Ex. No : 1 (a) **CAESAR CIPHER**

Date :

AIM

To perform encryption and decryption using Caesar Cipher.

ALGORITHM

1. Encryption:

   C = E(k, p) = (p + k) mod 26

   C - Cipher Text, p – Plain Text, k – key and E – Encryption Function
2. Decryption:

   p = D(k, C) = (C - k) mod 26

   D – Decryption Function
3. For ease of encryption and decryption:
   - i) Input string is converted into uppercase and then to character array.
   - ii) Each character is converted into its appropriate ASCII character. So A = 65, B = 66 and Z = 90.
   - iii) Before Encryption/Decryption operation, input ASCII character is subtracted by 65 to make A = 0, B = 1 and Z = 25. (Since key space = {0,1,2,..,25})
   - iv) After Encryption/Decryption operation, output ASCII character are added by 65 to compensate for earlier subtraction. (Since ASCII for Upper case alphabets are from 65-90)

/* Ex.No.1(a) CeaserCipher */

```
import java.util.Scanner;

class CeaserCipher {
	public String encryption(String plainText, int key) {
		String cipherText = "";
		char[] plainTextArray = plainText.toUpperCase().toCharArray();
		char[] cipherTextArray = new char[plainTextArray.length];
		for (int i = 0; i < plainTextArray.length; i++) {
			/*
			 * cipherTextArray[i] = (char) (((int) plainTextArray[i]
+ key - 65) % 26 + 65);
			 */
			cipherTextArray[i] = (char) (((int)
Math.floorMod(plainTextArray[i] + key - 65, 26)) + 65);
		}
		cipherText = String.valueOf(cipherTextArray);
		return cipherText;
	}

	public String decryption(String cipherText, int key) {
		String plainText = "";
		char[] cipherTextArray = cipherText.toCharArray();
		char[] plainTextArray = new char[cipherTextArray.length];
		for (int i = 0; i < cipherTextArray.length; i++) {
			/*
			 * plainTextArray[i] = (char) (((int) cipherTextArray[i]
- key - 65) % 26 + 65);
			 * Negative value produces wrong value
			 */
			plainTextArray[i] = (char) (((int)
Math.floorMod(cipherTextArray[i] - key - 65, 26)) + 65);
		}
		plainText = String.valueOf(plainTextArray).toLowerCase();
		return plainText;
	}
```

```
}

public class CeaserCipherDemo {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		CeaserCipher ceaserCipher = new CeaserCipher();
		Scanner scanner = new Scanner(System.in);
		String plainText = scanner.nextLine();
		int key = scanner.nextInt();
		String cipherText = ceaserCipher.encryption(plainText, key);
		System.out.println("CipherText=" + cipherText);
		String recoveredPlainText = 
ceaserCipher.decryption(cipherText, key);
		System.out.println("PlainText=" + recoveredPlainText);
		scanner.close();
	}

}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> CeaserCipherDemo (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 16, 2020, 10:33:19 PM – 10:33:28 PM)
vanakkam
3
CipherText=YDQDNNDP
PlainText=vanakkam
```

RESULT

The Java program to perform encryption and decryption using Caesar Cipher was successfully implemented.

Ex. No : 1 (b) **HILL CIPHER**

Date :

AIM

To perform encryption using Hill Cipher.

ALGORITHM

1. Encryption:

   $c1 = (k11p1 + k21p2 + k31p3) \bmod 26$

   $c2 = (k12p1 + k22p2 + k32p3) \bmod 26$

   $c3 = (k13p1 + k23p2 + k33p3) \bmod 26$

   c1, c2 and c3 are cipher text matrix elements

   k11, k21, k31, k12, k22, k32, k13, k23 and k33 are key matrix elements

   p1, p2, p3 are plain text matrix elements

2. For ease of encryption and decryption:
   i) Input string is converted into uppercase and then to character array.
   ii) Each character is converted into its appropriate ASCII character. So A = 65, B = 66 and Z = 90.
   iii) Before Encryption/Decryption operation, input ASCII character is subtracted by 65 to make A = 0, B = 1 and Z = 25. (Since key space = {0,1,2,..,25})
   iv) After Encryption/Decryption operation, output ASCII character are added by 65 to compensate for earlier subtraction. (Since ASCII for Upper case alphabets are from 65-90)

/*Ex.No.1(b) Hill Cipher*/

```java
import java.util.Scanner;

public class HillCipher {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter the plaintext message:");
		String plainText = sc.nextLine();
		int[][] key = new int[3][3];
		System.out.println("Enter the Key in 3 X 3 Matrix Format:");
		for (int i = 0; i < 3; i++) {
			for (int j = 0; j < 3; j++) {
				key[i][j] = sc.nextInt();
			}
		}
		String cipherText = encrypt(plainText.toUpperCase(), key);
		System.out.println("Cipher Text=" + cipherText);
	}

	public static String encrypt(String plainText, int key[][]) {
		char[] text = plainText.toCharArray();
		int c1, c2, c3, p1, p2, p3;
		p1 = (int) text[0] - 65;
		p2 = (int) text[1] - 65;
		p3 = (int) text[2] - 65;
		c1 = (key[0][0] * p1 + key[1][0] * p2 + key[2][0] * p3) % 26;
		c2 = (key[0][1] * p1 + key[1][1] * p2 + key[2][1] * p3) % 26;
		c3 = (key[0][2] * p1 + key[1][2] * p2 + key[2][2] * p3) % 26;
		char[] cipherText = new char[3];
		cipherText[0] = (char) (c1 + 65);
		cipherText[1] = (char) (c2 + 65);
		cipherText[2] = (char) (c3 + 65);
		return String.valueOf(cipherText);
	}
}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> HillCipher [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (Nov 17, 2020, 9:51:29 AM – 9:52:08 AM)
Enter the plaintext message:pay
Enter the Key in 3 X 3 Matrix Format:
17 17 5
21 18 21
2 2 19
Cipher Text=RRL
```

RESULT

The Hill Cipher encryption was successfully implemented.

Ex. No : 1 (c) **VIGENERE CIPHER**

Date :

AIM

To perform encryption and decryption using Vigenere Cipher.

ALGORITHM

1. Encryption

   $C_i = (p_i + k_{i \bmod m}) \bmod 26$

2. Decryption

   $p_i = (C_i - k_{i \bmod m}) \bmod 26$

3. For ease of encryption and decryption:

   i) Input string is converted into uppercase and then to character array.

   ii) Each character is converted into its appropriate ASCII character. So A = 65, B = 66 and Z = 90.

   iii) Before Encryption/Decryption operation, input ASCII character is subtracted by 65 to make A = 0, B = 1 and Z = 25. (Since key space = {0,1,2,..,25})

   iv) After Encryption/Decryption operation, output ASCII character are added by 65 to compensate for earlier subtraction. (Since ASCII for Upper case alphabets are from 65-90)

