Vel Tech High Tech

Dr.Rangarajan Dr.Sakunthala Engineering College

An Autonomous Institution

(AN ISO 9001-2008 CERTIFIED INSTITUTION)

(Approved by AICTE, New Delhi & Affiliated to Anna University)

No.60, Avadi – Vel Tech Road, Chennai-600 062.

21CS77P–SECURITY LABORATORY

NAME	**:**	
REGISTER NO	**:**	
ROLL NO	**:**	
BRANCH	**:**	**B. E Computer Science and Engineering**
YEAR	**:**	**IV**
SEMESTER	**:**	**VII**

Vel Tech High Tech
Dr.Rangarajan Dr.Sakunthala Engineering College
An Autonomous Institution

(AN ISO 9001-2008 CERTIFIED INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to Anna University)
No.60, Avadi – Vel Tech Road, Chennai-600 062.

BONAFIDE CERTIFICATE

Name……………………..…………………………….….……………………..…………………………….

Year:……….. Semester:………………… Branch: ……..**B.E–Computer Science and Engineering**…..

University Register No: | | College Roll No: | VH |

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

21CS77P–Security Laboratory during the academic year **2025-2026.**

Signature of Lab Incharge **Signature of Head of the Department**

Submitted for the University Practical Examination held on ………………... at **VEL TECH HIGH TECH Dr. RANGARAJAN Dr. SAKUNTHALA ENGINEERING COLLEGE, No.60, AVADI – VEL TECH ROAD, AVADI, CHENNAI-600062.**

Signature of Examiners

Internal Examiner:………………… **External Examiner:……………..……**

Date:……………….……..

INDEX

S. No	Date	Name Of The Experiment	Page No	Marks	Sign
1		**Perform encryption, decryption using the following substitution techniques**			
		a) Caesar Cipher			
		b) Playfair Cipher			
		c) Hill Cipher			
2		**Perform encryption and decryption using following transposition techniques**			
		a) Rail fence technique –Row major transformation			
		b) Rail fence technique - Column major transformation			
3		a) DES algorithm			
		b) Apply DES algorithm for practical applications			
4		Implement RSA Algorithm using HTML and JavaScript			
5		implement the Diffie-Hellman Key Exchange mechanism. Consider one of the parties as Alice and the other party as bob.			
6		Calculate the message digest of a text using the SHA-1,MD5 algorithm.			
7		Implement the SIGNATURE SCHEME - Digital Signature Standard.			
8		Demonstrate intrusion detection system (ids) using any tool.Snort or any other s/w..			
9		Automated Attack and Penetration Tools Exploring N-Stalker, a Vulnerability Assessment Tool			
10		Implement the blowfish algorithm			
11		**Defeating Malware** i) Building Trojans			
		ii) Rootkit Hunter			
12		Perform ethical hacking-based a) Port scanning using the Nmap tool			
		b) Network sniffing using Wireshark.			
		Content Beyond The Syllabus			
13		Perform encryption, decryption using the following substitution techniques- Vigenere Cipher			
14		Implement Euclid Algorithm With Time			

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO 1:

Embark upon successful professional practice in Computer Science and Engineering, displaying supportive and leadership roles.

PEO 2:

Engage in professional projects requiring teamwork and making valuable contributions to design, development, and production in the practice of Computer Science and Engineering or application areas.

PEO 3:

Equip to adapt and grow with changes in technology and globalization, and to pursue higher studies and research activities.

PEO 4:

Be capable of productive employment in the field of Computer Science and Engineering with competing technical expertise, good interpersonal skill.

PEO 5:

Utilize their broad educational experience, ethics, and professionalism to make a positive impact on their local and professional communities.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

By the time of graduation, the undergraduate Computer Science and Engineering students can have the ability of

PSO’s	PROGRAMME SPECIFIC OUTCOMES (PSOs)
PSO1	Designing Computer/Electronic based components which would serve social environment.
PSO2	Applying the current and gained knowledge and modern techniques not only in the Computer but in all related fields.

COURSE OUTCOMES:

At the end of the course, the student will be able to:

CO1: Develop a code for classical encryption techniques

CO2: Build a symmetric and asymmetric algorithms

CO3: Construct a code for various Authentication schemes

CO4: Apply the principles of digital signature.

Mapping of Course Outcomes with Program Outcomes

CO	PO1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO1	PSO2
CO1	3	3	3	3	3	-	-	-	3	-	-	1	3	3
CO2	3	3	3	3	3	-	-	-	3	-	1	-	3	3
CO3	3	3	3	3	3	-	-	-	3	-	1	-	3	3
CO4	3	3	3	3	3	-	-	-	3	-	1	-	3	3

EX. NO: 1(A) DATE:	**ENCRYPTION AND DECRYPTION USING CEASER CIPHER**

AIM:

ALGORITHM:

Step 1: Enter the plain text P.

Step 2: Select the key value from the range (1-25).

Step 3 : Perform Encryption by substituting each letter in the plaintext by a letter some fixed number of positions down the alphabet .

C= (P+K) mod 26.

Step4: Repeat step 3 to convert pain text to cipher text.

Step5: Perform Decryption P**= (C-K) mod26.**

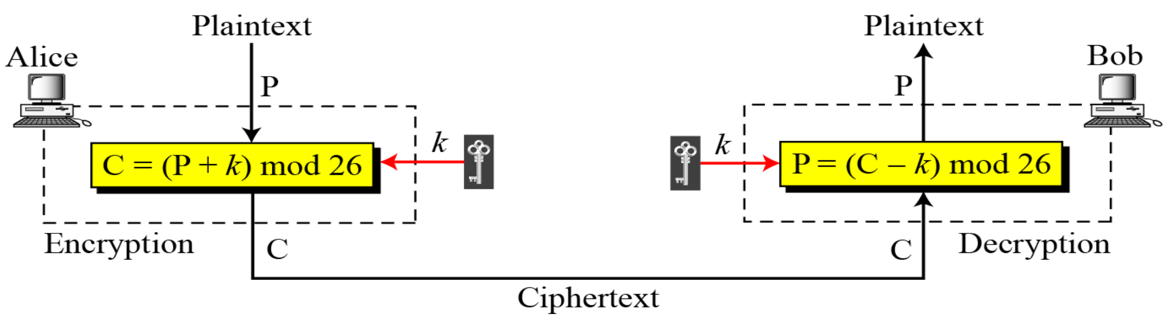
The action of a Caesar cipher is to replace each plain text letter with one fixed number of places down the alphabet. This example is with a shift of three, so that a B in the plain text becomes E in the cipher text.

Plain:

ABCDEFGHIJKLMNOPQRSTUVWXY

Z Cipher:

DEFGHIJKLMNOPQRSTUVWXYZABC



How to Use the Program

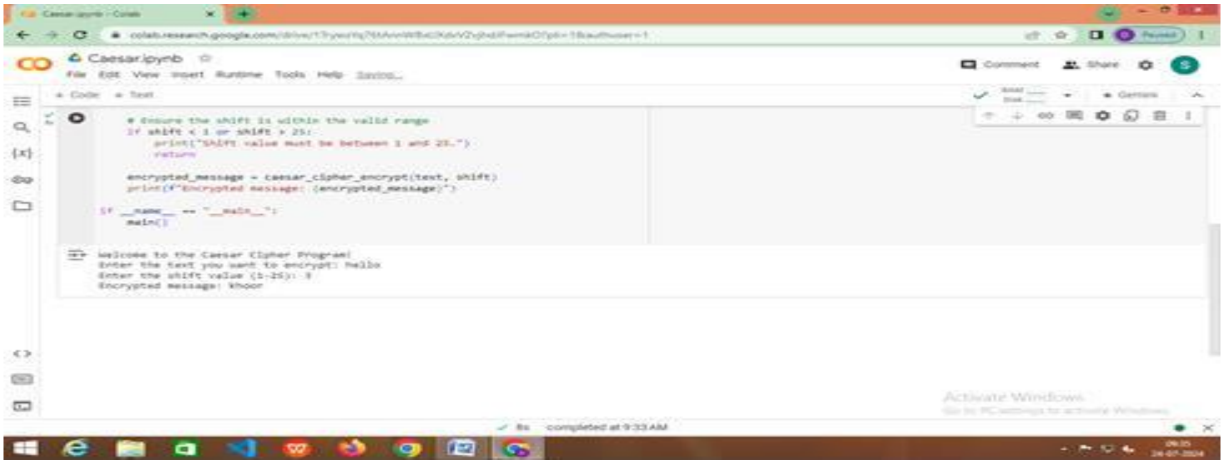
1. Input the Text: When prompted, enter the text you want to encrypt. This can include letters, spaces, and punctuation.
2. Input the Shift Value: Enter a shift value between 1 and 25. This determines how many positions each letter in the text will be shifted.
3. View the Encrypted Message: The program will display the encrypted message based on the Caesar Cipher algorithm.

PROGRAM:

Encryption ## Caesar Cipher Program in Python

```
def caesar_cipher_encrypt(text, shift):
    encrypted_text = ""
    for char in text:
        if char.isalpha(): # Check if the character is a
            letter shift_base = ord('A') if char.isupper()
            else ord('a')
            # Encrypt the character and wrap around using modulo
            encrypted_char = chr((ord(char) - shift_base + shift) % 26 + shift_base)
            encrypted_text += encrypted_char
        else:
            # Non-alphabetical characters are added
            unchanged encrypted_text += char
    return
encrypted_texT def
main():
    print("Welcome to the Caesar Cipher
    Program!") text = input("Enter the text you
    want to encrypt: ") shift = int(input("Enter
    the shift value (1-25): "))
    # Ensure the shift is within the valid
    range if shift < 1 or shift > 25:
        print("Shift value must be between 1 and
        25.") return
    encrypted_message =
    caesar_cipher_encrypt(text, shift)
    print(f"Encrypted message:
    {encrypted_message}")

if __name__== "__main__":
        main()
```

OUTPUT:



RESULT:

EX. NO: 1(B) DATE:	PLAYFAIR CIPHER

AIM:

ALGORITHM

Step 1: Construct 5 x 5 matrix of letters constructed using a keyword. Fill the remaining cells with the letters in the alphabetic order No duplication to letters is allowed, one cell is filled with two letters(I/J)

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Step 2: Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.

Step 3: Replace two plaintext letters, that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, ar is encrypted as RM.

Step 4: Replace two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.

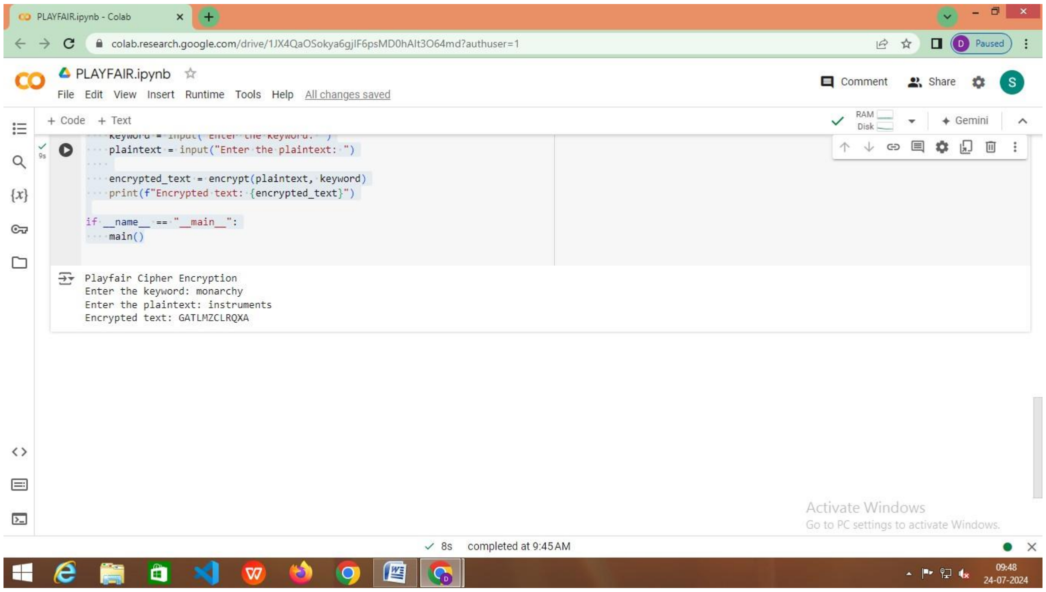
Step 5: Else, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs become BP and ea becomes IM (or JM, as the enciphered wishes).

PROGRAM:

```
def create_matrix(keyword):
    keyword = ''.join(sorted(set(keyword), key=keyword.index)).replace(' ',
    '').upper() keyword = keyword.replace('J', 'I')
   matrix = []
    for char in keyword:
        if char not in matrix and char.isalpha():
            matrix.append(char)
        for char in 'ABCDEFGHIKLMNOPQRSTUVWXYZ':
        if char not in matrix:
            matrix.append(char)
    return matrix
def format_plaintext(plaintext):
    plaintext = plaintext.replace(' ',  '').upper()
    plaintext = plaintext.replace('J', 'I')
    digraphs = []
    i = 0
    while i < len(plaintext):
        char1 = plaintext[i]
        if i + 1 < len(plaintext):
            char2 = plaintext[i +
            1]
            if char1 == char2:  # If both characters are the
                same digraphs.append(char1 + 'X')  # Add
                'X' as filler  i += 1
            else:
                digraphs.append(char1 +
                char2) i += 2
        else:
            digraphs.append(char1 + 'X')  # Add 'X' as filler for the last
            character i += 1
    return digraphs
def encrypt(plaintext, keyword):
    matrix =
    create_matrix(keyword)
    digraphs =
    format_plaintext(plaintext)
    encrypted_text = ''
    for digraph in digraphs:
        row1,                          col1                          =
        divmod(matrix.index(digraph[0]),   5)   row2,
        col2 = divmod(matrix.index(digraph[1]), 5) if
        row1 == row2:  # Same row
            encrypted_text += matrix[row1 * 5 + (col1 + 1)
            % 5] encrypted_text += matrix[row2 * 5 + (col2
            + 1) % 5]
        elif col1 == col2:  # Same column
            encrypted_text += matrix[((row1 + 1) % 5) * 5 +
            col1] encrypted_text += matrix[((row2 + 1) % 5)
            * 5 + col2]
        else:  # Rectangle rule
```

```
            encrypted_text  +=  matrix[row1  *  5  +
            col2] encrypted_text += matrix[row2 *
            5 + col1]
    return
encrypted_text       def
main():
    print("Playfair Cipher Encryption")
    keyword = input("Enter the
    keyword: ") plaintext = input("Enter
    the plaintext: ")
        encrypted_text = encrypt(plaintext, keyword)
    print(f"Encrypted text: {encrypted_text}")
if __name__== "__main__
    ": main()
```

OUTPUT:



RESULT:

NAME: SUKESH ST REG NO: 113022104150

EX. NO: 1(C) DATE:	HILL CIPHER

AIM:

ALGORITHM:

Step 1: Enter the plain text and Key.
Step 2: Take m successive plaintext letters and substitutes for them m cipher text letters.
Step 3: The substitution is determined by m linear equations in which each character is assigned a numerical value (a = 0, b = 1 ... z = 25).
This can be expressed in term of column vectors and matrices:

$$C = KP \bmod 26$$

where C and P are column vectors of length 3, representing the plaintext and ciphertext, and K is a 3 x 3 matrix, representing the encryption key.

Encryption ALGORITHM

$$\mathbf{C = E(K, P) = KP \bmod 26}$$

Step 4: Repeat Step 2 and Step 3 to encrypt the given plain text the cipher text.
Step 5: Perform decryption on the cipher text to convert to the given Plain text.

