DHAANISH AHMED COLLEGE OF ENGINEERING

Dhaanish Nagar, Padappai, Chennai – 601301
Approved By AICTE, New Delhi,
Affiliated to Anna University, Chennai.
www.dhaanish.in

Name : .....

Dept. / Year / Sec :.....

Subject Code & Name :.....

Register No. :.....

DHAANISH AHMED COLLEGE OF ENGINEERING

Dhaanish Nagar, Padappai, Chennai – 601301

BONAFIDE CERTIFICATE

This is to certify that, this a Bonafide Record Work done by

Mr./Ms.....

Year.....Semester.....Register No.....

Department of .....in the

.....during the

academic year 20 - 20

_____ _____

Signature Signature

Lab-In-Charge Head of the Department

Submitted for the Anna University practical Examination held on.....

Signature of
Internal Examiner
with Date

Signature of
External Examiner
with Date

S.No	Date	Experiment	Pg No	Marks	Signature
1		Implement symmetric key algorithms			
2		Implement asymmetric key algorithms and key exchange algorithms			
3		Implement digital signature schemes			
4		Installation of Wire shark, tcpdump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram.			
5		Check message integrity and confidentiality using SSL			
6		Experiment Eavesdropping, Dictionary attacks,MITM attacks			
7		Experiment with Sniff Traffic using ARP Poisoning			
8		Demonstrate intrusion detection system using any tool.			
9		Explore network monitoring tools			
10		Study to configure Firewall, VPN			

EX NO:1	Implement symmetric key algorithms
DATE:	

Aim: To implement symmetric key algorithms

Procedure/Features:

1. Install Required Libraries: Begin by installing any necessary libraries or cryptographic modules. For Python, you can use libraries like `cryptography` or `PyCryptodome`.

2. Generate a Symmetric Key:Use a cryptographic library to generate a random symmetric key. You can typically do this with a key generation function provided by the library.

3. Create an Encryption Object: Initialize an encryption object using the generated symmetric key. This object should be capable of both encryption and decryption.

4. Encrypt a Message: Convert the message you want to encrypt to bytes and use the encryption object to encrypt the message. This typically involves calling an encryption function/method with the message and key as parameters.

5. Decrypt a Message: To decrypt the encrypted message, use the same encryption object and method but provide the encrypted data and the key as input. The method should return the original message.

6. Secure Key Management: In a production environment, it's crucial to manage the symmetric key securely. Key management involves generating, storing, and sharing the key with authorized parties in a secure manner. Using secure key exchange protocols and practices is essential.

7. Testing: Before deploying your encryption program, thoroughly test it to ensure that encryption and decryption work correctly. Test with various messages and edge cases to validate the program's reliability.

8. Integration: Integrate your encryption program into your application or system as needed. Ensure that the program can handle encryption and decryption seamlessly.

9. Secure Key Storage:When not in use, securely store the symmetric key. It's important to protect the key from unauthorized access.

10. Documentation: Document the encryption process, including how keys are generated, distributed, and managed, for future reference and maintenance.

11. Security Review: Consider a security review or audit of your encryption program to identify and address potential vulnerabilities and ensure it meets the required security standards.

Program:

```
from cryptography.fernet import Fernet
key = Fernet.generate_key()
cipher_suite = Fernet(key)
message = "Hello, World!".encode()
encrypted_message = cipher_suite.encrypt(message)
decrypted_message = cipher_suite.decrypt(encrypted_message)
print("Original message:", message.decode())
print("Encrypted message:", encrypted_message)
print("Decrypted message:", decrypted_message.decode())
```

Output:-

Original message: Hello, World!

Encrypted message:b'gAAAAABlUaAQUyuMedIMOnbJtW8fgaPDT1s4h9Oh0KA
WZUfTET-MPPDqKl_obhP2pANVMpSfdTeimBpRvR6V-UMpeMHbGYe-9w=='

Decrypted message: Hello, World!

Result:

Thus the above program is executed successfullly

EX NO:2	Implement asymmetric key algorithms and key exchange algorithms
DATE:	

Aim: To Implement asymmetric key algorithms and key exchange algorithms

Procedure:

Asymmetric Key Algorithm (RSA):

1. Generate RSA Keys:

- Use a cryptography library like cryptography in Python.
- Generate a private key with a specified key size and public exponent.
- Derive the corresponding public key from the private key.
- Serialize both private and public keys to a format like PEM for storage or sharing.

