

**Ex. No: 1a****Date:****CAESAR CIPHER****AIM**

To encrypt and decrypt the given message by using Caesar Cipher encryption algorithm.

**ALGORITHM**

1. Input the plain text to be encrypted.
2. Input the key to be used for Caesar Cipher.
3. Process each character of the plain text and append the encrypted values
  - a. If the input character is uppercase, encrypt using:  
$$('A' + (i - 'A' + \text{offset}) \% 26)$$
  - b. If the input character is lowercase, encrypt using:  
$$('a' + (i - 'a' + \text{offset}) \% 26)$$
4. Return the appended values as the encrypted message.
5. Decryption process is the reverse of the encryption process.
6. Return the appended values as the decrypted message.
7. Display the encrypted message.
8. Display the decrypted message

**PROGRAM**

```
import java.util.Scanner;
class Main {
    public static String encode(String enc, int offset) {
        StringBuilder encoded = new StringBuilder();//to build a new string
        for (char i : enc.toCharArray()) {
            if (Character.isLetter(i)) {
                if (Character.isUpperCase(i)) {
                    encoded.append((char) ('A' + (i - 'A' + offset) % 26)); // ASCII A=65
                }
                else {
                    encoded.append((char) ('a' + (i - 'a' + offset) % 26)); // ASCII a=97
                }
            }
            else {
                encoded.append(i);
            }
        }
    }
}
```

```
        return encoded.toString();
    }

    public static String decode(String enc, int offset) {
        return encode(enc, 26 - offset);
    }

    public static void main(String[] args) throws java.lang.Exception {
        Scanner scan = new Scanner(System.in);
        System.out.println("Simulating Caesar Cipher\n-----");
        System.out.println("Enter the plain text");
        String msg = scan.next();
        System.out.println("Enter the key");
        Integer key = scan.nextInt();
        String cipher = encode(msg, key);
        System.out.printf("Encrypted Message: ");
        System.out.println(cipher);
        System.out.printf("Decrypted Message: ");
        System.out.println(decode(cipher, key));
    }
}
```

**OUTPUT**

```
Simulating Caesar Cipher
-----
Enter the plain text
GIPSON
Enter the key
8
Encrypted Message: OQXAWV
Decrypted Message: GIPSON

...Program finished with exit code 0
Press ENTER to exit console.█
```

**RESULT**

Thus the program for Caesar cipher encryption and decryption algorithm has been implemented and the output verified successfully.

**Ex. No: 1b****Date:****PLAYFAIR CIPHER****AIM**

To encrypt and decrypt the given message by using Playfair encryption algorithm.

**ALGORITHM**

1. Input the key for Playfair cipher.
2. Input the plain text for encryption.
3. Create the key square matrix.
4. Prepare the text before processing
  - a. Change to uppercase.
  - b. Remove all non-alphabetic characters.
  - c. Replace the letter J by I.
5. Prepare the digrams and process them as follows; append the values returned to get the encrypted message:
  - a. If the digrams are in the same row; return the values in the next column.
  - b. If the digrams are in the same column; return the values in the next row.
  - c. If the digrams are neither in the same row or column; return the values in the horizontal diagonal.
6. Return the appended values as the encrypted message.
7. Decryption process is the reverse of the encryption process.
8. Return the appended values as the decrypted message.
9. Display the encrypted message.
10. Display the decrypted message

**PROGRAM**

```
import java.awt.Point;
import java.util.Scanner;
class Main {
    private static char[][] charTable;
    private static Point[] positions;

    private static String prepareText(String s) {
        s = s.toUpperCase().replaceAll("[^A-Z]", ""); // Using regex to remove non-alphabetic
                                                    characters
        return s.replace("J", "I");
    }
}
```

```
private static void createTbl(String key) {
    charTable = new char[5][5];
    positions = new Point[26]; // Point array stores (x,y) value
    String s = prepareText(key + "ABCDEFGHIJKLMNOPQRSTUVWXYZ");
    int len = s.length();
    for (int i = 0, k = 0; i < len; i++) {
        char c = s.charAt(i);
        if (positions[c - 'A'] == null) { // Check if characters are in key square matrix
            charTable[k / 5][k % 5] = c;
            positions[c - 'A'] = new Point(k / 5, k % 5);
            k++;
        }
    }
    System.out.println("The key square");
    for(int i=0;i<5;i++){
        for(int j=0;j<5;j++){
            System.out.print(charTable[i][j] + " ");
        }
        System.out.println();
    }
}

private static String codec(StringBuilder txt, int dir) {
    int len = txt.length();
    for (int i = 0; i < len; i += 2) {
        char a = txt.charAt(i);
        char b = txt.charAt(i + 1);
        int row1 = positions[a - 'A'].x;
        int row2 = positions[b - 'A'].x;
        int col1 = positions[a - 'A'].y;
        int col2 = positions[b - 'A'].y;
        if (row1 == row2) { // Digrams in same row
            col1 = (col1 + dir) % 5;
            col2 = (col2 + dir) % 5;
        }
        else if (col1 == col2) { // Digrams in same column
```

```
        row1 = (row1 + dir) % 5;
        row2 = (row2 + dir) % 5;
    }
    else { // Digrams neither in same row or column
        int tmp = col1;
        col1 = col2;
        col2 = tmp;
    }
    txt.setCharAt(i, charTable[row1][col1]);
    txt.setCharAt(i + 1, charTable[row2][col2]);
}
return txt.toString();
}

private static String encode(String s) {
    StringBuilder sb = new StringBuilder(s);
    for (int i = 0; i < sb.length(); i += 2) {
        if (i == sb.length() - 1) {
            sb.append(sb.length() % 2 == 1 ? 'Z' : "");
        }
        else if (sb.charAt(i) == sb.charAt(i + 1)) {
            sb.insert(i + 1, 'Z');
        }
    }
    return codec(sb, 1);
}

private static String decode(String s) {
    return codec(new StringBuilder(s), 4);
}

public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter the key");
    String key = scan.next();
    System.out.println("Enter the plain text");
    String txt = scan.next();
}
```

```
        createTbl(key);
        String enc = encode(prepareText(txt));
        System.out.println("Simulating Playfair Cipher\n ----- ");
        System.out.println("Input Message : " + txt);
        System.out.println("Encrypted Message : " + enc);
        System.out.println("Decrypted Message : " + decode(enc));
    }
}
```

**OUTPUT**

```
Enter the key
Monarchy
Enter the plain text
GIPSON
The key square
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z
Simulating Playfair Cipher
-----
Input Message : GIPSON
Encrypted Message : IKQTNA
Decrypted Message : GIPSON

...Program finished with exit code 0
Press ENTER to exit console.
```

**RESULT**

Thus the program for Playfair cipher encryption and decryption algorithm has been implemented and the output verified successfully.



**Ex. No: 1c****Date:****HILL CIPHER****AIM**

To implement a program to encrypt a message using the Hill cipher substitution technique

**ALGORITHM**

1. Input a 3 x 3 matrix as the key matrix.
2. Input a message to be encrypted.
3. Change all the characters in the message to uppercase and remove all the spacing.
4. Add padding if necessary to the text.
5. Arrange the plain text as a matrix with 3 columns; and convert the values to their equivalent ASCII values.
6. Compute the encrypted message by multiplying the plain text matrix by the key matrix:  
$$C = P \times K$$
7. Return the encrypted values.
8. Display the encoded cipher text.

**PROGRAM**

```
import java.util.Scanner;
class Main {
    public static int[][] keymat = new int[3][3];
    private static String encode(char a, char b, char c) {
        String ret = "";
        int x, y, z;
        int posa = (int) a - 65;
        int posb = (int) b - 65;
        int posc = (int) c - 65;
        x = posa * keymat[0][0] + posb * keymat[1][0] + posc * keymat[2][0];
        y = posa * keymat[0][1] + posb * keymat[1][1] + posc * keymat[2][1];
        z = posa * keymat[0][2] + posb * keymat[1][2] + posc * keymat[2][2];
        a = (char)(x % 26 + 65);
        b = (char)(y % 26 + 65);
        c = (char)(z % 26 + 65);
        ret = "" + a + b + c;
        return ret;
    }
}
```

```
public static void main(String[] args) {
    String msg;
    String enc = "";
    int n;
    System.out.println("Simulation of Hill Cipher\n -----");
    System.out.println("Enter the 3x3 key matrix(0 to 25)");
    Scanner scan=new Scanner(System.in);
    for(int i=0; i<3;i++){
        for(int j=0; j<3;j++){
            keymat[i][j]=scan.nextInt();
        }
    }
    System.out.println("Enter the plain text");
    msg=scan.next();
    msg+=scan.nextLine();//to get the entire line as input
    msg = msg.toUpperCase();
    msg = msg.replaceAll("\\s", ""); // remove spaces
    n = msg.length() % 3;
    if (n != 0) { // append padding text X
        for (int i = 1; i <= (3 - n); i++) {
            msg += 'X';
        }
    }
    System.out.println("Padded plain text: " + msg);
    char[] plaintext = msg.toCharArray();
    for (int i = 0; i < msg.length(); i += 3) {
        enc += encode(plaintext[i], plaintext[i + 1], plaintext[i + 2]);
    }
    System.out.println("Encoded cipher text: " + enc);
}
```

**OUTPUT**

```
Simulation of Hill Cipher
-----
Enter the 3x3 key matrix(0 to 25)
1
2
3
4
5
6
7
8
9
Enter the plain text
Gipson
Padded plain text: GIPSON
Encoded cipher text: NQTJCV

...Program finished with exit code 0
Press ENTER to exit console.
```

**RESULT**

Thus the program for Hill cipher has been implemented and the output verified successfully.

**Ex. No: 1d****Date:****VIGENERE CIPHER****AIM**

To implement a program for encryption and decryption using Vigenere cipher substitution technique

**ALGORITHM**

1. Input the key for Vigenere cipher.
2. Change the key to uppercase.
3. Input the plain text for encryption.
4. Change the plain text to uppercase.
5. Process each character of the plain text for encryption:
  - a. Skip if there are any non-alphabetic characters.
  - b. For all characters perform:
$$c_{text} += (\text{char}) ((c + \text{key.charAt}(j) - 2 * 'A') \% 26 + 'A')$$
  - c. Return the cipher text and display it.
6. Process each character of the plain text for decryption:
  - a. Skip if there are any non-alphabetic characters.
  - b. For all characters perform:
$$p_{text} += (\text{char}) ((c - \text{key.charAt}(j) + 26) \% 26 + 'A')$$
  - c. Return the plain text and display it.