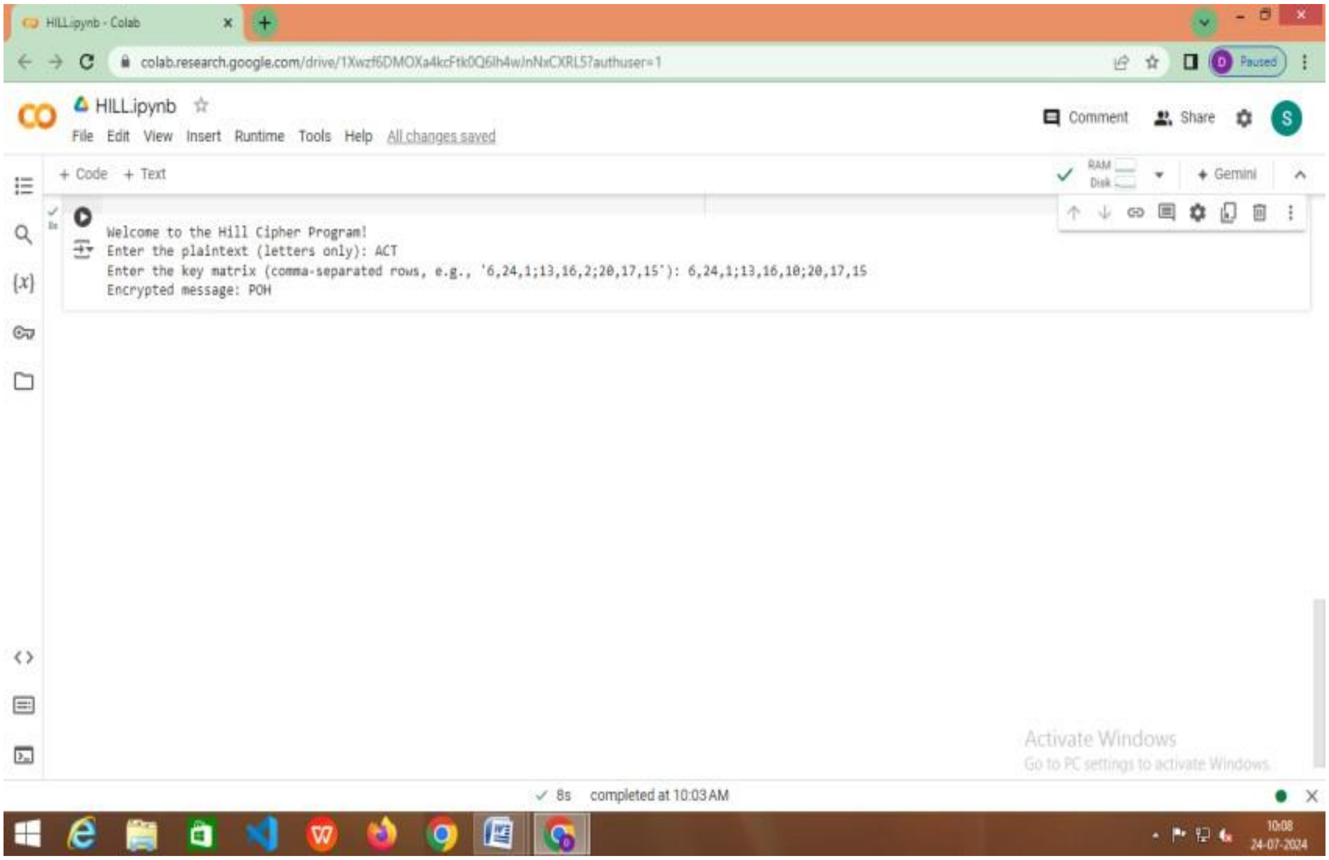
Decryption ALGORITHM

$$\mathbf{D(K, P) = K^{1}C \bmod 26 = K^{1}KP = P}$$

PROGRAM:

```
import numpy as np
def  matrix_mod_inv(matrix,  mod):
    det = int(np.round(np.linalg.det(matrix)))  # Determinant of the
    matrix det_inv = pow(det, -1, mod)  # Modular inverse of the
    determinant return matrix_mod
def hill_cipher_encrypt(plaintext,
   key_matrix): mod = 26  # For English
   alphabet
    plaintext = plaintext.replace(" ", "").upper()
    while len(plaintext) % key_matrix.shape[0] !=
    0:
        plaintext += 'X'  # Padding with 'X'
    ciphertext = ""
    for num in encrypted_vector.flatten():
    ciphertext += chr(num + ord('A'))
    return ciphertext
def main():
    print("Welcome to the Hill Cipher Program!")
plaintext = input("Enter the plaintext (letters only): ")
    if key_matrix.shape[0]  !=  key_matrix.shape[1]:

        print("Key matrix must be square.")
        return
        ciphertext = hill_cipher_encrypt(plaintext, key_matrix)
    print(f"Encrypted message: {ciphertext}")
if _name___== "_main_":
    main()
```

OUTPUT:



RESULT:

EX. NO: 2(A) DATE:	RAIL FENCE TECHNIQUE –ROW MAJOR TRANSFORMATION

AIM:

ALGORITHM:

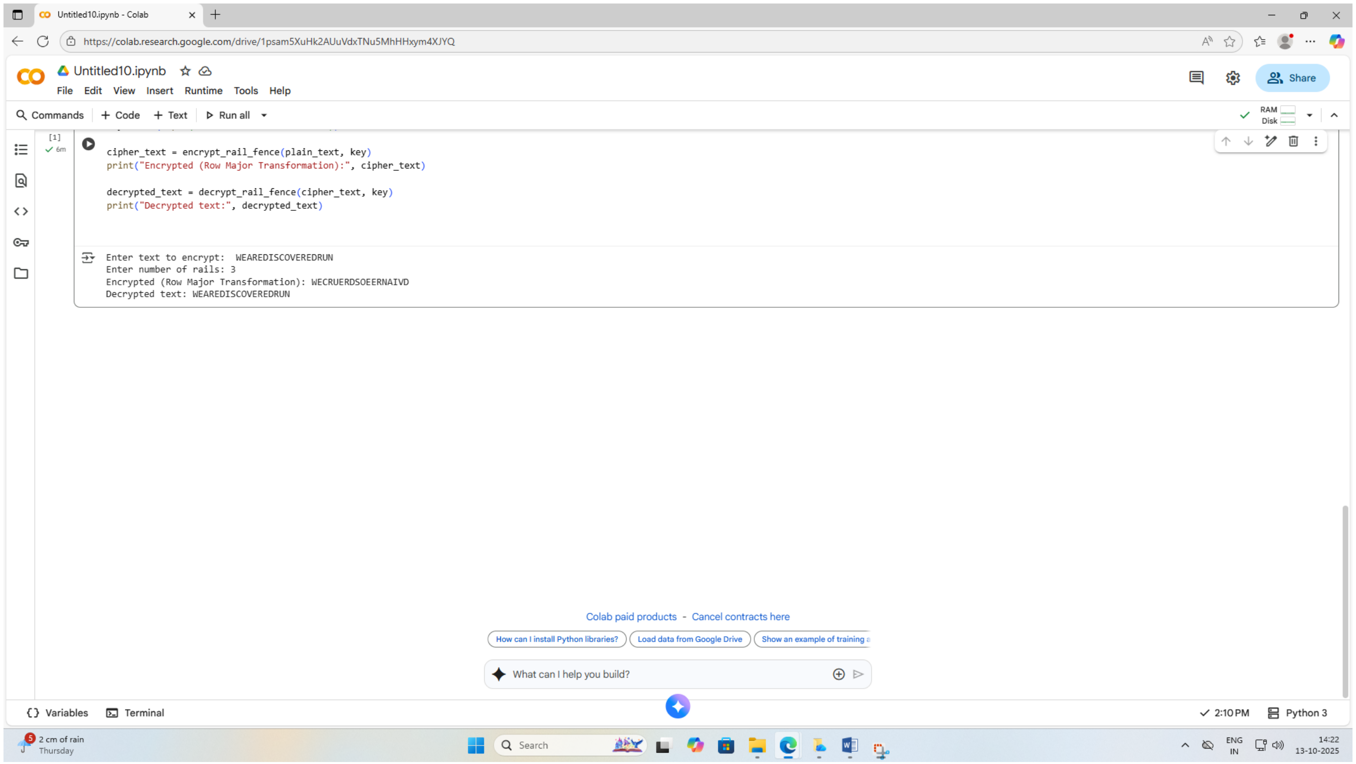
Step 1: Enter the Plain text.
Step 2: Write the Pain text letters out diagonally over a number of rows.
Step 3: Then read off cipher row by row to create a cipher text.
Step4: Thus converted plain text to cipher text using Rail Fence method.
Given a plain-text message and a numeric key, cipher/de-cipher the given text using Rail Fence algorithm.
The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded.

PROGRAM:

```
def encrypt_rail_fence(text, key):
# Create the matrix
rail = [['\n' for i in range(len(text))]
        for j in range(key)]

# Fill the rail matrix in zigzag manner
dir_down = False
row, col = 0, 0

for char in text:
    # Check direction
    if row == 0 or row == key - 1:
        dir_down = not dir_down

    # Place the character
    rail[row][col] = char
    col += 1

    # Move up or down
    row += 1 if dir_down else -1

# Now collect row-major order characters
result = []
for i in range(key):
    for j in range(len(text)):
        if rail[i][j] != '\n':
            result.append(rail[i][j])
return "".join(result)


def decrypt_rail_fence(cipher, key):
    # Create empty matrix
    rail = [['\n' for i in range(len(cipher))]
            for j in range(key)]

    # Mark positions
    dir_down = None
    row, col = 0, 0

    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False

        # Place marker
```

```
        rail[row][col] = '*'
        col += 1

        row += 1 if dir_down else -1

    # Fill the marked positions with cipher characters
    index = 0
    for i in range(key):
        for j in range(len(cipher)):
            if (rail[i][j] == '*') and (index < len(cipher)):
                rail[i][j] = cipher[index]
                index += 1

    # Read zigzag to get plaintext
    result = []
    row, col = 0, 0
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False

        if rail[row][col] != '\n':
            result.append(rail[row][col])
            col += 1

        row += 1 if dir_down else -1

    return "".join(result)


# --- Main Program ---
plain_text = input("Enter text to encrypt: ").replace(" ", "")
key = int(input("Enter number of rails: "))

cipher_text = encrypt_rail_fence(plain_text, key)
print("Encrypted (Row Major Transformation):", cipher_text)

decrypted_text = decrypt_rail_fence(cipher_text, key)
print("Decrypted text:", decrypted_text)
```

OUTPUT:



RESULT:

EX. NO: 2(B) DATE:	RAIL FENCE TECHNIQUE - COLUMN MAJOR RANSFORMATION

AIM:

ALGORITHM:

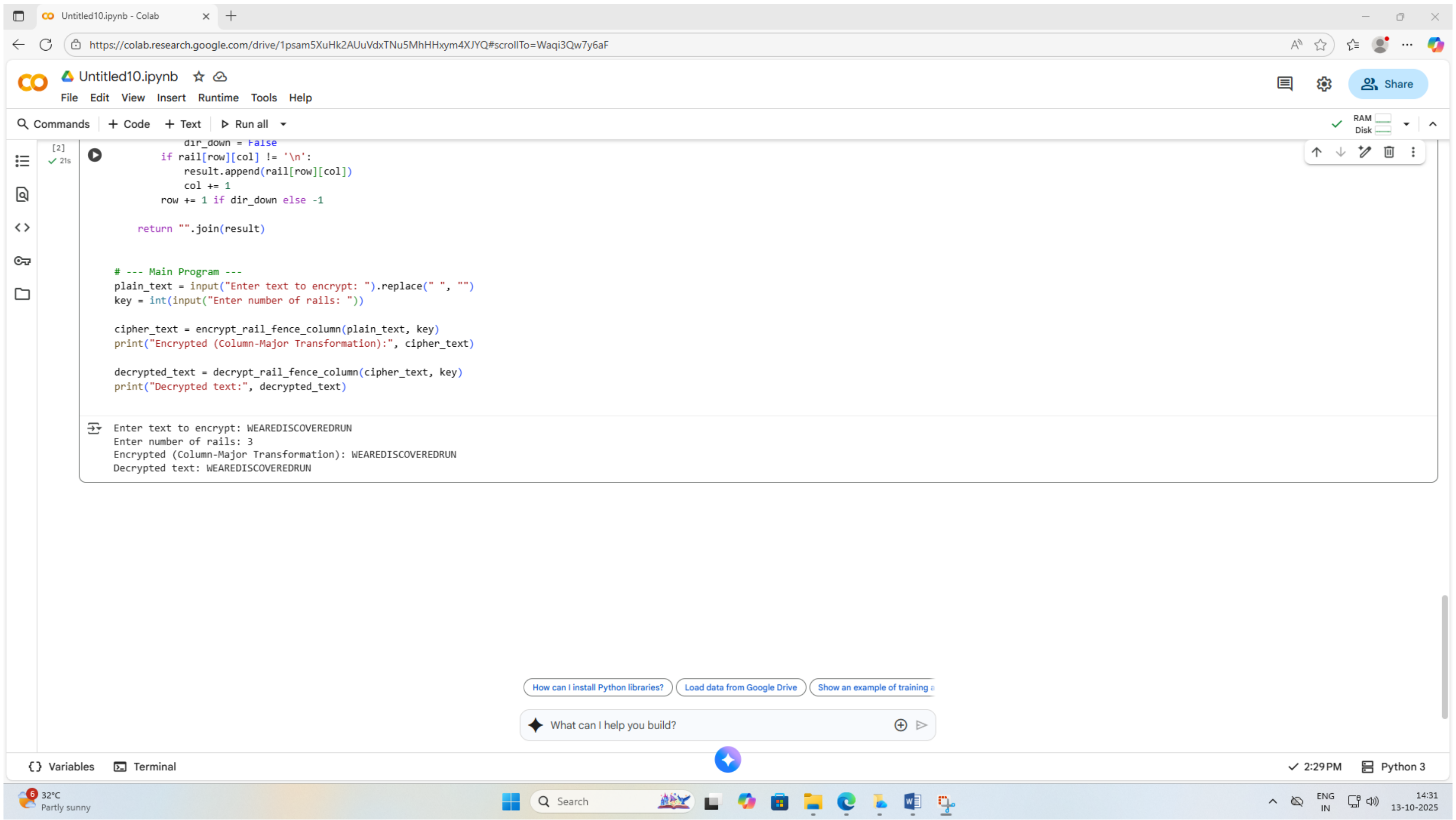
Step 1: Enter the plain text
Step 2: Select the number of columns.
Step 3: Write letters of plain text out in rows over a selected number of columns.
Step 4: Select the column Key order.
Step 5: Reorder the columns according to the key order before reading off the rows.
Step 6: Thus converted the given plain text to cipher text using Row transposition Ciphers.
Plaintext=attack postponed until twoam.

PROGRAM:

```
def encrypt_rail_fence_column(text, key):
    # Create the matrix
    rail = [['\n' for i in range(len(text))]
            for j in range(key)]

    # Fill the rail matrix in zigzag manner
    dir_down = False
    row, col = 0, 0

    for char in text:
        if row == 0 or row == key - 1:
            dir_down = not dir_down
        rail[row][col] = char
        col += 1
        row += 1 if dir_down else -1

    # Read column-major (top to bottom, left to right)
    result = []
    for j in range(len(text)):
        for i in range(key):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    return "".join(result)


def decrypt_rail_fence_column(cipher, key):
    # Create empty matrix
    rail = [['\n' for i in range(len(cipher))]
            for j in range(key)]

    # Mark zigzag positions
    dir_down = None
    row, col = 0, 0
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False
        rail[row][col] = '*'
        col += 1
        row += 1 if dir_down else -1

    # Fill the marked positions column-major
    index = 0
    for j in range(len(cipher)):
        for i in range(key):
            if rail[i][j] == '*' and index < len(cipher):
```

```
                rail[i][j] = cipher[index]
                index += 1

    # Read zigzag to reconstruct plaintext
    result = []
    row, col = 0, 0
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False
        if rail[row][col] != '\n':
            result.append(rail[row][col])
            col += 1
        row += 1 if dir_down else -1

    return "".join(result)


# --- Main Program ---
plain_text = input("Enter text to encrypt: ").replace(" ", "")
key = int(input("Enter number of rails: "))

cipher_text = encrypt_rail_fence_column(plain_text, key)
print("Encrypted (Column-Major Transformation):", cipher_text)

decrypted_text = decrypt_rail_fence_column(cipher_text, key)
print("Decrypted text:", decrypted_text)
```

OUTPUT:



RESULT:

NAME: SUKESH ST REG NO: 113022104150

EX. NO: 3(A) DATE:	DATA ENCRYPTION STANDARD (DES) ALGORITHM (USER MESSAGE ENCRYPTION)

AIM:

ALGORITHM:

Step1: Enter the plain text P.
Step 2: Select the key K of key length 64 bits.
Step3: Take a fixed-length string 64 bits from plaintext.
Step4: Perform Initial Permutation IP.
Step5: Transform 64-bit input in a series of steps into a 64-bit output
DES Round Structure:
It uses two 32-bit L & R halves as for any Feistel cipher can describe as:

***Li* = *Ri*–1**

***Ri*= *Li*–1 Å F (*Ri*–1, *Ki*)**

F takes 32-bit R half and 48-bit subkey:expands R to 48-bits using perm E adds to subkey using XORpasses through 8 S-boxes to get 32-bit result finally permutes using 32-bit perm P. **Step 6:** Repeat Step 5 sixteen times to convert 64 bits of plain text to 64 bits of cipher text.