2. Encrypt and Decrypt Data:

- Select data to be encrypted.
- Use the public key to encrypt the data.
- Transmit or store the encrypted data securely.
- Use the private key to decrypt the data, ensuring confidentiality.

Key Exchange Algorithm (Diffie-Hellman):

1. Generate Diffie-Hellman Keys:

- Choose a backend and generate Diffie-Hellman parameters with a generator and key size.
- Generate a private key based on the parameters.
- Derive the corresponding public key from the private key.
- Serialize and share the public key with the other party.

2. Perform Key Exchange:

- The other party receives the shared public key.
- Load the received public key.
- Use the private key to generate a shared secret.
- Transmit the shared secret securely to the other party.
- Both parties now have a shared secret derived independently.

Program:-

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives  import  serialization
from cryptography.hazmat.primitives.asymmetric import rsa
private_key = rsa.generate_private_key(public_exponent=65537,key_size=2048,
 backend=default_backend())
public_key = private_key.public_key()
private_pem = private_key.private_bytes(encoding=serialization.Encoding.PEM,
 format=serialization.PrivateFormat.TraditionalOpenSSL,
   encryption_algorithm=serialization.NoEncryption())
public_pem = public_key.public_bytes(encoding=serialization.Encoding.PEM,
   format=serialization.PublicFormat.SubjectPublicKeyInfo)
with open('private_key.pem', 'wb') as f: f.write(private_pem)
with open('public_key.pem', 'wb') as f: f.write(public_pem)
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
data_to_encrypt = b"Hello, RSA!"
ciphertext =
public_key.encrypt(data_to_encrypt,padding.OAEP(mgf=padding.MGF1(algorithm=hashes.
SHA256()),
       algorithm=hashes.SHA256(),
       label=None
   )
)
decrypted_data = private_key.decrypt(ciphertext,padding.OAEP(
       mgf=padding.MGF1(algorithm=hashes.SHA256()),
       algorithm=hashes.SHA256(),
       label=None
   )
)
print(f"Original Data:  {data_to_encrypt}")
print(f"Decrypted Data: {decrypted_data.decode('utf-8')}")
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import dh
from cryptography.hazmat.primitives import serialization


parameters = dh.generate_parameters(generator=2, key_size=2048,
backend=default_backend())
private_key = parameters.generate_private_key()
public_key  =  private_key.public_key()
public_pem = public_key.public_bytes(
   encoding=serialization.Encoding.PEM,
   format=serialization.PublicFormat.SubjectPublicKeyInfo
)
with open('dh_public_key.pem', 'wb') as f:
   f.write(public_pem)
```

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
with open('dh_public_key.pem', 'rb') as f:
    other_public_key_bytes = f.read()

other_public_key = serialization.load_pem_public_key(
    other_public_key_bytes,
    backend=default_backend()
)
shared_key = private_key.exchange(other_public_key)
print(f"Shared Key: {shared_key.hex()}")
```

Output:-

Original Data: b'Hello, RSA!'

Decrypted Data: Hello, RSA!

Shared Key:
6a312826847b8e412129b838956812523ef80c72aef405c14334adf881dd2b36cafe7f805098
8dee8bbab7b474d385195cfb60dc54cdcf6fab4ea11a2b6994aaeae4285ed8916627fb4f051a
f55a1f1c607a554d2e369d7c7520e6f31a07c87de87e990fe94acf2a6157b253e9768a44514a
52ba079b46a5f4fe1c1c1dcf8ed42b813f0edc1631a2f797e34dadf9cd7cda1da24bb895ca23
6efb341d1562d555425c9a0e37a2cc8b27567e4bf7b689d5edc01ddacdada38068dc6eb395
a86b7c4f2c484b5f719b3d739ed33a012674217ba90c7fb6190833f7c01e6e67a358be1cf5e1
cb59d72b9e6229940ad3c4994aba11f2123658d4e19d8f3ad8b7278b41

Result:-

The above program was executed successfully

EX NO:3	Implement digital signature schemes
DATE:	

Aim: To Implement digital signature schemes

Procedure:

1. Choose Cryptographic Algorithms:

- Select appropriate cryptographic algorithms for the digital signature scheme. Common choices include:
- Hash Functions: For generating a fixed-size hash of the message.
- Public-Key Cryptography: For creating and verifying the digital signature.

