V.S.B. ENGINEERING COLLEGE

An Autonomous Institution

KARUDAYAMPALAYAM, KARUR - 639 111.

RECORD NOTE BOOK

Name : _____

Reg.No. : _____

Subject : _____

Semester : _____

V.S.B. ENGINEERING COLLEGE

An Autonomous Institution

KARUDAYAMPALAYAM, KARUR - 639 111.

_____ RECORD

NAME..... CLASS .....

REGISTER No..... BRANCH.....

ROLL No.....

Certified that this is a bonafide record of work done by the above student during the year 20 - 20

Date : Lab In-charge Head of the Department

Submitted for the "UNIVERSITY PRACTICAL EXAMINATION"

held on.....

Internal Examiner. External Examiner.

CONTENTS

V.S.B. ENGINEERING COLLEGE, KARUR

An Autonomous Institution

DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR 2023 – 2024 EVEN

SEMESTER(REGULATION - 2021)

CCS354 – Network Security Laboratory

Student Name : _____

Register Number : _____

Year / Semester : _____

Department : _____

V.S.B. ENGINEERING COLLEGE
An Autonomous Institution
DEPARTMENT OF INFORMATION TECHNOLOGY

ACADEMIC YEAR: 2023-2024 (EVEN Semester)

VISION, MISSION, PEOs, POs and PSOs

Vision of the Institution:

We endeavor to impart futuristic technical education of the highest quality to the student community and to inculcate discipline in them to face the world with self-confidence and thus we prepare them for life as responsible citizens to uphold human values and to be of service at large. We strive to bring of the Institution as an Institution of academic excellence of International standard.

Mission of the Institution:

We transform persons into personalities by the state-of the art infrastructure, time consciousness, quick response and the best academic practices through assessment and advice.

Vision of the Department:

To provide professional computing training to make competent IT engineers and thus prepare them to work in the emerging trends of information technology field.

Mission of the Department:

1. Producing quality IT engineers with good technical knowledge by effective teaching.
2. Making competent engineers to adapt to the dynamic needs of industries.
3. Developing extensive learning skills in studies.
4. Inculcating strong ethical values and professionalism to serve society with responsibility.

Program Educational Objectives (PEOs)

PEO 1: To make graduates to be proficient in utilizing the fundamental knowledge of various streams in engineering and technology.

PEO2: To enrich graduates with the core competencies necessary for applying knowledge of computers and technologies in the context of business enterprise.

PEO 3: To enable graduates think logically and to pursue lifelong learning to understand technical issues related to computing systems and to provide optimal solutions.

PEO 4: To enable graduates develop hardware and software system by understanding the importance of social, business and environmental needs in the social context.

Program Outcomes (POs)

PO1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3. Design/development of solutions: Design solutions for complex engineering problems & design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcome (PSOs)

PSO1: Apply the mathematical and the computing knowledge to identify and provide solutionsfor computing problems.

PSO2: Design and develop computer programs in the areas related to algorithms, networking, webdesign and data analytics of varying complexity.

PSO3: Apply standard Engineering practices and strategies in software project using open sourceenvironment to deliver a quality product for business success.

CCS354	NETWORK SECURITY	L	T	P	C
		2	0	2	3

COURSE OBJECTIVES:

1. To learn the fundamentals of cryptography.
2. To learn the key management techniques and authentication approaches.
3. To explore the network and transport layer security techniques.
4. To understand the application layer security standards.
5. To learn the real time security practices.

LIST OF EXPERIMENTS:

1. Implement symmetric key algorithms
2. Implement asymmetric key algorithms and key exchange algorithms
3. Implement digital signature schemes
4. Installation of Wire shark, tcp dump 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

TOTAL: 30 PERIODS

CO, CO-PO Matrix & PSO Matrix

COURSE OUTCOMES: On Completion of the course, the students are able to:

CO1: Build cryptosystems by applying symmetric and public key encryption algorithms**.**

CO2: Understanding the Construct code for authentication algorithms.

CO3: Develop the ability to understand the security techniques applied to network and transport layer

CO4: Explain the application layer security standards.