**PROGRAM**

```
import java.util.Scanner;
public class Main {

    static String encode(String ptext, String key) {
        String ctext = "";
        ptext = ptext.toUpperCase();
        for (int i = 0, j = 0; i < ptext.length(); i++) {
            char c = ptext.charAt(i);
            if (c < 'A' || c > 'Z') {
                continue;
            }
            ctext += (\text{char}) ((c + \text{key.charAt}(j) - 2 * 'A') \% 26 + 'A');
            j = ++j \% \text{key.length()};
        }
    }
}
```

```
        return ctext;
    }

    static String decode(String ctext, String key) {
        String ptext = "";
        ctext = ctext.toUpperCase();
        for (int i = 0, j = 0; i < ctext.length(); i++) {
            char c = ctext.charAt(i);
            if (c < 'A' || c > 'Z') {
                continue;
            }
            ptext += (char) ((c - key.charAt(j) + 26) % 26 + 'A');
            j = ++j % key.length();
        }
        return ptext;
    }

    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        System.out.println("Simulating Vigenere Cipher\n----- ");
        System.out.println("Enter the key");
        String key = scan.next();
        key=key.toUpperCase();// Change the key to uppercase
        System.out.println("Enter the plain text");
        String msg = scan.next();
        msg+=scan.nextLine();// To get the entire line as input
        String cipher = encode(msg, key);// Encryption
        System.out.println("Encrypted cipher text\n" + cipher);
        String plain = decode(cipher, key);// Decryption
        System.out.println("Decrypted plain text\n" + plain);
    }
}
```

**OUTPUT**

```
Simulating Vigenere Cipher
-----
Enter the key
COMPUTER
Enter the plain text
GIPSON
Encrypted cipher text
IWBHIG
Decrypted plain text
GIPSON

...Program finished with exit code 0
Press ENTER to exit console.█
```

**RESULT**

Thus the program for Vigenere cipher encryption and decryption algorithm has been implemented and the output verified successfully.

**Ex. No: 2a****Date:****RAILFENCE CIPHER****AIM**

To implement a program for encryption and decryption using Railfence transposition technique.

**ALGORITHM**

1. Input the plain text to be encrypted using Railfence cipher.
2. For encryption, process the message till the length of message:
  - a. Start at index 0 and append each character into a new string by incrementing the index by 2.
  - b. Restart the index to 1 and again append each character by incrementing the index by 2.
  - c. Return and display the encrypted message.
3. For decryption
  - a. Divide the encrypted message into two halves.
  - b. Alternate between the first half and the last half of the message.
  - c. Append characters to a new string.
  - d. Return and display the decrypted message.

**PROGRAM**

```
import java.util.Scanner;
class railfenceCipher {
    String encode(String msg) {
        int len = msg.length();
        int k = 0, i = 0;
        String enc = "";
        while(k!=len){
            enc += msg.charAt(i);
            i+=2;
            if(i>=len){
                i=1;
            }
            k++;
        }
        return enc;
    }
    String decode(String encmsg) {
```

```
        int len = encmsg.length();
        int k = 0,i = 0;
        int j = (int)Math.ceil((double)len/2);
        String dec = "";
        while(k!=len){
            if(k%2==0){ // first half of encrypted message
                dec+=encmsg.charAt(i);
                i++;
            }
            else{// last half of encrypted message
                dec+=encmsg.charAt(j);
                j++;
            }
            k++;
        }
        return dec;
    }
}

class Main {
    public static void main(String[] args) {
        railfenceCipher rf = new railfenceCipher();
        String msg, enc, dec;
        Scanner scan=new Scanner(System.in);
        System.out.println("Simulating Railfence Cipher\n-----");
        System.out.print("Enter the plain text: ");
        msg = scan.next();
        msg+=scan.nextLine();
        enc = rf.encode(msg);
        dec = rf.decode(enc);
        System.out.println("Encrypted Message : " + enc);
        System.out.printf("Decrypted Message : " + dec);
    }
}
```



**OUTPUT**

```
Simulating Railfence Cipher
-----
Enter the plain text: JAVA IS OBJECT ORIENTED PROGRAMMING LANGUAGE
Encrypted Message : JV SOJC RETDPORMIGLNUGAAI BETOINE RGAMN AGAE
Decrypted Message : JAVA IS OBJECT ORIENTED PROGRAMMING LANGUAGE

...Program finished with exit code 0
Press ENTER to exit console.█
```

**RESULT**

Thus the Java program for Railfence Transposition Technique has been implemented and the output verified successfully.

**Ex. No: 2b****Date:****ROW AND COLUMN TRANSFORMATION****AIM**

To implement a program for encryption and decryption by using row and column transformation technique.

**ALGORITHM**

1. Input the plain text to be encrypted using row and column transformation cipher.
2. Input the number of columns in which the plain text is to be divided.
3. Remove all blank spaces present in the plain text using replaceAll() function.
4. Calculate the number of rows by dividing the length of the plain text by the number of columns.
5. Form a two-dimensional character matrix with the number of rows and columns values and store the values of the plain text row-wise.
6. Use 'X' as filler character for excess spaces available in the matrix.
7. Display the row and column matrix by traversing the matrix row-wise.
8. Display the cipher text by traversing the matrix column-wise.

**PROGRAM**

```
import java.util.*;
public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("SIMULATING ROW & COLUMN TRANSPOSITION CIPHER");
        System.out.println("-----");
        System.out.print("Enter the plain text: ");
        String pl = sc.next();
        pl += sc.nextLine();
        System.out.print("Enter the number of columns: ");
        int col = sc.nextInt();
        String s = "";
        s = pl.replaceAll(" ", "");
        int len = s.length();
        int k = 0;
        int row = (int) Math.ceil((double) len / col);
        char ch[][] = new char[row][col];
        for (int i = 0; i < row; i++) {
```

```
        for (int j = 0; j < col; j++) {
            if (k < len) {
                ch[i][j] = s.charAt(k);
                k++;
            }
            else {
                ch[i][j] = 'X';
            }
        }
    }
    System.out.println("The row and column matrix");
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            System.out.print(ch[i][j] + " ");
        }
        System.out.println();
    }
    System.out.print("The cipher text is: ");
    for (int i = 0; i < col; i++) {
        for (int j = 0; j < row; j++) {
            System.out.print(ch[j][i]);
        }
    }
}
```

**OUTPUT**

```
SIMULATING ROW AND COLUMN TRANSPOSITION CIPHER
-----
Enter the plain text: GIPSON RAHUL J M
Enter the number of columns: 4
The row and column matrix
G I P S
O N R A
H U L J
M X X X
The cipher text is: GOHMINUXPRLXSAJX

...Program finished with exit code 0
Press ENTER to exit console.
```

**RESULT**

Thus the java program for Row and Column Transposition Technique has been implemented and the output verified successfully.

**Ex. No: 3****Date:****DATA ENCRYPTION STANDARD (DES)****AIM**

To use Data Encryption Standard (DES) Algorithm for a practical application like User Message Encryption.

**ALGORITHM**

1. Generate key using KeyGenerator.
2. The KeyGenerator class provides getInstance() method which accepts a String variable representing the required key-generating algorithm and returns a KeyGenerator object that generates secret keys.  
`KeyGenerator keygenerator = KeyGenerator.getInstance("DES");`  
`SecretKey myDesKey = keygenerator.generateKey();`
3. Create a Cipher object by calling the Cipher's getInstance method, and pass the name of the requested transformation to it.  
`desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");`
4. Initialize a cipher by calling init().  
`desCipher.init(Cipher.ENCRYPT_MODE, myDesKey);`
5. Input the plain text to be encrypted.
6. Convert the plain text to bytes using getBytes().
7. Call the doFinal() method with the data to encrypt or decrypt.
8. Display the encrypted and decrypted form of plain text.

**PROGRAM**

```
import java.util.Scanner;
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 Main
{
    public static void main(String[] argv) {
        try{
            System.out.println("Message Encryption Using DES Algorithm\n----- ");
            KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
```

```
SecretKey myDesKey = keygenerator.generateKey();
Cipher desCipher;
desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
desCipher.init(Cipher.ENCRYPT_MODE, myDesKey);
Scanner scan = new Scanner(System.in);
System.out.print("Enter the plain text: ");
String ptext = scan.next();
ptext+= scan.nextLine();
byte[] text = ptext.getBytes();
System.out.println("Plain text [Byte Format] : " + text);
byte[] textEncrypted = desCipher.doFinal(text);
System.out.println("Encrypted plain text: " + textEncrypted);
desCipher.init(Cipher.DECRYPT_MODE, myDesKey);
byte[] textDecrypted = desCipher.doFinal(textEncrypted); // Encrypt or decrypt data with a
                                                         Cipher instance
System.out.println("Decrypted cipher text: " + new String(textDecrypted));
}
catch(Exception e){
    e.printStackTrace();
}
}
}
```

**OUTPUT**

```
Message Encryption Using DES Algorithm
-----
Enter the plain text: GIPSON RAHUL J M
Plain text [Byte Format] : [B@47fd17e3
Encrypted plain text: [B@7cdb5d3
Decrypted cipher text: GIPSON RAHUL J M

...Program finished with exit code 0
Press ENTER to exit console.□
```

**RESULT**

Thus the Java program for DES Algorithm has been implemented and the output verified successfully.

**Ex. No: 4****Date:****ADVANCED ENCRYPTION STANDARD (AES)****AIM**

To use Advanced Encryption Standard (AES) Algorithm for a practical application like message Encryption.