2. Key Generation:

- Generate a key pair for the digital signature scheme.
- Private Key: Used to create digital signatures.
- Public Key: Shared openly for signature verification.

3. Signing a Message:

- To create a digital signature for a message:
- Hash the message using a secure hash function.
- Encrypt the hash value using the private key to create the digital signature.

4. Signature Verification: To verify a digital signature:

- Decrypt the digital signature using the sender's public key to obtain the hash value.
- Hash the received message using the same hash function.
- Compare the computed hash with the decrypted hash. If they match, the signature is valid.

5. Ensure Key Security:

- Keep the private key secure. It should only be accessible to the entity that signs messages.
- Share the public key openly or through a secure channel.

Program:-

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend()
)
public_key = private_key.public_key()
private_pem = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.NoEncryption()
)
public_pem = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding
def sign_message(private_key, message):
    signature = private_key.sign(
        message,
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        hashes.SHA256()
    )
    return signature
def verify_signature(public_key, message, signature):
    try:
        public_key.verify(
            signature,
            message,
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            hashes.SHA256()
        )
        return True
    except Exception:
        return False
message_to_sign = b"Hello, Digital Signature!"
signature = sign_message(private_key, message_to_sign)
verification_result = verify_signature(public_key, message_to_sign, signature)
print(f"Original Message: {message_to_sign}")
print(f"Signature Verification Result: {verification_result}")
```

Output:-

Original Message: b'Hello, Digital Signature!'

Signature Verification Result: True

Result:-

Thus the above program is executed successfully.

EX NO:4	Installation of Wire shark, tcpdump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram.
DATE:	

