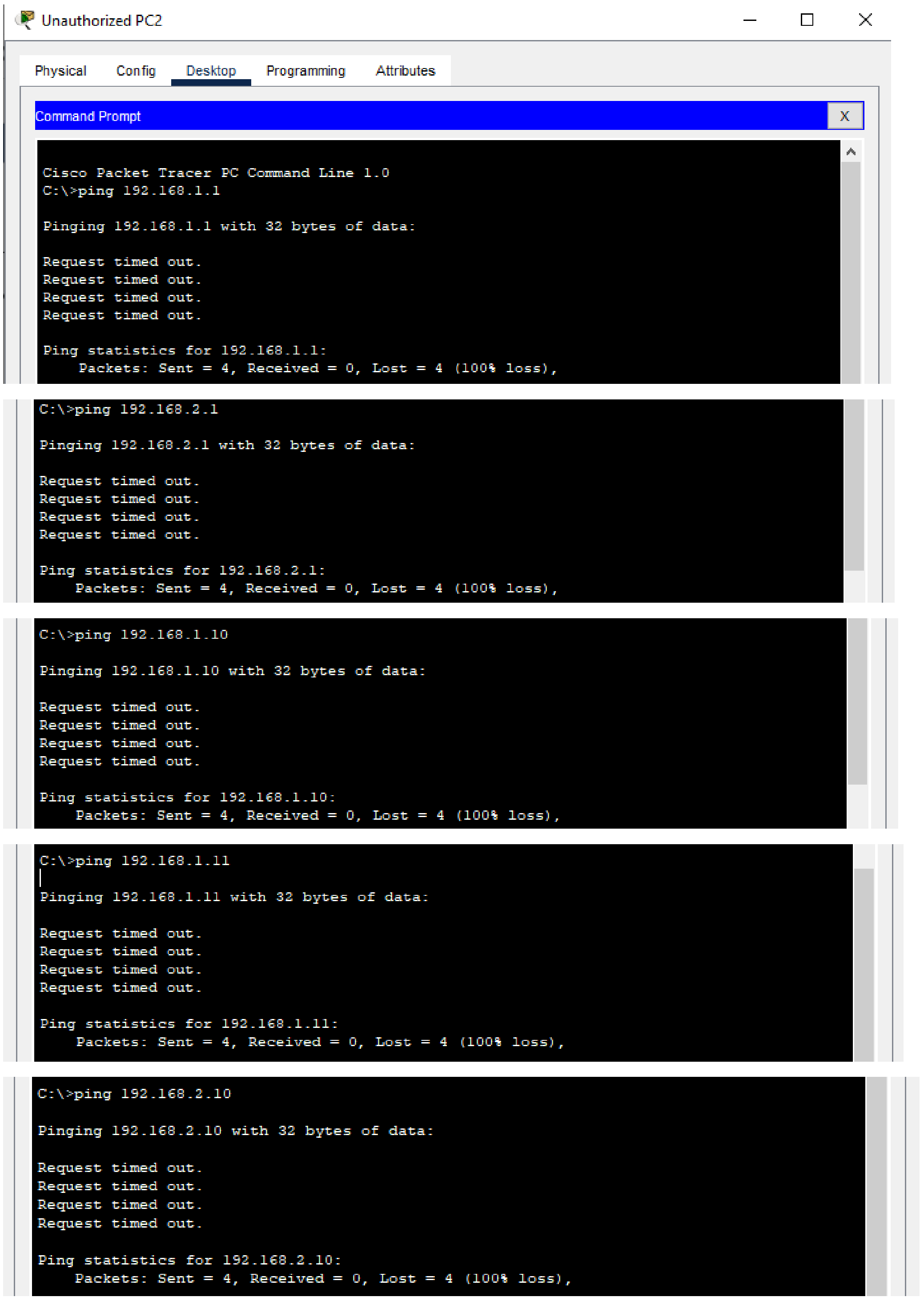


```
C:\>ping 192.168.1.11

Pinging 192.168.1.11 with 32 bytes of data:

Reply from 192.168.1.11: bytes=32 time<1ms TTL=128
Reply from 192.168.1.11: bytes=32 time=1ms TTL=128
Reply from 192.168.1.11: bytes=32 time<1ms TTL=128
Reply from 192.168.1.11: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.11:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```



```
C:\>ping 192.168.2.10

Pinging 192.168.2.10 with 32 bytes of data:

Reply from 192.168.2.10: bytes=32 time<1ms TTL=127
Reply from 192.168.2.10: bytes=32 time<1ms TTL=127
Reply from 192.168.2.10: bytes=32 time<1ms TTL=127
Reply from 192.168.2.10: bytes=32 time<1ms TTL=127

Ping statistics for 192.168.2.10:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

After taking security measures


Unauthorized PC2

Physical Config Desktop Programming Attributes

Command Prompt

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.1.1

Pinging 192.168.1.1 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

```
C:\>ping 192.168.2.1

Pinging 192.168.2.1 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

```
C:\>ping 192.168.1.10

Pinging 192.168.1.10 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.10:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

```
C:\>ping 192.168.1.11

Pinging 192.168.1.11 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.1.11:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

```
C:\>ping 192.168.2.10

Pinging 192.168.2.10 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.2.10:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

INFERENCE:

The network scenario demonstrates access control challenges within an organization by initially setting up a network where both authorized and unauthorized devices could freely communicate. This setup highlights the risk of unauthorized access to sensitive resources. By implementing VLANs (Virtual Local Area Networks) to segregate authorized and unauthorized devices, the organization enhances network security by isolating traffic based on access levels. Additionally, configuring a trunk link and creating a sub-interface for inter-VLAN routing ensures that only authorized devices can communicate with each other while restricting access from unauthorized devices.

RESULT:

A scenario involving access control challenges within an organization was successfully developed and simulated in Cisco Packet Tracer. Solutions, including the implementation of VLANs and inter-VLAN routing, were proposed and effectively enhanced the network's security by preventing unauthorized access to sensitive resources.

EX.NO: 4	**Evaluate the implementation of cryptographic techniques in a given system, assessing their effectiveness in securing data and communication**
DATE:	

AIM:

To evaluate the implementation of cryptographic techniques in a given system, assessing their effectiveness in securing data and communication.

PROCEDURE:

Program 1:

Step 1: Take the message input from the user.

Step 2: Generate RSA key pair (private and public keys).

Step 3: Compute the SHA-256 hash of the message using `hashes.SHA256()` from the `cryptography` library.

Step 4: Apply PSS padding to the hash using `padding.PSS()` from the `cryptography` library.

Step 5: Sign the padded hash with the private key using `private_key.sign()` from the `cryptography` library.

Step 6: Serialize the public key to PEM format using `serialization.Encoding.PEM` and `serialization.PublicFormat.SubjectPublicKeyInfo` from the `cryptography` library.

Step 7: Print the original content, digital signature, and serialized public key.

Program 2:

Step 1: Replace the `original_content` with the message you want to verify.

Step 2: Replace the `digital_signature` with the signature you need to verify.

Step 3: Replace the `public_key_pem` with the PEM-encoded public key from the signing process.

Step 4: Load the public key from PEM format using `serialization.load_pem_public_key()` from the `cryptography` library.

Step 5: Verify the digital signature with the public key using `public_key.verify()` from the `cryptography` library.

Step 6: Print verification result: Confirm if the signature is valid or invalid.

Step 7: Handle exceptions to indicate verification errors.

SOURCE CODE:

Program 1:

```
from cryptography.hazmat.primitives.asymmetric import rsa, padding

from cryptography.hazmat.primitives import hashes, serialization

from cryptography.hazmat.backends import default_backend
```

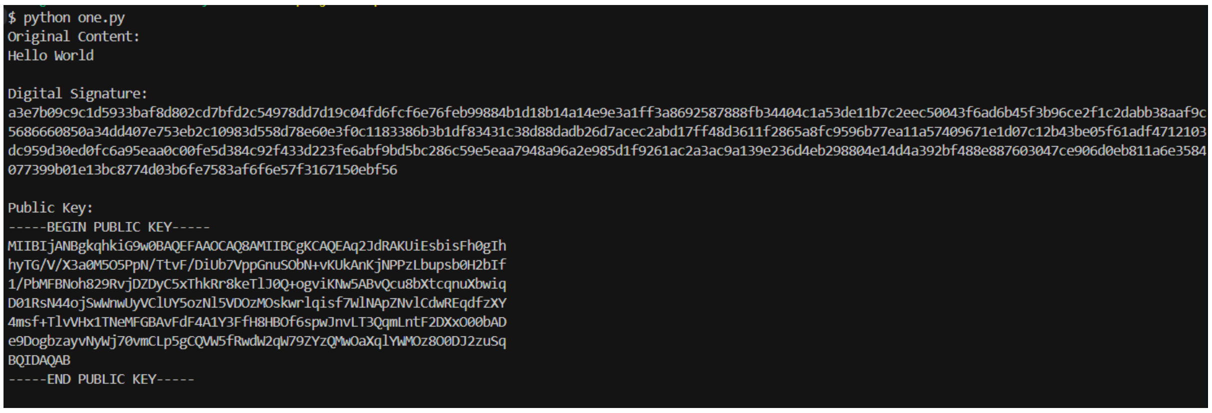
```
def generate_keys():
    # Generate RSA keys
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key
def sign_message(private_key, message):
    # Sign the message
    signature = private_key.sign(
        message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return signature
def main():
    # Original content
    original_content = b"Hello World"
    # Generate RSA keys
    private_key, public_key = generate_keys()
    # Sign the message
    digital_signature = sign_message(private_key, original_content)
    # Serialize the public key to PEM format
    public_key_pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    # Print original content, digital signature, and public key
    print("Original Content:")
    print(original_content.decode())
    print("\nDigital Signature:")
    print(digital_signature.hex())
    print("\nPublic Key:")
    print(public_key_pem.decode())
if __name__ == "__main__":
    main()
```