Step 7: Go to step 3, repeat the process until all the plain text is converted to cipher text.

PROGRAM:

```
pip install pycryptodome
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad

# DES key must be exactly 8 bytes
key = b'8bytekey'
# Data to encrypt
data = "Hello, DES Encryption!"

# Convert data to bytes
data_bytes = data.encode('utf-8')

# --- Encryption ---
cipher = DES.new(key, DES.MODE_ECB)   # ECB mode
padded_text = pad(data_bytes, DES.block_size)  # Padding
encrypted_text = cipher.encrypt(padded_text)
print("Encrypted (bytes):", encrypted_text)
print("Encrypted (hex):", encrypted_text.hex())

# --- Decryption ---
decipher = DES.new(key, DES.MODE_ECB)
decrypted_padded = decipher.decrypt(encrypted_text)
decrypted_text = unpad(decrypted_padded, DES.block_size)
print("Decrypted text:", decrypted_text.decode('utf-8'))
```

OUTPUT:

```
Encrypted (bytes): b'\x9f\x8b\x1a...'
Encrypted (hex): 9f8b1a...
Decrypted text: Hello, DES Encryption!
```

RESULT:

EX. NO: 3(B) DATE:	ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

AIM:

ALGORITHM:

1. Key Generation: The `generate_key` function creates a random 256-bit key for AES encryption.
2. Encryption Function: The `encrypt` function:
   - Creates a new AES cipher object in CBC mode.
   - Pads the plaintext to ensure its length is a multiple of 16 bytes.
   - Encrypts the padded text and returns both the initialization vector (IV) and the encrypted text, both encoded in base64 for easy representation.
3. Decryption Function: The `decrypt` function:
   - Decodes the base64 encoded IV and ciphertext.
   - Creates a new AES cipher object with the same key and IV.
   - Decrypts the ciphertext and removes the padding, returning the original plaintext.

PROGRAM:

First, ensure you have the `pycryptodome` library installed:

pip install pycryptodome

```
from Crypto.Cipher import DES
from Crypto.Util.Padding import pad, unpad

# --- DES Key (must be 8 bytes) ---
key = b'8bytekey'

# --- Function to encrypt file ---
def encrypt_file(input_file, output_file):
    cipher = DES.new(key, DES.MODE_ECB)  # ECB mode
    with open(input_file, 'rb') as f:
        plaintext = f.read()
    padded_text = pad(plaintext, DES.block_size)
    ciphertext = cipher.encrypt(padded_text)
    with open(output_file, 'wb') as f:
        f.write(ciphertext)
    print(f"File '{input_file}' encrypted successfully -> '{output_file}'")

# --- Function to decrypt file ---
def decrypt_file(input_file, output_file):
    cipher = DES.new(key, DES.MODE_ECB)
    with open(input_file, 'rb') as f:
        ciphertext = f.read()
    decrypted_padded = cipher.decrypt(ciphertext)
    plaintext = unpad(decrypted_padded, DES.block_size)
    with open(output_file, 'wb') as f:
        f.write(plaintext)
    print(f"File '{input_file}' decrypted successfully -> '{output_file}'")

# --- Example Usage ---
# Create a sample text file
with open('sample.txt', 'w') as f:
    f.write("This is a secret message for DES encryption demo!")

# Encrypt the file
encrypt_file('sample.txt', 'sample_encrypted.des')

# Decrypt the file
decrypt_file('sample_encrypted.des', 'sample_decrypted.txt')

# Read decrypted content
with open('sample_decrypted.txt', 'r') as f:
    print("Decrypted file content:", f.read())
```

OUTPUT:

```
File 'sample.txt' encrypted successfully -> 'sample_encrypted.des'
File 'sample_encrypted.des' decrypted successfully -> 'sample_decrypted.txt'
Decrypted file content: This is a secret message for DES encryption demo!
```

RESULT:

EX. NO: 4 DATE:	IMPLEMENT RSA ENCRYPTION TECHNIQUE (RIVEST-SHAMIR-ADLEMAN)

AIM:

ALGORITHM:

Step1: Choose two distinct prime numbers p and q.For security purposes, the integers p and q shouldbe chosen at random, and should be of similar bit-length.

Compute $n = pq$.

- n is used as the modulus for both the public and private keys

Step 2: Compute $\varphi(n) = (p-1)(q-1)$, where φ is Euler's totient function.

Step 3: Choose an integer e such that $1 < e < \varphi(n)$ and greatest common divisor of $(e, \varphi(n)) = 1$;i.e., e and $\varphi(n)$ are co prime.e is released as the public key exponent.

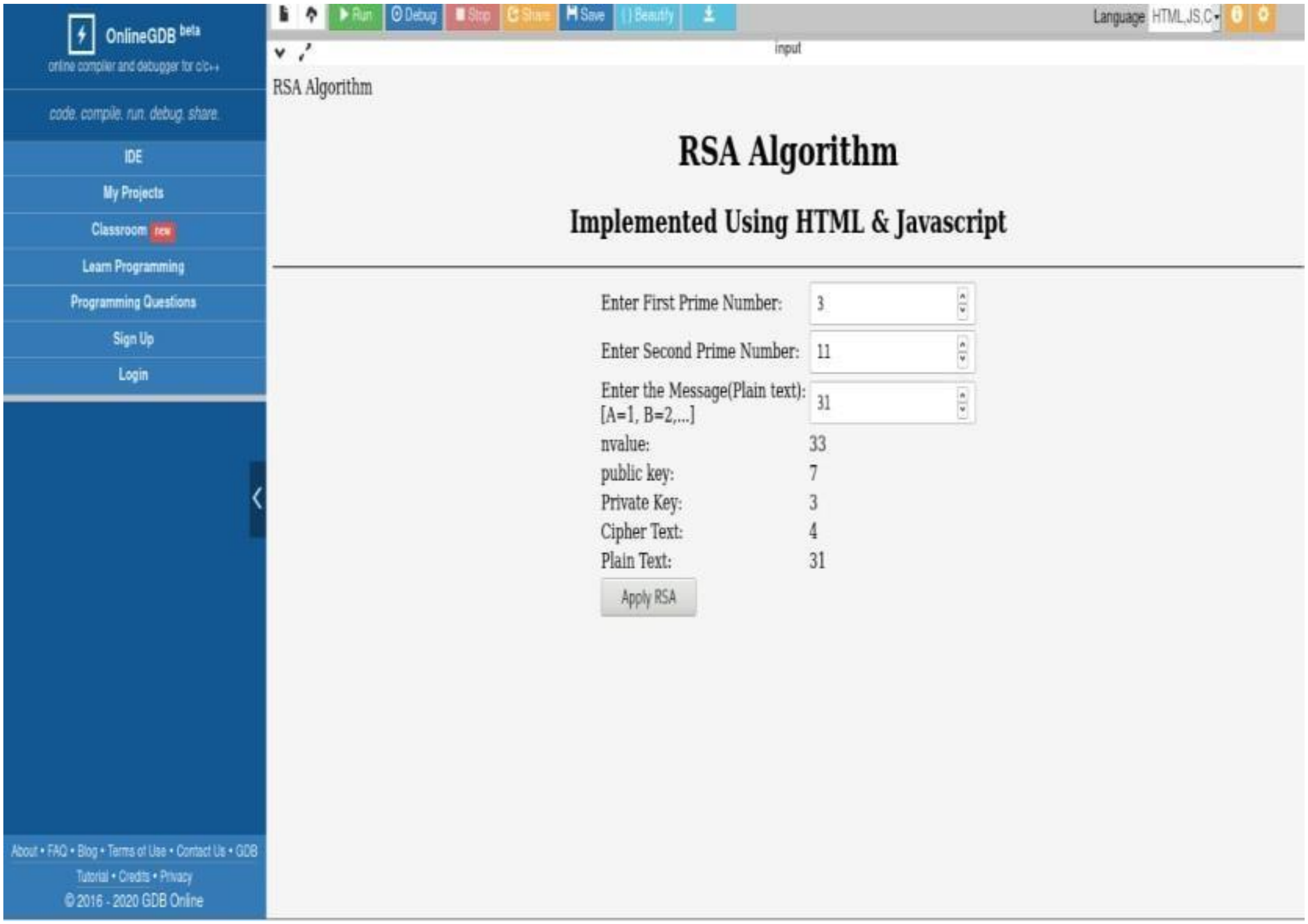
Step 4: Determine d as:

PROGRAM:

```
<html>

<head>

<title>RSA  Encryption</title>

<meta  name="viewport"  content="width=device-width,  initial-scale=1.0">

</head>

<body>

<center>

</table>

</center> </body>

<script type="text/javascript">

function RSA() {

var p, q, no, n, t, e, i, x, d;

p = document.getElementById('p').value;

q = document.getElementById('q').value;

no =

document.getElementById('msg').value;  n

= p * q;

t = (p - 1) * (q - 1);

e = 7

for (i = 0; i < 10; i++)

{ x = 1 + i * t

if (x % e == 0) {
```

```
d = x / e;

break; }}

ctt =

Math.pow(no,e).toFixed(0); ct

= ctt % n;

dtt =

Math.pow(ct,d).toFixed(0); dt

= dtt % n;

document.getElementById('nvalue').innerHTML = n;

document.getElementById('publickey').innerHTML = e;

document.getElementById('privatekey').innerHTML = d;

document.getElementById('ciphertext').innerHTML = ct;

document.getElementById('plaintext').innerHTML = dt;

}</script> </html>
```

OUTPUT:



RESULT:

EX. NO: 5 **DATE:**	**IMPLEMENT THE DIFFIE-HELLMAN KEY EXCHANGE MECHANISM. CONSIDER** **ONE OF THE PARTIES AS ALICE AND THE OTHER PARTY AS BOB.**

AIM:

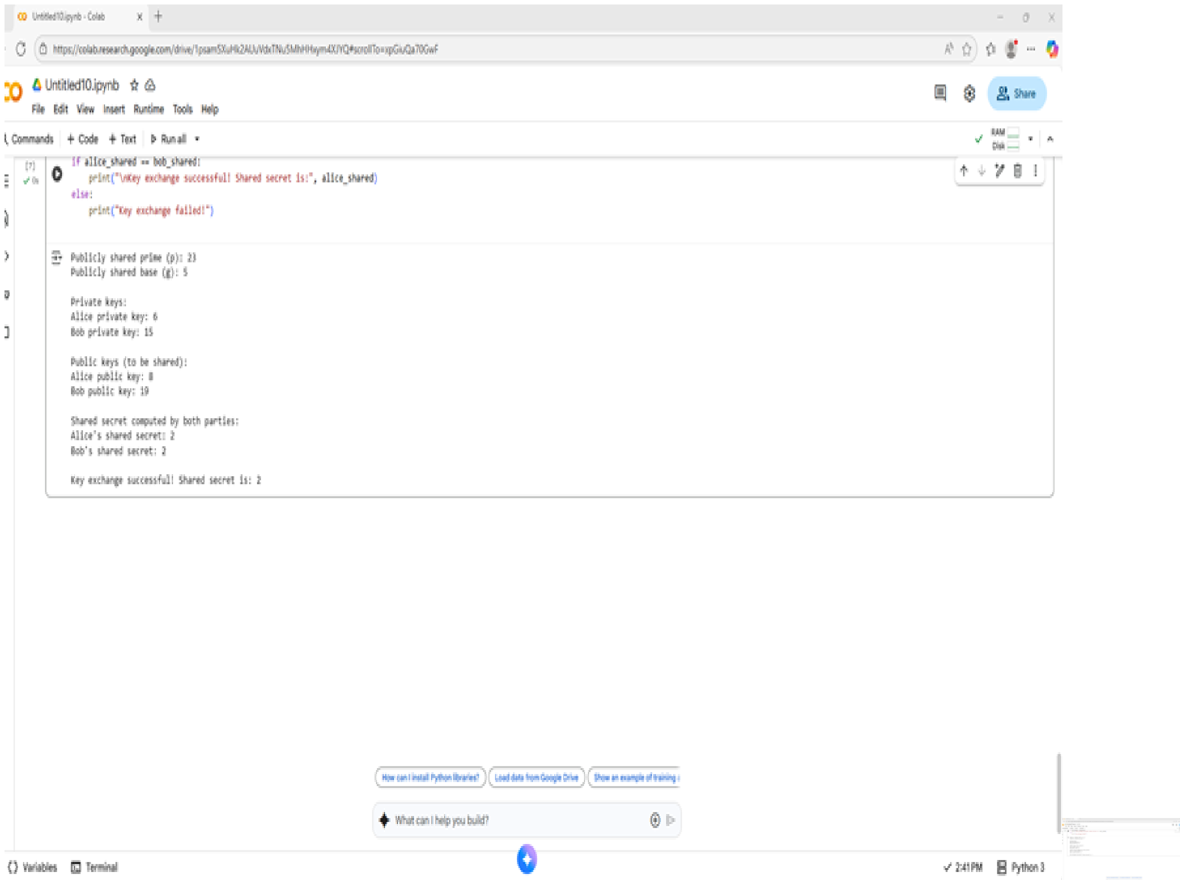
ALGORITHM:

Step 1 : GLOBAL PUBLIC ELEMENTS
Select any prime no : 'q'. Calculate the primitive root of q : 'a' such that a<q
Step 2 : ASYMMETRIC KEY GENERATION BY USER 'A'
Select a random number as the private key X_A where $X_{A.}$

PROGRAM:

```
# Diffie-Hellman Key Exchange
# Public parameters (shared)
# In real-world, these should be very large primes
prime = 23       # A prime number (p)
base = 5         # Primitive root modulo prime (g)
print("Publicly shared prime (p):", prime)
print("Publicly shared base (g):", base)
# --- Private keys (kept secret) ---
alice_private = 6  # Alice's private key (a)
bob_private = 15   # Bob's private key (b)
print("\nPrivate keys:")
print("Alice private key:", alice_private)
print("Bob private key:", bob_private)
# --- Compute public keys ---
alice_public = (base ** alice_private) % prime
bob_public = (base ** bob_private) % prime
print("\nPublic keys (to be shared):")
print("Alice public key:", alice_public)
print("Bob public key:", bob_public)
# --- Compute shared secret ---
alice_shared = (bob_public ** alice_private) % prime
bob_shared = (alice_public ** bob_private) % prime
print("\nShared secret computed by both parties:")
print("Alice's shared secret:", alice_shared)
print("Bob's shared secret:", bob_shared)
# Verify that both are equal
if alice_shared == bob_shared: print("\nKey exchange successful! Shared secret is:",
alice_shared)
        else: print("Key exchange failed!")
```

OUTPUT



Result:

EX. NO: 6(a) DATE:	SHA-1 ALGORITHM

AIM:

ALGORITHM:

Step 1: Append padding
bitsStep 2: Append length
Step 3: Initialize hash buffer
Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithmStep 5: Output the final state value as the resulting hash.

Step-1:Appending Padding Bits. The original message is "padded" (extended) so that its length (in bits) consists of a single 1-bit followed by the necessary number of 0-bits, so that its length is congruent to 896 modulo 1024

Step-2: **Append length:** a block of 64 bits is appended to the message. This block is treated asunsigned 64 bit integers (most significant byte first) and contains the length of the original message.