/*Ex.No.1(c) Vigenere Cipher*/

```
package com.securitylab.classical;

import java.util.Arrays;
import java.util.Scanner;

public class VigenereCipher {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the plaintext:");
		String plainText = sc.nextLine();
		System.out.println("Enter the key:");
		String key = sc.nextLine();
		String cipherText = encrypt(plainText, key);
		System.out.println("Cipher Text=" + cipherText);
		System.out.println("Recovered Plain Text=" +
decrypt(cipherText, key));
		sc.close();
	}

	public static String encrypt(String plainText, String key) {
		char[] plainTextChar = plainText.toUpperCase().toCharArray();
		char[] keyChar = padKey(key, plainTextChar.length);
		char[] cipherTextChar = new char[keyChar.length];
		for (int i = 0; i < cipherTextChar.length; i++) {
			cipherTextChar[i] = (char) (int) ((plainTextChar[i] +
keyChar[i] - 130) % 26 + 65);
		}
		return (String.valueOf(cipherTextChar));
	}

	public static String decrypt(String cipherText, String key) {
		char[] recoveredPlainTextChar = cipherText.toCharArray();
		char[] keyChar = padKey(key, recoveredPlainTextChar.length);
		for (int i = 0; i < recoveredPlainTextChar.length; i++) {
```

```
                recoveredPlainTextChar[i] = (char) (int)
((cipherText.charAt(i) - keyChar[i] + 26) % 26 + 65);
            }
            return String.valueOf(recoveredPlainTextChar).toLowerCase();
      }

      // Making length of Key same as length of Plain Text message
      public static char[] padKey(String key, int length) {
            char[] keyChar =
Arrays.copyOf(key.toUpperCase().toCharArray(), length);
            int i = 0;
            for (int j = key.toCharArray().length; j < keyChar.length;
j++) {
                  keyChar[j] = keyChar[i];
                  i++;
            }
            return keyChar;
      }
}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console ☒  Error Log  JUnit  Coverage
<terminated> VigenereCipher [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (Nov 17, 2020, 3:55:53 PM – 3:56:23 PM)
Enter the plaintext:
thisisvigenerecipher
Enter the key:
rmkcet
Cipher Text=KTSUMLMUQGRXIQMKTAVD
Recovered Plain Text=thisisvigenerecipher
```

RESULT

The Vigenere Cipher has been successfully implemented.

Ex. No : 2 **RAIL FENCE CIPHER**

Date :

AIM

To perform encryption and decryption using Rail Fence technique.

ALGORITHM

1. Encryption
    - The plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.
    - For example, to encipher the message "meet me after the toga party" with a rail fence of depth 2, we write the following:

      m e m a t r h t g p r y

        e t e f e t e o a a t

    - The encrypted message is MEMATRHTGPRYETEFETEOAAT.

/*Ex.No.2 Rail Fence Cipher*/

```
import java.util.Scanner;

public class RailFence {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the plaintext(Even Number of characters):");
		String plainText = sc.nextLine();
		char[] plainTextChar = plainText.toCharArray();
		char[] cipherTextChar = new char[plainTextChar.length];
		int i, j = 0;
		for (i = 0; i < (cipherTextChar.length / 2); i++) {
			cipherTextChar[i] = plainTextChar[j];
			cipherTextChar[i + plainTextChar.length / 2] = plainTextChar[j + 1];
			j = j + 2;
		}

		System.out.println("CipherText=" + String.valueOf(cipherTextChar));
		char[] recoveredPlainTextChar = new char[cipherTextChar.length];
		j = 0;
		for (i = 0; i < recoveredPlainTextChar.length; i = i + 2) {
			recoveredPlainTextChar[i] = cipherTextChar[j];
			recoveredPlainTextChar[i + 1] = cipherTextChar[j +
(plainTextChar.length / 2)];
			j++;
		}
		System.out.println("Recovered Plain Text=" +
String.valueOf(recoveredPlainTextChar));
		sc.close();
	}
}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> RailFence [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 17, 2020, 3:48:06 PM – 3:48:25 PM)
Enter the plaintext(Even Number of characters):
meetmeafterparty
CipherText=mematratetefepry
Recovered Plain Text=meetmeafterparty
```

RESULT

The Rail Fence Cipher has been successfully implemented using Java.

Ex. No : 3 **DATA ENCRYPTION STANDARD**

Date :

AIM

To perform encryption and decryption using DES.

ALGORITHM

1. Using Java’s in-built packages, DES algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get key and other cryptographic details using appropriate methods.
6. Perform encryption using appropriate methods.
7. To recover the plain text message, perform decryption using appropriate method.

/*Ex.No.3 DES*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;

public class DESDemo {
      public static void main(String[] args) throws IOException,
NoSuchAlgorithmException {
            Key key = getKey();
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Enter the plaintext message");
            String plainTextMessage = br.readLine();
            byte[] cipherText = encrypt(plainTextMessage, key);
            BigInteger b = new BigInteger(1, cipherText);
            System.out.println("CipherText in Hexadecimal Form:" +
b.toString(16));
            System.out.println("Recovered Plain Text Message:" +
decrypt(cipherText, key));
      }

      public static Key getKey() throws NoSuchAlgorithmException {
            KeyGenerator kg = KeyGenerator.getInstance("DES");
            kg.init(56);
            Key k = kg.generateKey();
```

```
            return k;
      }

      public static byte[] encrypt(String plainTextMessage, Key key) {
            byte[] cipherText = null;
            try {
                  Cipher cipher = Cipher.getInstance("DES");
                  byte[] input = plainTextMessage.getBytes();
                  BigInteger b = new BigInteger(1, input);
                  System.out.println("PlainText in the hexadecimal form: "
+ b.toString(16));
                  cipher.init(Cipher.ENCRYPT_MODE, key);
                  cipherText = cipher.doFinal(input);
            } catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
                        | NoSuchPaddingException e) {
                  e.printStackTrace();
            }
            return cipherText;
      }

      public static String decrypt(byte[] cipherTextMessage, Key key) {
            byte[] plainText = null;
            try {
                  Cipher cipher = Cipher.getInstance("DES");
                  cipher.init(Cipher.DECRYPT_MODE, key);
                  plainText = cipher.doFinal(cipherTextMessage);
            } catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
                        | NoSuchPaddingException e) {
                  e.printStackTrace();
            }
            return new String(plainText);
      }

}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> DESDemo [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 16, 2020, 10:45:04 PM – 10:45:42 PM)
Enter the plaintext message
Vanakkam to RMKCET
PlainText in the hexadecimal form: 56616e616b6b616d20746f20524d4b434554
CipherText in Hexadecimal Form:9a07ca17e1f3f6f3fdea84d892afaa2f85af9e73497877e4
Recovered Plain Text Message:Vanakkam to RMKCET
```

RESULT

The Java program to perform encryption and decryption using DES was successfully implemented.

Ex. No : 4 **ADVANCED ENCRYPTION STANDARD**

Date :

AIM

To perform encryption and decryption using AES.

ALGORITHM

1. Using Java’s in-built packages, AES algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get key and other cryptographic details using appropriate methods.
6. Perform encryption using appropriate methods.
7. To recover the plain text message, perform decryption using appropriate method.

/*Ex.No.4 AES*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class AESDemo {

	public static void main(String[] args) throws IOException,
NoSuchAlgorithmException {
		SecretKey key = getKey();
		BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
		System.out.println("Enter the plaintext message");
		String plainTextMessage = br.readLine();
		byte[] cipherText = encrypt(plainTextMessage, key);
		BigInteger b = new BigInteger(1, cipherText);
		System.out.println("CipherText in Hexadecimal Form:" +
b.toString(16));
		System.out.println("Recovered Plain Text Message:" +
decrypt(cipherText, key));
	}

	public static SecretKey getKey() throws NoSuchAlgorithmException {
		KeyGenerator kg = KeyGenerator.getInstance("AES");
		kg.init(128);
```

```
		SecretKey k = kg.generateKey();
		return k;
	}

	public static byte[] encrypt(String plainTextMessage, SecretKey key)
{
		byte[] cipherText = null;
		try {
			Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
			byte[] input = plainTextMessage.getBytes();
			BigInteger b = new BigInteger(1, input);
			System.out.println("PlainText in the hexadecimal form: "
+ b.toString(16));
			cipher.init(Cipher.ENCRYPT_MODE, key);
			cipherText = cipher.doFinal(input);
		} catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
				| NoSuchPaddingException e) {
			e.printStackTrace();
		}
		return cipherText;
	}

	public static String decrypt(byte[] cipherTextMessage, SecretKey
key) {
		byte[] plainText = null;
		try {
			Cipher cipher =
Cipher.getInstance("AES/ECB/PKCS5Padding");
			cipher.init(Cipher.DECRYPT_MODE, key);
			plainText = cipher.doFinal(cipherTextMessage);
		} catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
				| NoSuchPaddingException e) {
			e.printStackTrace();
```

```
        }

        return new String(plainText);
    }

}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> DESDemo [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 17, 2020, 7:49:20 AM – 7:49:41 AM)
Enter the plaintext message
This is AES Cipher
PlainText in the hexadecimal form: 546869732069732041455320436970686572
CipherText in Hexadecimal Form:1a2c000fecbc5f38e4d9c4acb1d94c20c1bc6d82bb406e8
Recovered Plain Text Message:This is AES Cipher
```

RESULT

The Java program to perform encryption and decryption using AES was successfully implemented.