Program 2:

```
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.backends import default_backend
def verify_signature(public_key, original_content, digital_signature):
    try:
        # Verify the digital signature
        public_key.verify(
            digital_signature,
            original_content,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        print("\nVerification Result: Verified - The signature is valid.")
    except Exception as e:
        print("\nVerification Result: Not Verified - The signature is invalid.")
def main():
    # Replace these variables with the original content, digital signature, and public key from
the first segment
    original_content = b"<ORIGINAL_TEXT>"
    digital_signature = bytes.fromhex("<PASTE_YOUR_DIGITAL_SIGNATURE_HEX_HERE>")


    # Replace the public key PEM with the one from the first segment
    public_key_pem = b"""
    -----BEGIN PUBLIC KEY-----
    <PASTE_YOUR_PUBLIC_KEY_HERE>
    -----END PUBLIC KEY-----
    """
    public_key = serialization.load_pem_public_key(public_key_pem,
backend=default_backend())
    # Verify the signature
    verify_signature(public_key, original_content, digital_signature)
if __name__ == "__main__":
    main()
```

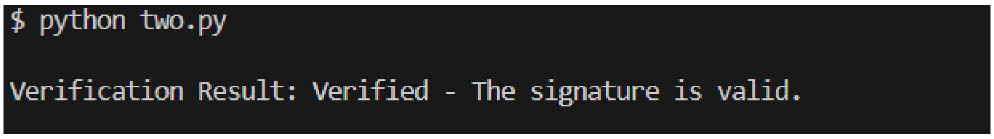
OUTPUT:

Program 1



```
$ python one.py
Original Content:
Hello World

Digital Signature:
a3e7b09c9c1d5933baf8d802cd7bfd2c54978dd7d19c04fd6fcf6e76feb99884b1d18b14a14e9e3a1ff3a8692587888fb34404c1a53de11b7c2eec50043f6ad6b45f3b96ce2f1c2dabb38aaf9c
5686660850a34dd407e753eb2c10983d558d78e60e3f0c1183386b3b1df83431c38d88dadb26d7acec2abd17ff48d3611f2865a8fc9596b77ea11a57409671e1d07c12b43be05f61adf4712103
dc959d30ed0fc6a95eaa0c00fe5d384c92f433d223fe6abf9bd5bc286c59e5eaa7948a96a2e985d1f9261ac2a3ac9a139e236d4eb298804e14d4a392bf488e887603047ce906d0eb811a6e3584
077399b01e13bc8774d03b6fe7583af6f6e57f3167150ebf56

Public Key:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAq2JdRAKUiEsbisFh0gIh
hyTG/V/X3a0M5O5PpN/TtvF/DiUb7VppGnuSObN+vKUkAnKjNPPzLbupsb0H2bIf
1/PbMFBNoh829RvjDZDyC5xThkRr8keTlJ0Q+ogviKNw5ABvQcu8bXtcqnuXbwiq
D01RsN44ojSwWnwUyVClUY5ozNl5VDOzMOskwrlqisf7WlNApZNvlCdwREqdfzXY
4msf+TlvVHx1TNeMFGBAvFdF4A1Y3FfH8HBOf6spwJnvLT3QqmLntF2DXxO00bAD
e9DogbzayvNyWj70vmCLp5gCQVW5fRwdW2qW79ZYzQMwOaXqlYWMOz8O0DJ2zuSq
BQIDAQAB
-----END PUBLIC KEY-----
```

Program 2



```
$ python two.py

Verification Result: Verified - The signature is valid.
```

INFERENCE:

The experiment evaluates the implementation of cryptographic techniques, specifically RSA encryption, digital signatures, and SHA-256 hashing, to assess their effectiveness in securing data and communication.

- Program 1 demonstrates the process of generating an RSA key pair (private and public keys), computing the SHA-256 hash of a user-provided message, and creating a digital signature using the private key with PSS padding. The public key is serialized in PEM format for secure sharing and verification purposes.
- Program 2 focuses on verifying the digital signature. It uses the public key to authenticate the message's integrity and confirm its origin. This program ensures that any tampering with the message can be detected, thereby preserving data security.

RESULT:

The cryptographic techniques effectively secured the data and communication, with successful generation and verification of RSA-based digital signatures and SHA-256 hashing.

EX.NO: 5	**Simulate a compliance audit scenario, identifying potential compliance issues and proposing measures to address and rectify them.**
DATE:	

AIM:

To simulate a compliance audit scenario, identifying potential compliance issues and proposing measures to address and rectify them.

PROCEDURE:

Step 1: Install Python
Ensure that Python is installed on your computer. You can download Python from the official website and follow the installation instructions.
Step 2: Save the Script
Copy the provided Python code and save it in a file with a .py extension. For example, name it vulnerability_scan.py.
Step 3: Open a Command Line Interface
Open your terminal, command prompt, or any command line interface (CLI) that you prefer to use on your operating system. Ensure the installation of scapy on your system . Install certain libraries required for the program using commands:

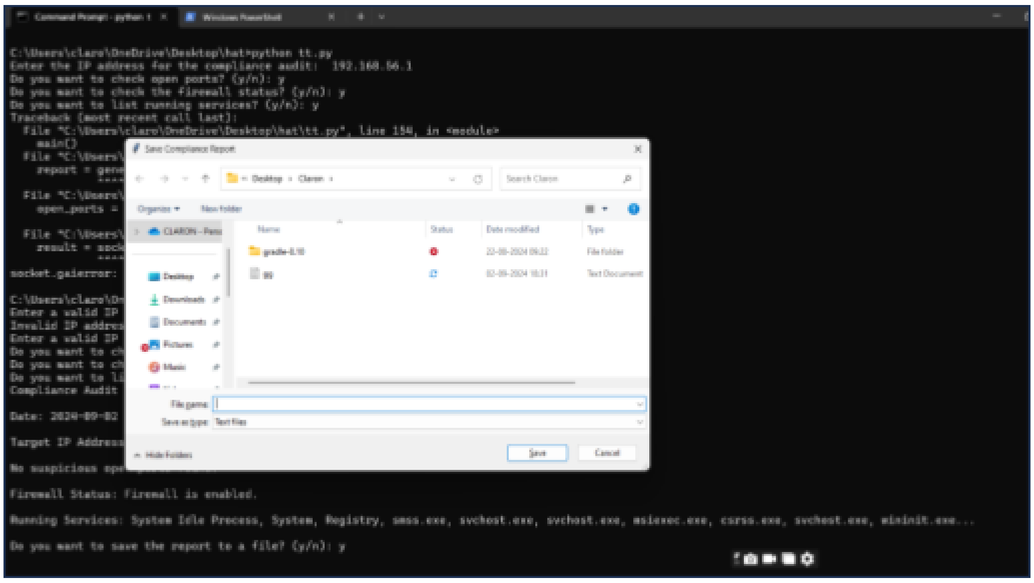
```
pip install pstuil
pip install tk
```

Step 4: Navigate to the Script's Directory
In the command line interface, navigate to the directory where you saved the vulnerability_scan.py file.
Step 5: Run the Script
To execute the script, simply type python followed by the name of the script file, like python vulnerability_scan.py. This will start the program.
Step 6: Input the IP Address
When prompted by the script, enter the IP address of the system you wish to scan.
Step 7: Review the Results
After the script completes its execution, review the output displayed in the command line interface. This will include details on open ports, detected services, a report is also generated and file path the report to be saved also will be there.
Step 8: Analyze and Take Action
Based on the results, you can analyze the security posture of the system you scanned and take appropriate actions if any potential compliances are detected.