Aim: To show the Installation of Wire shark, tcpdump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram.

Procedure:

1. Install Wireshark:

- Download Wireshark:
- Visit the Wireshark download page.
- Download the latest stable version for Windows.
- Run the downloaded installer.
- Follow the installation wizard instructions.
- During installation, you might be prompted to install WinPcap or Npcap (packet capture libraries). Install the recommended version.

Run Wireshark:

- After installation, launch Wireshark.
- Wireshark will ask you to select an interface for capturing. Choose the appropriate network interface (e.g., Ethernet or Wi-Fi).

2. Capture UDP/TCP Traffic:

Start a Capture:

- Click on the interface you want to capture traffic from.
- Click the green "Start" button to begin capturing packets.
-

Generate UDP/TCP Traffic:

- Initiate client-server communication using UDP or TCP protocols. This could be achieved by running applications that communicate over these protocols, such as a web browser (TCP) or a DNS query (UDP).

Stop the Capture:

- Once you've generated enough traffic, click the red "Stop" button to stop the capture.

3. Analyze UDP/TCP Datagrams:

Filter for UDP or TCP Traffic:

- Apply a display filter to show only UDP or TCP traffic.
- For UDP: udp
- For TCP: tcp

Analyze Packets:

- Click on individual packets to view details.
- Expand the packet details to see the UDP/TCP headers and payload.

Identify Datagrams:

- Observe the source and destination ports to identify UDP/TCP datagrams.
- For UDP, look for packets with the UDP protocol and analyze the UDP header information.
- For TCP, examine the TCP headers, including source and destination ports.

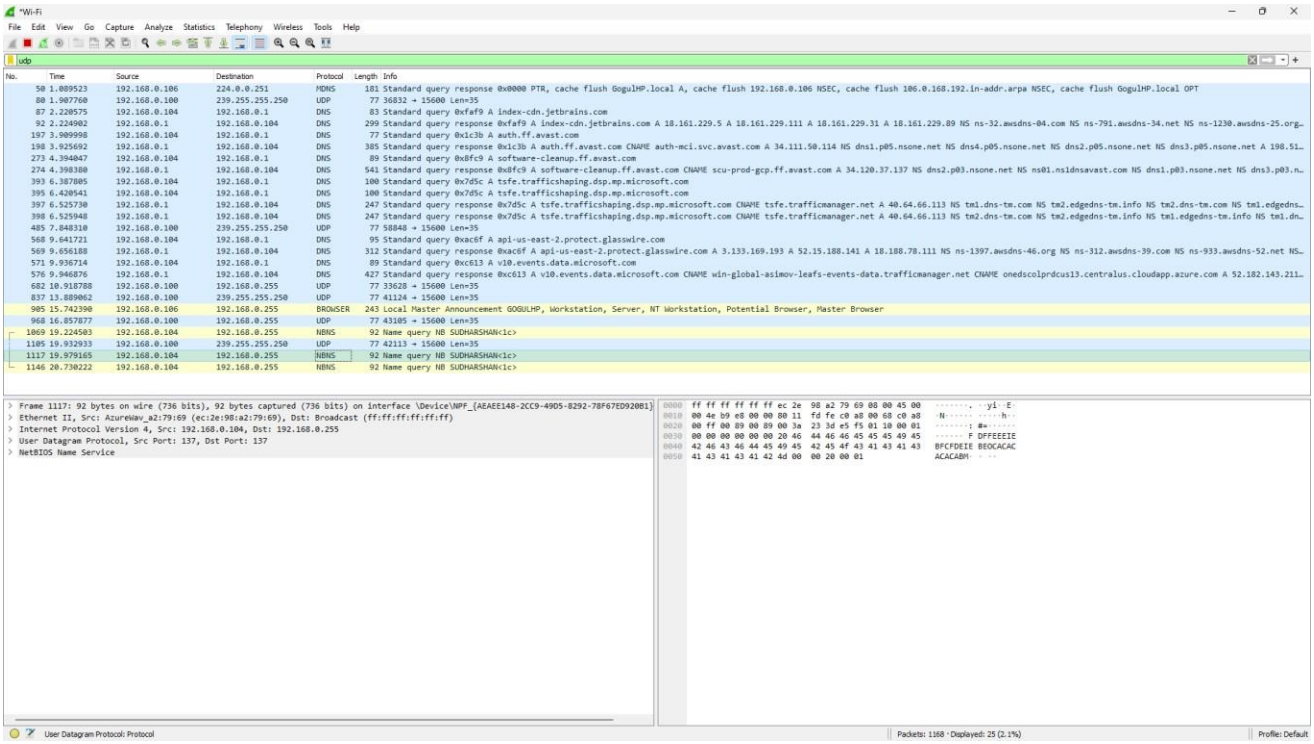
Save Captures:

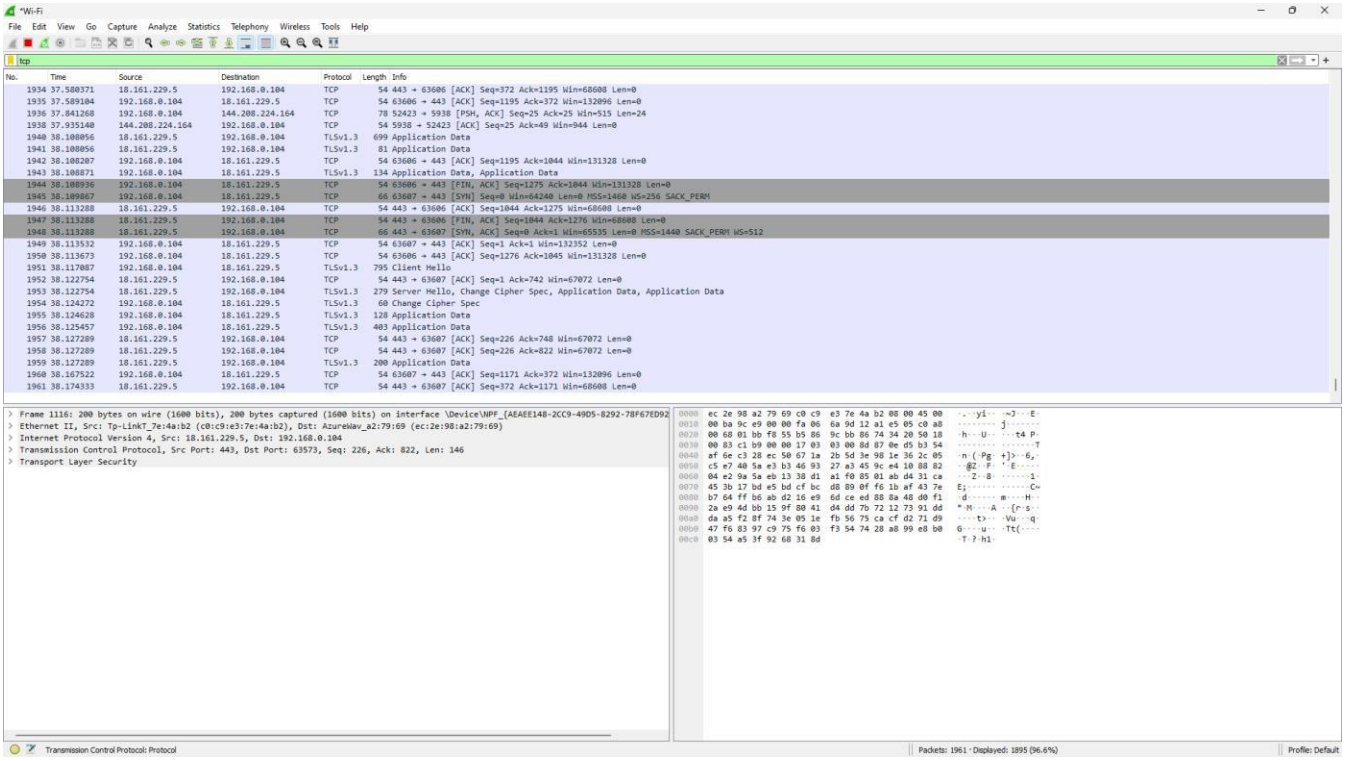
- Save the captured packets for later analysis.
- File > Save As to save the capture in a PCAP file.

Colorization:

- Wireshark uses colorization to represent different protocols.
  Green for TCP, Dark Blue for UDP, etc.

Output:-





Result:-

Thus the above program is executed successfully

EX NO:5	Check message integrity and confidentiality using SSL
DATE:	

Aim: To Check message integrity and confidentiality using SSL

Procedure:

Hash Functions:

- SSL/TLS uses cryptographic hash functions to ensure the integrity of messages.
- Before transmitting data, SSL/TLS calculates a hash value (MAC - Message Authentication Code) of the message using a secure hash function (e.g., SHA-256).
- The hash value is sent along with the message.

Digital Signatures:

- For non-symmetric key exchange methods, digital signatures may be used to ensure message integrity.
- The sender signs the hash value of the message with their private key.
- The recipient can verify the signature using the sender's public key.

HMAC (Hash-based Message Authentication Code):

- SSL/TLS can use HMAC for symmetric key algorithms.
- Both the sender and receiver share a secret key.
- The hash value is computed using this shared key.

Message Confidentiality:

1. **Encryption:**

- SSL/TLS encrypts the data being transmitted between the client and the server.
- This prevents eavesdroppers from understanding the content of the messages.
- The level of encryption depends on the chosen cipher suite and key exchange method.

2. **Symmetric Encryption:**

- SSL/TLS often uses symmetric-key encryption for data confidentiality.
- A symmetric key is exchanged securely between the client and server.
- This shared key is used for encrypting and decrypting the actual data.

Asymmetric Encryption (Public-Key Cryptography):

- Public-key cryptography is used in the initial handshake to exchange a pre-master secret.
- This pre-master secret is then used to derive symmetric keys for encryption.
- Asymmetric encryption is computationally more expensive and is typically used for key exchange rather

than encrypting the entire data stream.

SSL/TLS Handshake:

1. **ClientHello:**

- The client initiates a connection by sending a "ClientHello" message to the server.

2. **ServerHello:**

- The server responds with a "ServerHello" message.
- During this exchange, the client and server agree on cryptographic parameters, including the encryption algorithms, keys, and other settings.
-

Key Exchange:

- The client and server perform key exchange based on the agreed-upon parameters.
- This can involve asymmetric encryption (RSA, Diffie-Hellman) or pre-shared keys.

Program:-

SSL SERVER:-