CO5: Ability to demonstrate the network security system using open source tools**.**

CO-PO MATRIX

Course Outcomes	Program Outcomes												PSOs		
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
CO1	3	3	2	2	2	-	-	-	2	1	2	1	2	3	1
CO2	1	1	3	2	2	-	-	-	2	2	1	1	3	1	2
CO3	1	2	1	1	2	-	-	-	3	3	1	3	2	1	3
CO4	2	2	3	2	3	-	-	-	3	3	2	1	2	1	3
CO5	2	1	3	2	2	-	-	-	2	1	1	3	2	1	1
C312 (Average)	**2**	**2**	**3**	**2**	**2**	**-**	**-**	**-**	**2**	**2**	**1**	**2**	**2**	**1**	**2**

CO-PSO MATRIX

Course Outcomes	PSOs		
	1	**2**	**3**
CO1	2	3	1
CO2	3	1	2
CO3	2	1	3
CO4	2	1	3
CO5	2	1	1
C312 (Average)	**2**	**1**	**2**

LIST OF EXPERIMENTS

S.No	Name of the Experiments	COs	POs	PSOs
1	Implement symmetric key algorithms	CO1	PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO3
2	Implement asymmetric key algorithms and key exchange algorithms	CO2	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
3	Implement digital signature schemes	CO1	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
4	Installation of Wire shark, tcp dump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram.	CO3	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
5	Check message integrity and confidentiality using SSL	CO3	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
6	Experiment Eavesdropping, Dictionary attacks, MITM attacks	CO1 & CO5	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
7	Experiment with Sniff Traffic using ARP Poisoning	CO5	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
8	Demonstrate intrusion detection system using any tool	CO5	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
9	Explore network monitoring tools	CO5	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3
10	Study to configure Firewall, VPN	CO5	PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12	PSO2 & PSO3

Ex. No: 1 **Implement Symmetric Key Algorithms**
Date:

Aim:

The aim of the program is to demonstrate the implementation of a symmetric key algorithm using Java

Procedure:

1. **Key Generation:**

   The program generates a secret key using the AES algorithm with a key length of 256 bits.

2. **Encryption:**

- A plaintext message is provided ("This is a secret message").
- The program encrypts the plaintext message using the AES algorithm and the generated secret key.
- The encrypted ciphertext is then printed to the console.

3. **Decryption:**

- The encrypted ciphertext obtained from the encryption step is decrypted using the same secret key.
- The decrypted plaintext is printed to the console.

Program:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import java.util.Base64;
public class SymmetricKeyAlgorithm
{
   public static void main(String[] args) throws Exception
{
      // Step 1: Generate a symmetric key
      SecretKey secretKey = generateSecretKey();
      // Step 2: Encrypt plaintext using the secret key
      String plaintext = "This is a secret message";
      String ciphertext = encrypt(plaintext, secretKey);
      System.out.println("Encrypted text: " + ciphertext);
      // Step 3: Decrypt the ciphertext using the same secret key
      String decryptedText = decrypt(ciphertext, secretKey);
      System.out.println("Decrypted text: " + decryptedText);
   }
   // Method to generate a secret key
   public static SecretKey generateSecretKey() throws Exception {
      KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
      keyGenerator.init(256); // AES key length is 256 bits
      return keyGenerator.generateKey();
   }
   // Method to encrypt plaintext using AES and the secret key
   public static String encrypt(String plaintext, SecretKey secretKey) throws Exception {
      Cipher cipher = Cipher.getInstance("AES");
      cipher.init(Cipher.ENCRYPT_MODE, secretKey);
      byte[] encryptedBytes = cipher.doFinal(plaintext.getBytes());
      return Base64.getEncoder().encodeToString(encryptedBytes);
   }
```

```
    // Method to decrypt ciphertext using AES and the secret key
    public static String decrypt(String ciphertext, SecretKey secretKey) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        byte[] encryptedBytes = Base64.getDecoder().decode(ciphertext);
        byte[] decryptedBytes = cipher.doFinal(encryptedBytes);
        return new String(decryptedBytes);
    }
}
```

Output:

Encrypted text: 8pAR5zHpaTcN6TQ1qVzSU+RRWe8yWLWl8VvJm4w8uuI=

Decrypted text: This is a secret message

Particulars	**Marks allocated**	**Marks Obtained**
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

The program demonstrates the successful encryption and decryption of the message using symmetric key cryptography, specifically the AES algorithm.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO10 PO11 & PO12.

Ex.No: 2 **Implement asymmetric key algorithms and key**

Date: **exchange algorithms**

Aim:

To demonstrate the implementation of asymmetric key algorithms using RSA for encryption and decryption, as well as the implementation of a key exchange algorithm using Diffie-Hellman for secure key establishment between RSA algorithm for asymmetric encryption.

Procedure 1 : Asymmetric Key Algorithm

1. Generate Key Pair: Use KeyPairGenerator to generate a public-private key pair.

2. Encryption: Use the recipient's public key to encrypt the message.

3. Decryption: Use the recipient's private key to decrypt the encrypted message.

4. Digital Signatures (Optional): Use the sender's private key to sign the message and the recipient's public key to verify the signature.

5. Secure Key Exchange (Optional): Use asymmetric key exchange algorithms like Diffie-Hellman to establish a shared secret key between two parties.

6. Secure Storage and Management: Ensure private keys are securely stored and managed.

Procedure : Key Exchange Algorithm

1. Alice and Bob both generate their own DH key pairs.
2. Alice sends her public key to Bob, and Bob sends his public key to Alice.
3. Alice and Bob use each other's public keys and their own private keys to generate shared secrets.
4. They then validate that their shared secrets match, indicating a successful key exchange.

Program 1: Asymmetric Key Algorithm