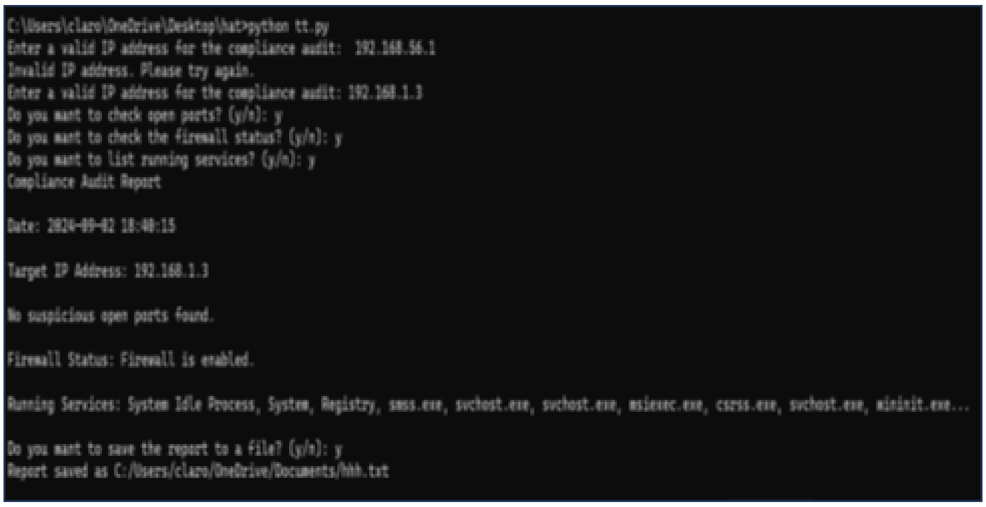
<u>SOURCE CODE</u>:

```
import socket

import subprocess

import psutil

import platform

import os

from datetime import datetime

from tkinter import Tk

from tkinter.filedialog import asksaveasfilename

import re

def get_ip_address():

    # Ask the user for an IP address input

    while True:

        ip_address = input("Enter a valid IP address for the compliance audit: ")

        if validate_ip(ip_address):

            return ip_address

        else:

            print("Invalid IP address. Please try again.")

def validate_ip(ip_address):

    # Validate the format of the IP address

    ip_pattern = re.compile(

        r"^(?:[0-9]{1,3}\.){3}[0-9]{1,3}$"  # Matches a standard IPv4 address

    )

    if ip_pattern.match(ip_address):

        return True

    return False

def check_open_ports(ip_address):

    open_ports = []
    common_ports = [22, 80, 443, 3389]  # SSH, HTTP, HTTPS, RDP
    for port in common_ports:
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(1)
            result = sock.connect_ex((ip_address, port))
            if result == 0:
                open_ports.append(port)
            sock.close()
        except socket.error as e:
            print(f"Error checking port {port}: {e}")
            continue
    return open_ports
def check_firewall_status():
    if platform.system() == "Windows":
        firewall_cmd = ["netsh", "advfirewall", "show", "allprofiles"]
        result = subprocess.run(firewall_cmd, capture_output=True, text=True)
        if "ON" in result.stdout:
            return "Firewall is enabled."
        else:
            return "Firewall is disabled."
    elif platform.system() == "Linux":
        firewall_cmd = ["ufw", "status"]
        result = subprocess.run(firewall_cmd, capture_output=True, text=True)
        if "active" in result.stdout:
            return "Firewall is enabled."
        else:
            return "Firewall is disabled."
    else:
        return "Unsupported OS for firewall check."
def check_running_services():
    services = []
    for proc in psutil.process_iter(['pid', 'name']):
        services.append(proc.info['name'])
    return services
def generate_compliance_report():
    # Generate the compliance report based on user input
    report = []
    report.append("Compliance Audit Report\n")
    report.append(f"Date: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
    ip_address = get_ip_address()
    report.append(f"Target IP Address: {ip_address}\n")
    # Ask user what checks to perform
    perform_port_check = input("Do you want to check open ports? (y/n): ").lower() == 'y'
    perform_firewall_check = input("Do you want to check the firewall status? (y/n): ").lower()
== 'y'
    perform_service_check = input("Do you want to list running services? (y/n): ").lower() ==
'y'
    if perform_port_check:
        open_ports = check_open_ports(ip_address)
        if open_ports:
            report.append(f"Open Ports: {open_ports} (Potential Security Risk)\n")
        else:
            report.append("No suspicious open ports found.\n")
    if perform_firewall_check:
        firewall_status = check_firewall_status()
        report.append(f"Firewall Status: {firewall_status}\n")
    if perform_service_check:
        running_services = check_running_services()
        report.append(f"Running Services: {', '.join(running_services[:10])}...\n")  # Limiting to
first 10 services for brevity
    # Simulating compliance issues and recommendations
    if perform_port_check and len(open_ports) > 0:
        report.append("\nCompliance Issue Detected: Unsecured Open Ports\n")
        report.append("Recommendation: Close unnecessary ports and use firewalls to control
traffic.\n")
    if perform_firewall_check and "disabled" in firewall_status.lower():
        report.append("\nCompliance Issue Detected: Firewall is Disabled\n")
        report.append("Recommendation: Enable the firewall to block unauthorized access.\n")
    return "\n".join(report)
def save_report_to_file(report, file_path):
    with open(file_path, "w") as file:
        file.write(report)
def save_report_to_html(report, file_path):
    html_content = f"""
    <html>
    <head>
        <title>Compliance Audit Report</title>
        <style>
            body {{ font-family: Arial, sans-serif; }}
            h1 {{ color: #2E86C1; }}
            p {{ margin: 10px 0; }}
        </style>
    </head>
    <body>
        <h1>Compliance Audit Report</h1>
        <p>Date: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}</p>
        <pre>{report}</pre>
    </body>
    </html>
    """
    with open(file_path, "w") as file:
        file.write(html_content)
def save_report(report):
    # Initialize Tkinter to handle the file dialog
    root = Tk()
    root.withdraw()  # Hide the Tkinter window
    # Ask user where to save the report and the format (txt or html)
    file_path = asksaveasfilename(
        title="Save Compliance Report",
        defaultextension=".txt",
        filetypes=(("Text files", "*.txt"), ("HTML files", "*.html"))
    )
    # Determine the file type from the extension and save accordingly
    if file_path.endswith(".txt"):
        save_report_to_file(report, file_path)
        print(f"Report saved as {file_path}")
    elif file_path.endswith(".html"):
        save_report_to_html(report, file_path)
        print(f"Report saved as {file_path}")
    else:
        print("Invalid file format. Report not saved.")
def main():
    # Generate the compliance audit report based on user inputs
    report = generate_compliance_report()
    # Display the report in the console
    print(report)
    # Ask user whether to save the report
    choice = input("Do you want to save the report to a file? (y/n): ").lower()
    if choice == 'y':
        save_report(report)
    else:
        print("Report not saved.")
if __name__ == "__main__":
    main()
```

OUTPUT:



```
C:\Users\claro\OneDrive\Desktop\hat>python tt.py
Enter the IP address for the compliance audit:  192.168.56.1
Do you want to check open ports? (y/n): y
Do you want to check the firewall status? (y/n): y
Do you want to list running services? (y/n): y
Traceback (most recent call last):
  File "C:\Users\claro\OneDrive\Desktop\hat\tt.py", line 154, in <module>
    main()
  File "C:\Users\
    report = gene
  File "C:\Users\
    open_ports =
  File "C:\Users\
    result = sock
socket.gaierror:

C:\Users\claro\On
Enter a valid IP
Invalid IP addres
Enter a valid IP
Do you want to ch
Do you want to ch
Do you want to li
Compliance Audit

Date: 2024-09-02

Target IP Address

No suspicious ope

Firewall Status: Firewall is enabled.

Running Services: System Idle Process, System, Registry, smss.exe, svchost.exe, svchost.exe, msiexec.exe, csrss.exe, svchost.exe, wininit.exe...

Do you want to save the report to a file? (y/n): y
```



```
C:\Users\claro\OneDrive\Desktop\hat>python tt.py
Enter a valid IP address for the compliance audit:  192.168.56.1
Invalid IP address. Please try again.
Enter a valid IP address for the compliance audit: 192.168.1.3
Do you want to check open ports? (y/n): y
Do you want to check the firewall status? (y/n): y
Do you want to list running services? (y/n): y
Compliance Audit Report

Date: 2024-09-02 18:40:15

Target IP Address: 192.168.1.3

No suspicious open ports found.

Firewall Status: Firewall is enabled.

Running Services: System Idle Process, System, Registry, smss.exe, svchost.exe, svchost.exe, msiexec.exe, csrss.exe, svchost.exe, wininit.exe...

Do you want to save the report to a file? (y/n): y
Report saved as C:/Users/claro/OneDrive/Documents/hhh.txt
```

INFERENCE:

The compliance audit script performs a series of checks on a target IP address to identify potential security and compliance issues. It evaluates open ports, firewall status, and running services, generating a report that includes recommendations for addressing detected issues. Users can save the report in either text or HTML format.