**ALGORITHM**

1. The cipher class of the javax.crypto package is used to encrypt the given data.
2. Input the secret key and plaintext for encryption.
3. The setKey function is used to generate the keys that are used to encrypt and decrypt the data.
4. In the encrypt function the plaintext is converted to cipher text.
5. Create a Cipher object by calling the Cipher's getInstance method, and pass the name of the requested transformation to it.
6. Initialize a cipher by calling init() in ENCRYPT mode.
7. Encode the byte data by using Base64.getEncoder().
8. In the decrypt function the cipher text is converted to plain text.
9. Create a Cipher object by calling the Cipher's getInstance method, and pass the name of the requested transformation to it.
10. Initialize a cipher by calling init() in DECRYPT mode.
11. Decode the byte data by using Base64.getDecoder().
12. Display the Encrypted Cipher text and Decrypted Plain text.

**PROGRAM**

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
import java.util.Scanner;

public class Main {
    private static SecretKeySpec secretKey;
    private static byte[] key;
    public static void setKey(String myKey) {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
```



```
        sha = MessageDigest.getInstance("SHA-1");
        key = sha.digest(key);
        key = Arrays.copyOf(key, 16);
        secretKey = new SecretKeySpec(key, "AES");
    }
    catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    }
    catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

public static String encrypt(String strToEncrypt, String secret) {
    try {
        setKey(secret);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
    }
    catch (Exception e) {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return null;
}

public static String decrypt(String strToDecrypt, String secret) {
    try {
        setKey(secret);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
    }
    catch (Exception e) {
        System.out.println("Error while decrypting: " + e.toString());
    }
    return null;
}

public static void main(String[] args) {
```

```
Scanner scan = new Scanner(System.in);
System.out.println("Encryption Using AES Algorithm\n ----- ");
System.out.print("Enter the secret key: ");
String secretKey = scan.next();
secretKey += scan.nextLine();
System.out.print("Enter the plain text: ");
String originalString = scan.next();
originalString += scan.nextLine();
String encryptedString = encrypt(originalString, secretKey);
String decryptedString = decrypt(encryptedString, secretKey);
System.out.println("Original plain text: " + originalString);
System.out.println("Encrypted cipher text: " + encryptedString);
System.out.println("Decrypted plain text: " + decryptedString);
}
}
```

**OUTPUT**

```
Encryption Using AES Algorithm
-----
Enter the secret key: FUTURE
Enter the plain text: COMPUTERS ARE OUR FUTURE
Original plain text: COMPUTERS ARE OUR FUTURE
Encrypted cipher text: crWS+qEpAD3ghT/VqdSTvjfy9wJlFfKEwN/m7gmhluw=
Decrypted plain text: COMPUTERS ARE OUR FUTURE

...Program finished with exit code 0
Press ENTER to exit console.
```

**RESULT**

Thus the Java program for AES Algorithm has been implemented for message Encryption and the output verified successfully.

**Ex. No: 5****Date:****RSA ALGORITHM****AIM**

To implement RSA (Rivest-Shamir-Adleman) algorithm by using HTML and Javascript.

**ALGORITHM**

1. Input two prime number p and q.
2. Input the plain text to be encrypted using RSA algorithm.
3. Check whether p and q are prime numbers or not.
4. Compute the value of n and t as,  
$$n = p * q$$
$$t = (p-1) * (q-1)$$
5. Find the value of e from t value.
6. Compute the value of d using gcd() function.
7. Encryption is given as,  
$$C = M^e \pmod n$$
8. Display (e,n) as public key.
9. Display (d,n) as private key.
10. Display C as cipher text for the given input message.

**PROGRAM**

```
<html>
<head>
<title>RSA Encryption</title>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
<center>
<h1>RSA Algorithm</h1>
<h2>Implemented Using HTML & Javascript</h2>
<hr>
<table>
<tr>
<td>Enter First Prime Number (p):</td>
<td><input type="number" id="p"></td>
</tr>
<tr>
```

```

<td>Enter Second Prime Number (q):</td>
<td><input type="number" id="q"></p> </td>
</tr>
<tr>
<td>Enter the Message(cipher text):<br>[A=1, B=2,...]</td>
<td><input type="number" id="msg"></p> </td>
</tr>
<tr>
<td>Public Key (e,n):</td>
<td><p id="publickey"></p> </td>
</tr>
<tr>
<td>Private Key (d,n):</td>
<td><p id="privatekey"></p> </td>
</tr>
<tr>
<td>Cipher Text:</td>
<td>
<p id="ciphertext"></p>
</td>
</tr>
<tr>
<td><button onclick="RSA();">Apply RSA</button></td>
</tr>
</table>
</center>
</body>
<script type="text/javascript">
function checkPrime(n) {
var i, flag = true;
n = parseInt(n)
for(i = 2; i <= n - 1; i++)
if (n % i == 0) {
flag = false;
break;
}
return flag;

```

```
}  
function RSA() {  
var gcd, p, q, M, n, t, e, i, x, C, d, dt;  
gcd = function (a, b) { return (!b) ? a : gcd(b, a % b); };  
p = document.getElementById('p').value;  
q = document.getElementById('q').value;  
if(!(checkPrime(p) && checkPrime(q))){  
alert("p & q should always be prime numbers");  
return;  
}  
M = document.getElementById('msg').value;  
n = p * q;  
t = (p - 1) * (q - 1);  
for (e = 2; e < t; e++) {  
if (gcd(e, t) == 1) {  
break;  
}}  
i=0;  
while(1)  
{  
x = 1 + i * t;  
if (x % e == 0) {  
d = x / e;  
break;  
}  
i++;  
}  
C = (Math.pow(M, e).toFixed(0)) % n;  
document.getElementById('publickey').innerHTML = e + "," + n;  
document.getElementById('privatekey').innerHTML = d + "," + n;  
document.getElementById('ciphertext').innerHTML = C;  
}  
</script>  
</html>
```

**OUTPUT**

**RSA Algorithm**  
**Implemented Using HTML & Javascript**

---

Enter First Prime Number (p):   
Enter Second Prime Number (q):   
Enter the Message(cipher text):   
[A=1, B=2,...]  
Public Key (e,n): 7,77  
Private Key (d,n): 43,77  
Cipher Text: 10

---

**RESULT**

Thus the RSA algorithm has been implemented using HTML & CSS and the output has been verified successfully.

**Ex. No: 6****Date:****DIFFIE-HELLMAN KEY EXCHANGE****AIM**

To implement the Diffie-Hellman Key Exchange algorithm for a given problem.

**ALGORITHM**

1. Select the global public element  $q$ ; which is a prime number. Here we have set the value of  $q$  as 23.
2. Select the global public element  $a$ ; where  $a < q$  and  $a$  is a primitive root of  $q$ .
3. Input the private key of User A as  $X_a$ ; where  $X_a < q$ .
4. Input the private key of User B as  $X_b$ ; where  $X_b < q$ .
5. Calculate the public key of User A  

$$Y_a = (\text{Math.pow}(a, X_a)) \% q$$
6. Calculate the public key of User B  

$$Y_b = (\text{Math.pow}(a, X_b)) \% q$$
7. Calculate the secret key by user A  

$$K_a = (\text{Math.pow}(Y_b, X_a)) \% q$$
8. Calculate the secret key by user B  

$$K_b = (\text{Math.pow}(Y_a, X_b)) \% q$$
9. Check if  $K_a = K_b$ 
  - a. Display Success: Shared key matches! if they are equal
  - b. Display Error: Shared key does not match if they are not equal.

**PROGRAM**

```
import java.util.Scanner;
class Main {
    public static void main(String args[]) {
        int q = 23; /* Global public element prime number q*/
        int a = 5; /* Global public element a which is a primitive root of q*/
        System.out.println("Simulation of Diffie-Hellman key exchange algorithm\n-----
        -----");
        System.out.println("Global public element prime number q = " + q);
        System.out.println("Global public element a = " + a);
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter the private key of user A ( $X_a < q$  Here the value of q is 23): ");
        int Xa = scan.nextInt(); /* Private key of User A */
        System.out.print("Enter the private key of user B ( $X_b < q$  Here the value of q is 23): ");
```



```
int Xb = scan.nextInt(); /* Private key of User A */
double Ya = (Math.pow(a, Xa)) % q;
double Kb = (Math.pow(Ya, Xb)) % q;
double Yb = (Math.pow(a, Xb)) % q;
double Ka = (Math.pow(Yb, Xa)) % q;
System.out.println("Public key of User A (Ya): " + Ya);
System.out.println("User B computes signature using public key of user A (Kb): " + Kb);
System.out.println("Public key of User B (Yb): " + Yb);
System.out.println("User A computes signature using public key of user B (Ka): " + Ka);
if (Ka == Kb)
System.out.println("Success: Shared key matches! " + Ka);
else
System.out.println("Error: Shared key does not match");
}
}
```

**OUTPUT**

```
Simulation of Diffie-Hellman key exchange algorithm
-----
Global public element prime number q = 23
Global public element a = 5
Enter the private key of user A (<q=23): 15
Enter the private key of user B (<q=23): 03
Public key of User A (Ya): 19.0
User B computes signature using public key of user A (Kb): 5.0
Public key of User B (Yb): 10.0
User A computes signature using public key of user B (Ka): 5.0
Success: Shared key matches! 5.0

...Program finished with exit code 0
Press ENTER to exit console.█
```

**RESULT**

Thus the Diffie-Hellman key exchange algorithm has been implemented using Java and the output has been verified successfully.

**Ex. No: 7****Date:****SECURE HASH ALGORITHM (SHA-1)****AIM**

To Calculate the message digest of a text using the SHA-1 algorithm.

**ALGORITHM**

1. Generate MessageDigest object using MessageDigest.getInstance() function.
2. The MessageDigest class provides getInstance() method which accepts a String variable representing the required hashing algorithm and returns a MessageDigest object that generates the message digest(hash).  
`MessageDigest md = MessageDigest.getInstance('SHA1');`
3. Display the Algorithm, Provider using getAlgorithm() and getProvider() functions respectively.
4. Get the message to be hashed from the user as input.
5. Convert the message to bytes using getBytes() and pass it as parameter to the update() function of the MessageDigest object.
6. Call the digest() method to produce the message digest(hash) and store the output as byte array.
7. Display the hashed message by using the bytesToHex() function.

**PROGRAM**