Ex. No : 5 **RSA**

Date :

AIM

To perform encryption and decryption using RSA.

ALGORITHM

Version 1:

1. Using Java's in-built packages, RSA algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get key and other cryptographic details using appropriate methods.
6. Perform encryption using appropriate methods.
7. To recover the plain text message, perform decryption using appropriate method.

Version 2:

1. Use BigInteger class to perform RSA encryption and decryption.
2. Encryption

   $C = M^e \bmod n$
3. Decryption

   $M = C^d \bmod n$
4. Get encryption constant, prime numbers p and q from the user.
5. Calculate n and phi(n).
6. Calculate decryption constant.
7. Use various methods of BigInteger to perform modular arithmetic.

Version 3:

1. Use HTML for creating user interaction form and JavaScript for computing the formulae.
2. Encryption: $C = M^e \bmod n$
3. Decryption: $M = C^d \bmod n$
4. Get encryption constant, prime numbers p and q from the user.
5. Calculate n, phi(n), decryption constant.
6. Compute encryption and decryption using the above formulae.

/*Ex.No.5(a) RSA Using Java Crypto & Security Packages*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.Key;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

public class RSADemo {

      public static void main(String[] args) throws IOException,
NoSuchAlgorithmException, NoSuchPaddingException,
                  InvalidKeyException, IllegalBlockSizeException,
BadPaddingException, NoSuchProviderException {
            KeyPair kp = getKeys();
            BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
            System.out.println("Enter the plaintext message");
            String plainTextMessage = br.readLine();
            byte[] cipherText = encrypt(plainTextMessage, kp.getPublic());
            System.out.println("CipherText in the hexadecimal form: " +
new BigInteger(1, cipherText).toString(16));
            byte[] recoveredPlainText = decrypt(cipherText,
kp.getPrivate());
            System.out.println("Recovered PlainText : " + new
String(recoveredPlainText));
```

```java
	}

	public static KeyPair getKeys() throws NoSuchAlgorithmException {
		KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
		kpg.initialize(2048);
		KeyPair kp = kpg.genKeyPair();
		return kp;
	}

	public static byte[] encrypt(String plainTextMessage, Key publicKey)
{
		byte[] cipherText = null;
		try {
			Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-
256AndMGF1Padding");
			byte[] input = plainTextMessage.getBytes();
			BigInteger b = new BigInteger(1, input);
			System.out.println("PlainText in the hexadecimal form: "
+ b.toString(16));
			cipher.init(Cipher.ENCRYPT_MODE, publicKey);
			cipherText = cipher.doFinal(input);
			// System.out.println(new String(cipherText));
		} catch (InvalidKeyException | IllegalBlockSizeException |
BadPaddingException | NoSuchAlgorithmException
				| NoSuchPaddingException e) {
			e.printStackTrace();
		}
		return cipherText;
	}

	public static byte[] decrypt(byte[] cipherText, Key privateKey) {
		byte[] plainText = null;
		try {
			Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-
256AndMGF1Padding");
			cipher.init(Cipher.DECRYPT_MODE, privateKey);
```

```
                plainText = cipher.doFinal(cipherText);
            } catch (NullPointerException | InvalidKeyException |
IllegalBlockSizeException | BadPaddingException
                        | NoSuchAlgorithmException |
NoSuchPaddingException e) {
                e.printStackTrace();
            }
            return plainText;
      }

}
```

/*Ex.No.5(b): RSA Using Java BigInteger*/

```
import java.io.BufferedReader;
import java.io.IOException;
import java.math.BigInteger;

public class RSABigIntegerDemo {

	public static void main(String[] args) throws IOException {
		// TODO Auto-generated method stub
		BufferedReader br = new BufferedReader(new
java.io.InputStreamReader(System.in));
		System.out.println("Enter Prime p:");
		BigInteger p = new BigInteger(br.readLine());
		System.out.println("Enter Prime q:");
		BigInteger q = new BigInteger(br.readLine());
		BigInteger phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
		BigInteger n = p.multiply(q);
		System.out.println("Enter Encryption Constant e:");
		BigInteger e = new BigInteger(br.readLine());
		BigInteger d = e.modInverse(phi);
		System.out.println("Enter Plaintext Message M:");
		BigInteger M = new BigInteger(br.readLine());
		BigInteger C = encrypt(M, e, n);
		System.out.println("Encrypted Message C=" + C);
		BigInteger M2 = decrypt(C, d, n);
		System.out.println("Decrypted Message M=" + M2);
	}

	public static BigInteger encrypt(BigInteger M, BigInteger e,
BigInteger n) {
		BigInteger C = M.modPow(e, n);
		return C;
	}
```

```
	public static BigInteger decrypt(BigInteger C, BigInteger d,
BigInteger n) {
		BigInteger M = BigInteger.ONE;
		M = C.modPow(d, n);
		return M;
	}

}
```

/*Ex.No.5(c) RSA Using HTML and JavaScript*/

```
<!-- index.html -->
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>RSA</title>
<link href="style.css" rel="stylesheet">
<script type="text/javascript" src="rsaencrypt.js"></script>
</head>
<body>
<form name="rsaform">
<fieldset>
<legend>Selecting Primes and Key Generation by Receiver:</legend>
<label for="p">Enter Prime p:</label>
<input type="text" id="p" name="p"><br>
<label for="q">Enter Prime q:</label>
<input type="text" id="q" name="q"><br>
<label for="e">Enter Encryption Constant e:</label>
<input type="text" id="e" name="e"><br>
</fieldset> <br>
<fieldset>
<legend>Encryption by Sender:</legend>
<label for="M">Enter Message:</label>
<input type="text" id="M" name="M"><br>
<input type="button" value="Encrypt" onClick="encrypt()">
<input type="reset" value="Clear"><br>
<label for="M">Encrypted Message:</label>
<input type="text" id="C" name="C"><br>
</fieldset><br>

<fieldset>
<legend>Decryption by Sender:</legend>
<label for="C2">Cipher Text Message Received:</label>
<input type="text" id="C2" name="C2"><br>
```

```
<input type="button" value="Calculate Decryption Constant d"
onClick="getDecryptionConstant()"><br>
<label for="d">Decryption Constant:</label>
<input type="text" id="d" name="d"><br>
<input type="button" value="Decrypt" onClick="decrypt()">
<input type="reset" value="Clear"><br>
<label for="M2">Decrypted Message:</label>
<input type="text" id="M2" name="M2">
</fieldset>

</form>
</body>
</html>
```

/*rsaencrypt.js*/

```
function getPrimes() {
      var p = parseInt(document.rsaform.p.value);
      var q = parseInt(document.rsaform.q.value);
      return [p,q];
}

function calculateN(){
      let p,q;
      [p,q] = getPrimes();
      var n = p * q;
      return n;
}

function calculatePhi(){
      let p,q;
      [p,q] = getPrimes();
      var phi = (p-1) * (q-1);
      return phi;
}

function getEncryptionConstant(){
```

```
        var e = parseInt(document.rsaform.e.value);
        return e;
}

function getDecryptionConstant() {
        var phi = calculatePhi();
        var e = getEncryptionConstant();
        var d = (modInverse(e, phi) + phi) % phi;
        document.getElementById('d').value = d;
        return d;
}

function getPublicKey() {
        var e = getEncryptionConstant();
        var n = calculateN();
        return [e, n];
}

function getPrivateKey() {
        var d = getDecryptionConstant();
        var n = calculateN();
        return [d, n];
}

function encrypt() {
        let e, n;
        [e,n] = getPublicKey();
        var M = parseInt(document.rsaform.M.value);
        const C = powerMod(M, e, n);
        //alert("Encrypted Cipher Text="+ C);
        document.getElementById('C').value = C;
        document.getElementById('C2').value = C;
        return C;
}
```