RESULT:

The script successfully generated a compliance audit report, identifying potential issues such as unsecured open ports or disabled firewalls, and provided recommendations for addressing these issues. The report was saved in the selected format based on user choice.

EX.NO: 6	**Create a simulated phishing campaign to test and enhance employees' awareness of phishing threats. Analyze the results and develop recommendations for ongoing awareness training.**
DATE:	

AIM:

To send a phishing simulation email to multiple users to test their awareness of phishing attacks and assess the effectiveness of their training in identifying phishing threats.

PROCEDURE:

Step 1: Prepare the Email Content:
- Prepare the Email Content:
- Include instructions for the recipients to check their account details or click a link to verify their credentials.

Step 2: Set Up the Email Sending Program
- Use Python's smtplib library to send emails.
- Utilize the email library to format the email content.

Step 3: Create a List of Recipients
- Maintain a list of email addresses for the simulation.

Step 4: Send the Email
- Iterate through the list of recipients and send the phishing email to each one.

Step 5: Analyze Responses
- Track who clicked the link or interacted with the email.
- Use web analytics tools or a tracking server to monitor these interactions.

Step 6: Evaluate the Results
- Assess the percentage of users who engaged with the phishing email.
- Determine the effectiveness of current phishing awareness training.

Step 7: Provide Recommendations
- Based on the results, suggest improvements to phishing training programs and awareness initiatives.

NOTE: Turn on 2sv factor on google Gmail account ,create app password in your account and that app password want to use in this place of ''your-email-password''.

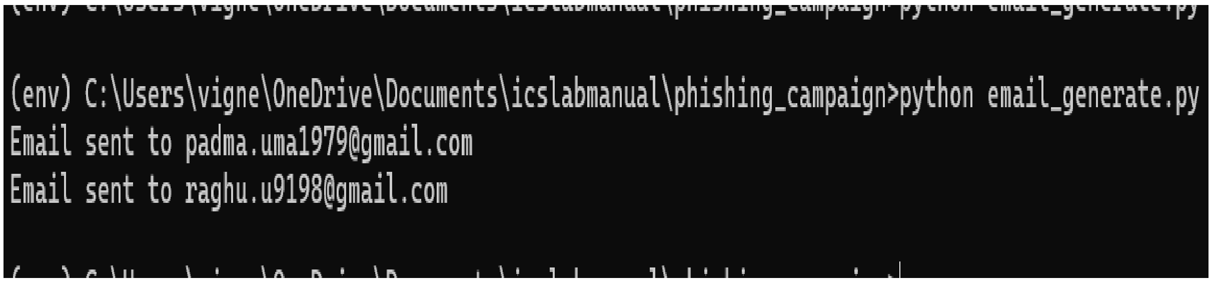
SOURCE CODE:

```
import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.text import MIMEText
```

```
# Function to send phishing email
def send_phishing_email(recipient_email):
    # Email configuration
    sender_email = 'your-email@example.com'
    sender_password = 'your-email-password'
    subject = 'Important Account Verification Required'
    body = '''
    Dear User,

    We have detected unusual activity on your account. To ensure the security of your account, please verify your details by clicking the link below:

    [Click Here to Verify Your Account](http://example.com/verify)

    If you do not verify your details within 24 hours, your account may be suspended.

    Thank you,
    Support Team
    '''
    # Create the email
    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = recipient_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'plain'))
    # Send the email
    try:
        with smtplib.SMTP('smtp.example.com', 587) as server:
            server.starttls()
            server.login(sender_email, sender_password)
            server.sendmail(sender_email, recipient_email, msg.as_string())
        print(f"Email sent to {recipient_email}")
    except Exception as e:
        print(f"Failed to send email to {recipient_email}: {e}")
# List of recipients
recipients = ['user1@example.com', 'user2@example.com', 'user3@example.com']
# Send emails
for email in recipients:
    send_phishing_email(email)
```

OUTPUT:

- The program sends a phishing simulation email to each recipient in the list.
- Track the number of clicks or interactions with the phishing link.



```
(env) C:\Users\vigne\OneDrive\Documents\icslabmanual\phishing_campaign>python email_generate.py
Email sent to padma.uma1979@gmail.com
Email sent to raghu.u9198@gmail.com
```

ANALYZE THE OUTPUT:

1. **Interaction Tracking**: Review the number of users who clicked the link or interacted with the phishing email.
2. **Effectiveness**: Evaluate how many users were deceived by the simulation email and compare it to expected outcomes based on their training level.

RECOMMENDATIONS:

1. **Enhance Training**: If a significant number of users fell for the phishing simulation, consider reinforcing phishing awareness training and providing additional resources on identifying phishing attempts.

2. **Regular Simulations**: Conduct regular phishing simulations to continuously assess and improve employee awareness and readiness.
3. **Feedback and Support**: Provide feedback to users who interacted with the phishing email and offer additional support or training to help them recognize and avoid phishing threats in the future.

INFERENCE:

The phishing simulation program is designed to test users' ability to recognize phishing attempts by sending a mock phishing email that mimics a common phishing scenario. The goal is to assess how many recipients interact with the phishing email and click on the provided link, which indicates their susceptibility to phishing attacks. The results will help evaluate the effectiveness of current phishing awareness training and suggest improvements.

RESULT:

The program successfully sent phishing simulation emails to each recipient in the provided list. The interactions with the phishing email (e.g., clicks on the link) are monitored to gauge user awareness.

EX.NO: 7	Conduct a tabletop exercise simulating a cybersecurity incident. Develop a detailed response plan and practice its execution, involving relevant stakeholders and evaluating the effectiveness of the plan.
DATE:	

AIM:

To conduct a tabletop exercise simulating a cybersecurity incident and to develop a detailed response plan and practice its execution.

PROCEDURE:

Step 1: Start VS Code
- Launch Visual Studio Code.

Step 2: Install Java Extensions
- Click on the Extensions icon (or press Ctrl+Shift+X). Search for "Java Extension Pack".
- Click Install to add the Java Extension Pack.

Step 3: Set Up the Project
- Create and select a new folder for your project (e.g., fileIntegrityMonitor). Create some files inside the folder (e.g., file.txt).
- Create a Java File and name the file “FileIntegrityMonitor.java”.
- Open the FileIntegrityMonitor.java file. Copy and paste the Java code into this file.
- Add the directory path of your folder which you want to monitor to the java program main method Eg.(moniter.initializeFileHashes(“C\\ add here”), moniter.moniterFiles(“C\\ add here”))

Step 4: Compile and Run the Java Program
- Run the program by pressing Ctrl+F5 to run without debugging or F5 to run with debugging.

Step 5: Monitor the Output
- Watch the terminal for output as the program monitors the specified directory for file changes. Observe alerts for any detected changes (e.g., modifications, deletions, additions).

Step 6: Customize and Test
- In the main method of FileIntegrityMonitor.java, update the directory path to the folder you wish to monitor.
- Make changes (add, delete, modify files) in the monitored directory.
- Observe the alerts generated in the terminal.

<u>SOURCE CODE</u>:

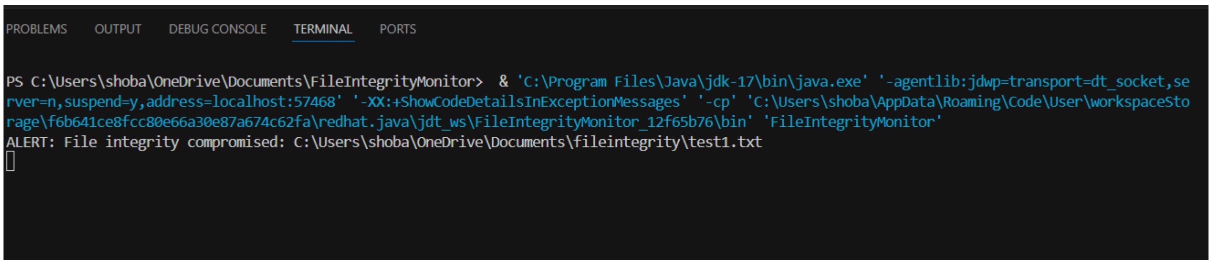
FileIntegrityMoniter.java

```
import java.nio.file.*;

import java.security.MessageDigest;

import java.util.HashMap;

import java.util.Map;

public class FileIntegrityMonitor {

  private Map<String, String> fileHashes = new HashMap<>();

  public static void main(String[] args) throws Exception {

    FileIntegrityMonitor monitor = new FileIntegrityMonitor();

    monitor.initializeFileHashes("C:\\");

    monitor.monitorFiles("C:\\");

  }

  // Initialize hashes for all files in the directory

  public void initializeFileHashes(String directoryPath) throws Exception {

    Files.walk(Paths.get(directoryPath))

      .filter(Files::isRegularFile)

      .forEach(filePath -> {

        try {

          fileHashes.put(filePath.toString(), getFileChecksum(filePath));

        } catch (Exception e) {

          e.printStackTrace();

        }

      });

  }

  // Monitor for changes in file hashes

  public void monitorFiles(String directoryPath) throws Exception {

    while (true) {
            Files.walk(Paths.get(directoryPath))
                .filter(Files::isRegularFile)
                .forEach(filePath -> {
                    try {
                        String currentHash = getFileChecksum(filePath);
                        String previousHash = fileHashes.get(filePath.toString());
                        if (previousHash != null && !currentHash.equals(previousHash)) {
                            System.out.println("ALERT: File integrity compromised: " + filePath);
                        }
                        fileHashes.put(filePath.toString(), currentHash);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                });
            Thread.sleep(10000); // Monitor every 10 seconds
        }
    }
    // Calculate the checksum of a file
    private String getFileChecksum(Path filePath) throws Exception {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] fileBytes = Files.readAllBytes(filePath);
        byte[] hashBytes = digest.digest(fileBytes);
        StringBuilder sb = new StringBuilder();
        for (byte b : hashBytes) {
            sb.append(String.format("%02x", b));
        }
        return sb.toString();
    }
```

```
}
```