```
import java.security.*;
import java.util.Scanner;
public class Main {
    public static void main(String[] a) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA1");
            System.out.println("Message digest object info:\n----- ");
            System.out.println("Algorithm=" + md.getAlgorithm());
            System.out.println("Provider=" + md.getProvider());
            System.out.println("ToString=" + md.toString());
            Scanner scan = new Scanner(System.in);
            System.out.print("Enter the message: ");
            String input = "scan.next()";
            input += scan.nextLine();
            md.update(input.getBytes());
            byte[] output = md.digest();
```

```
System.out.println("SHA1(\"" + input + "\")=" + bytesToHex(output));
} catch (Exception e) {
System.out.println("Exception:" + e);
}
}

private static String bytesToHex(byte[] b) {
char hexDigit[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
StringBuffer buf = new StringBuffer();
for (byte aB : b) {
buf.append(hexDigit[(aB >> 4) & 0x0f]);
buf.append(hexDigit[aB & 0x0f]);
}
return buf.toString();
}
}
```

**OUTPUT**

```
Message digest object info:
-----
Algorithm=SHA1
Provider=SUN version 1.8
ToString=SHA1 Message Digest from SUN, <initialized>

Enter the message: Java is a class-based, object-oriented programming languages
SHA1("scan.next()Java is a class-based, object-oriented programming languages")=D0E5E6025CC1151C2B9A9724D6189BA413ED6CF7

...Program finished with exit code 0
Press ENTER to exit console.█
```

**RESULT**

Thus the Secure Hash Algorithm (SHA-1) has been implemented and the output has been verified successfully.

**Ex. No: 8****Date:****DIGITAL SIGNATURE STANDARD (DSS)****AIM**

To implement the Digital signature scheme - Digital Signature Standard.

**ALGORITHM**

1. Input the message to be signed from the user.
2. Generate key pair using KeyPairGenerator.
3. The KeyPairGenerator class provides getInstance() method which accepts a String variable representing the required key pair-generating algorithm and returns a KeyPairGenerator object that generates the key pair.  
`KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance('DSA');`
4. Initialize the KeyPairGenerator object with the number of bytes using the initialize() function.
5. Get the KeyPair object from the KeyPairGenerator object using generateKeyPair() function.
6. Get the PrivateKey object from the KeyPair object using getPrivate() function.
7. Create a Signature object by calling the Signature's getInstance method, and pass the name of the requested algorithm to it.  
`Signature sign = Signature.getInstance('SHA256withDSA');`
8. Initialize the sign by calling initSign() function.  
`sign.initSign(privKey);`
9. Convert the message to bytes using getBytes().
10. Call the update() method with the data to sign.
11. Call the sign() method to get the digital signature in bytes.
12. Display the digital signature by converting it to String.

**PROGRAM**

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.Signature;
import java.util.Scanner;

public class Main {
    public static void main(String args[]) throws Exception {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the message to be signed");
        String msg = sc.next();
```

```
msg += sc.nextLine();
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("DSA");
keyPairGen.initialize(2048);
KeyPair pair = keyPairGen.generateKeyPair();
PrivateKey privKey = pair.getPrivate();
Signature sign = Signature.getInstance("SHA256withDSA");
sign.initSign(privKey);
byte[] bytes = msg.getBytes();
sign.update(bytes);
byte[] signature = sign.sign();
System.out.println("Digital signature for the given message: "+new String(signature,"UTF8"));
}
}
```

**OUTPUT**

```
Enter the message to be signed
C IS A PROCEDURAL COMPUTER PROGRAMMING LANGUAGE
Digital signature for the given message: 0=S0iA)R.c]H;u只OO]GGNm?
7
g

...Program finished with exit code 0
Press ENTER to exit console.█
```

**RESULT**

Thus the Digital Signature Standard Signature Scheme has been implemented and the output has been verified successfully.



**Ex. No: 9****Date:****DEMONSTRATION OF INTRUSION DETECTION SYSTEM(IDS)****AIM**

To demonstrate Intrusion Detection System (IDS) using Snort software tool.

**PROCEDURE****Step 1**

Goto <https://www.snort.org/downloads> to download Snort.

**Step 2**

Under Binaries, Click the link depending on the type of OS

Snort\_2\_9\_16\_1\_Installer.x86.exe (Windows 32 bit)

Snort\_2\_9\_16\_1\_Installer.x64.exe (Windows 64 bit)

**Step 3**

Run the downloaded .exe file to install Snort in your system.

Note: Choose location as C:/Snort (default location)

**Step 4**

In <https://www.snort.org/downloads>, under rules, signin (signup and also confirm email address) and then download

snortrules-snapshot-29161.tar.gz

and extract the same using Winrar.

Note: If Winrar is not installed. Goto <https://www.win-rar.com/download.html> and install the same.

**Step 5**

Copy all files from the 'rules' folder of the extracted folder and paste the rules into 'C:/Snort/rules' folder.

**Step 6**

Copy 'snort.conf' file from the 'etc' folder of the extracted folder and paste it into 'C:/Snort/etc' folder. Overwrite any existing file.

**Step 7**

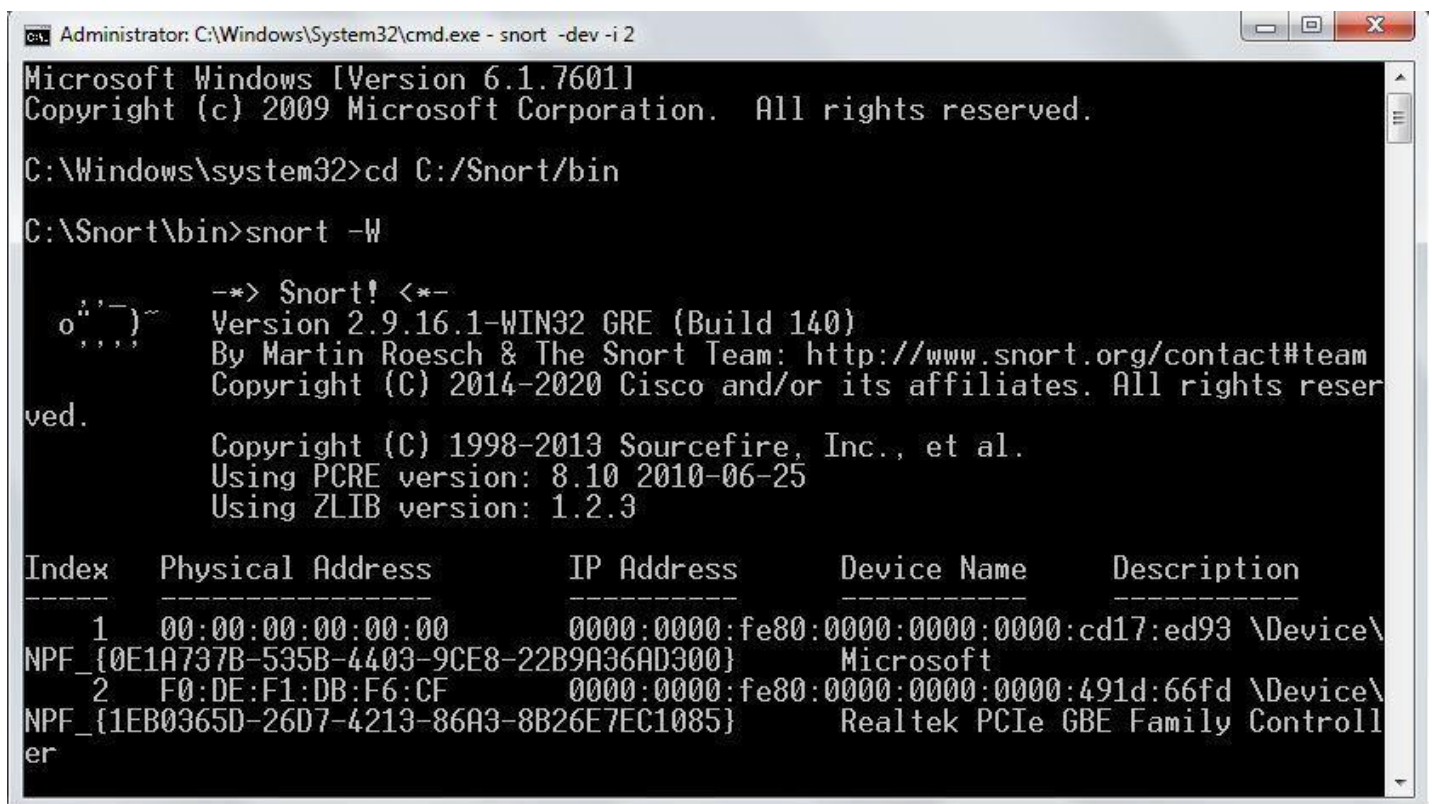
Goto <https://nmap.org/npcap/> and download Npcap 1.00 installer for Windows 7/2008R2, 8/2012, 8.1/2012R2, 10/2016, 2019 (x86 and x64) and install the same.