```
function decrypt(){
      let d, n;
      [d, n] = getPrivateKey();
      var C = document.getElementById('C2').value;
      const  M = powerMod(C, d, n);
      document.getElementById('M2').value = M;
}

/*Source: http://umaranis.com/2018/07/12/calculate-modular-exponentiation-
powermod-in-javascript-ap-n/*/
function powerMod(base, exponent, modulus) {
    if (modulus === 1) return 0;
    var result = 1;
    base = base % modulus;
    while (exponent > 0) {
        if (exponent % 2 === 1)  //odd number
            result = (result * base) % modulus;
        exponent = exponent >> 1; //divide by 2
        base = (base * base) % modulus;
    }
    return result;
}

function modInverse(b, m) {
      let A1, A2, A3;
      [A1, A2, A3] = [1, 0, m];
      let B1, B2, B3;
      [B1, B2, B3] = [0, 1, b];
      let T1, T2, T3;
      var Q;
      while (B3 != 0 || B3 != 1) {
            if (B3 == 0)
                  return 0;
            if (B3 == 1)
                  return B2;
            Q = Math.floor(A3 / B3);
```

```
            [T1, T2, T3] = [A1 - Q * B1, A2 - Q * B2, A3 - Q * B3];
            [A1, A2, A3] = [B1, B2, B3];
            [B1, B2, B3] = [T1, T2, T3];
            }
            return 0;
}
```

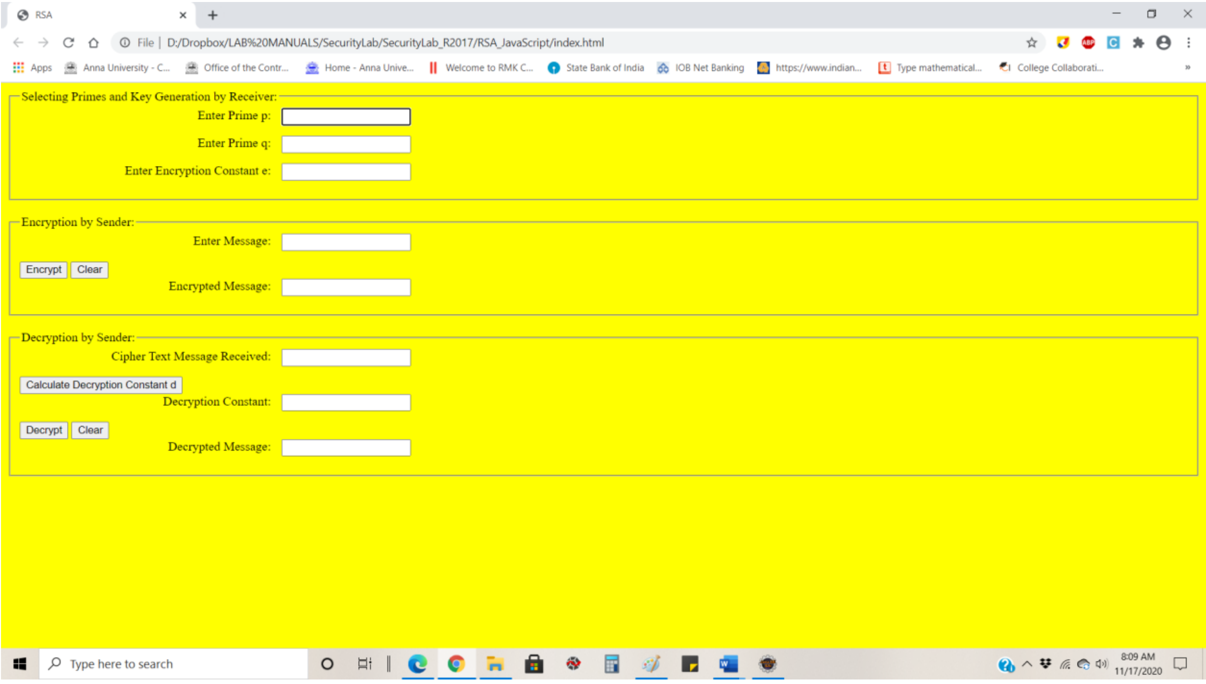
/*style.css*/

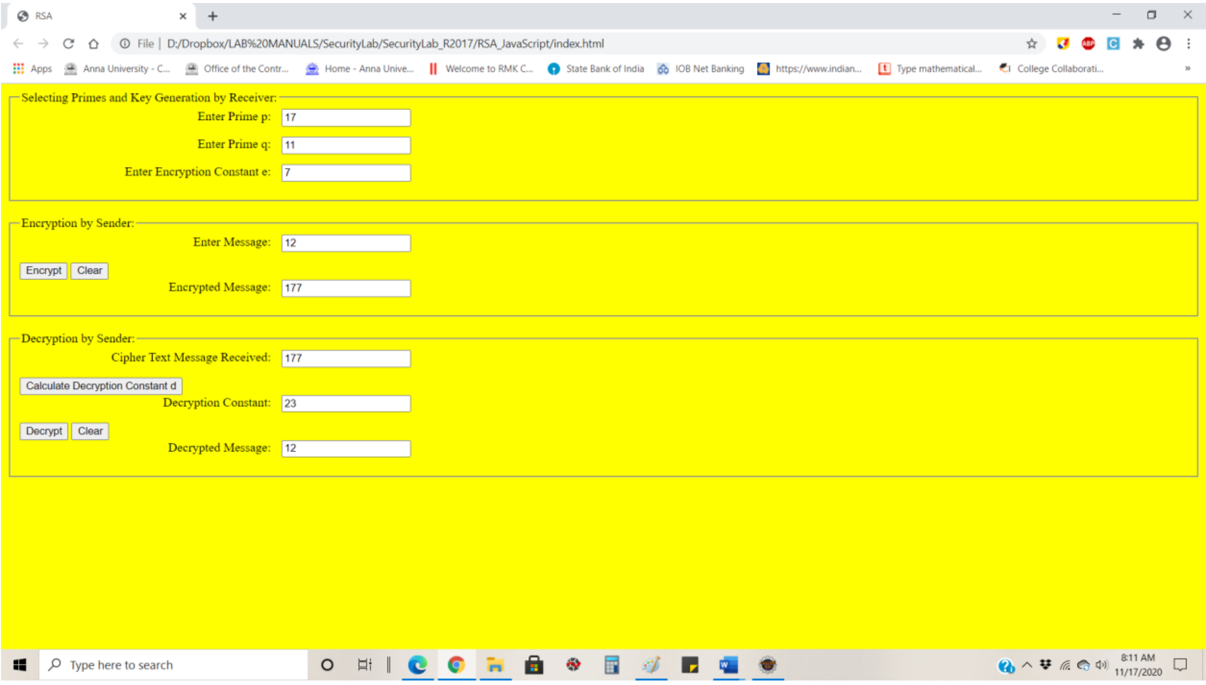
```
body {
      background-color: yellow;
}

.clearfix::after {
  clear: both;
}

label {
      float:left;
      width:20em;
      text-align:right;
}

input[type=text] {
      margin-left: 1em;
      margin-bottom: 1em;
}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> RSADemo (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (Nov 17, 2020, 7:59:29 AM – 7:59:54 AM)
Enter the plaintext message
This is RSA Algorithm using inbuilt Cryptographic libraries
PlainText in the hexadecimal form: 546869732069732052534120416c676f726974686d207573696e6720696e6275696c742
CipherText in the hexadecimal form: 8e9992e5830b70a720a65f555387476e5b7c7634c5cdf8f645afb72e8e8de78a3bffe
Recovered PlainText : This is RSA Algorithm using inbuilt Cryptographic libraries
```

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> RSABigIntegerDemo (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (Nov 17, 2020, 8:04:37 AM – 8:04:57 AM)
Enter Prime p:
17
Enter Prime q:
11
Enter Encryption Constant e:
7
Enter Plaintext Message M:
12
Encrypted Message C=177
Decrypted Message M=12
```





RESULT

The RSA algorithm was successfully implemented by (i) Java crypto and security packages, (ii) Java BigInteger methods, and (iii) HTML and JavaScript successfully.