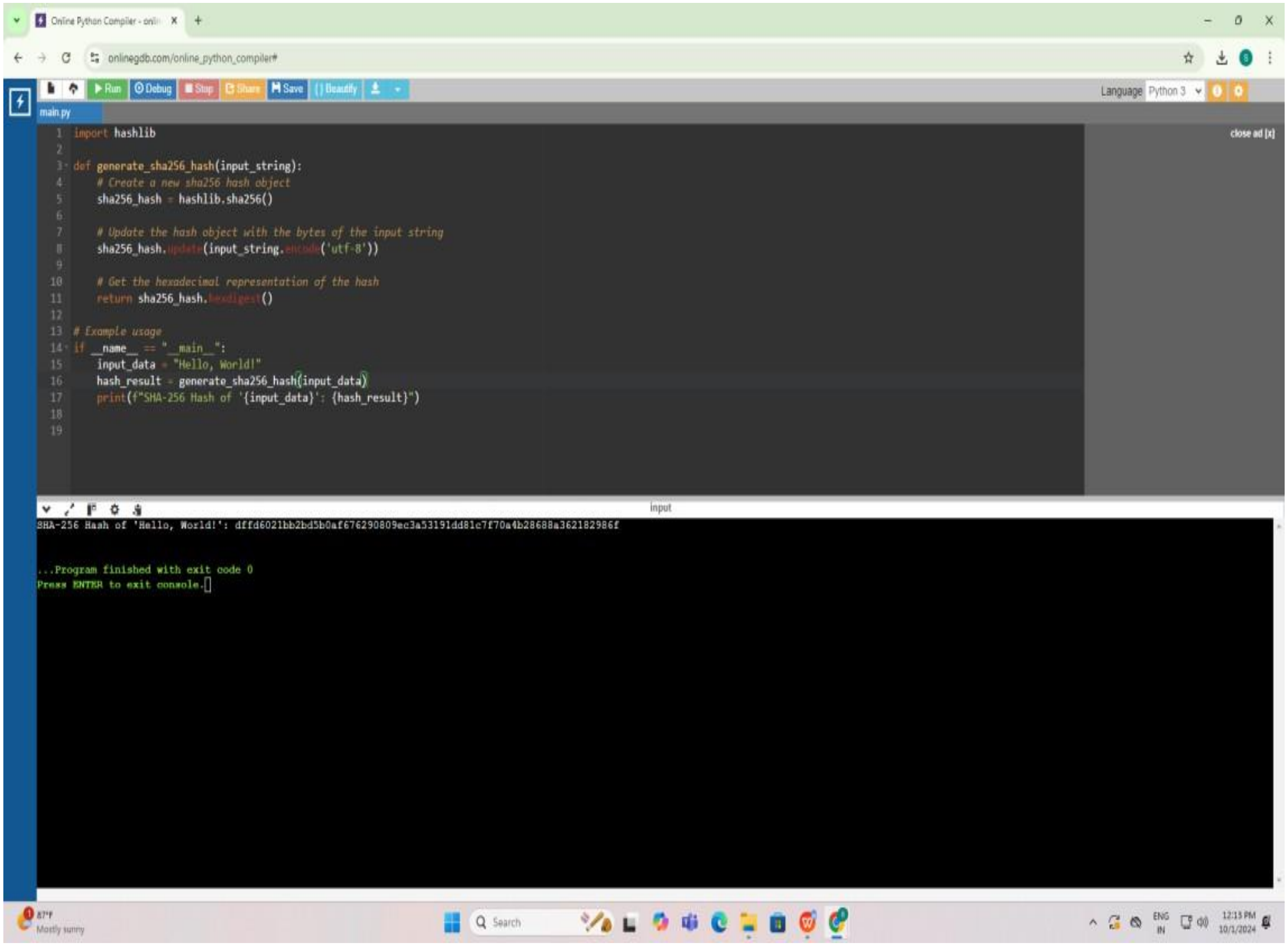
Step-3: Initialize hash buffer

Step 4: Process Message in 512 bits.

PROGRAM:

```
iimport hashlib
def
    generate_sha256_hash(input_string):
    sha256_hash = hashlib.sha256()
        sha256_hash.update(input_string.encode('utf-
    8'))   return sha256_hash.hexdigest()
if _name___== "_main_":
    input_data = "Hello,
    World!"
    hash_result = generate_sha256_hash(input_data)
    print(f"SHA-256  Hash  of '{input_data}':
    {hash_result}")
```

OUTPUT:



RESULT:

EX. NO: 6(b) DATE:	**MD5 (MESSAGE DIGEST ALGORITHM)**

AIM:

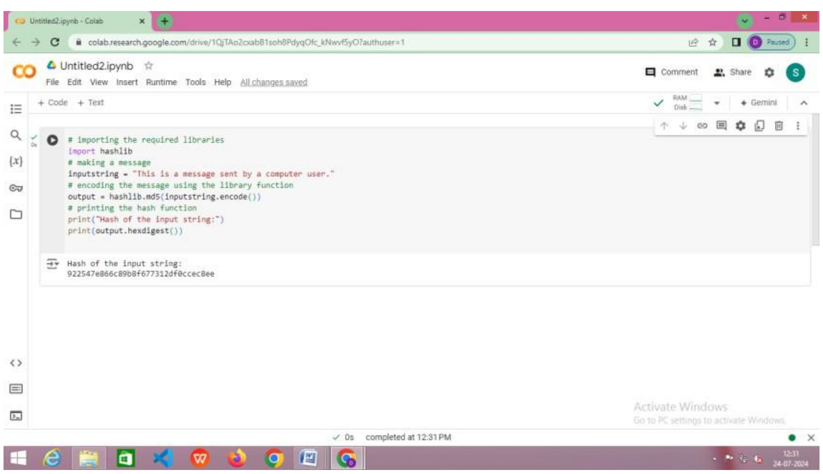
ALGORITHM:

Step1: Append Padding
Step 2: Append Length
Step 3: Initialize MD
Buffer
Step 4: Process Message in 512 bit (16-Word)
Step 5: Output

PROGRAM:

```
import hashlib
inputstring = "This is a message sent by a computer
user." output = hashlib.md5(inputstring.encode())
print("Hash of the input string:")
print(output.hexdigest())
```

OUTPUT:

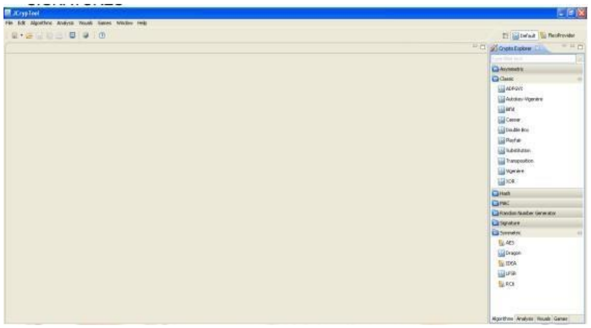


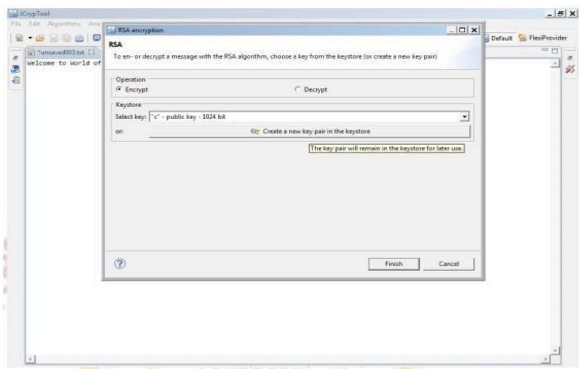
RESULT:

EX. NO: 7 DATE:	**DIGITAL SIGNATURE STANDARD**

AIM:

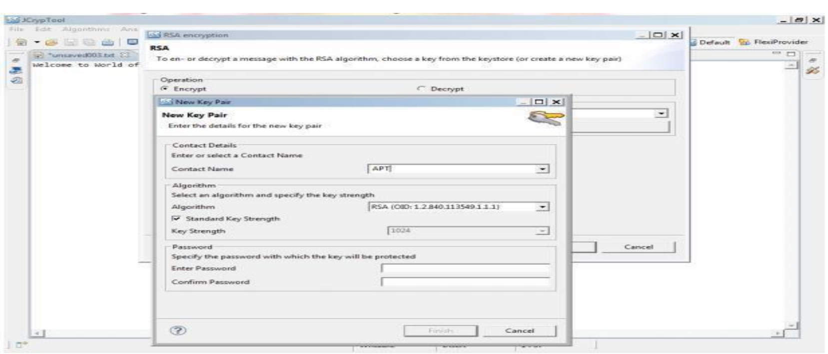
PROCEDURE:



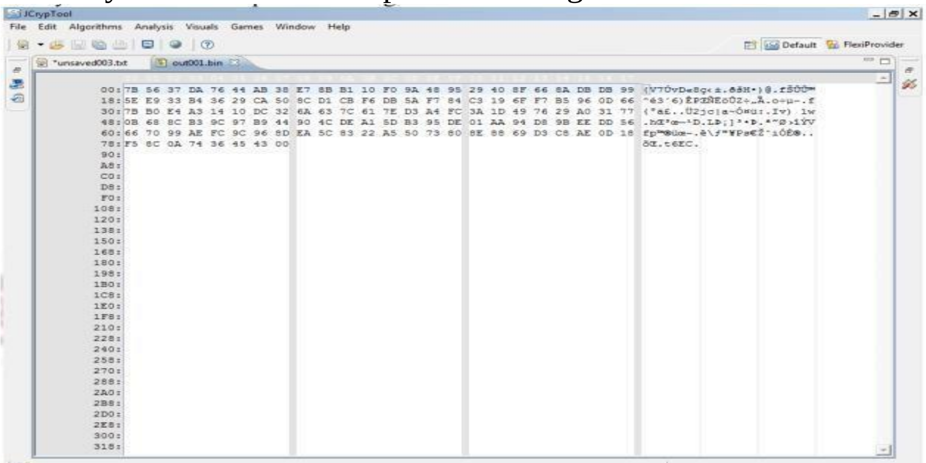


Click on the ALGORITHM menu bar and Select Asymmetric ALGORITHM RSA for encryption.

1. Click create a New KeyPair and type in the contact name[xxxxx] and enter the password and confirm password, then Click finish again.



2. Now you can see RSA output bin file is generated.



The same output bin file to decrypt select RSA ALGORITHM and Click on Decrypt, Select keyname you have declared earlier and Click Finish.

3. Enter the password to Decrypt and see the output with original Decrypted text on theScreen.

SYMMETRIC ALGORITHM

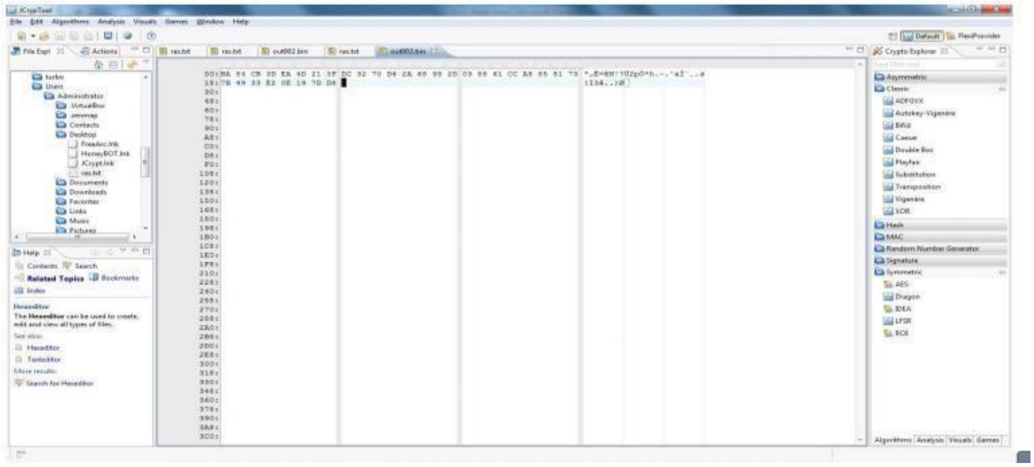
Click on ALGORITHM Menu bar Select Symmetric AES and Click on it.
Click on create a new key, type contact name and enter the password and confirm, Click Finish
Click finish again.
Enter the password to open the output file.
To Decrypt Select ALGORITHMs Symmetric Select the key which you have created and Click Finish.
Enter the password and see the result in output bin file with hexadecimal values and plain text.

HASH GENERATION

Click on ALGORITHMs, Click on Hash Select MD5 Click Finish.
Now view the output bin file HASH generated.
Practice using SHA and SHA3 and verify the result on the screen.

DIGITAL SIGNATURE

Click on ALGORITHMs, Click on Signature, Select DSA and Click on it. Select sign operation and Click on create a new key.
Enter the password and save the file and Click finish.
To verify Click on ALGORITHM, Click on Signature and Click DSA.
Select verify operation, Click open and type the password and Click finish. The Signature file is opened and verified.



RESULT:

EX. NO: 8 DATE:	**DEMONSTRATION OF INTRUSION DETECTION SYSTEM(IDS)**

AIM:

PROCEDURE:

SNORT can be configured to run in three modes:

1. Sniffer mode
2. Packet Logger mode
3. Network Intrusion Detection System mode

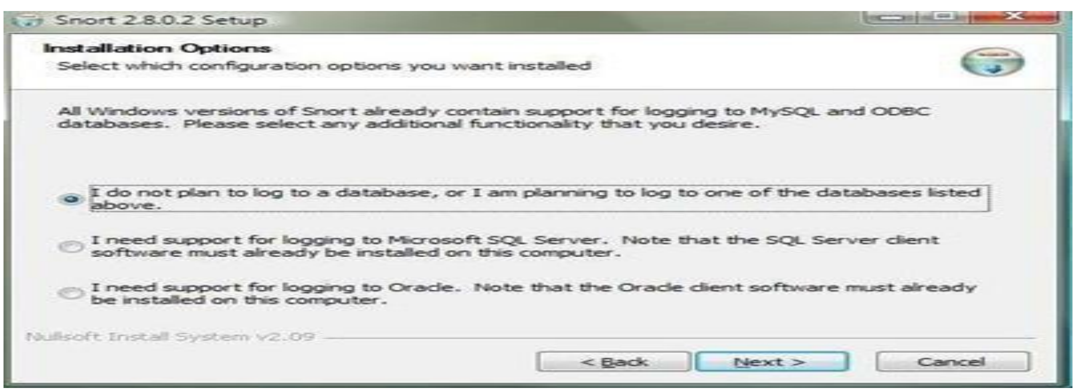
Sniffer mode snort –v Print out the TCP/IP packets header on the screen Snort –vd show the TCP/IP ICMP header with application data in transit.

Packet Logger mode

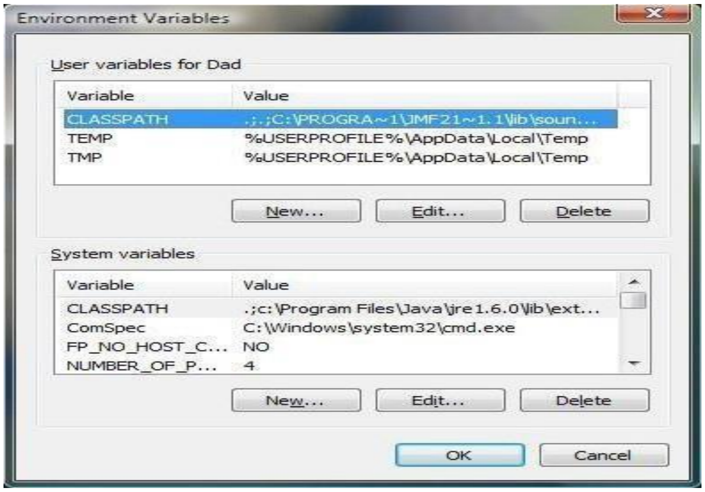
snort –dev –l c:\log [create this directory in the C drive] and snort will automatically know to go into packet logger mode, it collects every packet it sees and places it in log directory.snort –dev –l c:\log –h ipaddress/24 This rule tells snort that you want to print out the data link and TCP/IP headers as well as application data into the log directory.snort –l c:\log –b This is binary mode logs everything into a single file.