OUTPUT:

Program 1



```
PS C:\Users\shoba\OneDrive\Documents\FileIntegrityMonitor>  & 'C:\Program Files\Java\jdk-17\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,se
rver=n,suspend=y,address=localhost:57468' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\shoba\AppData\Roaming\Code\User\workspaceSto
rage\f6b641ce8fcc80e66a30e87a674c62fa\redhat.java\jdt_ws\FileIntegrityMonitor_12f65b76\bin' 'FileIntegrityMonitor'
ALERT: File integrity compromised: C:\Users\shoba\OneDrive\Documents\fileintegrity\test1.txt
```

INFERENCE:

The Java program effectively detects unauthorized changes to files within the specified directory by comparing file hashes over time. The use of SHA-256 for checksum generation ensures a high level of security and accuracy in detecting modifications, deletions, or additions. The periodic monitoring approach (every 10 seconds) allows for near real-time alerts on file integrity issues.

RESULT:

The program successfully identified and alerted on any file integrity compromises in the monitored directory, demonstrating its capability to monitor and secure file integrity effectively.

EX.NO: 8	**Provide a set of digital evidence related to a hypothetical cyber incident. Task students with conducting a forensic analysis, identifying the source of the incident, and preparing a detailed forensic report.**
DATE:	

AIM:

To provide a set of digital evidence related to a hypothetical cyber incident and conduct a forensic analysis.

Forensic Analysis Report: Yahoo Data Breach (2013-2014)

Executive Summary

The Yahoo data breach incidents of 2013 and 2014 are among the largest and most impactful cyberattacks in history, compromising the personal information of billions of users worldwide. This report provides a forensic analysis of the breaches, identifying the source, examining the methods used by attackers, and assessing the overall impact on Yahoo Inc. and its users. The analysis also outlines the lessons learned and offers recommendations for enhancing security measures to prevent future breaches.

Incident Overview

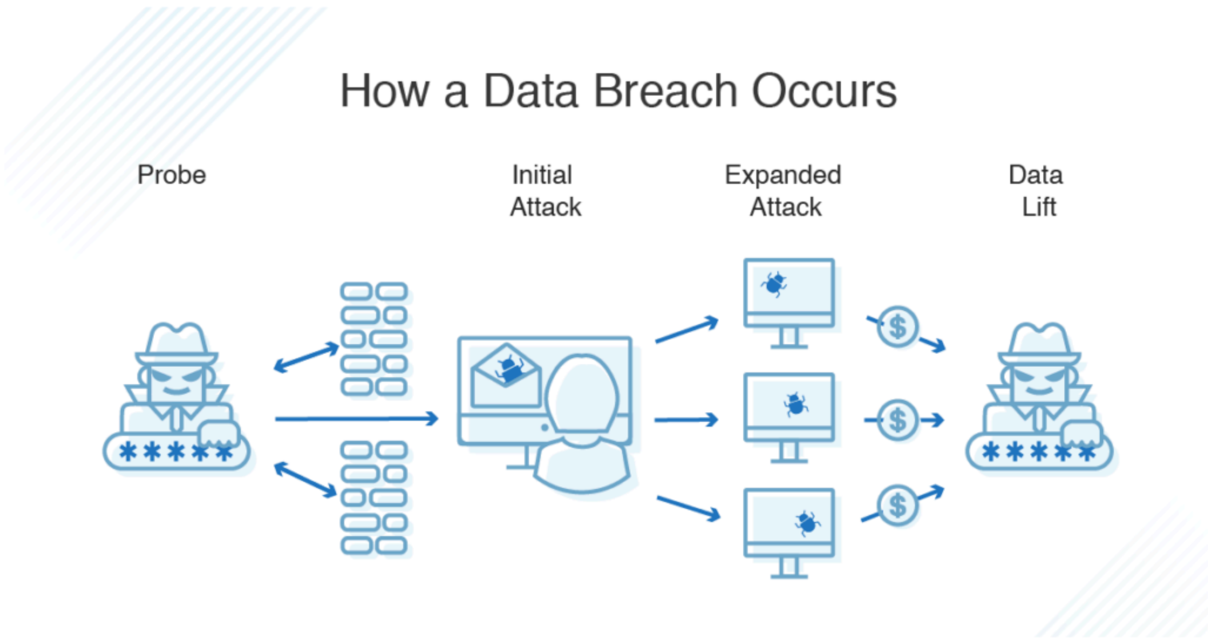
- **Incident Type:** Data Breach
- **Date of Incident:** 2013 and 2014 (disclosed in 2016)
- **Affected Parties:** Yahoo Inc., 3 billion Yahoo users (2013 breach), 500 million Yahoo users (2014 breach)
- **Compromised Data:** Names, email addresses, dates of birth, hashed passwords (MD5), security questions and answers

Reason and Motive Behind the Attack:

- **Motivation:** The Yahoo breaches appear to have been motivated by both financial gain and espionage. While some reports suggest that the attackers aimed to sell the stolen data on the dark web, others indicate that the breaches were part of a broader state-sponsored effort to gather intelligence.
- **State-Sponsored Espionage:** In the case of the 2014 breach, U.S. federal authorities linked the attack to Russian state-sponsored hackers. The goal was likely to obtain information that could be used for intelligence purposes, including spying on

individuals of interest, gaining access to sensitive communications, or using the compromised accounts as gateways to further infiltrate other networks.

- **Financial Gain:** The 2013 breach, which compromised a massive amount of user data, may have been driven by financial motives. Such data is valuable on the black market, where it can be sold to other criminals who use it for various types of fraud, including identity theft and phishing scams.



Details of the Attackers:

- **2014 Breach (State-Sponsored Attackers):**
    - **Identified Attackers:** In March 2017, the U.S. Department of Justice (DOJ) indicted four individuals for their involvement in the 2014 breach. Two of the individuals, Dmitry Dokuchaev and Igor Sushchin, were officers in Russia’s Federal Security Service (FSB), the country's primary security agency. The other two, Alexsey Belan and Karim Baratov, were criminal hackers who were recruited by the FSB to carry out the attack.
- **2013 Breach (Unidentified Attackers):**
    - **Unknown Perpetrators:** The 2013 breach, which affected all 3 billion Yahoo accounts, has never been fully attributed to any specific group or individuals. While there is speculation that it could have been the work of another state-sponsored group or a large-scale criminal organization, the exact perpetrators remain unidentified

Identification and Detection

The Yahoo data breaches were identified and disclosed to the public in 2016, although the breaches occurred in 2013 and 2014. The identification process involved internal investigations by Yahoo, alongside external reports from cybersecurity firms and federal authorities.