Ex. No : 6 **DIFFIE – HELLMAN KEY EXCHANGE**

Date :

AIM

To perform Diffie-Hellman Key Exchange.

ALGORITHM

1. Use various methods of BigInteger to perform modular arithmetic.
2. Get the prime number and one of its primitive root from the user.
3. Get the private keys of User A and User B.
4. Calculate public keys of User A and User B.
5. Calculate shared secret key.
6. Algorithm Formulae

   X_A - User A’s Private Key

   X_B - User B’s Private Key

   User A’s Public Key: $Y_A = \alpha^{X_A} \bmod q$

   User B’s Public Key: $Y_B = \alpha^{X_B} \bmod q$

   Shared Secret Key calculated by User A: $K = (Y_B)^{X_A} \bmod q$

   Shared Secret Key calculated by User B: $K = (Y_A)^{X_B} \bmod q$

/*Ex.No.6 Diffie-Hellman*/

```
import java.math.BigInteger;
import java.util.Scanner;

public class DiffieHellmanDemo {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the primitive root");
		BigInteger g = sc.nextBigInteger();
		System.out.println("Enter the prime number");
		BigInteger q = sc.nextBigInteger();
		System.out.println("Enter the private key of User A");
		BigInteger xA = sc.nextBigInteger();
		System.out.println("Enter the private key of User B");
		BigInteger xB = sc.nextBigInteger();
		BigInteger yA = computePublicKey(g, xA, q);
		BigInteger yB = computePublicKey(g, xB, q);
		System.out.println("Public Key of A: " + yA);
		System.out.println("Public Key of B: " + yB);
		BigInteger K1 = computeSharedKey(xA, yB, q);
		BigInteger K2 = computeSharedKey(xB, yA, q);
		System.out.println("Shared Key Computed by User A = " + K1);
		System.out.println("Shared Key Computed by User B = " + K2);
		sc.close();
	}

	public static BigInteger computePublicKey(BigInteger g, BigInteger 
xA, BigInteger q) {
		BigInteger Y = g.modPow(xA, q);
		return Y;
	}
```

```
	public static BigInteger computeSharedKey(BigInteger X, BigInteger
Y, BigInteger q) {
		BigInteger K = Y.modPow(X, q);
		return K;
	}

}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console ☒  Error Log  JUnit  Coverage
<terminated> DiffieHellmanDemo [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (Nov 17, 2020, 8:24:04 AM – 8:24:29 AM)
Enter the primitive root
7
Enter the prime number
71
Enter the private key of User A
5
Enter the private key of User B
12
Public Key of A: 51
Public Key of B: 4
Shared Key Computed by User A = 30
Shared Key Computed by User B = 30
```

RESULT

The Diffie-Hellman algorithm was successfully implemented using Java.

Ex. No : 7 **SHA-1**

Date :

AIM

To generate message digest value using SHA-1.

ALGORITHM

1. Using Java's in-built packages, SHA-1 algorithm is implemented.
2. Import necessary packages.
3. Convert the input message to byte format for easy manipulation.
4. Using appropriate methods, message digest is generated.
5. Message digest is displayed in hexadecimal format.

/*Ex.No.7 SHA-1*/

```java
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class MessageDigestDemo {
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the message");
		String message = sc.nextLine();
		byte[] hash = generateMessagedigest(message);
		BigInteger b = new BigInteger(1, hash);
		String hashValue = b.toString(16);
		System.out.println("MessageDigest in HexaDecimal Format: " +
hashValue);
	}

	public static byte[] generateMessagedigest(String message) {
		byte[] hash = null;
		byte[] input = message.getBytes();
		System.out.println("Message: " + new String(input));
		MessageDigest md;
		try {
			md = MessageDigest.getInstance("SHA-1");
			md.update(input);
			hash = md.digest();
		} catch (NoSuchAlgorithmException e) {
			e.printStackTrace();
		}
		return hash;
	}

}
```

OUTPUT

```
Markers  Properties  Servers  Data Source Explorer  Problems  Console  Error Log  JUnit  Coverage
<terminated> MessageDigestDemo [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 17, 2020, 8:29:11 AM – 8:29:50 AM)
Enter the message
This is SHA-1 algorithm
Message: This is SHA-1 algorithm
MessageDigest in HexaDecimal Format: a64fa09cdb195e96d2a695cae6eda3077c82ebe3
```

RESULT

The SHA-1 algorithm was successfully implemented using Java.

Ex. No : 8 **DIGITAL SIGNATURE STANDARD**

Date :

AIM

To generate and verify digital signature using DSS.

ALGORITHM

1. Using Java's in-built packages, DSS algorithm is implemented.
2. Import necessary packages.
3. Add security provider.
4. Convert the input message to byte format for easy manipulation.
5. Get cryptographic details using appropriate methods.
6. Sign and verify the message using appropriate methods.

/*Ex.No.8 Digital Signature Standard*/

```
import java.math.BigInteger;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.util.Scanner;

public class DSSDemo {

	public static void main(String[] args) throws Exception {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter the message");
		String inputMessage = sc.nextLine();
		Signature signAlgorithm =
Signature.getInstance("SHA256withDSA");
		KeyPair kp = getKeys();
		byte[] signBytes = generateSignature(inputMessage,
kp.getPrivate(), signAlgorithm);
		BigInteger signedMessage = new BigInteger(1, signBytes);
		System.out.println("Digital Signature generated by sender:" +
signedMessage.toString(16));

		if (verifySignature(inputMessage, signBytes, kp.getPublic(),
signAlgorithm))
			System.out.println("Signature is verified");
		else
			System.out.println("Signature is not matching");
		sc.close();
	}

	public static byte[] generateSignature(String inputMessage,
PrivateKey privateKey, Signature signAlgorithm) {
```

```
		byte[] sigBytes = null;
		try {
			signAlgorithm.initSign(privateKey);
			byte[] message = inputMessage.getBytes();
			signAlgorithm.update(message);
			sigBytes = signAlgorithm.sign();
		} catch (Exception e) {
			e.printStackTrace();
		}
		return sigBytes;
	}

	public static boolean verifySignature(String inputMessage, byte[]
sigBytes, PublicKey publicKey,
			Signature signAlgorithm) {
		boolean result = true;
		try {
			signAlgorithm.initVerify(publicKey);
			signAlgorithm.update(inputMessage.getBytes());
			result = signAlgorithm.verify(sigBytes);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return result;
	}

	public static KeyPair getKeys() throws NoSuchAlgorithmException {
		KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
		kpg.initialize(512);
		KeyPair kp = kpg.genKeyPair();
		return kp;
	}

}
```

OUTPUT

```
Markers Properties Servers Data Source Explorer Problems Console Error Log JUnit Coverage
<terminated> DSSDemo [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Nov 17, 2020, 8:35:20 AM – 8:35:30 AM)
Enter the message
This is DSS algorithm
Digital Signature generated by sender:302d02146b4271686feea87c3d49f593d4037a60605f1c5b02150095f76f7311396
Signature is verified
```

RESULT

The Digital Signature Standard algorithm was successfully implemented using Java.