**Step 8**

Open command prompt as administrator(search 'cmd' in the start menu, right click and select run as administrator) and navigate to folder 'C:/Snort/bin' using the following command,  
cd C:/Snort/bin

**Step 9**

Run the following command to get the interface list,  
snort -W



```

Administrator: C:\Windows\System32\cmd.exe - snort -dev -i 2
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:/Snort/bin
C:\Snort\bin>snort -W

o''~  -*> Snort! <*-
o''~  Version 2.9.16.1-WIN32 GRE (Build 140)
      By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
      Copyright (C) 2014-2020 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      00:00:00:00:00:00      0000:0000:fe80:0000:0000:0000:cd17:ed93 \Device\
NPF_{0E1A737B-535B-4403-9CE8-22B9A36AD300} Microsoft
2      F0:DE:F1:DB:F6:CF      0000:0000:fe80:0000:0000:0000:491d:66fd \Device\
NPF_{1EB0365D-26D7-4213-86A3-8B26E7EC1085} Realtek PCIe GBE Family Controller
er
  
```

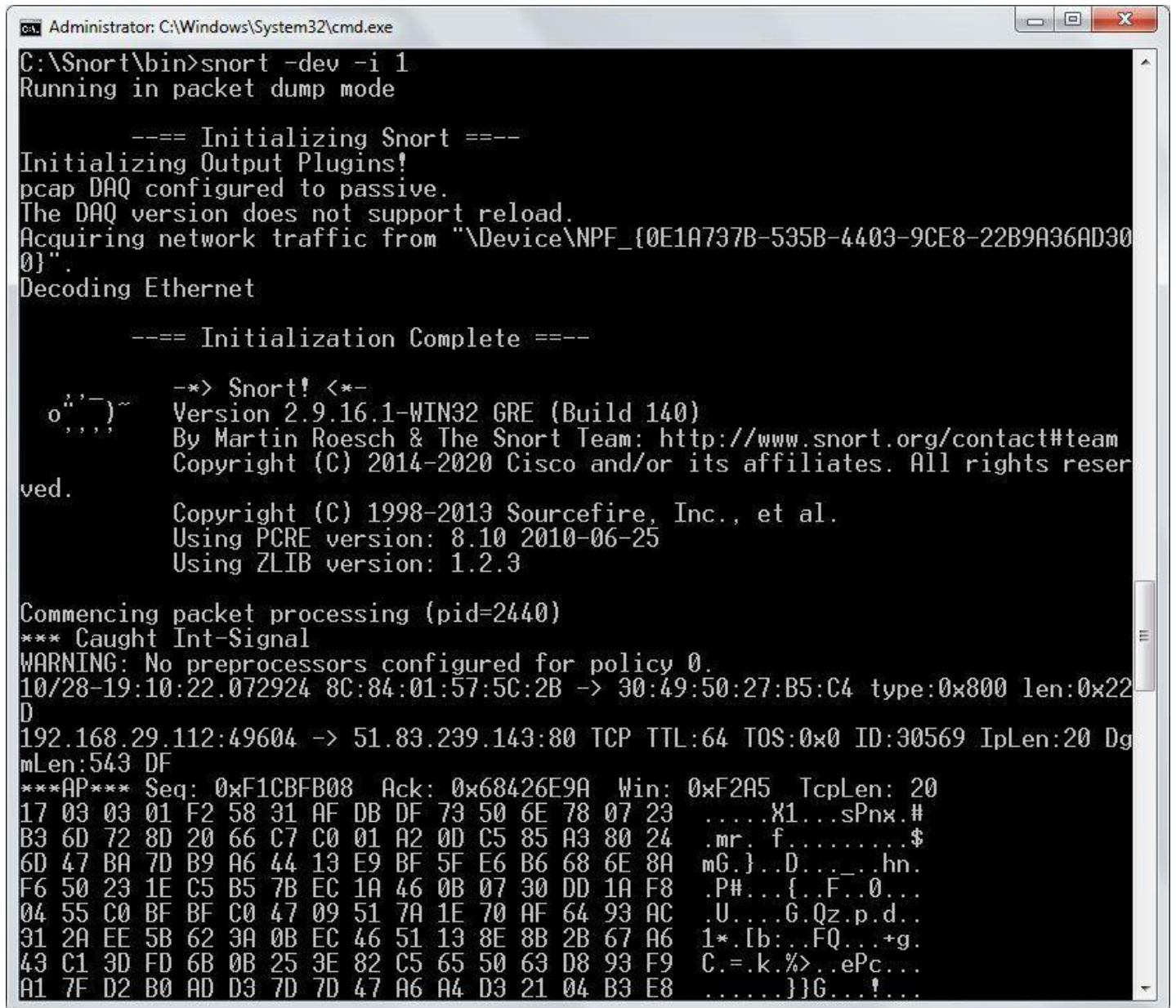
**Step 10**

To start Snort in sniffer mode use following command:

```
snort -dev -i 1
```

-i indicates the interface number.

Note: Use Ctrl+C to stop the execution



```

Administrator: C:\Windows\System32\cmd.exe
C:\Snort\bin>snort -dev -i 1
Running in packet dump mode

--== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to passive.
The DAQ version does not support reload.
Acquiring network traffic from "\Device\NPF_{0E1A737B-535B-4403-9CE8-22B9A36AD300}"
Decoding Ethernet

--== Initialization Complete ==--

o^,~)~
',',',
-*> Snort! <*-
Version 2.9.16.1-WIN32 GRE (Build 140)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2020 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

Commencing packet processing (pid=2440)
*** Caught Int-Signal
WARNING: No preprocessors configured for policy 0.
10/28-19:10:22.072924 8C:84:01:57:5C:2B -> 30:49:50:27:B5:C4 type:0x800 len:0x22
D
192.168.29.112:49604 -> 51.83.239.143:80 TCP TTL:64 TOS:0x0 ID:30569 IpLen:20 Dg
mLen:543 DF
***AP*** Seq: 0xF1CBFB08 Ack: 0x68426E9A Win: 0xF2A5 TcpLen: 20
17 03 03 01 F2 58 31 AF DB DF 73 50 6E 78 07 23 .....X1...sPnx.#
B3 6D 72 8D 20 66 C7 C0 01 A2 0D C5 85 A3 80 24 .mr. f.....$
6D 47 BA 7D B9 A6 44 13 E9 BF 5F E6 B6 68 6E 8A mG.}..D..._.hn.
F6 50 23 1E C5 B5 7B EC 1A 46 0B 07 30 DD 1A F8 .P#...{..F..0...
04 55 C0 BF BF C0 47 09 51 7A 1E 70 AF 64 93 AC .U....G.Qz.p.d..
31 2A EE 5B 62 3A 0B EC 46 51 13 8E 8B 2B 67 A6 1*.[b:...FQ...+g.
43 C1 3D FD 6B 0B 25 3E 82 C5 65 50 63 D8 93 F9 C.=.k.%>...ePc...
A1 7F D2 B0 AD D3 7D 7D 47 A6 A4 D3 21 04 B3 E8 .....}}G...!...

```

```

Administrator: C:\Windows\System32\cmd.exe
=====
Run time for packet processing was 1.13000 seconds
Snort processed 1 packets.
Snort ran for 0 days 0 hours 0 minutes 1 seconds
Pkts/sec:          1
=====
Packet I/O Totals:
  Received:        13
  Analyzed:         1 ( 7.692%)
  Dropped:          0 ( 0.000%)
  Filtered:         0 ( 0.000%)
  Outstanding:     12 ( 92.308%)
  Injected:         0
=====
Breakdown by protocol (includes rebuilt packets):
  Eth:              1 (100.000%)
  VLAN:             0 ( 0.000%)
  IP4:              1 (100.000%)
  Frag:             0 ( 0.000%)
  ICMP:             0 ( 0.000%)
  UDP:              0 ( 0.000%)
  TCP:              1 (100.000%)
  IP6:              0 ( 0.000%)
  IP6 Ext:          0 ( 0.000%)
  IP6 Opts:         0 ( 0.000%)
  Frag6:            0 ( 0.000%)
  ICMP6:            0 ( 0.000%)
  UDP6:             0 ( 0.000%)
  TCP6:             0 ( 0.000%)
  Teredo:           0 ( 0.000%)
  ICMP-IP:          0 ( 0.000%)
  EAPOL:            0 ( 0.000%)
  IP4/IP4:          0 ( 0.000%)
  IP4/IP6:          0 ( 0.000%)
  IP6/IP4:          0 ( 0.000%)
  IP6/IP6:          0 ( 0.000%)
  GRE:              0 ( 0.000%)
  GRE Eth:          0 ( 0.000%)

```

## RESULT

Thus the Intrusion Detection System(IDS) has been demonstrated by using the Open Source Snort Intrusion Detection Tool.

**Ex. No: 10****Date:****EXPLORING N-STALKER, A VULNERABILITY ASSESSMENT TOOL****AIM**

To download the N-Stalker Vulnerability Assessment Tool and exploring the features.

**PROCEDURE****Step 1**

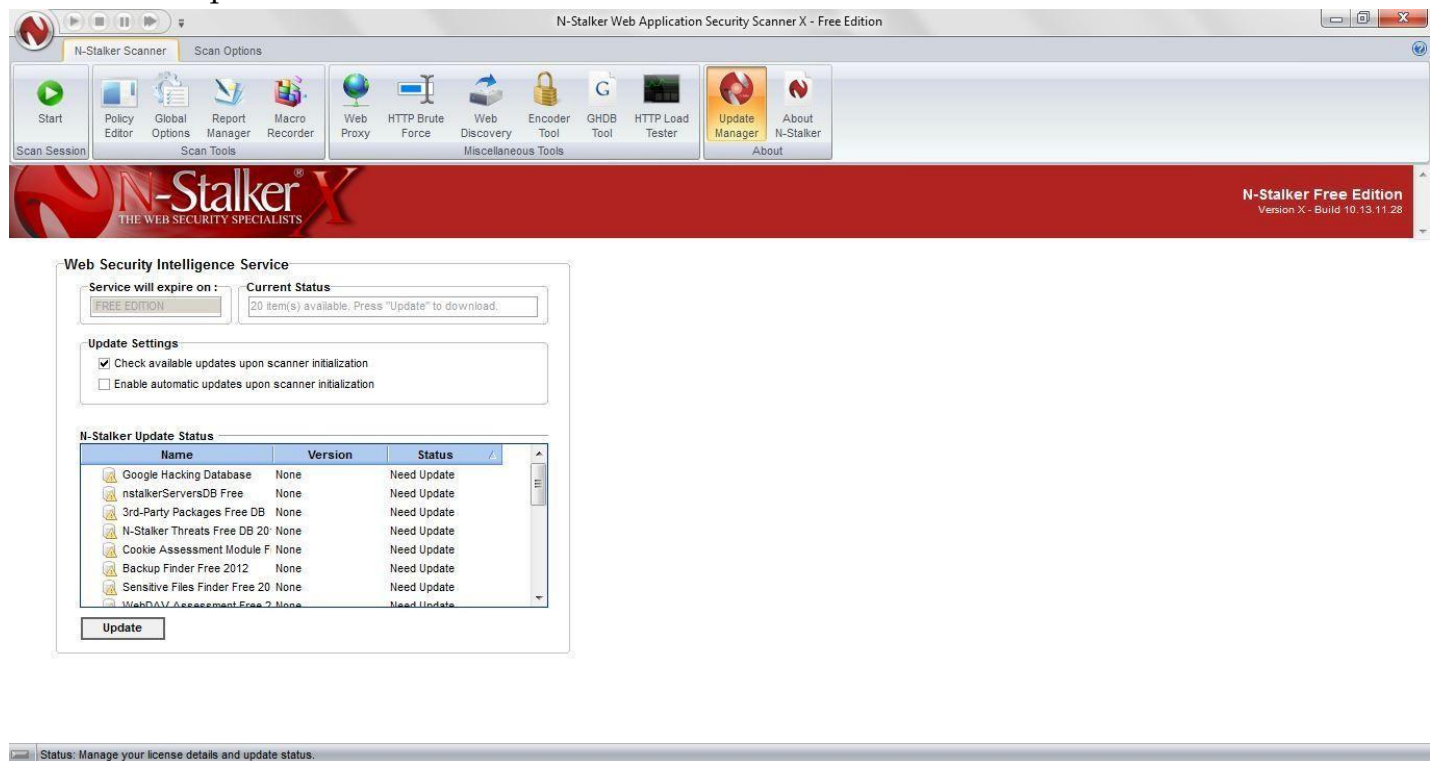
Goto [https://download.cnet.com/N-Stalker-Web-Application-Security-Scanner/3000-2653\\_4-10583954.html](https://download.cnet.com/N-Stalker-Web-Application-Security-Scanner/3000-2653_4-10583954.html) to download the N-Stalker Application and install the same.