Network Intrusion DetectionSystem mode snort –d c:\log –h ipaddress/24 –c snort.conf This is a configuration file applies rule to each packet to decide it an action based upon the rule type in the file.Snort –d –h ipaddress/24 –l c:\log –c snort.conf This will cnfigure snort to run in its most basic NIDS form, logging packets that trigger rules specifies in the snort.conf

- Download SNORT from snort.org
- Install snort with or without database support.



1. Select all the components and Click Next. Install andClose.
2. Skip the Win P cap driver installation
3. Add the path variable in windows environment variable by selecting new class path. Create a path variable and point it at snort.exe variable name path and a path variable



4. Click OK button and then close all dialogboxes.

5. Open command prompt and type the followingcommands:

RESULT:

EX. NO: 9 DATE:	EXPLORING N-STALKER, A VULNERABILITY ASSESSMENT TOOL

AIM:

EXPLORING N-STALKER:

N-Stalker Web Application Security Scanner is a Web security assessment tool.
It incorporates with a well-known N-Stealth HTTP Security Scanner and35,000 Web attack signature database.

This tool also comes in both free and paid version.

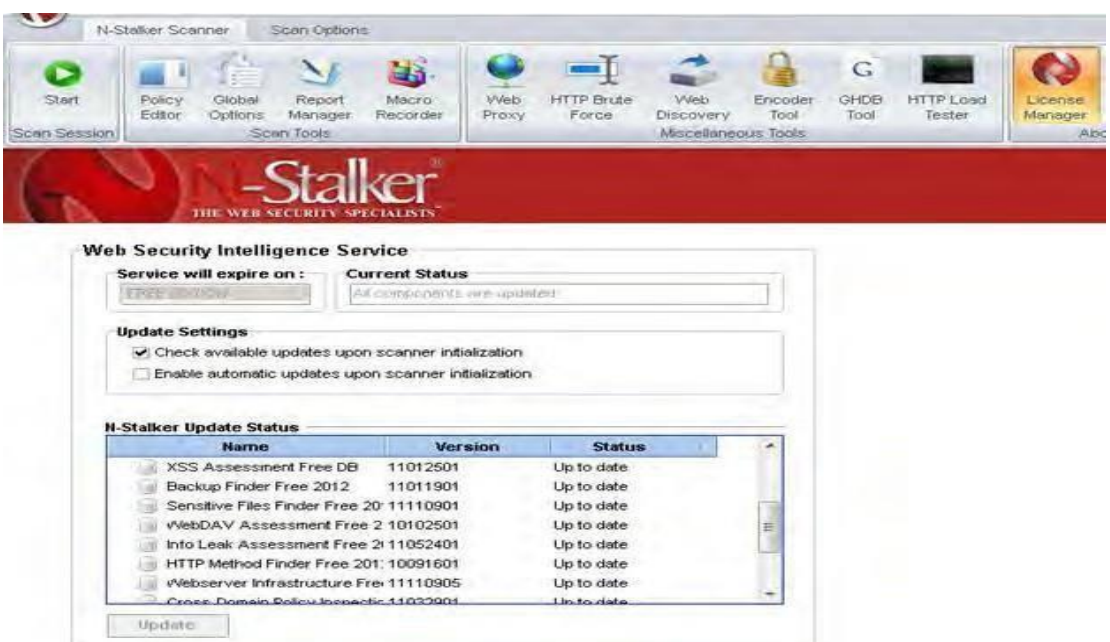
Before scanning the target, goto ―License Manager ‖tab, perform the

update. Once update, you will note the status as up to date.

You need to download and install N-Stalker from www.nstalker.com.

1. Start N-Stalker from a Windows computer. The program is installed under Start➪ Programs ➪N-Stalker ➪N-Stalker Free Edition.
2. Enter a host address or a range of addresses toscan.
3. Click Start Scan.
4. After the scan completes, the N-Stalker Report Manager will prompt
5. you to select a format for the resulting report as choose Generate HTML.
6. Review the HTML report for vulnerabilities.

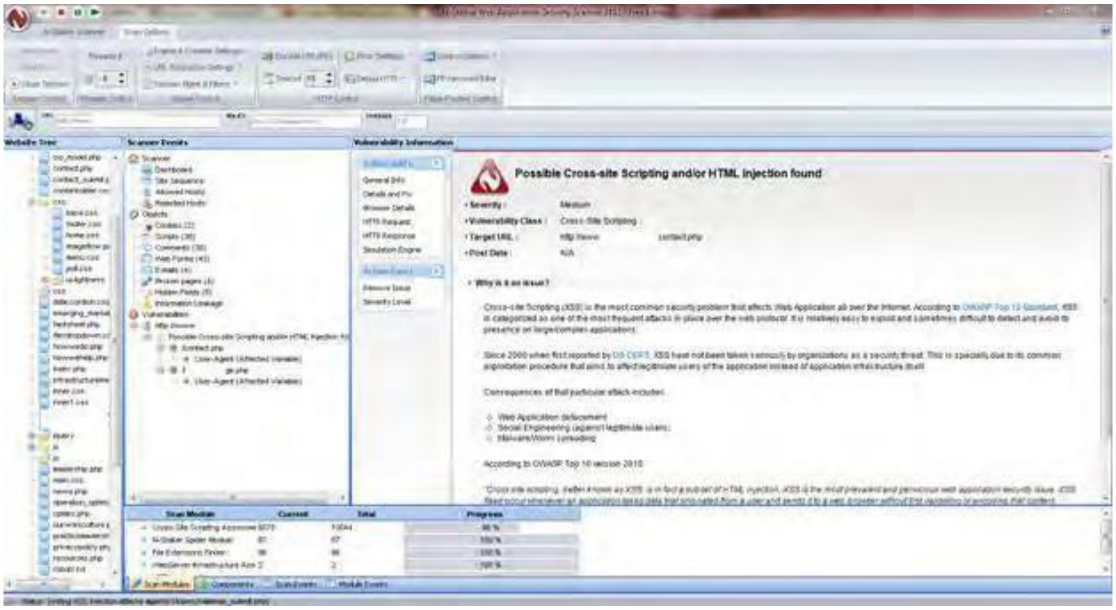
OUTPUT:



Nowgoto—Scan Session‖, enterthe target URL.
In scan policy, you can select from the four options,

- Manual test which will crawl the website and will be waiting formanualattacks.
- full xssassessment
- owasp policy
- Web server infrastructureanalysis.

Once,theoptionhasbeenselected,nextstepis—Optimizesettings‖whichwillcrawlthe whole website for furtheranalysis. In review option, you can get all the information like host information, technologies used, policy name, etc.



RESULT:

EX. NO: 10 DATE:	**IMPLEMENT THE BLOWFISH ALGORITHM**

AIM:

ALGORITHM:

step1: Generation of subkeys:

18 subkeys{P[0]…P[17]} are needed in both encryption as well as decryption process and the same subkeys are used for both the processes.

These 18 subkeys are stored in a P-array with each array element being a 32-bit entry. It is initialized with the digits of pi(?).

The hexadecimal representation of each of the subkeys is given by

P[0] = "243f6a88"

P[1] = "85a308d3"

.

.

P[17] = "8979fb1b"

Step2: initialise Substitution Boxes:

4 Substitution boxes(S-boxes) are needed{S[0]…S[4]} in both encryption aswell as decryption process with each S-box having 256 entries{S[i][0]…S[i][255], 0&lei&le4} where each entry is 32-bit.

It is initialized with the digits of pi(?) after initializing the P-array.

Step3: Encryption:

The encryption function consists of two parts:

a. Rounds: The encryption consists of 16 rounds with each round(Ri) taking inputs the plainText(P.T.) from previous round and corresponding subkey(Pi). The description of each round is as follows:

PROGRAM:

```
pip install pycryptodome
String S[][]= { { "d1310ba6", "98dfb5ac", "2ffd72db", "d01adfb7",
        "b8e1afed", from Crypto.Cipher import Blowfish
from Crypto.Util.Padding import pad, unpad

# --- Key (can be 4 to 56 bytes) ---
key = b'myblowfishkey'

# --- Data to encrypt ---
data = "Hello, this is Blowfish encryption demo!"
data_bytes = data.encode('utf-8')

# --- Create cipher object ---
cipher = Blowfish.new(key, Blowfish.MODE_ECB)

# --- Encryption ---
padded_data = pad(data_bytes, Blowfish.block_size)  # Pad to 8-byte
        block
encrypted_data = cipher.encrypt(padded_data)
print("Encrypted (bytes):", encrypted_data)
print("Encrypted (hex):", encrypted_data.hex())

# --- Decryption ---
decipher = Blowfish.new(key, Blowfish.MODE_ECB)
decrypted_padded = decipher.decrypt(encrypted_data)
decrypted_data = unpad(decrypted_padded, Blowfish.block_size)
print("Decrypted text:", decrypted_data.decode('utf-8'))
```

OUTPUT:

```
Encrypted (bytes): b'\x9f\x12...'
Encrypted (hex): 9f12...
Decrypted text: Hello, this is Blowfish encryption demo!
```

RESULT:

EX. NO: 11(a) DATE:	**DEFEATING MALWARE - BUILDING TROJANS**

AIM:

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 thistrojan.

TROJAN:

In computing, a Trojan horse,or trojan, is any malware which misleads users of its true intent.

Trojans are generally spread by some form of social engineering, for example where a user is duped into executing an email attachmentdisguised to appear not suspicious, (e.g., a routine form to be filled in), or by clicking on some fake advertisement on social media or anywhere else.

Although their payload can be anything, many modern forms act as a backdoor, contacting a controller which can then have unauthorizedaccess to the affected computer. Trojans may allow an attacker to access users' personal information suchas banking information, passwords, or personal identity.

Example: *Ransomware* attacks are often carried out using a *trojan*.

CODE:

Trojan.bat

```
@echo off
:x
start mspaint
start notepad
start cmd start
explorer start
control start
calc goto x
```

OUTPUT:

(MS-Paint, Notepad, Command Prompt, Explorer will open infinitely)

RESULT:

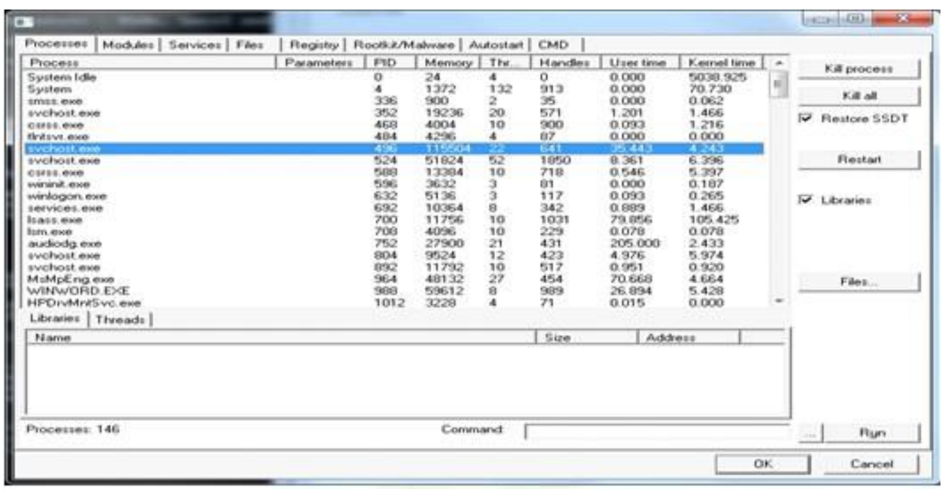
<table>
<tr><td>EX. NO:
11(b) DATE:</td><td>DEFEATING MALWARE - ROOTKIT HUNTER</td></tr>
</table>

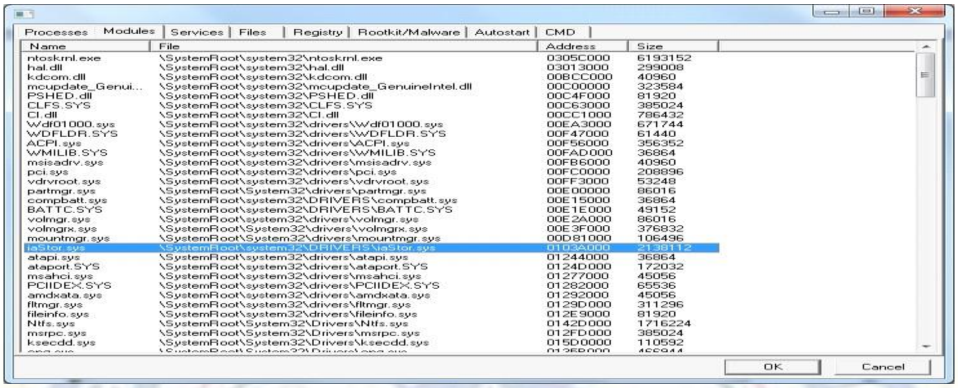
AIM:

PROCEDURE:

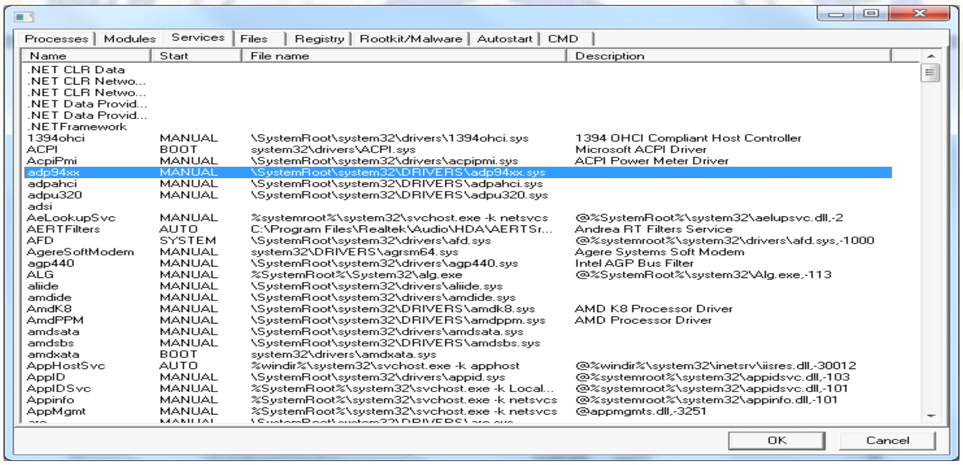
Rootkit is a stealth type of malicious software designed to hide the existence of certain process from normal methods of detection and enables continued privileged access to a computer.

1. Download Rootkit Tool from GMERwebsite.www.gmer.net
2. This displays the Processes, Modules, Services, Files, Registry, RootKit /Malwares, Auto start, and CMD of local host.
3. Select Processes menu and kill any unwanted process if any. Modules menu displays the various system files like .sys,.dll
4. Services menu displays the complete services runningwith Auto start, Enable, Disable, System, Boot.
5. Files menu displays full files on Hard-Disk volumes.
6. Registry displays Hkey_Current_user and Hkey_Local_Machine. Rootkits/Malwares scans the local drives selected.
7. Auto start displays the registry base Auto start applications.
8. CMD allows the user to interact with command line utilities or Registry.





Processes
Modules
Services
Files
Registry
Rootkit/Malware
Autostart
CMD
Name
File
Address
Size
OK
Cancel



Processes
Modules
Services
Files
Registry
Rootkit/Malware
Autostart
CMD
Name
Start
File name
Description
OK
Cancel

RESULT:

EX. NO: 12 (a) DATE:	PERFORM ETHICAL HACKING-BASED (i)PORT SCANNING USING THE NMAP TOOL

AIM:

To perform ethical port scanning using the **Nmap** tool to discover open, closed, and filtered ports on a target system, analyze running services and versions, and assess the target's attack surface—while following legal and ethical guidelines (authorized testing only).