Ex. No : 9 **SNORT**

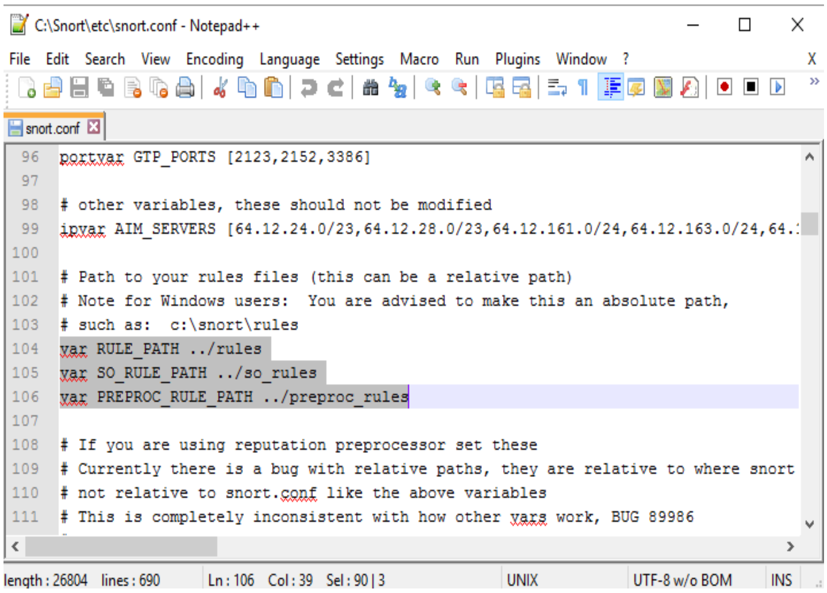
Date :

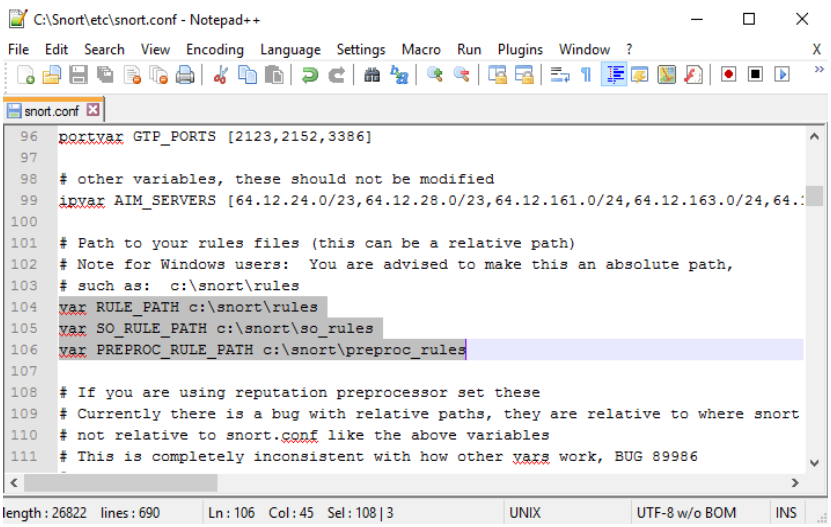
AIM

To demonstrate intrusion detection using Snort.

PROCEDURE

1. Install WinPcap libraries.
2. Install Snort
3. After installation, open ***snort.conf*** using any text editor. (It is present in the location C:\Snort\etc)
4. Edit the contents specifying rules path.

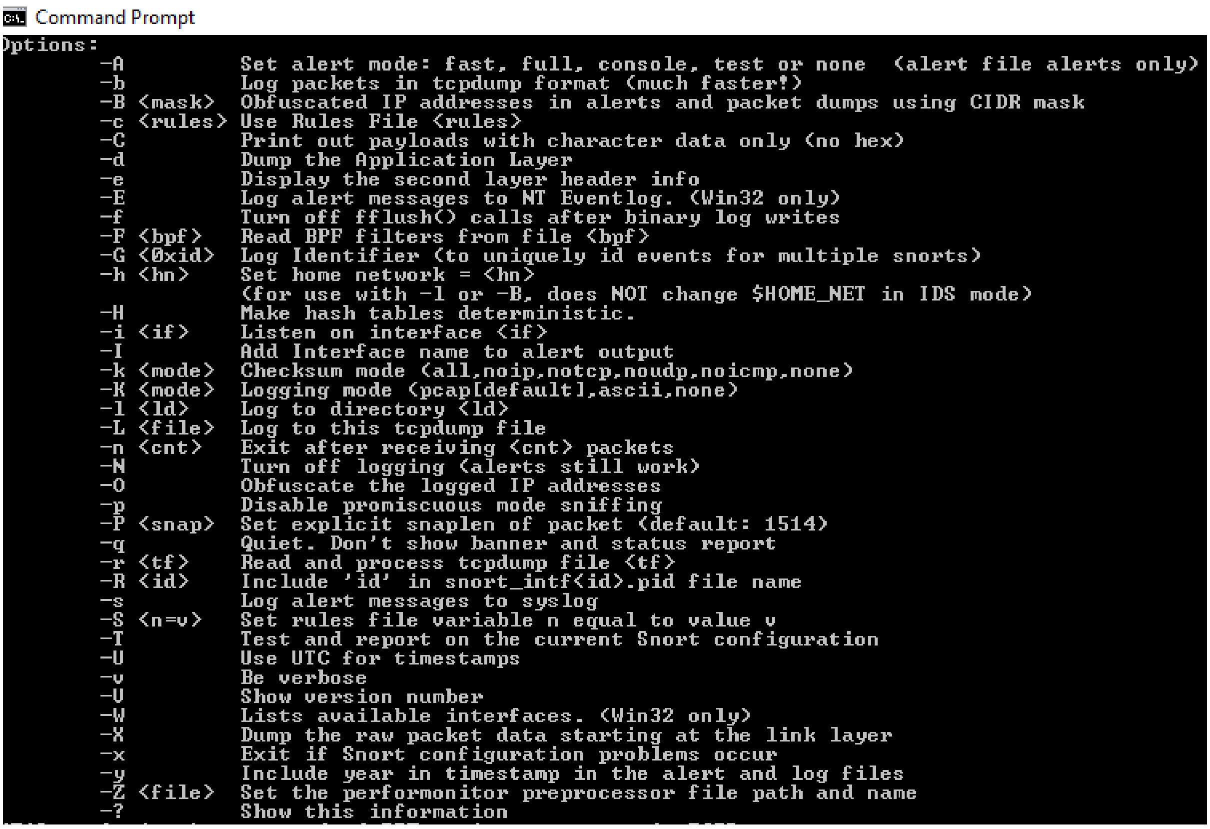




5. To use snort from any window path, set **path** variable as **C:\Snort\bin** under **Environment Variables**.
6. Run ***snort -W*** to see a list of interfaces available to Snort

```
Command Prompt
C:\Users\madhu.amarnath>snort -W

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.8.3-WIN32 GRE (Build 383)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2015 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.3

Index   Physical Address        IP Address      Device Name     Description
-----   ----------------        ----------      -----------     -----------
    1   E0:DB:55:B1:61:45       0000:0000:fe80:0000:0000:0000:7ca3:0d3d \Device\NPF_{0F5D141C-0CC8-4
425-A161-17E236D43FB0}  Realtek PCIe GBE Family Controller
    2   00:00:00:00:00:00       0000:0000:fe80:0000:0000:0000:e58c:ddfc \Device\NPF_{3358EE96-D9CD-4
97D-AA56-E40E12CAF40C}  Microsoft
    3   00:00:00:00:00:00       0000:0000:fe80:0000:0000:0000:349e:43eb \Device\NPF_{B715300D-2E72-4
CC7-A335-40CF98CCB04A}  Microsoft

C:\Users\madhu.amarnath>_
```