```
import java.security.*;
import javax.crypto.Cipher;
public class RSAExample
{
   public static void main(String[] args) throws Exception
{
      // Generating public and private keys
      KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
      keyPairGenerator.initialize(2048);
      KeyPair keyPair = keyPairGenerator.generateKeyPair();
      PublicKey publicKey = keyPair.getPublic();
      PrivateKey privateKey = keyPair.getPrivate();
      // Message to be encrypted
      String message = "Hello, this is a secret message.";
      // Encrypting the message using public key
      Cipher encryptCipher = Cipher.getInstance("RSA");
      encryptCipher.init(Cipher.ENCRYPT_MODE, publicKey);
      byte[] encryptedMessage = encryptCipher.doFinal(message.getBytes());
      // Decrypting the message using private key
      Cipher decryptCipher = Cipher.getInstance("RSA");
      decryptCipher.init(Cipher.DECRYPT_MODE, privateKey);
      byte[] decryptedMessage = decryptCipher.doFinal(encryptedMessage);
      // Printing results
      System.out.println("Original message: " + message);
      System.out.println("Encrypted message: " + new String(encryptedMessage));
      System.out.println("Decrypted message: " + new String(decryptedMessage));
   }
}
```

Sample output:

```
Original message: Hello, this is a secret message.
Encrypted message: 6??i      _?
Decrypted message: Hello, this is a secret message.
```

Program 2 : Key Exchange Algorithm

```
import javax.crypto.KeyAgreement;
import javax.crypto.SecretKey;
import javax.crypto.spec.DHParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.PrivateKey;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
public class KeyExchangeExample
{
   public static void main(String[] args) throws Exception
 {
      // Alice generates her key pair
      KeyPairGenerator aliceKpg = KeyPairGenerator.getInstance("DH");
      aliceKpg.initialize(2048);
      KeyPair aliceKp = aliceKpg.generateKeyPair();
      // Alice sends her public key to Bob
      byte[] alicePublicKeyBytes = aliceKp.getPublic().getEncoded();
      String alicePublicKeyBase64 = Base64.getEncoder().encodeToString(alicePublicKeyBytes);
      // Bob receives Alice's public key and generates his key pair
      byte[] receivedAlicePublicKeyBytes = Base64.getDecoder().decode(alicePublicKeyBase64);
      KeyFactory bobKeyFactory = KeyFactory.getInstance("DH");
      X509EncodedKeySpec x509KeySpec = new
X509EncodedKeySpec(receivedAlicePublicKeyBytes);
      PublicKey alicePublicKey = bobKeyFactory.generatePublic(x509KeySpec);
      DHParameterSpec dhParams = ((javax.crypto.interfaces.DHPublicKey)
alicePublicKey).getParams();
      KeyPairGenerator bobKpg = KeyPairGenerator.getInstance("DH");
      bobKpg.initialize(dhParams);
      KeyPair bobKp = bobKpg.generateKeyPair();
```