```
import ssl
import socket
HOST = '127.0.0.1'
PORT = 12345
CERT_FILE = 'server.crt'
KEY_FILE = 'server.key'
context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
context.load_cert_chain(certfile=CERT_FILE, keyfile=KEY_FILE)
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
    server_socket.bind((HOST, PORT))
    server_socket.listen()
    print(f"Server listening on {HOST}:{PORT}...")
    with context.wrap_socket(server_socket, server_side=True) as ssl_socket:
        print(f"SSL connection from {ssl_socket.getpeername()}.")
        data = ssl_socket.recv(1024)
        print(f"Received encrypted data: {data.decode()}")
        response = "Hello, client! This is a secure connection."
        ssl_socket.sendall(response.encode())
```

SSL CLIENT:-

```
import ssl
import socket
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 12345
CERT_FILE = 'client.crt'
KEY_FILE = 'client.key'
context = ssl.create_default_context(ssl.Purpose.SERVER_AUTH)
context.load_cert_chain(certfile=CERT_FILE, keyfile=KEY_FILE)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
    with context.wrap_socket(client_socket, server_hostname=SERVER_HOST) as ssl_socket:
        print(f"SSL connection to {SERVER_HOST}:{SERVER_PORT}.")
        message = "This is a confidential and secure message."
        ssl_socket.sendall(message.encode())
        response = ssl_socket.recv(1024)
        print(f"Received encrypted response: {response.decode()}")
```

Output:-

Server:-

```
Server listening on 127.0.0.1:12345...
SSL connection from ('127.0.0.1', 12345).
Received encrypted data: This is a confidential and secure message.
```

Client:-

```
SSL connection to 127.0.0.1:12345.
Received encrypted response: Hello, client! This is a secure connection.
```

Result:-

Thus the above program is executed successfully

EX NO:6	Experiment Eavesdropping, Dictionary attacks, MITM attacks
DATE:	

Aim: To Experiment Eavesdropping, Dictionary attacks, MITM attacks

Procedure:

- Man-in-the-middle (MITM) attacks, and eavesdropping on Wi-Fi networks. This experiment requires advanced knowledge of Wi-Fi security protocols, packet capture, and ethical hacking practices.
- Eavesdropping To perform eavesdropping, you can use tools like Aircrack-ng or Wireshark. These tools capture wireless packets and help in decoding the information transmitted over the air.
- Dictionary Attacks A dictionary attack is a brute-force technique where the attacker tries to guess the password by systematically trying all the words in the dictionary. This can be achieved using tools like Hydra or Aircrack-ng.

simple procedure to perform a dictionary attack:

- Start by gathering information about the target, such as the available Wi-Fi networks and the encryption type used.
- Create a list of possible passwords or use a pre-defined dictionary.
- Use the attack tool of your choice and specify the target network's ESSID (network name), the attack type (dictionary), and the list of possible passwords.
- Run the attack and wait for the tool to find the correct password.
- Once the correct password is found, you can connect to the target network using the credentials.
- Perform the necessary cleanup tasks, such as removing any malicious configurations or tools from the compromised device.

Man-in-the-Middle (MITM) Attacks A MITM attack occurs when an attacker intercepts and possibly alters the communication between two parties without their knowledge. This can be achieved using tools like ettercap or SSLstrip.

simple procedure to perform a MITM attack:

- Start by gathering information about the target, such as the available Wi-Fi networks and the devices connected to them.
- Set up a rogue access point (RAP) using a tool like Airbase-ng or createpacket.
- Force the target device to connect to the RAP using tools like ettercap or ARP spoofing.

- Once the target device is connected to the RAP, capture the Wi-Fi handshake using

Simple procedure to perform Eavesdropping Attack:-

- Open the Command Prompt by pressing the Windows key and typing "cmd". Right-click on "Command Prompt" and select "Run as administrator".

- Identify the target device's MAC address by typing the following command and pressing Enter:

arp -a

- Find the network interface name of the device you want to perform the attack on by typing the following command and pressing Enter:

- ipconfig