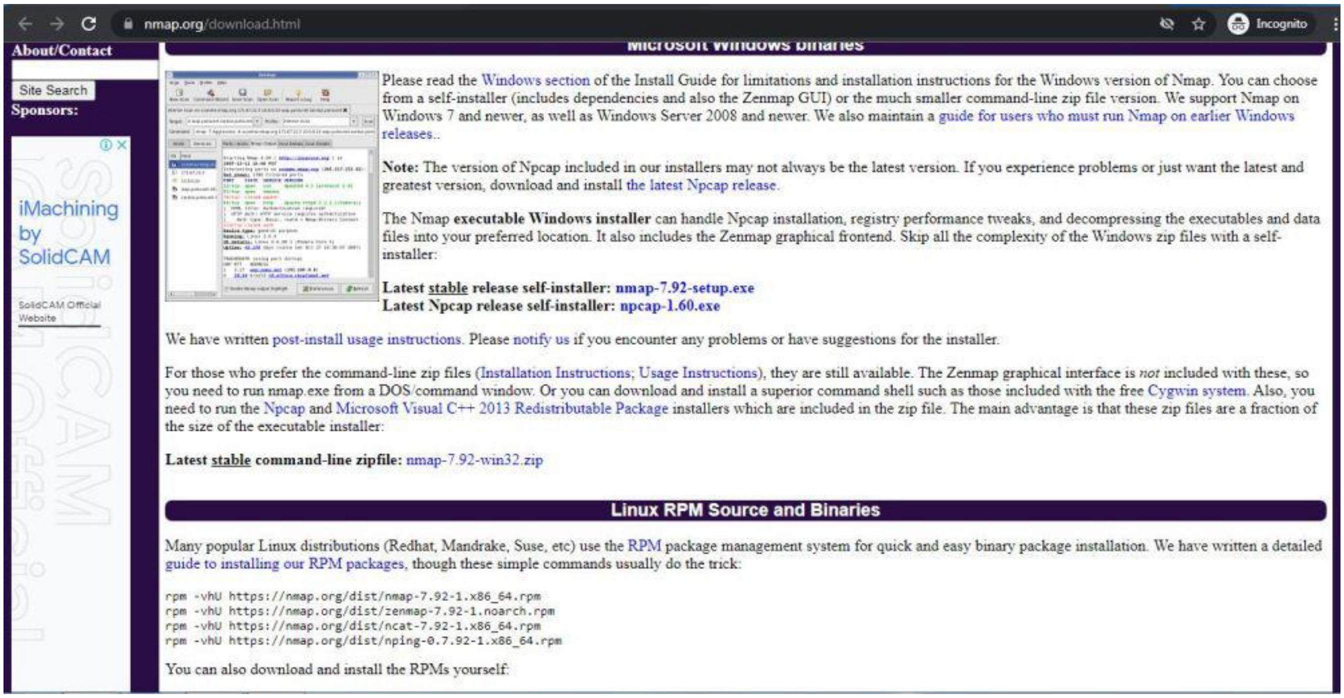
Nmap is computer software that is used to scan networks. It was developed by Gordon Lyon. It is written in C, C++, Python, and Lua. Its initial release was in 1997, and its stable release was in 2021. Its latest version is 7.92. It is free software used for security purposes of networks. It can be run on different operating systems like Windows, Mac, Linux, etc. It is used to protect computers by discovering hosts and services on a computer network by sending data packets and analyzing the responses. Some of the basic features of Nmap are:

- It discovers hosts on a network.
- It also scans open ports on the target hosts.
- It also finds the application name and the version number by interacting with network services on remote devices.
- It also finds the OS and hardware characteristics of the network devices.
- It also does scriptable interaction with the target with the help of NSE(Nmap Scripting Engine) and Lua programming languages.
- It is open-source software and is available for most operating systems.

Installing Nmap on Windows

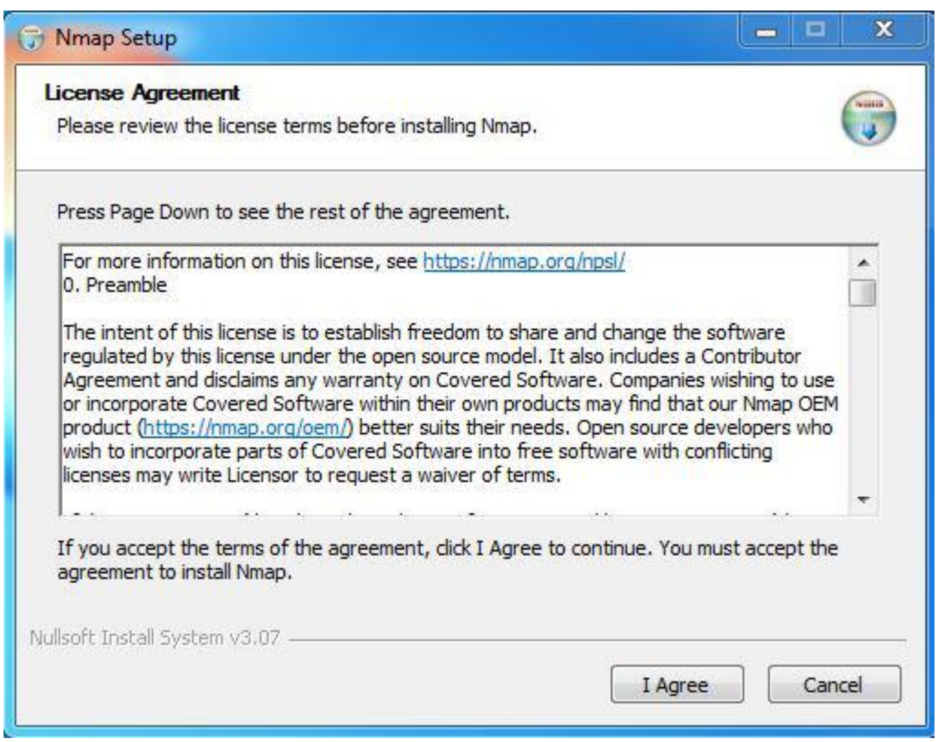
Follow the below steps to install Nmap on Windows:

Step 1: Visit the official website using the URL https://nmap.org/download.html on any web browser the click on **nmap-7.92-setup.exe**. Downloading of this executable file will start soon. It is a 21.8 MB file so it will take some minutes.



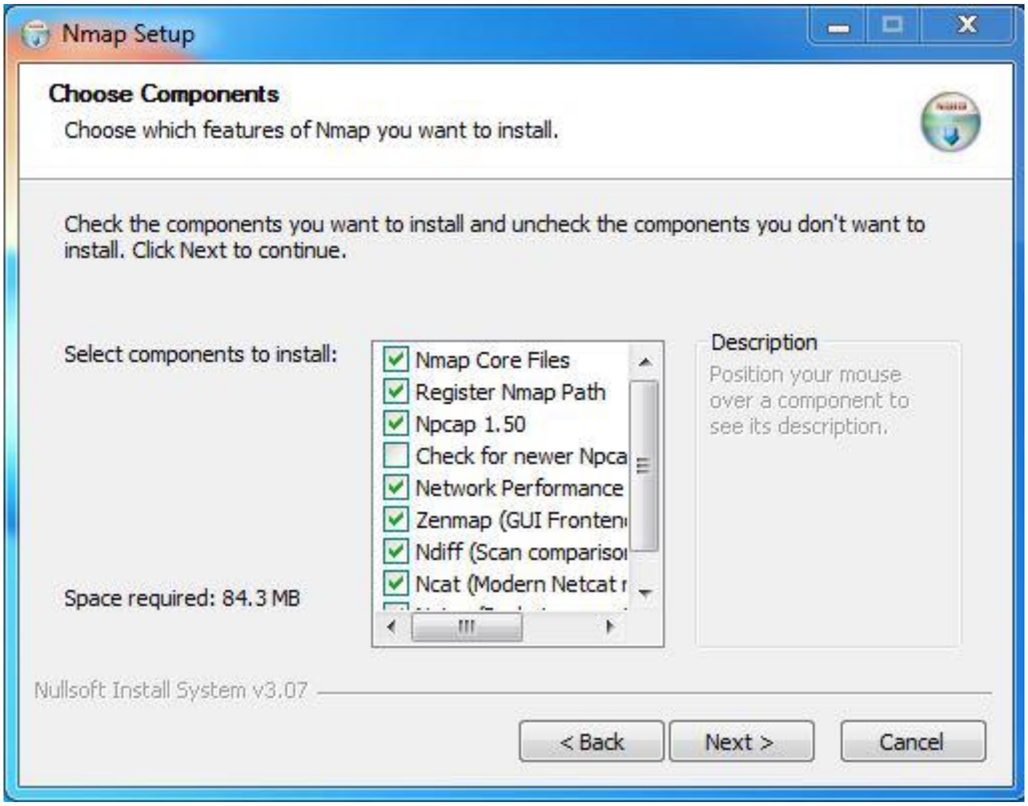
Step 2: Now check for the executable file in downloads in your system and run it.

Step 3: It will prompt confirmation to make changes to your system. Click on **Yes**.
Step 4: The next screen will be of License Agreement, click on **I Agree**.

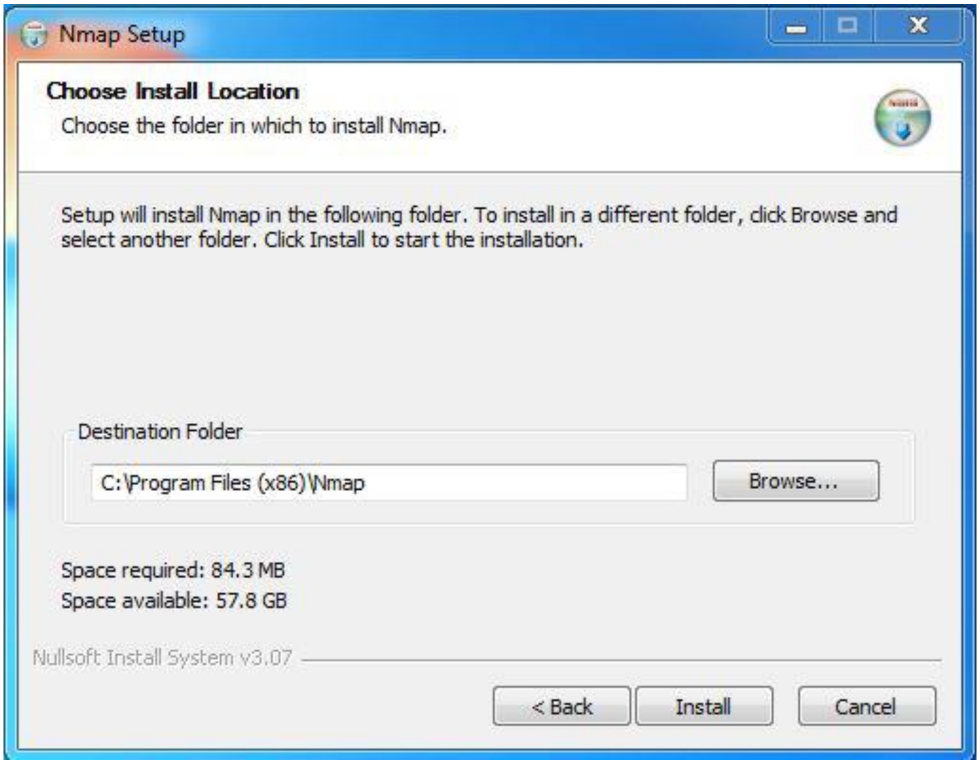


Step 5: Next screen is of choosing components, all components are already marked so don't change

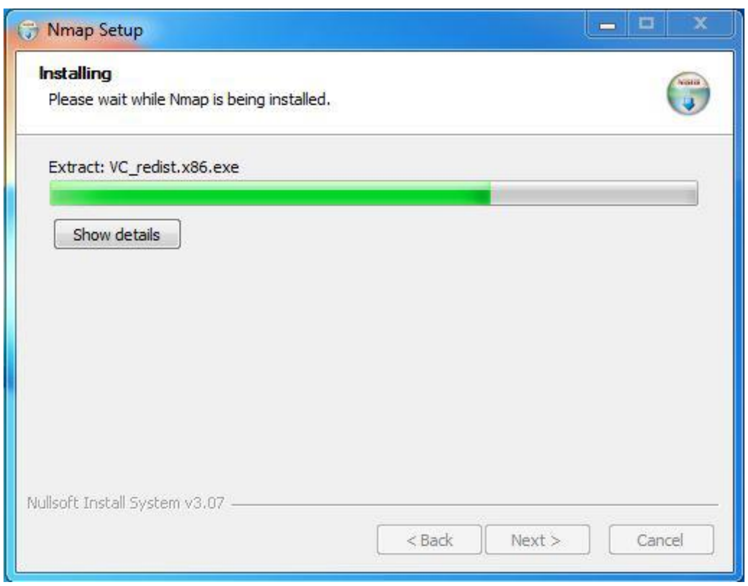
anything just click on the **Next** button.



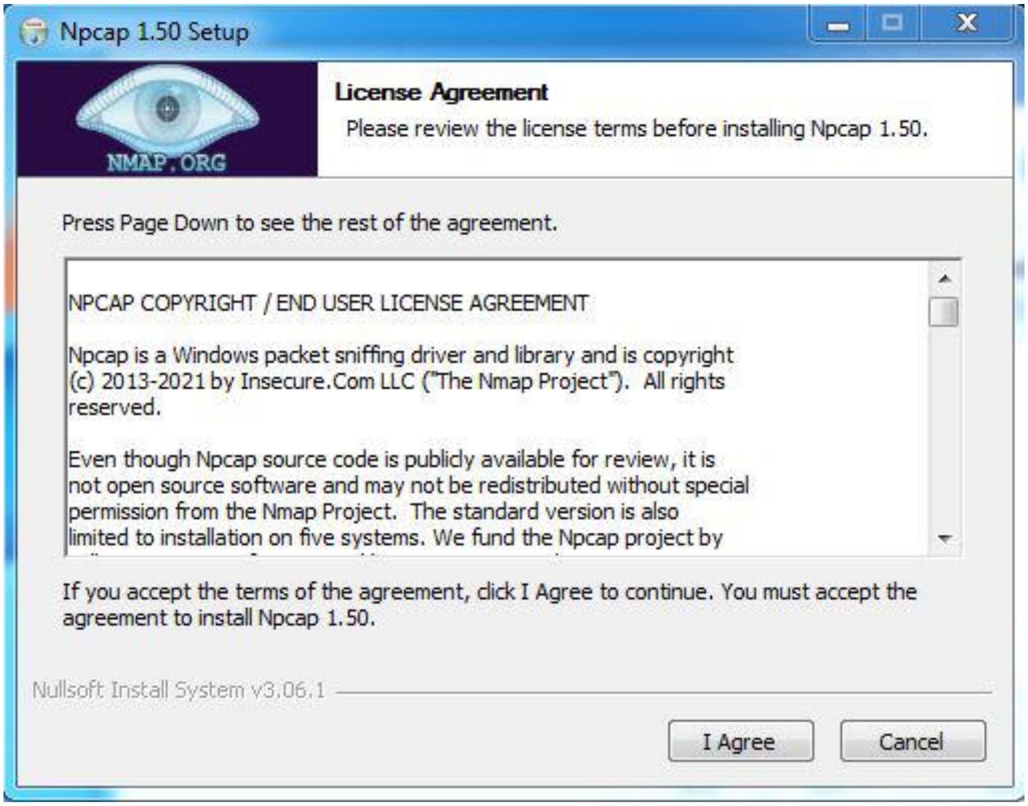
Step 6: In this step, we choose the installation location of Nmap. By default, it uses the C drive but you can change it into another drive that will have sufficient memory space for installation. It requires 84.3 MB of memory space.