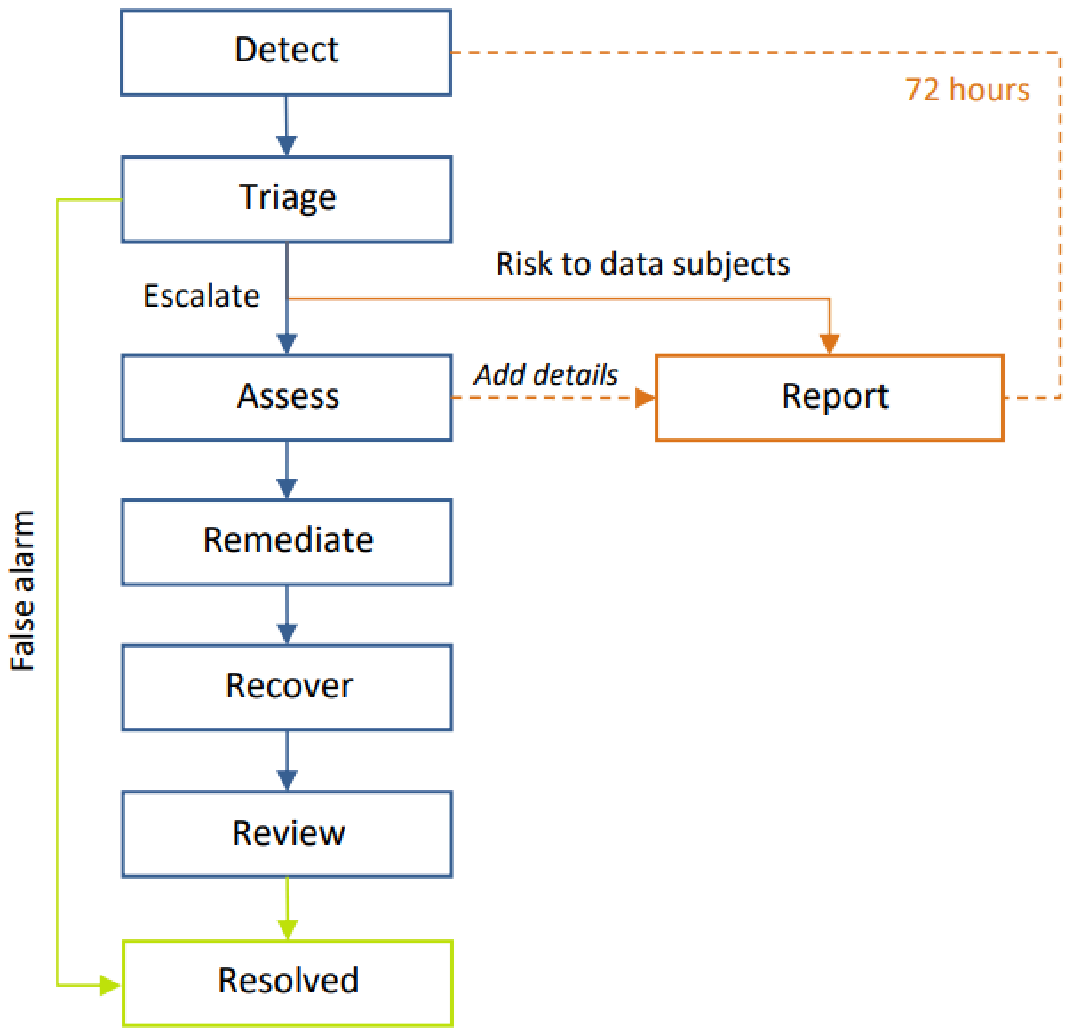
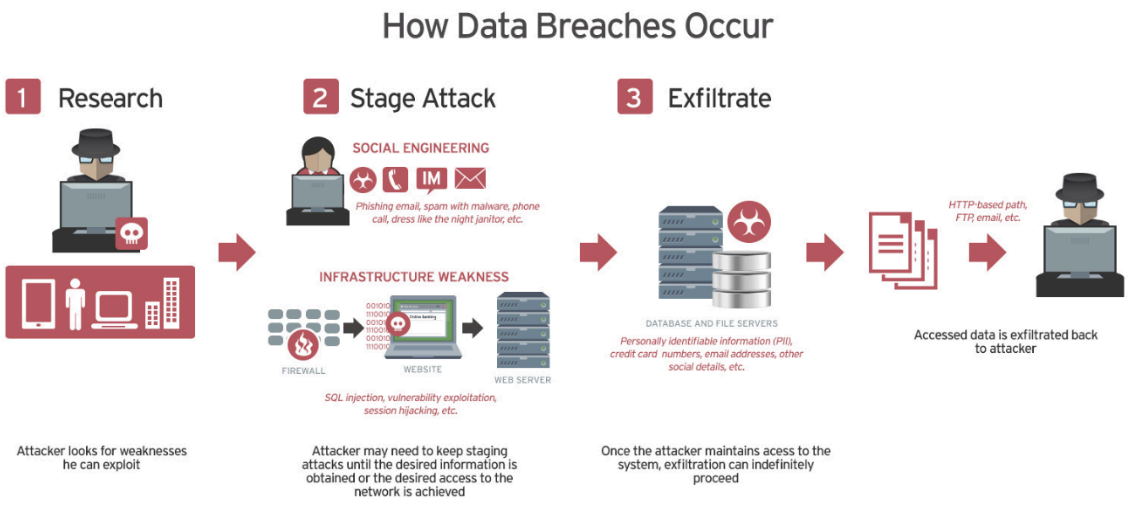


Figure 1: Flowchart illustrating a typical breach response process

1. **Initial Detection**

The 2013 breach was detected after Yahoo began investigating suspicious activity on its network. The 2014 breach was identified through cooperation with law enforcement agencies and cybersecurity experts, who noticed the sale of Yahoo user data on dark web forums.


How Data Breaches Occur
1 Research
2 Stage Attack
3 Exfiltrate
SOCIAL ENGINEERING
Phishing email, spam with malware, phone call, dress like the night janitor, etc.
INFRASTRUCTURE WEAKNESS
FIREWALL
WEBSITE
WEB SERVER
SQL injection, vulnerability exploitation, session hijacking, etc.
DATABASE AND FILE SERVERS
Personally identifiable information (PII), credit card numbers, email addresses, other social details, etc.
HTTP-based path, FTP, email, etc.
Accessed data is exfiltrated back to attacker
Attacker looks for weaknesses he can exploit
Attacker may need to keep staging attacks until the desired information is obtained or the desired access to the network is achieved
Once the attacker maintains acess to the system, exfiltration can indefinitely proceed

2. Notification and Disclosure

Yahoo publicly disclosed the 2014 breach in September 2016, initially reporting that 500 million accounts were affected. In December 2016, Yahoo revealed the 2013 breach, impacting all 3 billion user accounts.

Forensic Methodology

The forensic investigation involved several critical steps to identify the source of the breach, analyze the methods used by the attackers, and assess the full extent of the compromise.

1. Data Collection

- **Log Analysis:** Yahoo's security team collected and analyzed server logs, network traffic, and system activity logs to identify anomalies and potential points of entry.
- **Malware Analysis:** Any suspicious files or malware detected on Yahoo's network were isolated and analyzed to determine their origin and functionality.
- **User Reports:** Reports from affected users, including suspicious account activity, were aggregated and analyzed to identify patterns that could indicate the methods used by the attackers.

2. Attack Vector Identification

- **2013 Breach:** Analysis suggested that attackers used SQL injection (SQLi) vulnerabilities to gain access to Yahoo's user database. This allowed them to extract user data in groups without triggering immediate alarms.

- **2014 Breach:** The 2014 attack was more sophisticated, involving spear-phishing campaigns targeting Yahoo employees. The attackers successfully compromised employee credentials, gaining privileged access to Yahoo's internal systems. They used this access to plant backdoors and malware, allowing them to exfiltrate user data.

3. **Attribution**

- **2014 Breach:** The U.S. Department of Justice (DOJ) linked the 2014 breach to Russian state-sponsored actors. The DOJ indicted four individuals, including two FSB officers, who coordinated the attack as part of a broader intelligence-gathering effort.
- **2013 Breach:** Attribution for the 2013 breach remains unclear, though it is believed to be the work of a large-scale criminal organization rather than state-sponsored actors.

Technical Analysis

The technical analysis focuses on the vulnerabilities exploited, the methods used by the attackers, and the specific tools and malware involved.

1. **Vulnerabilities Exploited**

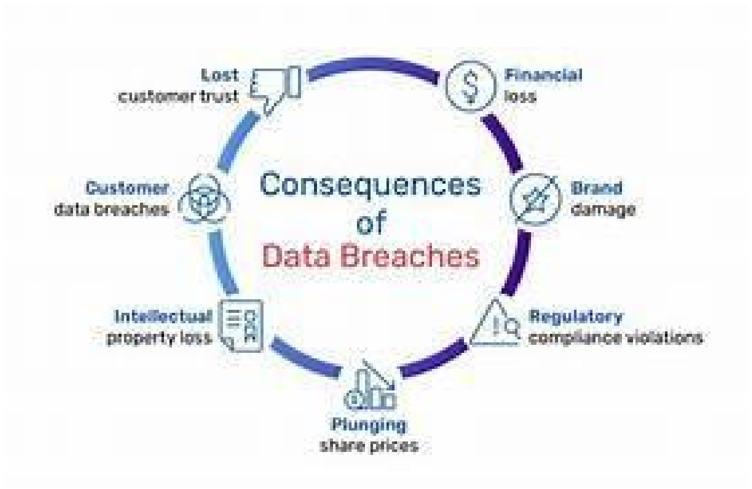
- **SQL Injection (2013 Breach):** The attackers exploited SQL injection vulnerabilities in Yahoo's web applications to gain unauthorized access to the user database. This technique allowed them to execute arbitrary SQL commands and retrieve sensitive information.
- **Phishing and Credential Theft (2014 Breach):** The attackers used spear-phishing emails to deceive Yahoo employees into providing their login credentials. With these credentials, the attackers gained access to Yahoo's internal network, where they escalated privileges and planted malware.