- Set the IP address of the attacker device to be in the same subnet as the target device. To do this, open the "Control Panel" and navigate to "Network and Internet" > "Network Connections". Select the network adapter and choose "Properties". Select "Internet Protocol Version 4 (TCP/IPv4)" and click on "Properties". Change the "IP address" and "Subnet mask" to values that correspond to the target device's subnet. Click "OK" to save the changes.

- Carry out the ARP Spoofing attack by typing the following command and pressing Enter. Replace <attacker interface name>, <target IP>, and <router IP> with the appropriate values:

arp -s <target IP> <router MAC> <attacker interface name>

- Redirect the packets of the target device to the attacker device by typing the following command and pressing Enter:

arp -s <target IP> <attacker MAC> <attacker interface name>

- you can capture and analyze the HTTP packets of the target device.

Result:-

Thus the above experiment is carried out successfully

EX NO:7	Experiment with Sniff Traffic using ARP Poisoning
DATE:	

Aim: To Experiment with Sniff Traffic using ARP Poisoning

Procedure:

- ARP Poisoning is a technique used to perform Man-in-the-Middle (MITM) attacks on networks. In this attack, an attacker sends spoofed ARP packets to a victim's machine, creating an entry in the victim's ARP cache that maps the attacker's MAC address to the IP address of the victim.
- The SniffTraffic.py script in this solution captures network traffic and prints the details of the packets in a readable format. This can be useful in identifying network issues or potential attacks.
- The ARPPoison.py script performs the ARP poisoning attack. This script needs to be executed with administrative privileges on the target network to modify the ARP cache.

Here is an alternative approach to experiment with Sniff Traffic using ARP Poisoning:

Step 1: Set up the attacking machine.

Step 2: Use a tool like Npcap to capture packets. Npcap is a free network packet capture library for Windows that supports packet capture in user-mode and is compatible with the old Microsoft Network Monitor 3 (NetMon) API.

Step 3: Capture ARP packets using Npcap. You can do this by writing a simple script that listens for ARP packets on the network interface and saves them to a file.

Step 4: Parse the captured ARP packets and extract the required information. You can do this by using the dpkt Python library, which is designed to be fast and easy to use.

Step 5: Send spoofed ARP packets to the victim and router using Scapy, a powerful packet manipulation tool.

Step 6: Analyze the captured network traffic to identify any malicious activity.

Step 7: ARP poisoning, will need two devices connected to the same network, with one of them (the attacker) having a spoofed IP address and MAC address.

Step 8: Network configuration. Start by gathering information about the target device (T) and the router (R) connected to the same network. This includes IP addresses, MAC addresses, and network interfaces.

Step 9: Set up ARP spoofing. Once you have the required information, use a tool like Scapy (a Python-based packet manipulation library) to perform ARP spoofing. This involves sending ARP replies with a fake MAC address (corresponding to the attacker's IP address) to the target and router devices.

Result:-

Thus the above experiment is carried out successfully

EX NO:8	onstrate intrusion detection system using any tool.
DATE:	

Aim: To Demonstrate intrusion detection system using any tool.

Procedure:

- Choose an intrusion detection system (IDS) tool, such as Snort or AIDE, for the specific OS environment.
- Configure the IDS tool according to your organization's security policy and system configuration. This includes specifying which events or files should trigger an alert.
- Deploy the IDS tool in your network, ideally in multiple locations to achieve comprehensive coverage. This ensures that all incoming and outgoing network traffic is monitored and analyzed for potential intrusions.
- Monitor the IDS tool's alerts and notifications. These can be viewed through the tool's console, or they can be forwarded to a centralized monitoring system.
- If the IDS tool triggers an alert, it indicates a potential security breach or intrusion attempt. At this point, you should take appropriate steps to investigate and mitigate the issue.
- Periodically review the IDS tool's logs and adjust the configuration settings as needed to maintain the system's effectiveness.
- Windows Subsystem for Linux (WSL):
- Make sure WSL is installed on your Windows system. You can install it from the Microsoft Store. Suricata Installation:
- Open your WSL terminal and install Suricata:

```
sudo apt-get update
sudo apt-get install suricata
```

1. Suricata config:

- Edit the Suricata configuration file to specify the network interface and other settings.

```
sudo nano /etc/suricata/suricata.yaml
```

- Modify the interface parameter to match your network interface (e.g., eth0).
- Save the changes and exit the text editor.

2. Start Suricata:

```
sudo suricata -c /etc/suricata/suricata.yaml -i eth0
```

- Suricata will start monitoring traffic on the specified interface.

Generate Traffic:

- Open a web browser or use tools like curl to generate network traffic.
- Suricata will analyze the traffic and generate alerts for suspicious activities.

View Alerts:

- Suricata logs alerts to the /var/log/suricata/fast.log file. View the alerts using:

```
cat /var/log/suricata/fast.log
```

Result:-

Thus the above experiment is carried out successfully

EX NO:9	Explore network monitoring tools
DATE:	

Aim: To Explore network monitoring tools

Procedure:

1. Wireshark:
   Type: Packet Analyzer
   Features:

- Captures and analyzes network packets in real-time.
- Provides detailed information about protocols, packet headers, and payloads.
- Useful for network troubleshooting and protocol analysis.

2. Nagios:
   Type: Infrastructure Monitoring
   Features:

- Monitors hosts, services, and network devices.
- Supports alerting, notification, and reporting.
- Extensible through plugins for various functionalities.

3. Zabbix:
   Type: Infrastructure Monitoring
   Features:

- Monitors servers, networks, and applications.
- Supports data visualization, alerting, and reporting.
- Provides a web interface for configuration and monitoring

.

4. PRTG Network Monitor:
   Type: Infrastructure Monitoring
   Features:

- Monitors network devices, bandwidth, and applications.
- Supports SNMP, WMI, packet sniffing, and custom sensors.
- Offers customizable dashboards and reports.

5. SolarWinds Network Performance Monitor (NPM):
   Type: Infrastructure Monitoring
   Features:

- Monitors network performance, devices, and interfaces.
- Provides real-time and historical data.
- Offers alerting, reporting, and network mapping.

6. Cacti:
   Type: Infrastructure Monitoring
   Features:

- Utilizes SNMP for monitoring network devices.
- Provides graphing and trending capabilities.
- Supports data collection and historical analysis.

7. Observium:
   Type: Infrastructure Monitoring
   Features:

- Monitors network devices and servers.
- Supports SNMP, LLDP, and syslog.
- Offers automatic discovery and mapping.

8. Snort:
   Type: Intrusion Detection System (IDS)
   Features:

- Analyzes network traffic for malicious activity.
- Detects and alerts on suspicious network behavior.
- Extensible with community rules for various threats.

9. Prometheus:
   Type: Infrastructure Monitoring
   Features:

- Collects and stores time-series data.
- Supports multi-dimensional data querying.
- Integrates with Grafana for visualization.

10. Grafana:
    Type: Visualization and Monitoring
    Features:

- Integrates with various data sources, including Prometheus.
- Provides customizable dashboards and panels.
- Supports alerting and notification.

Result:-

Thus the above experiment is carried out successfully

EX NO:10	Study to configure Firewall, VPN
DATE:	

Aim: To Study to configure Firewall, VPN

Procedure:

Configuring a Firewall:

Step 1: Identify Firewall Requirements

- Determine Security Policies
- Identify the security policies for your network, including which services and traffic should be allowed or denied.

Step 2: Choose a Firewall Solution

- Select a Firewall Solution
- Choose a firewall solution based on your requirements (e.g., hardware firewall, software firewall, cloud-based firewall).

Step 3: Configure Firewall Rules

- Access Firewall Interface:
- Access the firewall administration interface through a web browser or command-line interface.
- Create rules to allow or block specific traffic based on source/destination IP addresses, port numbers, and protocols.
- If needed, configure Network Address Translation (NAT) rules for mapping private to public IP addresses.
- Configure logging and alerts for firewall events to monitor and respond to potential security incidents.

Step 4: Test and Refine

- Test the firewall rules to ensure they are functioning as intended.
- Refine firewall rules based on testing and monitoring activities.

Setting Up a VPN:

Step 1: Choose a VPN Solution

- Choose between site-to-site, remote access, or other VPN types based on your network architecture.

- Select a VPN protocol (e.g., OpenVPN, IPsec, L2TP/IPsec) based on security requirements and compatibility.

Step 2: Configure VPN Server

- Install the VPN server software on the server that will act as the VPN gateway.
Configure Server Settings:

- Set up server settings, including authentication methods, IP addressing, and encryption parameters.

Step 3: Configure VPN Clients

- Install VPN client software on devices that need VPN access.
Configure Client Settings:

- Configure client settings, including connection details, authentication credentials, and encryption parameters.

Step 4: Establish and Test VPN Connection

- Establish VPN connections from client devices.

- Verify that the VPN connection is established successfully.

Step 5: Monitor and Maintain

- Monitor VPN traffic for anomalies and performance.

- Regularly update and patch both firewall and VPN software to address security vulnerabilities.

Security Considerations:

- Implement Strong Authentication:

- Use strong authentication methods for both firewall and VPN access.
Encrypt Sensitive Data:

- Encrypt sensitive data transmitted over VPN connections.
Regularly Audit and Review:

- Conduct regular audits of firewall and VPN configurations to identify and address security risks.

Result:-

Thus the above study to configure firewall and VPN was successfully