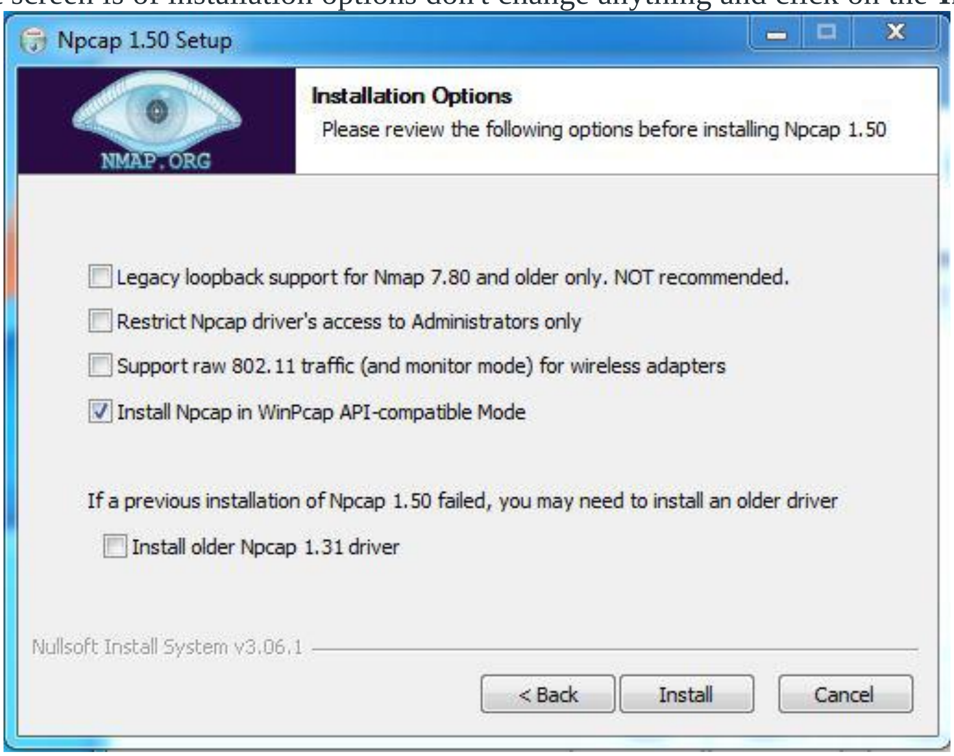
Step 7: After this installation process it will take a few minutes to complete the installation.



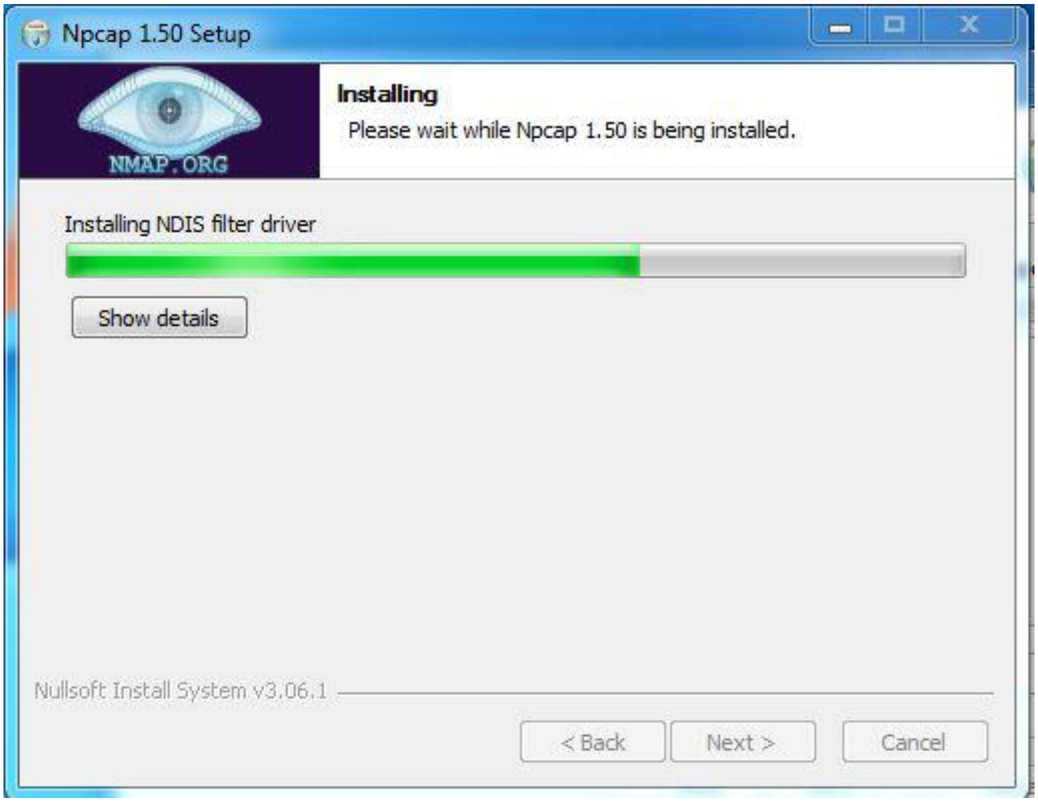
Step 8: Npcap installation will also occur with it, the screen of License Agreement will appear, click on **I Agree**.



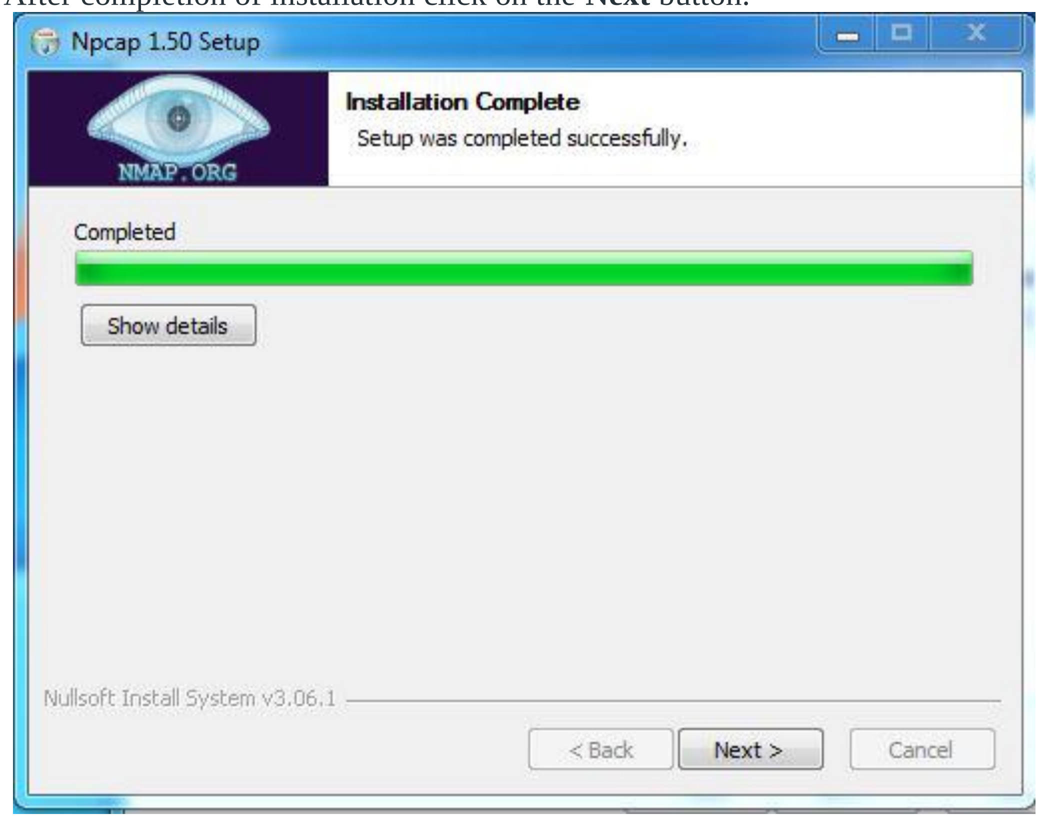
Step 9: Next screen is of installation options don't change anything and click on the **Install** button.



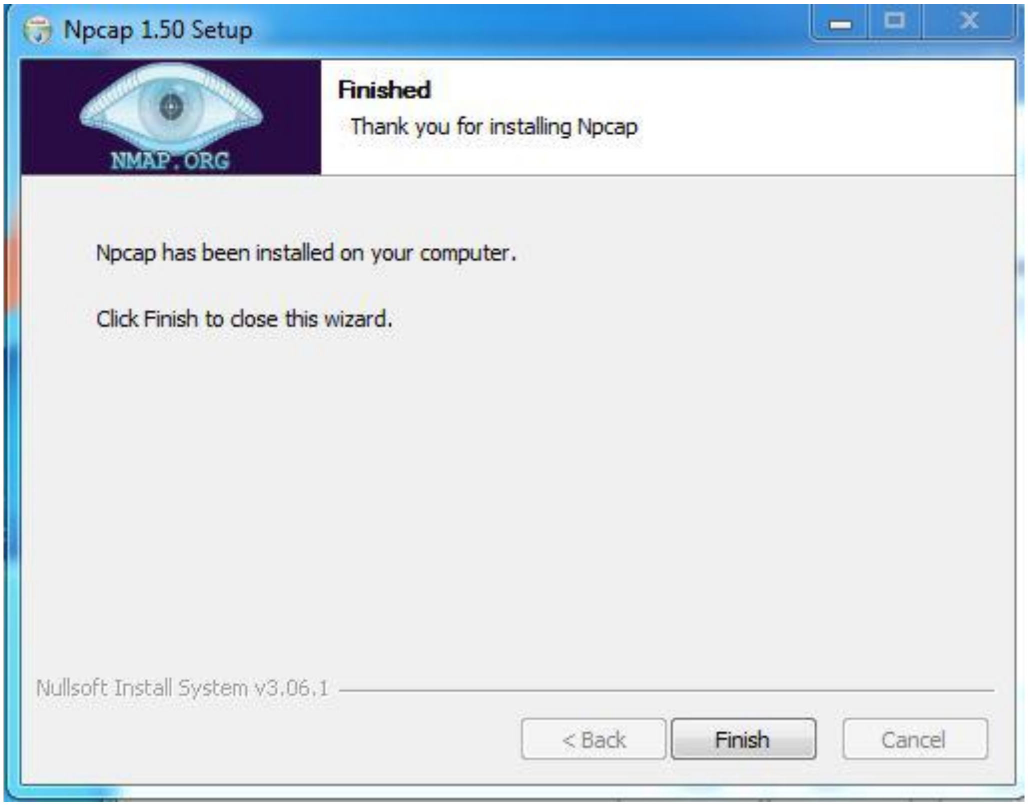
Step 10: After this installation process it will take a few minutes to complete the installation.



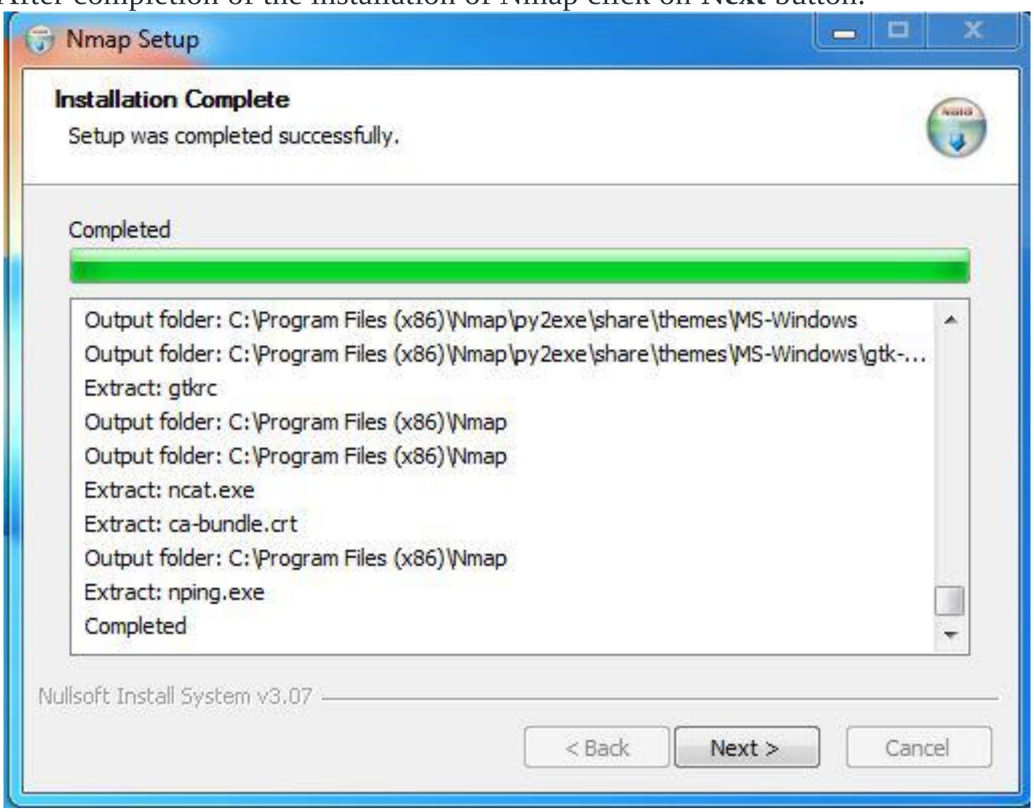
Step 11: After completion of installation click on the **Next** button.



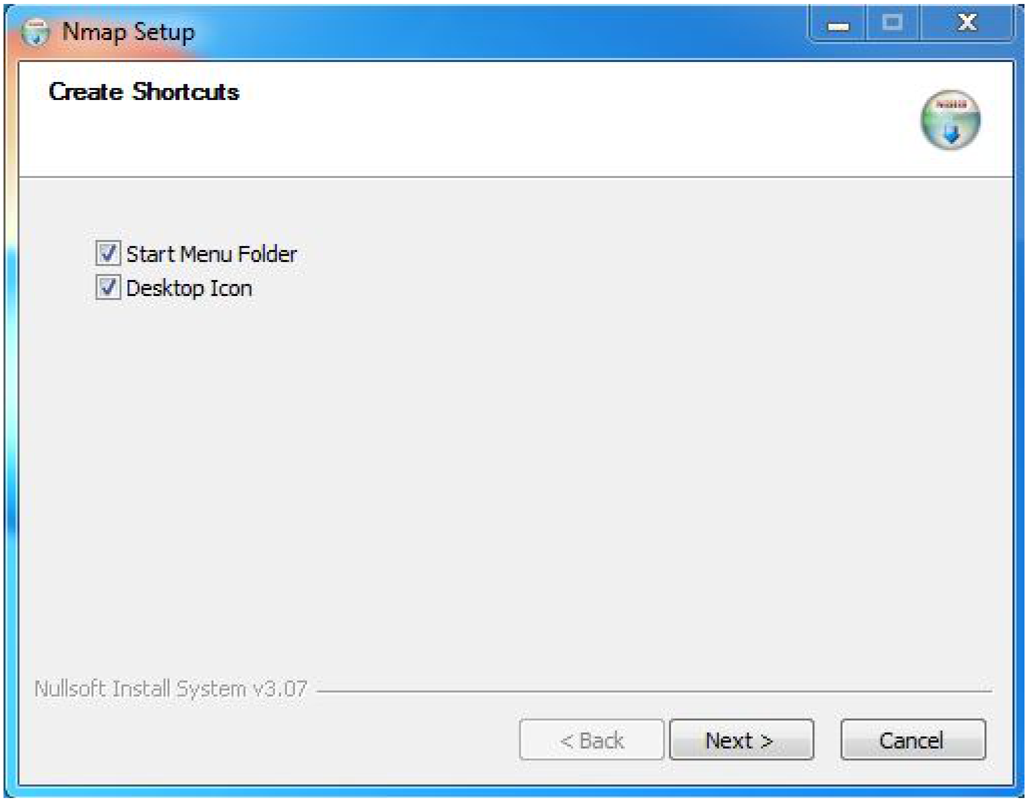
Step 12: Click on the **Finish** button to finish the installation of Npcap.