**Step 2**

Run the N-Stalker Application and goto 'Update Manager' present in the 'About' Section of the 'N-Stalker Scanner' tab.

**Step 3**

Click on the Update button and wait for all the files to be downloaded.





**Step 4**

Once the download is complete, click on 'Start' in the 'Scan Session' under 'N-Stalker Scanner' tab. The N-Stalker Scan Wizard pops-up.

**Step 5**

Enter the URL that you want to test in the 'Enter Web Application URL' textbox.

Eg. [www.google.com](http://www.google.com)

The screenshot shows the 'N-Stalker Scan Wizard' window. The title bar says 'N-Stalker Scan Wizard'. The main heading is 'Start Web Application Security Scan Session' with a subtitle 'You must enter an URL and choose policy. Scan Settings may be configured.' The window is divided into a left sidebar and a main content area. The sidebar has four buttons: 'Choose URL & Policy' (highlighted), 'Optimize Settings', 'Review Summary', and 'Start Scan Session'. The main content area has several sections: 1. 'Enter Web Application URL' with a text box containing 'www.google.com' and a note '(E.g: http://www.example.tl/, https://www.test.tl/VirtualDirectory/, etc)'. Below this are two checkboxes: 'Scan both HTTP and HTTPS locations' (checked) and 'Do not test web authentication forms' (unchecked). 2. 'Choose Scan Policy' with a dropdown menu showing 'Manual Test (Crawl through the URL and standby for manual attack)'. 3. 'Load Scan Session' with a dropdown menu and a note '(You may load scan settings from previously saved scan sessions)'. 4. 'Load Spider Data' with a dropdown menu showing 'Not available in N-Stalker Free Edition' and a note '(You may load spider data from previously saved scan sessions)'. Below this is an unchecked checkbox 'Use local cache from previously saved session (Avoid new web crawling)'. At the bottom are three buttons: 'Scan Settings', 'Cancel', and 'Next >>'.

**Step 6**

Select 'Manual Test (Crawl through the URL and standby for manual attack)' in the 'Choose Scan Policy' dropdown list box and click Next.

**Step 47**

Click on the 'Optimize' button and wait for the Optimization Progress to reach 100%. After that, Click Next.

The screenshot shows the 'N-Stalker Scan Wizard' window, specifically the 'Optimize Settings' step. The window has a title bar with 'N-Stalker Scan Wizard' and a close button. The main area is titled 'Start Web Application Security Scan Session' with a subtitle 'You must enter an URL and choose policy. Scan Settings may be configured.' Below this, there's a section 'Optimizing Settings' with a text box containing 'http://www.google.com/'. A note below the text box says '(You may choose to run a series of tests to allow for optimization or click Next to continue)'. There are five tabs: 'Optimize Results' (selected), 'Authentication', 'False Positive', 'Engine', and 'Miscellaneous'. Below the tabs is a section 'Optimization Progress' with a blue progress bar at 100% and the status 'Status: Optimization successfully completed.' Below that is a section 'Optimization Results' with three fields: 'Xfer Rate' (168.95 KB/s), 'Avg Response' (1.17 ms), and 'Conn Failures' (0%). Below these fields are two bullet points: 'Only few URLs found. You should consider using a Web Macro script.' and 'To enhance speed, we have restricted Spider to 30 pages variations per no...'. A note at the bottom of this section says '(Right-click over an item to see suggested actions)'. At the bottom of the window are four buttons: 'Scan Settings', 'Optimize', '<< Back', and 'Next >>'.

N-Stalker Scan Wizard

**Start Web Application Security Scan Session**  
You must enter an URL and choose policy. Scan Settings may be configured.

**Optimizing Settings**

http://www.google.com/  
(You may choose to run a series of tests to allow for optimization or click Next to continue)

Optimize Results | Authentication | False Positive | Engine | Miscellaneous

**Optimization Progress**

100 %  
Status: Optimization successfully completed.

**Optimization Results**

Xfer Rate 168.95 KB/s | Avg Response 1.17 ms | Conn Failures 0%

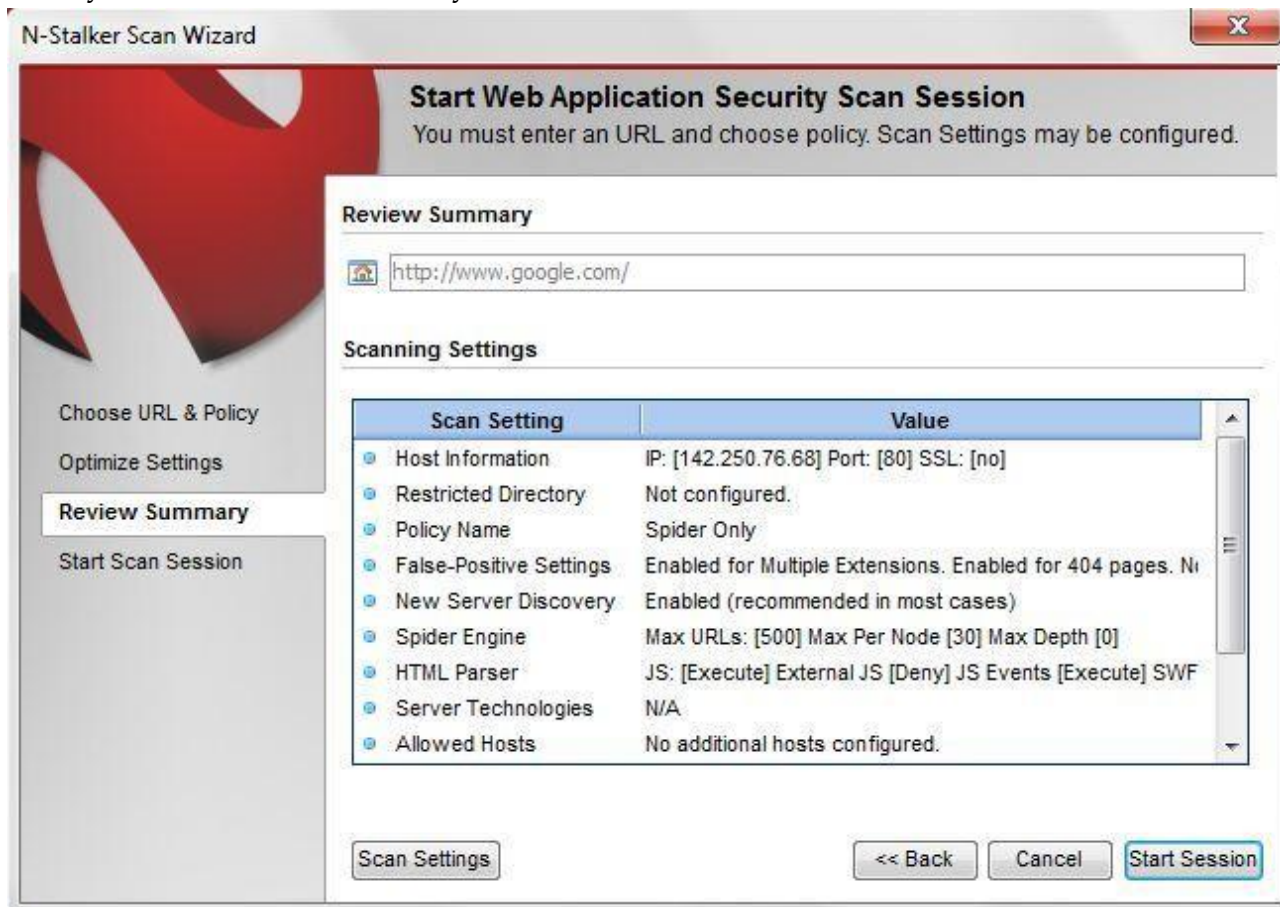
- Only few URLs found. You should consider using a Web Macro script.
- To enhance speed, we have restricted Spider to 30 pages variations per no...

(Right-click over an item to see suggested actions)

Scan Settings | Optimize | << Back | Next >>

**Step 48**

Finally, in the 'Review Summary' tab, Click on 'Start Session'.





**Step 49**

Click on 'Start Scan' in the 'Session Control' section under the 'Scan Options' tab, in order to initiate the scan.

The screenshot displays the N-Stalker Web Application Security Scanner X - Free Edition interface. The 'Scan Options' tab is active, showing various configuration sections like 'Session Control', 'Threads Control', 'Spider Control', 'HTTP Control', and 'False-Positive Control'. The 'Start Scan' button is visible in the 'Session Control' section. Below the configuration tabs, the 'Website Tree' on the left lists scanned URLs. The 'Scanner Dashboard' on the right provides a progress status, progress details, and a network statistics table.