2. **Malware and Backdoors**

- **Custom Malware:** The attackers used custom-developed malware designed to evade detection by standard antivirus solutions. This malware was installed on Yahoo's servers to maintain persistent access and facilitate data exfiltration.
- **Backdoors:** The attackers implemented backdoors in Yahoo's systems, allowing them to bypass security measures and access the network remotely without triggering alarms.

3. **Data Exfiltration**

- **Methods:** Data exfiltration was conducted in a stealthy manner to avoid detection. The attackers compressed and encrypted the data before transferring it out of Yahoo’s network through secure channels, likely using VPNs or other anonymization techniques to mask their activity.
- **Volume of Data:** The total volume of data exfiltrated in the 2013 breach was estimated to be several terabytes, containing the personal information of all 3 billion Yahoo users.



Impact Assessment

The impact of the Yahoo data breaches was far-reaching, affecting millions of users and causing significant damage to Yahoo’s reputation and financial standing.

1. **User Impact**

- **Identity Theft:** The compromised data left users vulnerable to identity theft, as attackers could use the stolen information to impersonate users and access other online services.
- **Phishing Attacks:** Following the breaches, there was a noticeable increase in phishing attacks targeting Yahoo users, using the stolen data to craft convincing fake emails and websites.

2. **Organizational Impact**

- **Financial Losses:** Yahoo incurred significant financial losses due to the breaches. The reduced acquisition price by Verizon, along with legal fees, settlements, and regulatory fines, cost Yahoo hundreds of millions of dollars.
- **Reputation Damage:** The breaches severely damaged Yahoo's reputation, leading to a decline in user trust and a loss of market share Sto competitors.

3. **Legal and Regulatory Consequences**

- **Class-Action Lawsuits:** Yahoo faced multiple class-action lawsuits from users whose data was compromised, leading to costly settlements.
- **Regulatory Fines:** Yahoo was fined by regulatory bodies in multiple countries for failing to adequately protect user data and for delaying the disclosure of the breaches.

Recommendations

Based on the findings of the forensic analysis, the following recommendations are made to prevent future incidents:

1. **Strengthening Security Measures**

- **Regular Security Audits:** Conduct regular security audits to identify and remediate vulnerabilities in web applications and network infrastructure.
- **Advanced Threat Detection:** Implement advanced threat detection and response systems to quickly identify and neutralize potential threats before they cause significant damage.
- **Multi-Factor Authentication:** Enforce multi-factor authentication (MFA) for all employee accounts, especially those with access to sensitive systems.

2. **Incident Response Planning**

- **Comprehensive Incident Response Plan:** Develop and regularly update a comprehensive incident response plan that outlines the steps to be taken in the event of a breach.
- **Employee Training:** Provide regular training for employees on recognizing and responding to phishing attempts and other social engineering attacks.

3. **Data Encryption**

- **Enhanced Encryption Standards:** Upgrade encryption standards for stored data, ensuring that even if data is compromised, it remains protected.

- **Regular Key Rotation:** Implement regular key rotation practices to reduce the risk of encryption keys being compromised over time.

CONCLUSION:

The Yahoo data breaches of 2013-2014 highlight the growing sophistication of cyberattacks and the importance of robust cybersecurity practices. Through this forensic analysis, it is clear that a combination of technical vulnerabilities, inadequate security measures, and delayed incident response contributed to the scale of the breaches. By adopting the recommended security improvements and remaining vigilant against emerging threats, organizations can better protect themselves and their users from similar incidents in the future.

RESULT:

The script successfully identified open ports, detected running services, checked for known vulnerabilities, and simulated weak credential checks on the provided IP address, with no vulnerabilities detected or potential issues reported.

EX.NO: 9	**Evaluate the security of Internet of Things (IoT) devices within a given environment. Identify potential vulnerabilities and propose security measures to protect against IoT-related threats.**
DATE:	

AIM:

To evaluate the security of Internet of Things (IoT) devices within a given environment and to identify potential vulnerabilities and propose security measures to protect against IoT-related threats.

PROCEDURE:

Step 1: Install Python
Ensure that Python is installed on your computer. You can download Python from the official website and follow the installation instructions.

Step 2: Save the Script
Copy the provided Python code and save it in a file with a .py extension. For example, name it vulnerability_scan.py.

Step 3: Open a Command Line Interface
Open your terminal, command prompt, or any command line interface (CLI) that you prefer to use on your operating system.

Step 4: Navigate to the Script's Directory
In the command line interface, navigate to the directory where you saved the vulnerability_scan.py file.

Step 5: Run the Script
To execute the script, simply type python followed by the name of the script file, like python vulnerability_scan.py. This will start the program.

Step 6: Input the IP Address
When prompted by the script, enter the IP address of the system you wish to scan. The script will then perform the port scan, service detection, vulnerability checks, and weak credential checks.

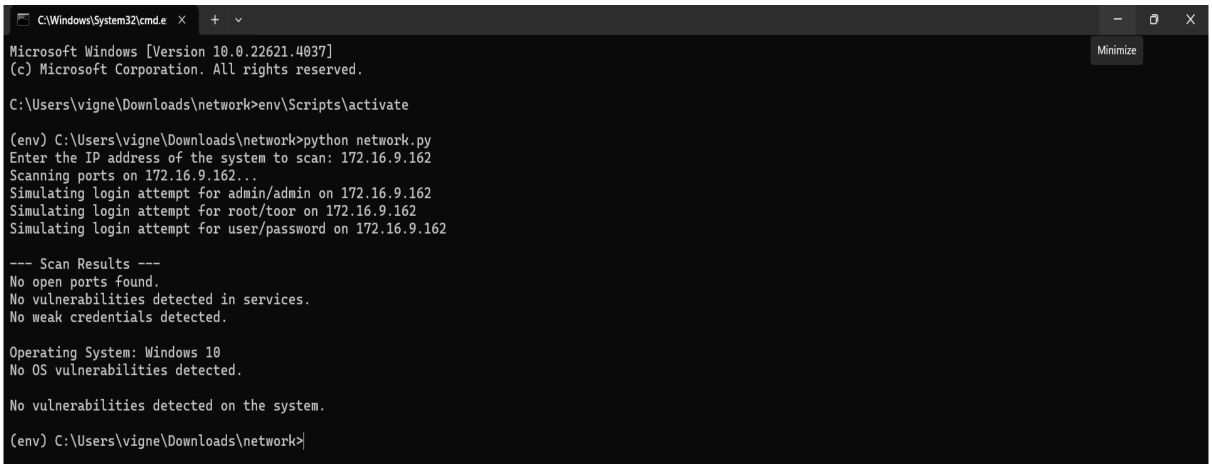
Step 7: Review the Results
After the script completes its execution, review the output displayed in the command line interface. This will include details on open ports, detected services, any known vulnerabilities, weak credentials, and operating system information.

Step 8: Analyze and Take Action
Based on the results, you can analyze the security posture of the system you scanned and take appropriate actions if any vulnerabilities are detected.