```
        // Bob sends his public key to Alice
        byte[] bobPublicKeyBytes = bobKp.getPublic().getEncoded();
        String bobPublicKeyBase64 = Base64.getEncoder().encodeToString(bobPublicKeyBytes);
        // Alice receives Bob's public key
        byte[] receivedBobPublicKeyBytes = Base64.getDecoder().decode(bobPublicKeyBase64);
        X509EncodedKeySpec x509KeySpecBob = new
X509EncodedKeySpec(receivedBobPublicKeyBytes);
        PublicKey bobPublicKey = bobKeyFactory.generatePublic(x509KeySpecBob);
        // Alice generates secret key
        KeyAgreement aliceKeyAgreement = KeyAgreement.getInstance("DH");
        aliceKeyAgreement.init(aliceKp.getPrivate());
        aliceKeyAgreement.doPhase(bobPublicKey, true);
           byte[] aliceSharedSecret = aliceKeyAgreement.generateSecret();
        // Bob generates secret key
        KeyAgreement bobKeyAgreement = KeyAgreement.getInstance("DH");
        bobKeyAgreement.init(bobKp.getPrivate());
        bobKeyAgreement.doPhase(alicePublicKey, true);
        byte[] bobSharedSecret = bobKeyAgreement.generateSecret();
        // Validate that the shared secrets match
        if (java.util.Arrays.equals(aliceSharedSecret, bobSharedSecret)) {
           System.out.println("Shared secrets match. Key exchange successful!");
           System.out.println("Shared Secret (Alice): " +
Base64.getEncoder().encodeToString(aliceSharedSecret));
           System.out.println("Shared Secret (Bob): " +
Base64.getEncoder().encodeToString(bobSharedSecret));
        } else {
           System.out.println("Shared secrets do not match. Key exchange failed!");
        }
     }
}
```

Sample output:

Shared secrets match: Key exchange successful!

Shared Secret (Alice): <Alice's Shared Secret>

Shared Secret (Bob): <Bob's Shared Secret>

Particulars	Marks allocated	Marks Obtained
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

The encrypted message, decrypted message data using RSA, and the shared secret key using Diffie-Hellman Key Exchange.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12.

Ex.No: 3 **Implement Digital Signature**

Date:

Aim:

The aim of this program is to demonstrate the generation, signing, and verification of digital signatures using the RSA algorithm.

Procedure:

1. Key Pair Generation:

Generate a pair of RSA public and private keys with a key size of 2048 bits.

2. Message Signing:

Create a sample message to be signed. Use the private key to generate a digital signature for the message.

3.Verification:

Use the public key to verify the digital signature against the original message.

Program

```
import java.security.*;

import java.security.spec.PKCS8EncodedKeySpec;

import java.security.spec.X509EncodedKeySpec;

import java.util.Base64;

public class DigitalSignatureExample {

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

        // Generate public and private keys

        KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");

        keyPairGenerator.initialize(2048);

        KeyPair keyPair = keyPairGenerator.generateKeyPair();

        PublicKey publicKey = keyPair.getPublic();

        PrivateKey privateKey = keyPair.getPrivate();

        // Sample message

        String message = "This is a secret message to be signed.";

        // Generate digital signature

        byte[] signature = sign(message, privateKey);

        // Verify the digital signature

        boolean isVerified = verify(message, signature, publicKey);