7. Snort Options



```
Command Prompt
Options:
        -A         Set alert mode: fast, full, console, test or none  (alert file alerts only)
        -b         Log packets in tcpdump format (much faster!)
        -B <mask>  Obfuscated IP addresses in alerts and packet dumps using CIDR mask
        -c <rules> Use Rules File <rules>
        -C         Print out payloads with character data only (no hex)
        -d         Dump the Application Layer
        -e         Display the second layer header info
        -E         Log alert messages to NT Eventlog. (Win32 only)
        -f         Turn off fflush() calls after binary log writes
        -F <bpf>   Read BPF filters from file <bpf>
        -G <0xid>  Log Identifier (to uniquely id events for multiple snorts)
        -h <hn>    Set home network = <hn>
                   (for use with -l or -B, does NOT change $HOME_NET in IDS mode)
        -H         Make hash tables deterministic.
        -i <if>    Listen on interface <if>
        -I         Add Interface name to alert output
        -k <mode>  Checksum mode (all,noip,notcp,noudp,noicmp,none)
        -K <mode>  Logging mode (pcap[default],ascii,none)
        -l <ld>    Log to directory <ld>
        -L <file>  Log to this tcpdump file
        -n <cnt>   Exit after receiving <cnt> packets
        -N         Turn off logging (alerts still work)
        -O         Obfuscate the logged IP addresses
        -p         Disable promiscuous mode sniffing
        -P <snap>  Set explicit snaplen of packet (default: 1514)
        -q         Quiet. Don't show banner and status report
        -r <tf>    Read and process tcpdump file <tf>
        -R <id>    Include 'id' in snort_intf<id>.pid file name
        -s         Log alert messages to syslog
        -S <n=v>   Set rules file variable n equal to value v
        -T         Test and report on the current Snort configuration
        -U         Use UTC for timestamps
        -v         Be verbose
        -V         Show version number
        -W         Lists available interfaces. (Win32 only)
        -X         Dump the raw packet data starting at the link layer
        -x         Exit if Snort configuration problems occur
        -y         Include year in timestamp in the alert and log files
        -Z <file>  Set the performonitor preprocessor file path and name
        -?         Show this information
```

8. Sniffer Mode

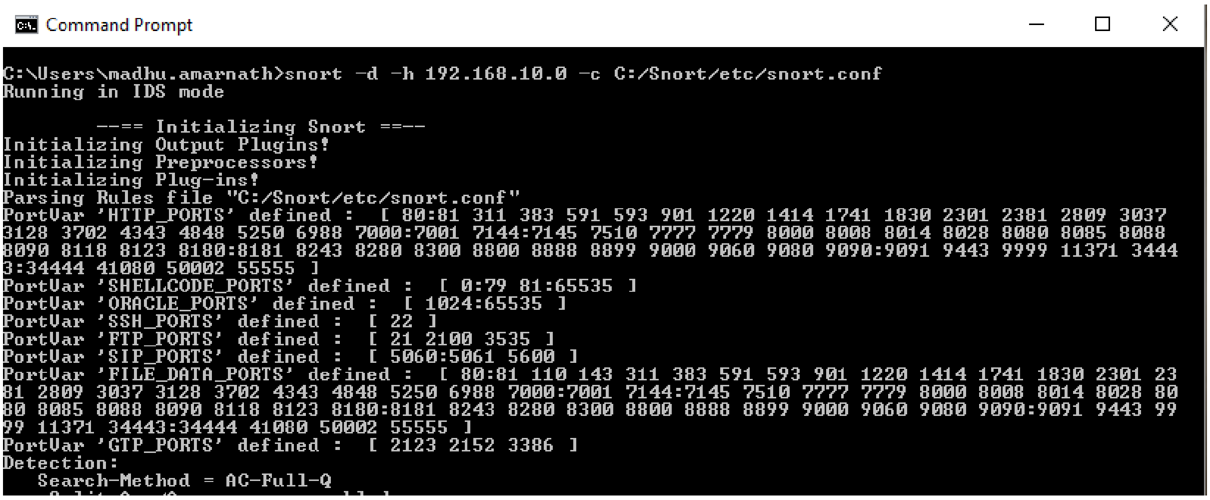
   snort -i 1 –v

9. Packet Logger Mode

   snort -l c:\snort\log -i1

10. Network IDS Mode

snort -d -h 192.168.10.0 -c C:/Snort/etc/snort.conf



RESULT

The Snort tool was used to demonstrate Intrusion Detection System.

Ex. No : 10 **Automated Attack and Penetration Tools: ZAP**

Date :

AIM

To perform penetration testing on a web application using ZAP.