**Progress Status:**

- Completed Spider
- Not Tested Info Gather
- Step 3 Run Modules
- Not Tested Sig Scanner

**Progress Details:**

Scan Session	Start Time	Duration
	Oct 28, 2020 15:52:47	0 Hours 51 Minutes

**Spider Engine:**

#	Crawled URLs	Crawled Hosts	Default Page Size
502	2	620,692 bytes	

**Scan Engine:**

#	Total Requests	Failed Requests	Attacks Sent	404 Errors	302 Redirection
2,041	0	4	182	147	

**Network Statistics:**

Network	#
Bytes Sent	1,318,236
Bytes Received	150,805,097
Avg Response Time	0.11 s
Avg Transfer Rate	5.60 Mb/s
Requests/Minute	40.00 req/min

**Scan Module Progress:**

Scan Module	Current	Total	Progress
N-Stalker Spider Module	500	500	100 %
File Extensions Finder	20	20	100 %
WebServer Infrastructure Ass	4	4	100 %
HTTP Method Finder	22	22	100 %

Status: Standby Mode - You may now run manual attacks or press "Close" to finish session.

## Step 10

Wait for the scan to complete and click the 'Hidden Fields' under 'Objects' in the 'Scanner Events' Section.

The screenshot displays the N-Stalker Web Application Security Scanner X - Free Edition interface. The 'Scanner Events' tab is active, showing a tree view on the left with 'Hidden Fields (330)' selected under 'Objects'. The main pane shows a table of hidden fields with columns: Source URL, Hidden Variable, and Value. Below this, a list of 8 hidden input fields is shown, each with a type of 'hidden', name of 'source', and value of 'hp'. At the bottom, a 'Scan Modules' table shows the progress of various modules.

Source URL	Hidden Variable	Value
https://www.google.com/?gws_rd=ssl	source	hp
https://www.google.com/?source=hp&ei=	source	hp
https://www.google.com/index.html?gws	source	hp
https://www.google.com/webhp?gws_r=	source	hp
https://www.google.com/shopping?gws	source	hp
https://www.google.com/shopping/seller	source	hp
https://www.google.com/?source=hp&ei=	source	hp
https://www.google.com/	source	hp
https://www.google.com/webhp?tab=wv	source	hp

Scan Module	Current	Total	Progress
N-Stalker Spider Module	500	500	100 %
File Extensions Finder	20	20	100 %
WebServer Infrastructure Ass	4	4	100 %
HTTP Method Finder	22	22	100 %

Status: Standby Mode - You may now run manual attacks or press "Close" to finish session.

## RESULT

Thus the N-Stalker Vulnerability Assessment tool has been downloaded, installed and the features has been explored by using a website.

**Ex. No: 11a****Date:****DEFEATING MALWARE - BUILDING TROJANS****AIM**

To build a Trojan and know the harm of the Trojan malwares in a computer system.

**PROCEDURE**

1. Create a simple Trojan by using Windows Batch File (.bat)
2. Type these below code in notepad and save it as Trojan.bat
3. Double click on Trojan.bat file.
4. When the Trojan code executes, it will open MS-Paint, Notepad, Command Prompt, Explorer, etc., infinitely.
5. Restart the computer to stop the execution of this Trojan.

**PROGRAM****Trojan.bat**

```
@echo off
```

```
:x
```

```
start mspaint
```

```
start notepad
```

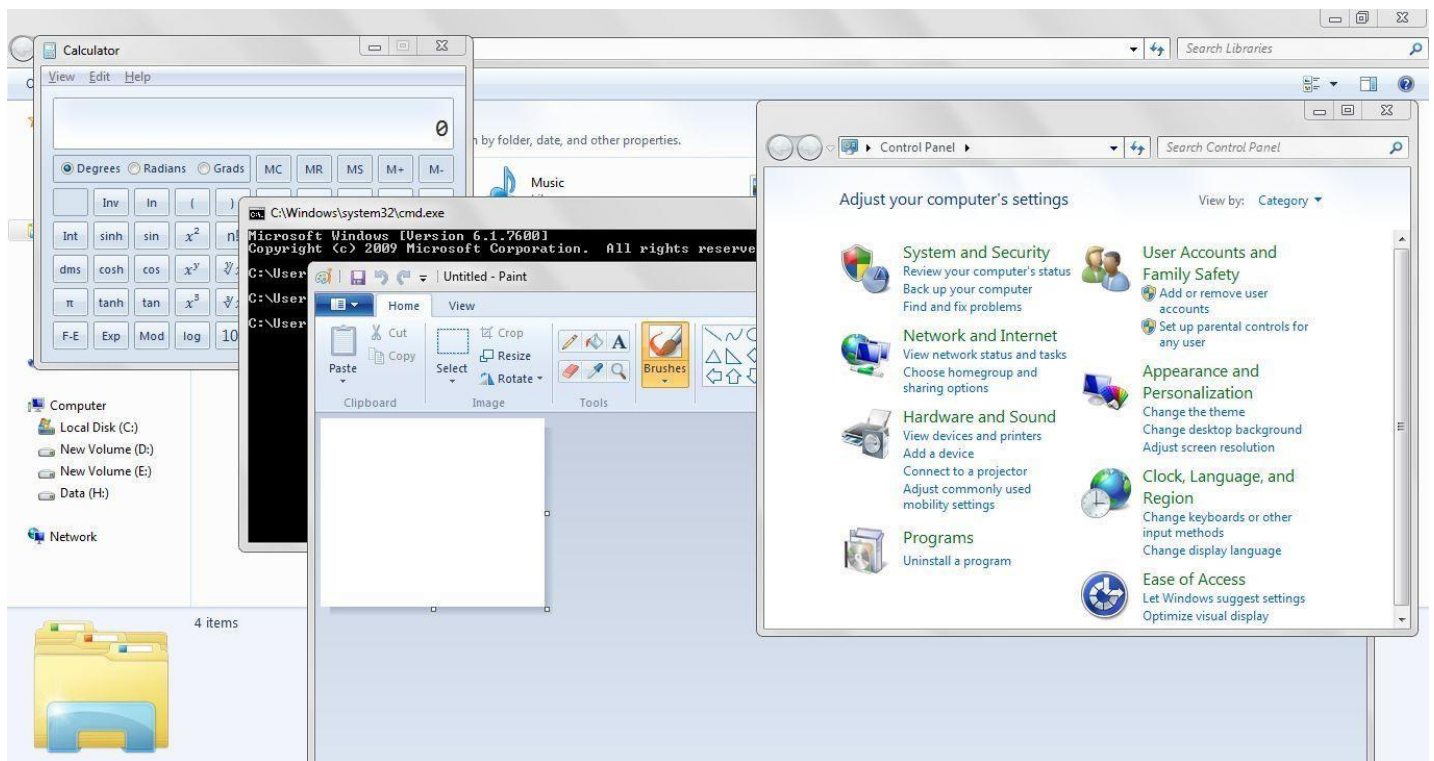
```
start cmd
```

```
start explorer
```

```
start control
```

```
start calc
```

```
goto x
```

**OUTPUT****RESULT**

Thus a Trojan has been built and the harm of the Trojan viruses has been explored.

Ex. No: 11b

Date:

## DEFEATING MALWARE - ROOTKIT HUNTER

### AIM

To install a rootkit hunter and find the malwares in a computer.

### PROCEDURE

#### Step 1

Goto <http://www.gmer.net/#files> and click Download EXE button to download GMER.

**GMER** <http://www.gmer.net>  
 all your rootkits are belong to us [\*]

**Start**

**Files**

**News**

**Rootkits**

**FAQ**

**Contact**

**Start**

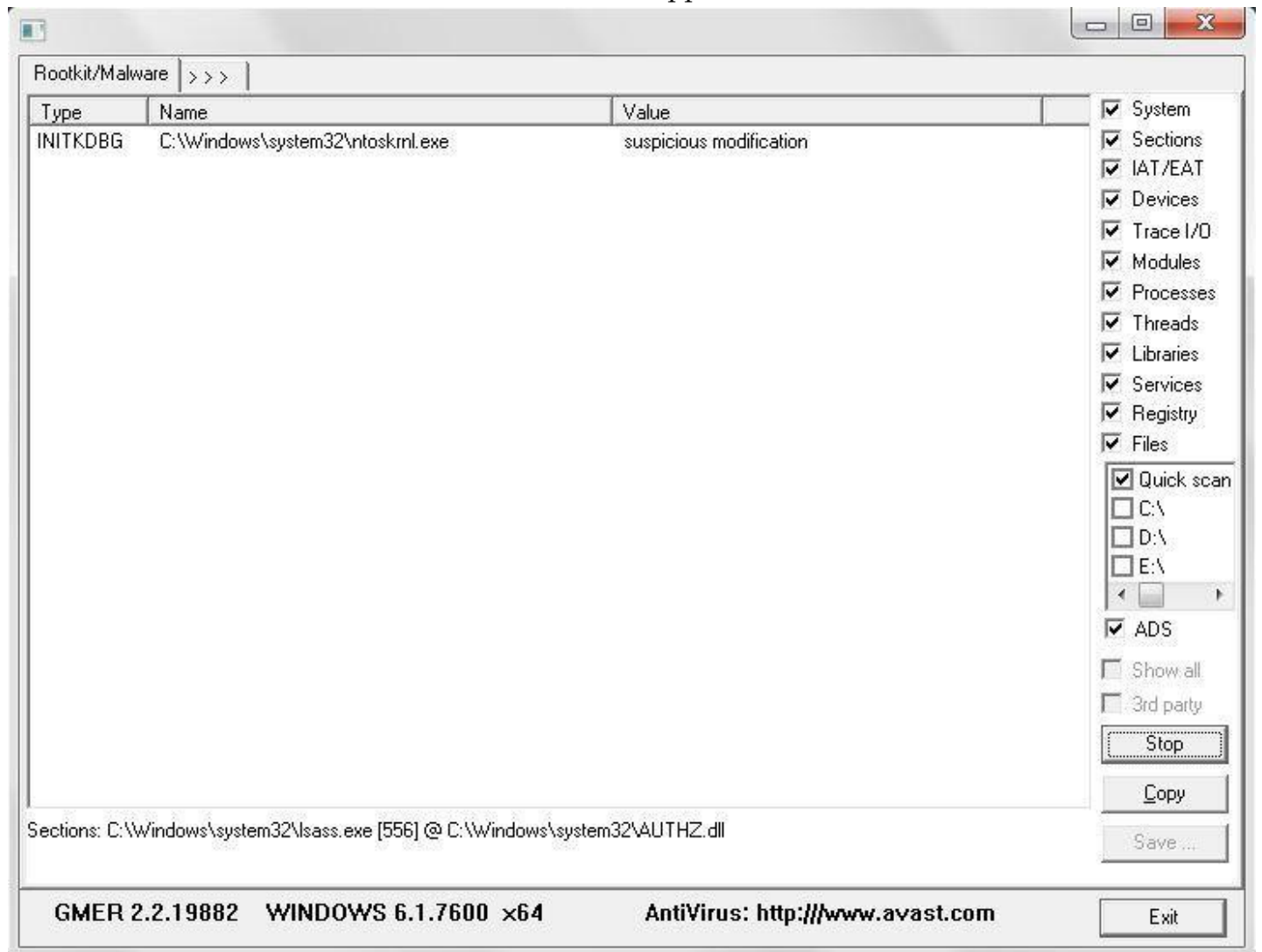
GMER is an application that detects and removes rootkits .