        // Print results
```

```
        System.out.println("Original Message: " + message);
        System.out.println("Digital Signature: " + Base64.getEncoder().encodeToString(signature));
        System.out.println("Is Verified: " + isVerified);
    }
    // Method to generate digital signature
    public static byte[] sign(String message, PrivateKey privateKey) throws Exception {
        Signature signature = Signature.getInstance("SHA256withRSA");
        signature.initSign(privateKey);
        signature.update(message.getBytes());
        return signature.sign();
    }
    // Method to verify digital signature
    public static boolean verify(String message, byte[] signature, PublicKey publicKey) throws
Exception {
        Signature verifier = Signature.getInstance("SHA256withRSA");
        verifier.initVerify(publicKey);
        verifier.update(message.getBytes());
        return verifier.verify(signature);
    }
}
```

Sample Output:

Original Message: This is a secret message to be signed.

Digital Signature:

IAAg8vmgejsOENQyfGWkBA5GHliChZ4uBdOM+iBvJmwVGOPnp+4Q9orwXlRbnO5Dv7CYa1P
t6OGEcb94LELb0T3rmisYkXUYRQLoQFiRGUMKaqeu1zDSI7vNAFO4rKtBfIm5MZFeBAb1uV
9xQ7PnCgdbgcgf6MBIqgElsXTYHqHqEV8qUMOssmcNxOgf+AmV4WuH9Pt9Dml6K3fIEQBKE
X82hGn6CKvc/rgMZkWN3NzyGMBfcK4GkPyk1V7Cbs4LAsTrgQgHZztOUDXWqITIyF4XKBa0
23Ve1V4apnI99wS7/Dr7VOWOc7RfawfM/Rpt9DBi0taMq40wmbnNEhr8oQ==

Is Verified: true

Particulars	Marks allocated	Marks Obtained
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

The program should output the original message, the generated digital signature, and whether the verification process succeeded or failed.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO8, PO9, PO110, PO11 & PO12.

Ex.No : 4 **Installation Of Wire Shark,Tcp Dump And Observe Data Transferred in**

Date: **Client Server Communication Using UDP / TCP**

and identity the UDP / TCP Datagram

Aim:

To observe data transfer in client-server communication using UDP and TCP protocols, you can use Wireshark and tcpdump. Here's a step-by-step guide to install Wireshark and tcpdump and then capture and analyze data transfer between a client and server:

Procedure:

1. Installing Wireshark: Windows

Go to the Wireshark official website and download the installer for Windows.

Run the installer and follow the installation instructions.

Linux:

Use your distribution's package manager to install Wireshark.

For example, on Ubuntu, you can use:

sudo apt-get install wireshark

After installation, you may need to add your user to the wireshark group to capture packets without root privileges:

sudousermod -aGwireshark $USER

Log out and log back in for the group change to take effect.

macOS:

Download the Wireshark installer for macOS from the official website.

Open the downloaded .dmg file and drag the Wireshark application to the Applications folder.

2. Installing tcpdump: Linux

tcpdump is usually pre-installed on most Linux distributions. If not, you can install it using your package manager. For example, on Ubuntu:

sudo apt-get install tcpdump

macOS:

tcpdump is also pre-installed on macOS. You can use it from the Terminal.

3. Capturing and Analyzing Traffic:

Start Wireshark:

Open Wireshark from the applications menu.

Select Interface:

Select the network interface you want to capture packets from (e.g., Ethernet, Wi-Fi).

Start Capture:

Click on the "Start" button to begin capturing packets.

4. Initiate Client-Server Communication:

Start the client-server communication using UDP or TCP protocols.

Stop Capture:

After you have captured enough packets, click on the "Stop" button in Wireshark.

Analyze Packets:

You can now analyze the captured packets in Wireshark. You can filter packets by protocol (e.g., UDP or TCP) and inspect the contents of individual packets to identify UDP/TCP datagrams.

5. Identifying UDP/TCP Datagrams:

- In Wireshark, UDP and TCP packets can be easily identified by their protocol type indicated in the packet list.
- To apply display filters in Wireshark to filter only UDP or TCP packets.
- Once you have filtered the packets, you can select a packet and inspect its details in the packet view pane to identify the UDP/TCP datagram, including source and destination ports, payload data, and other relevant information.

Particulars	Marks allocated	Marks Obtained
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

The Wireshark and tcpdump are installed, and data transferred between client and server using UDP and TCP can be observed and analysed.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO7,PO8, PO9, PO10, PO11 & PO12.

Ex.No:5 **Check message integrity and confidentiality using SSL**

Date:

Aim:

To demonstrate the implementation of a client-server communication using SSL (Secure Sockets Layer) in order to achieve message integrity and confidentiality.

Procedure:

1. Server Setup:

- Set up a server that listens for incoming connections from clients over a secure SSL connection.

Procedure:

- Load the server's keystore containing its SSL certificate and private key.
- Create an SSL context using the loaded keystore and initialize it.
- Create an SSL server socket and start listening for incoming connections.
- Accept incoming connections from clients.
- Read the message sent by the client over the SSL connection.

Implementation:

- SSL Server class encapsulates the server setup process.

2. Client Setup:

Set up a client that establishes a connection to the server over a secure SSL connection.

Procedure:

- Load the client's keystore containing its SSL certificate and private key.
- Create an SSL context using the loaded keystore and initialize it.
- Create an SSL socket and connect it to the server.
- Send a message to the server over the SSL connection.

Implementation:

- SSLClient class encapsulates the client setup process.

3. SSL Configuration:

- Configure SSL settings for both the server and the client to ensure secure communication.

Procedure:

- Load keystore files containing SSL certificates and private keys for both the server and the client.
- Initialize SSL contexts with the loaded keystore information.

Implementation:

- Keystore files (server_keystore.jks and client_keystore.jks) are loaded and used to initialize SSL contexts.

4. Message Exchange:

- Exchange messages securely between the client and server over the SSL connection.

Procedure:

- The client sends a message to the server over the SSL connection.
- The server receives the message from the client over the SSL connection and processes it.

Implementation:

- The client sends a message using a BufferedWriter and the server reads the message using a BufferedReader.

5. Message Integrity and Confidentiality:

Ensure that messages exchanged between the client and server are encrypted and secure from tampering.

Procedure:

- The SSL protocol handles encryption and decryption of data transmitted between the client and server, ensuring message confidentiality.
- Digital certificates and SSL handshakes are used to verify the identity of the server and establish a secure connection, ensuring message integrity.

Implementation:

- SSL encryption and decryption are automatically handled by the SSL protocol, and digital certificates ensure server authenticity and message integrity.

Sample Output:

The server will print "Received message from client: Hello from client!" upon receiving the message from the client.

The client will print "Message sent to server: Hello from client!" after sending the message to the server.

Particulars	Marks allocated	Marks Obtained
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

This demonstrates how to establish an SSL connection between a client and a server, exchange messages securely, ensuring message integrity and confidentiality.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO8, PO9, PO10, PO11 & PO12.

Ex.No:6 **Experiment eavesdropping, dictionary attacks, MITM attacks**

Date:

Aim:

To simulate and understand the concepts of eavesdropping, dictionary attacks, and Man-in-the-Middle (MITM) attacks in a controlled and educational environment. These simulations are intended for learning purposes only and should not be used for any unethical or illegal activities.

Procedure:

1. Eavesdropping Simulation:

Simulate the act of eavesdropping, where an unauthorized party intercepts and listens to communication between two parties.

Procedure:

- Generate a message (or simulate capturing a message) between two parties.
- Output the intercepted message to simulate eavesdropping.

Implementation:

The Eavesdropping Simulation class generates a message and outputs it as if it were intercepted by an eavesdropper.

2. Dictionary Attack Simulation:

Simulate a dictionary attack, where an attacker tries to guess a password by testing against a list of common passwords (dictionary).

Procedure:

- Prompt the user to input a password.
- Check if the entered password matches any entry in the simulated dictionary.
- Output whether the password is found in the dictionary or not.

Implementation:

The Dictionary Attack Simulation class prompts the user to input a password and checks if it matches any entry in the simulated dictionary.

3. Man-in-the-Middle (MITM) Attack Simulation:

Simulate a Man-in-the-Middle (MITM) attack, where an attacker intercepts and possibly alters communication between two parties without their knowledge.

Procedure:

- Simulate a client sending a message to a server.
- Intercept the message and forward it to the server.
- Simulate the server's response and forward it back to the client.

Implementation:

The MITM Attack Simulation class simulates a client sending a message to a server, and a MITM attacker intercepts the message, forwards it to the server, receives the server's response, and forwards it back to the client.

Particulars	**Marks allocated**	**Marks Obtained**
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

Gain insights into the vulnerabilities of communication networks to eavesdropping, dictionary attacks, and MITM attacks.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO7,PO8, PO9, PO10, PO11 & PO12.

Ex.No: 7 **Experiment With Sniff Traffic Using ARP Poisoning**

Date:

Aim:

The aim of an experiment with sniffing traffic using ARP poisoning is to demonstrate how an attacker can intercept network traffic between two communicating parties by manipulating the Address Resolution Protocol (ARP) cache of network devices.

Procedure:

1. Setup a Local Network: Set up a local network environment with at least two devices, such as computers or virtual machines, connected to the same network.

2. Identify Target and Attacker: Determine which device will act as the attacker and which device's traffic will be intercepted (the target).

3. Enable IP Forwarding (Optional): If necessary, enable IP forwarding on the attacker's machine to allow it to forward intercepted traffic to its intended destination. This step may vary depending on the operating system.

4. Observe ARP Cache: Use appropriate commands (e.g., arp -a on Windows or arp -n on Linux) to observe the ARP cache of devices in the network. Note the MAC addresses associated with IP addresses.

5. Perform ARP Poisoning: Using tools like arpspoof or ettercap, the attacker sends spoofed ARP packets to the target device and the gateway, tricking them into associating the attacker's MAC address with the IP address of the other party. This effectively redirects traffic intended for the other party through the attacker's machine.

6. Start Sniffing Traffic: Use a packet sniffing tool such as Wireshark on the attacker's machine to capture and analyze the intercepted network traffic. Configure Wireshark to listen on the network interface through which traffic is being routed (usually the interface connected to the local network).