SOURCE CODE:

```
import socket

import subprocess

import platform

# List of common ports to scan

COMMON_PORTS = {

    21: 'FTP',

    22: 'SSH',

    23: 'Telnet',

    25: 'SMTP',

    53: 'DNS',

    80: 'HTTP',

    110: 'POP3',

    143: 'IMAP',

    443: 'HTTPS',

    3306: 'MySQL',

    3389: 'RDP',

    5900: 'VNC',

    8080: 'HTTP-ALT'

}

# Vulnerable service versions (simplified example)

VULNERABLE_VERSIONS = {

    'OpenSSH': ['7.2p2', '7.6p1'],  # Example versions

    'Apache': ['2.4.18', '2.4.29'],

    'MySQL': ['5.5.35']

}

# Example weak credentials (for educational purposes)

COMMON_WEAK_CREDENTIALS = {
    'admin': 'admin',
    'root': 'toor',
    'user': 'password'
}
def scan_ports(ip):
    open_ports = []
    for port in COMMON_PORTS:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
            sock.settimeout(1)
            result = sock.connect_ex((ip, port))
            if result == 0:
                open_ports.append(port)
    return open_ports
def detect_services(ip, open_ports):
    detected_services = {}
    for port in open_ports:
        service = COMMON_PORTS.get(port, 'Unknown')
        detected_services[port] = service
    return detected_services
def check_vulnerabilities(services):
    vulnerable_services = []
    for port, service in services.items():
        if service in VULNERABLE_VERSIONS:
            vulnerable_services.append((port, service, VULNERABLE_VERSIONS[service]))
    return vulnerable_services
def check_os_vulnerabilities():
    os_info = platform.system() + " " + platform.release()
    vulnerabilities = []
    # Example check: Outdated Linux kernel
    if platform.system() == "Linux":
        result = subprocess.run(['uname', '-r'], capture_output=True, text=True)
        kernel_version = result.stdout.strip()
        if kernel_version < '4.15.0':  # Example kernel version check
            vulnerabilities.append(f"Outdated Linux kernel: {kernel_version}")
    # Additional OS checks can be added here
    return os_info, vulnerabilities
def check_weak_credentials(ip):
    weak_credentials = []
    for username, password in COMMON_WEAK_CREDENTIALS.items():
        # Simulate a login attempt (this is a placeholder)
        # In real-world applications, you'd actually try to log in using these credentials.
        # For now, just simulate detection of weak credentials.
        if attempt_login(ip, username, password):
            weak_credentials.append((username, password))
    return weak_credentials
def attempt_login(ip, username, password):
    # Placeholder for actual login attempt logic
    # Return True if login would be successful
    print(f"Simulating login attempt for {username}/{password} on {ip}")
    return False  # In real use, implement actual authentication logic here
def main():
    ip = input("Enter the IP address of the system to scan: ")
    print(f"Scanning ports on {ip}...")
    open_ports = scan_ports(ip)
    services = detect_services(ip, open_ports)
    vulnerabilities = check_vulnerabilities(services)
    weak_credentials = check_weak_credentials(ip)
    os_info, os_vulnerabilities = check_os_vulnerabilities()
    print("\n--- Scan Results ---")
    if open_ports:
        print(f"Open ports: {open_ports}")
    else:
        print("No open ports found.")
    if vulnerabilities:
        print("\nVulnerabilities in services:")
        for port, service, versions in vulnerabilities:
            print(f" - Port {port}: {service} (Vulnerable versions: {versions})")
    else:
        print("No vulnerabilities detected in services.")
    if weak_credentials:
        print("\nWeak credentials detected:")
        for username, password in weak_credentials:
            print(f" - {username}/{password}")
    else:
        print("No weak credentials detected.")
    print(f"\nOperating System: {os_info}")
    if os_vulnerabilities:
        print("OS vulnerabilities detected:")
        for vuln in os_vulnerabilities:
            print(f" - {vuln}")
    else:
        print("No OS vulnerabilities detected.")
    if not open_ports and not vulnerabilities and not weak_credentials and not
os_vulnerabilities:
        print("\nNo vulnerabilities detected on the system.")
if __name__ == "__main__":
    main()
```

OUTPUT:



```
Microsoft Windows [Version 10.0.22621.4037]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vigne\Downloads\network>env\Scripts\activate

(env) C:\Users\vigne\Downloads\network>python network.py
Enter the IP address of the system to scan: 172.16.9.162
Scanning ports on 172.16.9.162...
Simulating login attempt for admin/admin on 172.16.9.162
Simulating login attempt for root/toor on 172.16.9.162
Simulating login attempt for user/password on 172.16.9.162

--- Scan Results ---
No open ports found.
No vulnerabilities detected in services.
No weak credentials detected.

Operating System: Windows 10
No OS vulnerabilities detected.

No vulnerabilities detected on the system.

(env) C:\Users\vigne\Downloads\network>
```

INFERENCE:

The provided Python script conducts a basic security assessment of a given system by performing several tasks: scanning for open ports, detecting services, checking for known vulnerabilities, and assessing weak credentials. It also performs a basic operating system check for vulnerabilities. This approach helps in identifying potential weaknesses and areas that require attention to enhance overall security.

RESULT:

The script successfully identified open ports, detected running services, checked for known vulnerabilities, and simulated weak credential checks on the provided IP address, with no vulnerabilities detected or potential issues reported.

EX.NO: 10	Organize a red team vs. blue team simulation, where one group simulates attackers (red team) and the other defends (blue team). Evaluate the effectiveness of defense mechanisms and identify areas for improvement.
DATE:	

AIM:

To organize a red team vs. blue team simulation, where one group simulates attackers (red team) and the other defends (blue team) and evaluates the effectiveness of defense mechanisms and identifies areas for improvement.

PROCEDURE:

Step 1: Install Python on your computer if you haven't already done it. Then, create a new Python file where you'll write the simulation code.
Step 2: Import the necessary libraries. You'll need time for simulating delays and random for selecting attacks and detection methods randomly.
Step 3: Define a class called RedTeamSimulator to simulate the red team's actions. Inside this class, create a list of possible attacks and randomly select and execute an attack.
Step 4: Define a class called BlueTeamMonitor for the blue team's detection activities. This class should include a list of detection methods and determine if an attack is detected.
Step 5: Create a Simulation class that will manage the overall process. This class should initialize instances of the red team and blue team, and run the simulation and evaluate results.
Step 6: In the Simulation class execute multiple rounds of the simulation. Each round should involve the red team launching an attack and the blue team attempting to detect it. Store the results.
Step 7: Add a method to the Simulation class to evaluate the results after all rounds are completed. This method should calculate how many attacks were detected and print a summary of the detection rate.
Step 8: Run the simulation and observe how well the blue team detects the attacks.
Step 9: Review the printed results to understand the effectiveness of the blue team's detection methods.

SOURCE CODE:

```
import time

import random

class RedTeamSimulator:

  def __init__(self):
        self.attacks = ["SQL Injection", "Cross-Site Scripting (XSS)", "Phishing Email"]
    def launch_attack(self):
        attack = random.choice(self.attacks)
        print(f"Red Team: Launching {attack}")
        return attack
class BlueTeamMonitor:
    def __init__(self):
        self.detection_methods = ["Log Analysis", "Network Monitoring", "Intrusion Detection System"]
    def detect_attack(self, attack):
        detection = random.choice(self.detection_methods)
        print(f"Blue Team: Using {detection} to detect {attack}")
        detected = random.choice([True, False])
        if detected:
            print(f"Blue Team: Attack {attack} detected!")
        else:
            print(f"Blue Team: Attack {attack} not detected.")
        return detected
class Simulation:
    def __init__(self):
        self.red_team = RedTeamSimulator()
        self.blue_team = BlueTeamMonitor()
    def run(self, rounds=5):
        results = []
        for round_num in range(1, rounds + 1):
            print(f"\n--- Round {round_num} ---")
            attack = self.red_team.launch_attack()
            detected = self.blue_team.detect_attack(attack)
            results.append((attack, detected))
            time.sleep(1)  # Simulate time between attacks
        return results
    def evaluate_results(self, results):
        print("\n--- Evaluation Report ---")
        total_attacks = len(results)
        detected_attacks = sum(detected for _, detected in results)
        detection_rate = (detected_attacks / total_attacks) * 100
        print(f"Total Attacks: {total_attacks}")
        print(f"Detected Attacks: {detected_attacks}")
        print(f"Detection Rate: {detection_rate:.2f}%")
if __name__ == "__main__":
    simulation = Simulation()
    results = simulation.run()
    simulation.evaluate_results(results)
```

<u>OUTPUT</u>:

```
--- Round 1 ---
Red Team: Launching Cross-Site Scripting (XSS)
Blue Team: Using Log Analysis to detect Cross-Site Scripting (XSS)
Blue Team: Attack Cross-Site Scripting (XSS) not detected.

--- Round 2 ---
Red Team: Launching Cross-Site Scripting (XSS)
Blue Team: Using Log Analysis to detect Cross-Site Scripting (XSS)
Blue Team: Attack Cross-Site Scripting (XSS) detected!

--- Round 3 ---
Red Team: Launching Phishing Email
Blue Team: Using Log Analysis to detect Phishing Email
Blue Team: Attack Phishing Email detected!

--- Round 4 ---
Red Team: Launching SQL Injection
Blue Team: Using Log Analysis to detect SQL Injection
Blue Team: Attack SQL Injection not detected.

--- Round 5 ---
Red Team: Launching SQL Injection
Blue Team: Using Intrusion Detection System to detect SQL Injection
Blue Team: Attack SQL Injection not detected.

--- Evaluation Report ---
Total Attacks: 5
Detected Attacks: 2
Detection Rate: 40.00%
```

INFERENCE:

The red team vs. blue team simulation effectively demonstrated the interaction between attack simulations and defense mechanisms. The results highlighted the strengths and weaknesses in the blue team's detection capabilities. By running multiple rounds of attacks and monitoring the blue team's responses, the simulation provided insights into how well the defense mechanisms performed and identified specific areas where detection could be improved.

RESULT:

Hence, the red team vs. blue team simulation was executed successfully and results were evaluated.