It scans for:

- hidden processes
- hidden threads
- hidden modules
- hidden services
- hidden files
- hidden disk sectors (MBR)
- hidden Alternate Data Streams
- hidden registry keys
- drivers hooking SSDT
- drivers hooking IDT
- drivers hooking IRP calls
- inline hooks

**Step 54**

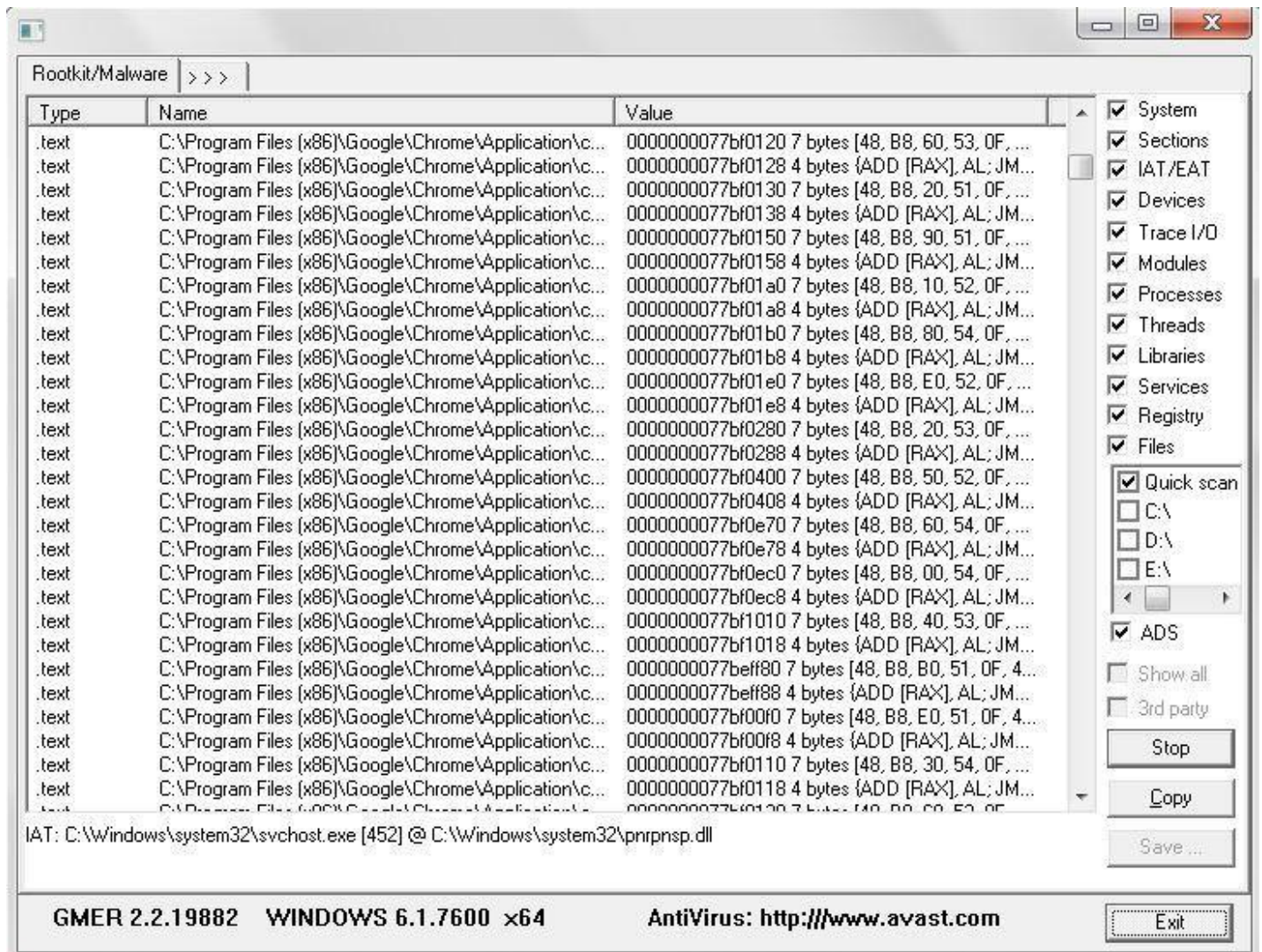
Double-click the downloaded .exe file to run GMER application.





**Step 55**

Click the 'Scan' button in the lower-right corner of the dialog box. Wait for the program to scan your entire hard drive.

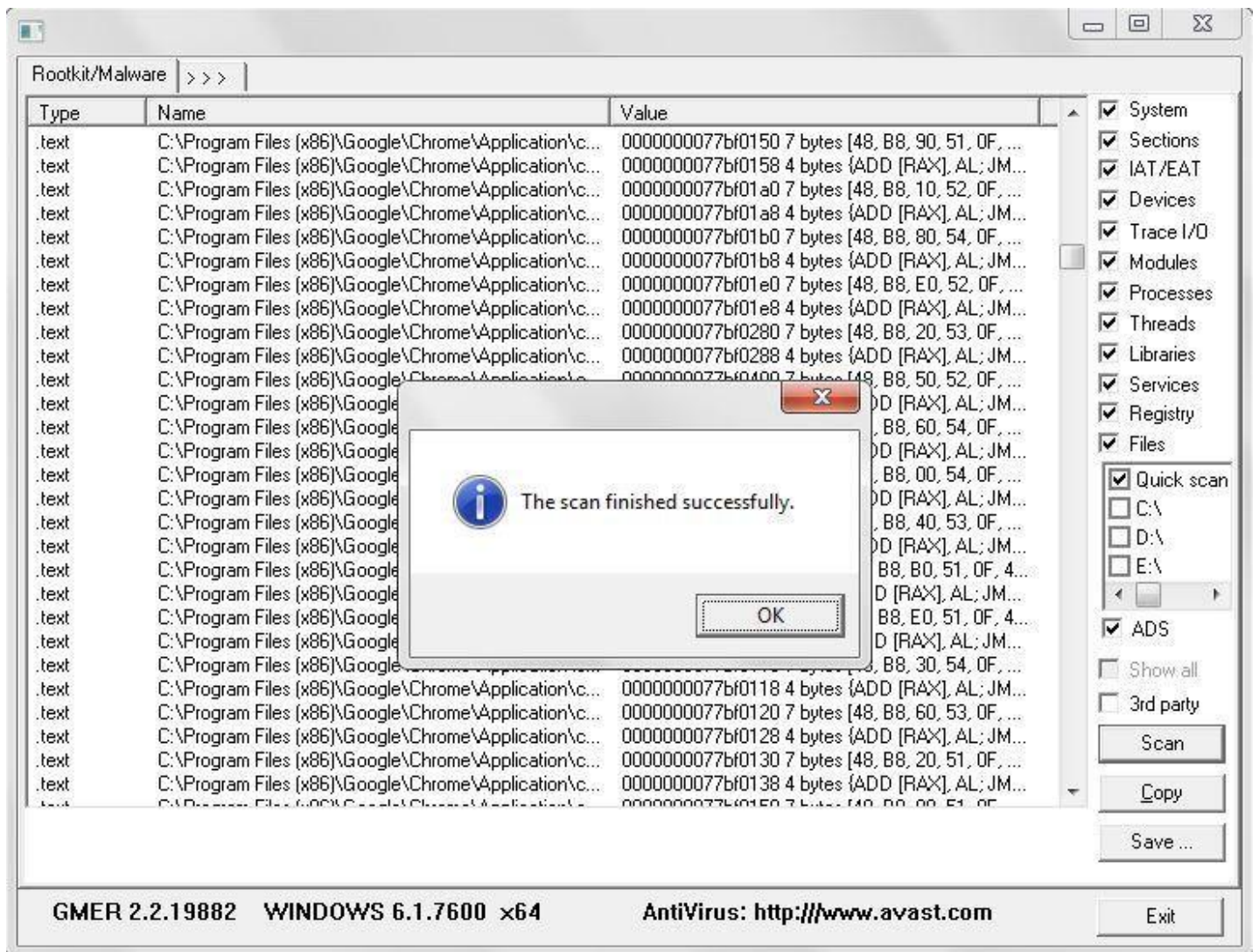


**Step 56**

When the program completes its scan, select any program or file listed in red.

Right-click it and select 'Delete'. If nothing is listed in red, you are good to go since no rootkit is found in your PC.

Note: If the red item is a service, it may be protected. Right-click the service and select 'Disable'. Reboot your computer and run the scan again, this time selecting 'Delete' when that service is detected.

**Step 5**

Close the program and restart your PC after deleting the rootkits.

**RESULT**

In this experiment a rootkit hunter software tool has been installed and the rootkits have been detected.