7. Analyze Intercepted Traffic: Analyze the captured packets to observe sensitive information, such as plaintext passwords, HTTP requests, or other confidential data, exchanged between the target and other network devices.

8. Cleanup: Once the experiment is complete, stop the ARP poisoning attack and restore the ARP caches of the target and gateway devices to their original state. This can usually be done by sending correct ARP packets or allowing the ARP caches to expire naturally.

To sniff network traffic on Windows

Use the netsh command-line tool along with the trace subcommand.

This allows you to capture network packets on a specified network interface.

Here's how you can use netsh to start capturing traffic:

netsh trace start capture=yes tracefile=C:\path\to\capture.etl maxsize=1024

This command starts capturing network traffic on all interfaces and saves the captured packets to a file named capture.etl in the specified directory (C:\path\to\).

The maxsize parameter specifies the maximum size of the capture file in megabytes.

To stop capturing traffic, you can use the following command:

netsh trace stop

This command stops the capture and saves the captured packets to the output file.

Particulars	Marks allocated	Marks Obtained
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

Captured network traffic showing ARP poisoning and its effects on communication betweenthe victim and the router.

Ex.No:8 **Demonstrate intrusion detection system using any tool**

Date:

1. Setting up Snort:

a. Install Snort:

You can install Snort on various operating systems like Linux, Windows, or macOS. The installation process may vary based on your OS. You can find installation instructions on the official Snort website or various online tutorials.

b. Configure Snort:

After installation, you need to configure Snort. The configuration file for Snort is usually located at /etc/snort/snort.conf (on Linux). You can modify this file to customize Snort's behavior, including network interfaces to monitor, rulesets to use, and logging settings.

c. Download Rulesets:

Snort uses rules to detect suspicious network traffic. You can download pre-defined rulesets from the Snort website or other sources. Make sure to keep your rulesets updated for the latest threats.

2. Start Snort:

Once configured, you can start Snort by running the command sudo snort -c /etc/snort/snort.conf -i<interface> (replace <interface> with the network interface you want Snort to monitor).

3. Using Snort:

Monitoring Network Traffic:

Snort will now monitor the network traffic on the specified interface according to the configured rulesets. It will log any suspicious activity it detects based on the rules.

Analyzing Alerts:

Snort will generate alerts for any suspicious activity detected. These alerts can be viewed in real-time in the terminal where Snort is running or logged to a file specified in the configuration.

Responding to Alerts:

Based on the severity of the alerts, you can take appropriate action. This may include blocking the source IP address, investigating the incident further, or updating your security measures to prevent similar attacks in the future.

Sample Output:

Here's an example of what Snort alerts might look like in the terminal:

12/01-12:34:56.789012 [**] [1:12345:6] ATTACK-RESPONSES Reverse TCP connection [**] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.1.10:1234 -> 203.0.113.5:80

This alert indicates that Snort has detected a reverse TCP connection attempt from IP address 192.168.1.10 to IP address 203.0.113.5 on port 80, matching a rule with ID 12345. The severity of this alert is classified as "Potentially Bad Traffic" with a priority of 2.

Particulars	**Marks allocated**	**Marks Obtained**
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

The demonstration of the intrusion detection system (IDS) using [insert tool name] provided valuable insights into the importance of proactive network security measures. Throughout the demonstration, we simulated various attack scenarios and monitored the system's response to detect and mitigate potential threats.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO7,PO8, PO9, PO10, PO11 & PO12

Ex.No:9 **Explore network monitoring tools**
Date:

Aim:

The aim of the provided Java code is to demonstrate how to capture network packets from a specified network interface using the pcap4j library in Java. This serves as a basic foundation for building network monitoring and analysis tools.

Procedure:

1. Setting Network Interface

2. Opening Network Interface

3. Packet Capture

4. Start Capturing Packets

5. Closing the Handle

Popular network monitoring tools across different categories:

1. Open-Source Network Monitoring Tools:

Nagios Core

Nagios is a widely used open-source monitoring system that allows you to monitor servers, switches, applications, and services.

It supports alerting, notification, and reporting features.

Zabbix

Zabbix is an enterprise-class open-source monitoring solution that offers real-time monitoring, alerting, and visualization of network and application performance.

It supports auto-discovery of devices, flexible alerting, and distributed monitoring.

Prometheus

Prometheus is an open-source monitoring and alerting toolkit designed for reliability and scalability.

It collects metrics from configured targets, stores them, and allows querying and alerting based on these metrics.

2. Paid Network Monitoring Tools:

SolarWinds Network Performance Monitor (NPM)

SolarWinds NPM is a comprehensive network monitoring solution that provides deep visibility into network performance, traffic, and devices.

It offers features like network mapping, performance analysis, and customizable alerts.

PRTG Network Monitor

PRTG Network Monitor is an all-in-one network monitoring solution that provides real-time visibility into your entire network infrastructure.