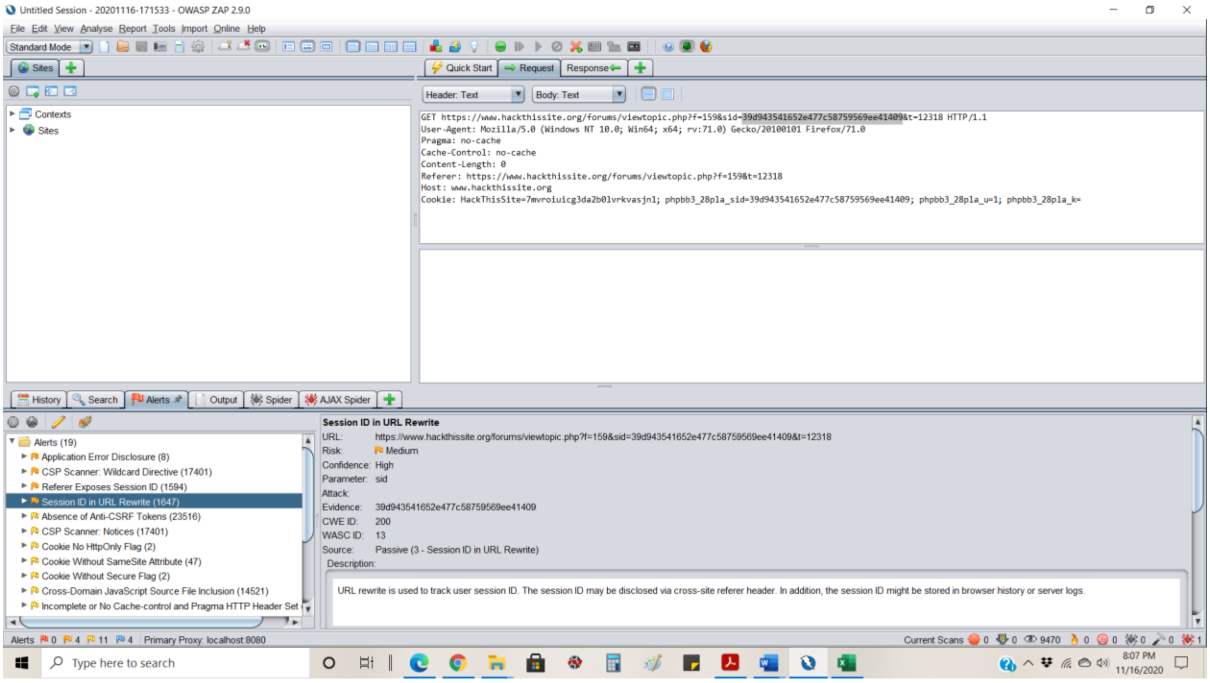
DESCRIPTION

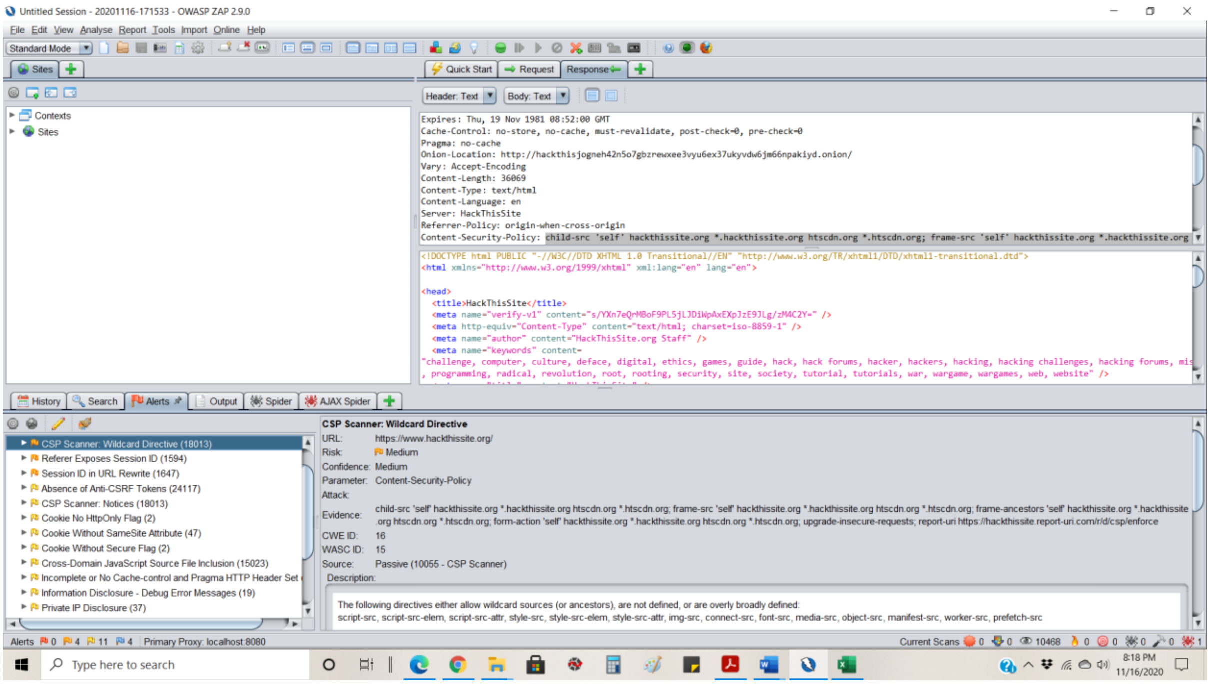
- Zed Attack Proxy (ZAP) is a free, open-source penetration testing tool being maintained under the umbrella of the Open Web Application Security Project (OWASP).
- ZAP is designed specifically for testing web applications and is both flexible and extensible.
- A penetration test, also known as a pen test or ethical hacking, is a simulated cyber- attack against your computer system to check for exploitable vulnerabilities.
- The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. (Website: https://owasp.org/)
- Requirements: The Windows and Linux versions require Java 8 or higher to run.
- Download Link: https://www.zaproxy.org/download/
- Getting Started: https://www.zaproxy.org/getting-started/

PROCEDURE

1. Start ZAP and click the Quick Start tab of the Workspace Window.
2. Click the large Automated Scan button.
3. In the URL to attack text box, enter the full URL of the web application you want to attack.
    a. Choose a website that you have permission to perform penetration testing.
    b. https://securitytrails.com/blog/vulnerable-websites-for-penetration-testing
    c. https://www.hackthissite.org/
4. Click the Attack.
5. Once the scanning is complete, click the “Alerts” tab to view the vulnerabilities details.

OUTPUT





RESULT

Thus, the OWASP ZAP tool was used to perform pen testing on a website and discover vulnerabilities successfully.

Ex. No : 11 **GMER – Rootkit Detection and Removal**

Date :

AIM

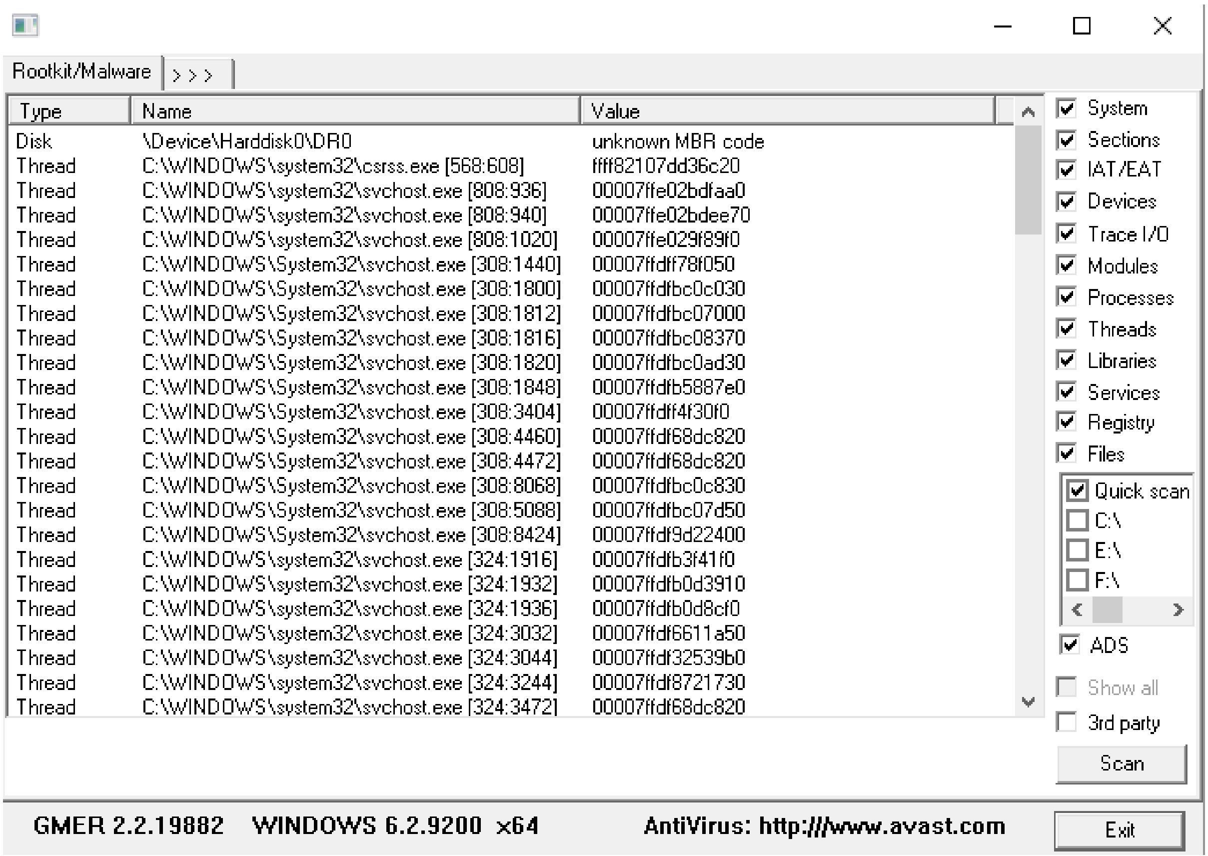
To use GMER application to detect and remove rootkits.

DESCRIPTION

- GMER is an application that detects and removes rootkits.
- A rootkit is a malicious software that allows an unauthorized user to have privileged access to a computer and to restricted areas of its software.
- A rootkit allows someone to maintain command and control over a computer without the computer user/owner knowing about it.

PROCEDURE

1. Click the **Scan** button.



RESULT

The GMER application was installed and run for detection of rootkits. No rootkits were found.

APPENDIX 1 – Important Links

1. Java JDK
   https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html

2. Eclipse IDE
   https://www.eclipse.org/downloads/packages/release/2020-09/r/eclipse-ide-enterprise-java-developers

3. ZAP – Penetration Testing Tool
   https://www.zaproxy.org/download/

4. Snort - Network Intrusion Detection & Prevention System
   https://www.snort.org/downloads/snort/Snort_2_9_8_3_Installer.exe

5. GMER – Application to detect and remove rootkits
   http://www.gmer.net/

6. WinPcap – Windows Packet Capture Library
   http://www.winpcap.org/install/default.htm