It supports SNMP, WMI, NetFlow, and other protocols for monitoring various devices and services.

ManageEngine OpManager

OpManager is a network monitoring software that offers network performance monitoring, traffic analysis, and fault management.

It provides out-of-the-box support for monitoring routers, switches, firewalls, servers, and more.

3. Packet Capture and Analysis Tools:

Wireshark

Wireshark is a popular packet analyzer that allows you to capture and interactively browse network traffic.

It supports hundreds of protocols and provides detailed packet-level analysis.

tcpdump

tcpdump is a command-line packet analyzer available on Unix-like operating systems.

It captures packets and displays them in real-time or saves them to a file for later analysis.

4. Security Information and Event Management (SIEM) Tools:

Splunk

Splunk is a SIEM solution that collects and analyzes data from various sources, including network devices, servers, and applications.

It provides real-time monitoring, alerting, and visualization of security events.

Elastic Security

Elastic Security (formerly known as Elasticsearch, Logstash, and Kibana - ELK Stack) is a SIEM platform that enables security analytics and threat hunting.It integrates with various data sources to collect, analyze, and visualize security-related data.

Particulars	Marks allocated	Marks Obtained
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

Graphs, charts, and reports generated by the network monitoring tool showing network performance metrics and status.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO7,PO8, PO9, PO10, PO11 & PO12.

Ex. No: 10 **Study to configure firewall, VPN**

Date:

Aim:

To Configuring the Windows Firewall is essential for securing your system against unauthorized access and malicious activities. To demonstrate how to configure a VPN using the built-in client in Windows 10.

Step 1: Open Windows Firewall Settings

Press Win + S on your keyboard to open the Windows search bar.
Type "Windows Defender Firewall" and press Enter.
Click on "Windows Defender Firewall with Advanced Security".

Step 2: Configure Inbound Rules

In the left pane, click on "Inbound Rules".
Click on "New Rule..." in the right pane.

Example: Allow Inbound Traffic for a Specific Program

Select "Program" and click Next.
Select "This program path:" and browse for the executable file of the program you want to allow.
Click Next.
Choose "Allow the connection" and click Next.
Select the network locations where you want to apply this rule (usually leave all checked) and click Next.
Name your rule (e.g., "Allow MyApp Inbound") and add a description if needed.
Click Finish.

Step 3: Configure Outbound Rules

In the left pane, click on "Outbound Rules".
Click on "New Rule..." in the right pane.

Example: Block Outbound Traffic for a Specific Program

Select "Program" and click Next.

Select "This program path:" and browse for the executable file of the program you want to block.

Click Next.

Choose "Block the connection" and click Next.

Select the network locations where you want to apply this rule (usually leave all checked) and click Next.

Name your rule (e.g., "Block MyApp Outbound") and add a description if needed.

Click Finish.

Step 4: Verify and Test Rules

After creating rules, ensure they appear in the list of inbound and outbound rules respectively.

Test the rules by running the associated programs and verifying whether they are allowed or blocked from accessing the network.

Step 5: Monitoring and Maintenance

Regularly review your firewall rules to ensure they align with your security policies. Monitor firewall logs for any suspicious activities.

Study to configure VPN in windows

Open Settings:

Press Win + I to open the Settings app.

Navigate to Network & Internet:

Click on "Network & Internet" from the Settings menu.

VPN Configuration:

In the Network & Internet settings, select the "VPN" tab from the left-hand menu.

Add a VPN Connection:

Click on "Add a VPN connection" to start configuring your VPN connection.

Fill in VPN Connection Details:

VPN Provider: Choose the appropriate VPN provider (e.g., Windows built-in, third-party VPN service).

Connection Name: Enter a name for your VPN connection.

Server Name or Address: Enter the server address provided by your VPN service.

VPN Type: Select the VPN type (e.g., IKEv2, L2TP/IPsec, SSTP, PPTP).

Type of Sign-in Info: Choose the type of sign-in information required (e.g., Username and password, Certificate, etc.).

Username and Password: Enter your VPN username and password if required.

Advanced Settings (Optional):

Click on "Advanced settings" if you need to configure additional options like proxy settings, DNS server, etc.

Save the Configuration:

Once you've entered all the necessary details, click on "Save" to save the VPN configuration.

Connect to the VPN:

After saving the configuration, you'll see your VPN connection listed under the VPN section in Settings. To connect to the VPN, click on it and then click on "Connect".

Authenticate:

If you configured your VPN connection to require authentication, you'll be prompted to enter your username and password. Enter them and click "OK" or "Connect" to establish the VPN connection.

Verify Connection:

Once connected, you should see a VPN icon in the system tray indicating that you are connected to the VPN. You can also verify your connection by visiting a website like https://www.whatismyip.com/ to confirm that your IP address has changed to the VPN server's IP address.

Particulars	**Marks allocated**	**Marks Obtained**
Performance	50	
Viva-voce	10	
Record	15	
Total	75	

Result:

Firewall rules and VPN configurations applied to the network environment.

This experiment is mapped with PO1, PO2, PO3, PO4, PO5, PO6, PO7,PO8, PO9, PO10, PO11 & PO12.