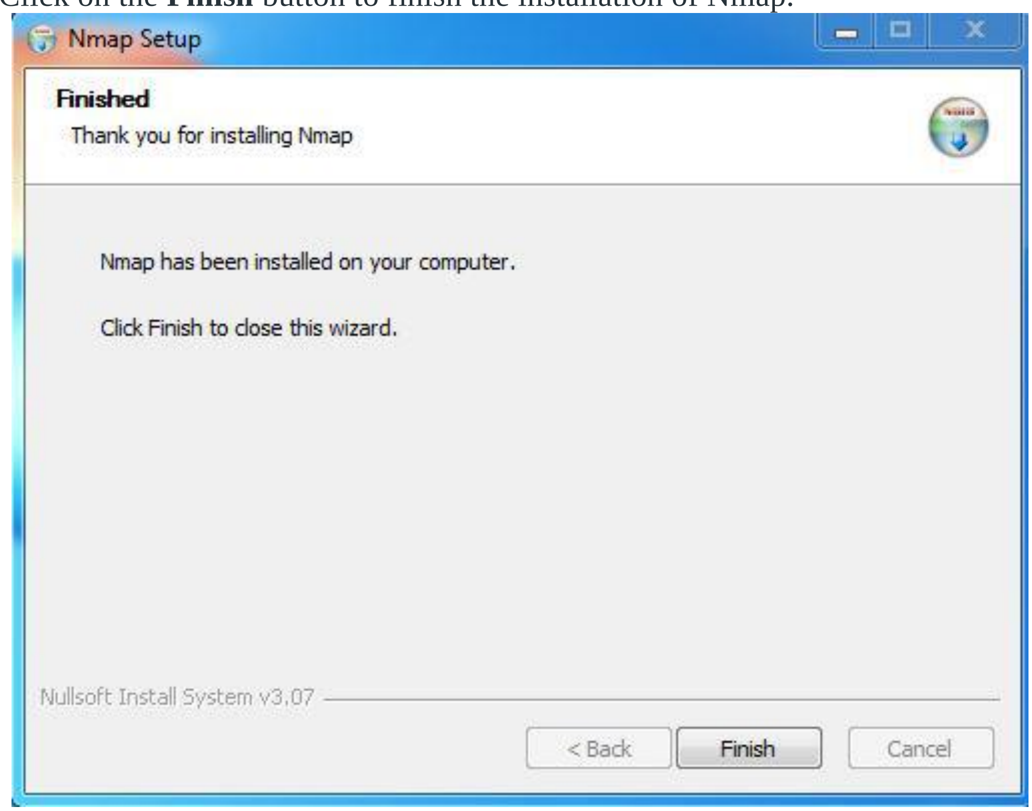
Step 13: After completion of the installation of Nmap click on **Next** button.



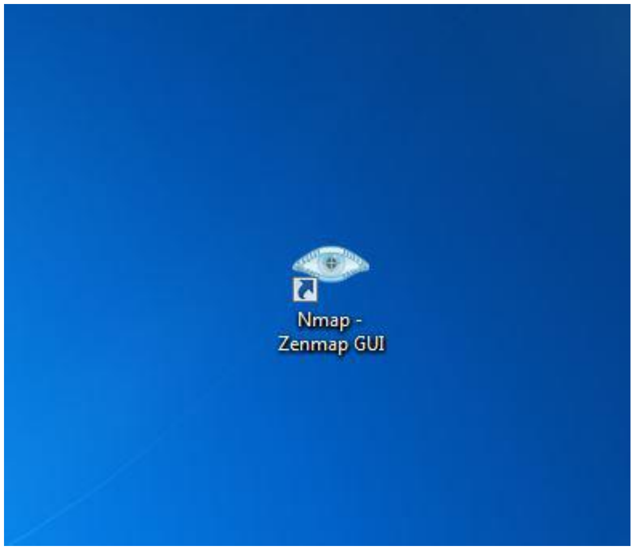
Step 14: Screen for creating shortcut will appear, click on **Next** button.



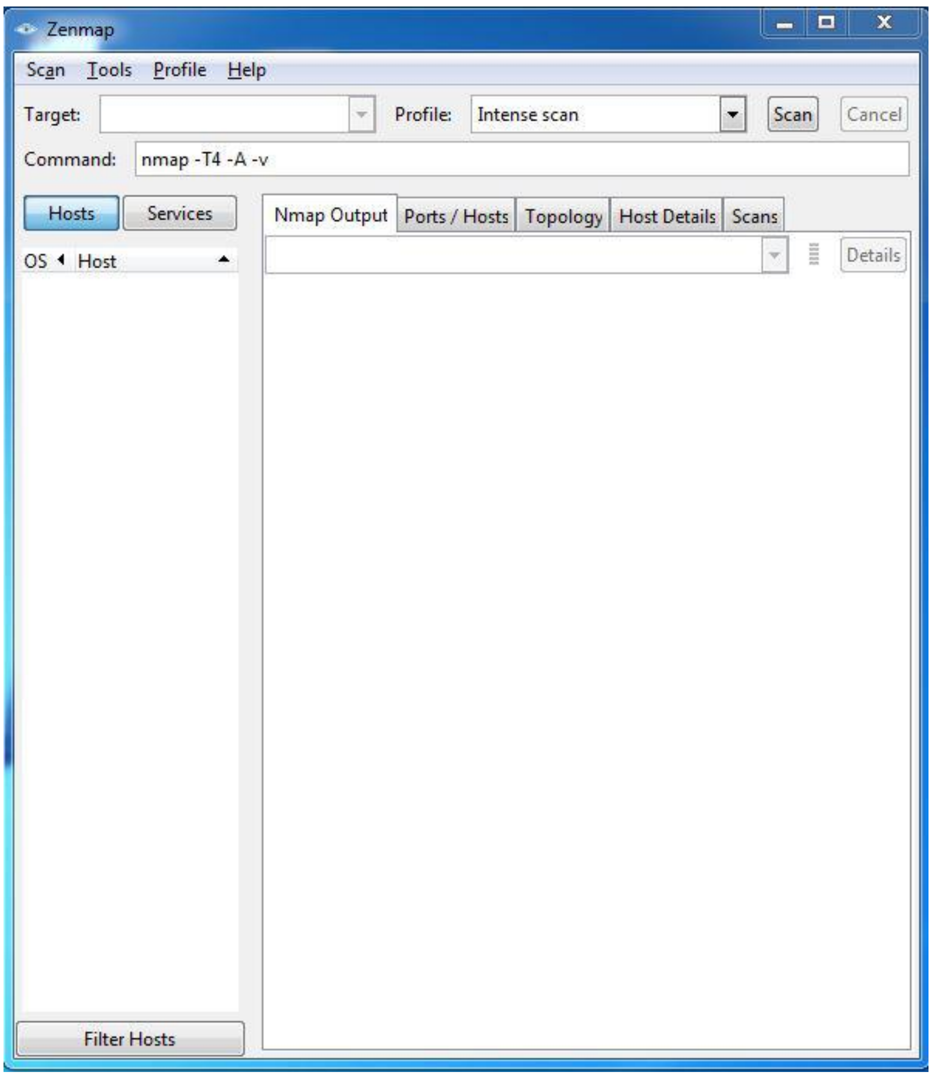
Step 15: Click on the **Finish** button to finish the installation of Nmap.



Step 16: Nmap is successfully installed on the system and an icon is created on the desktop.

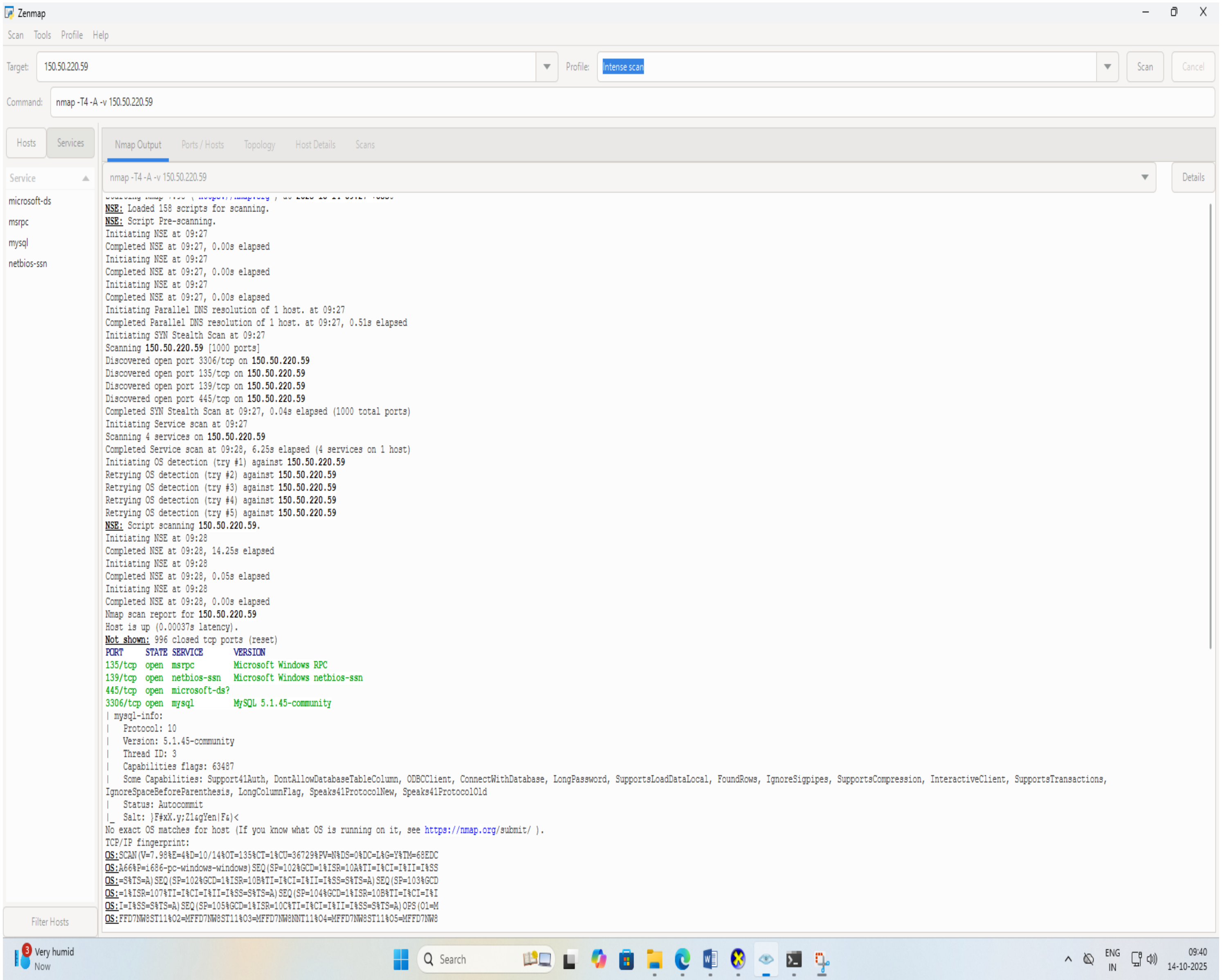


Step 17: Run the software and see the interface.



So this is how you have successfully installed Nmap on your windows system.

Output: Enter the IP Address: 150.50.220.59 and Scan



RESULT:

EX. NO: 12 (b) DATE:	PERFORM ETHICAL HACKING-BASED NETWORK SNIFFING USING WIRESHARK

AIM:

To analyze and understand the behavior of network protocols by capturing and examining real-time network traffic using **Wireshark** network protocol analyzer.

Wireshark

Wireshark is a network protocol analyzer, sometimes called a *packet analyzer*, designed to provide visibility into network traffic occurring on a network or between machines. It lets us peer inside network traffic and examine the details of wireless and wired network traffic at a variety of levels, ranging from connection-level information to the bits constituting a given packet to the payload (data) contained within those packets. Wireshark also lets us view the information at multiple levels of the stack so that we can isolate, identify and debug network connections from the lowest levels all the way up the stack to the application layer.

Wireshark is an important tool for cybersecurity professionals when used ethically and legally. Threat actors, however, also use Wireshark to cause harm or in furtherance of illegal and unethical activities. It's incumbent on you to use it ethically and responsibly. If you're not sure if the way you intend to use it is legal or not, don't do it until you are sure. Take steps to ensure you stay on the right side of ethical guidelines. This includes following the law and organizational policies.

Many organizations have a policy that spells out the rights of individuals using the corporate network, including requirements for obtaining, analyzing and retaining network traffic dumps. The policy could also address the conditions under which monitoring network traffic is acceptable. If the policy requires approval or permission, obtain it. If it requires that executive teams sign off, make that happen.

From a functional standpoint, Wireshark's capabilities are straightforward. It enables practitioners to do the following:

- **View data traversing various networks**, including wired networks, such as Ethernet; wireless networks; Bluetooth networks; or virtual network interfaces, such as with Docker or a hypervisor.
- **Navigate and view the various layers of the stack**, including application-level protocols, such as HTTP/HTTPS; mail protocols, such as Post Office Protocol 3 and SMTP; and file-sharing protocols, such as Server Message Block and Common Internet File System. Lower down in the stack, we can view TCP/IP and User Datagram Protocol. Even lower in the stack, we can view artifacts such as Ethernet frames.

- **Record and capture traffic** for subsequent analysis.

Installations

1. Download and install Wireshark

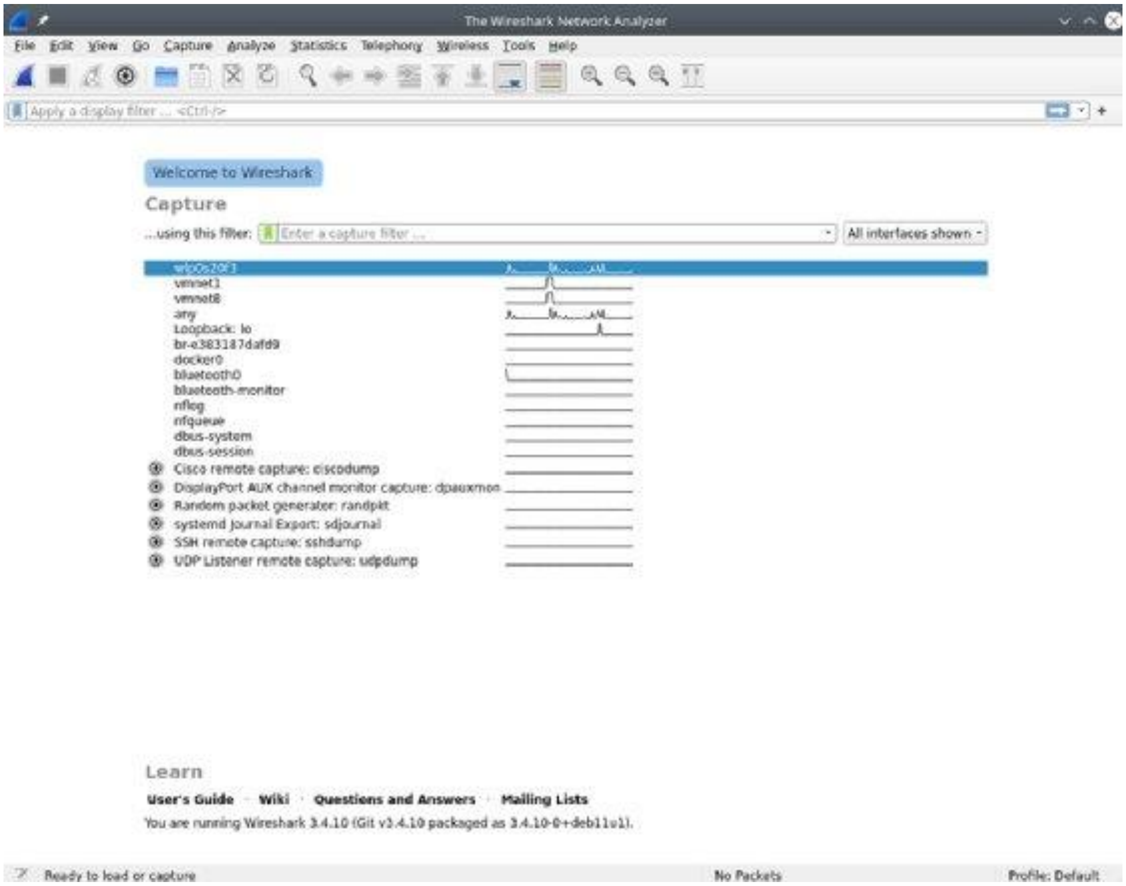
First, obtain and install the software:

- Download installation packages for Windows or Mac.
- Build your own Wireshark executable from source.
- Install the default package manager for many popular Linux distributions.
- Employ a specialized security Linux distribution, such as Kali, that has Wireshark installed by default.
- Add a portable copy of Wireshark on a USB drive to your incident response toolkit.
- Use a "live CD" or other bootable media as a portable network analyzer device.

. Run a simple packet capture

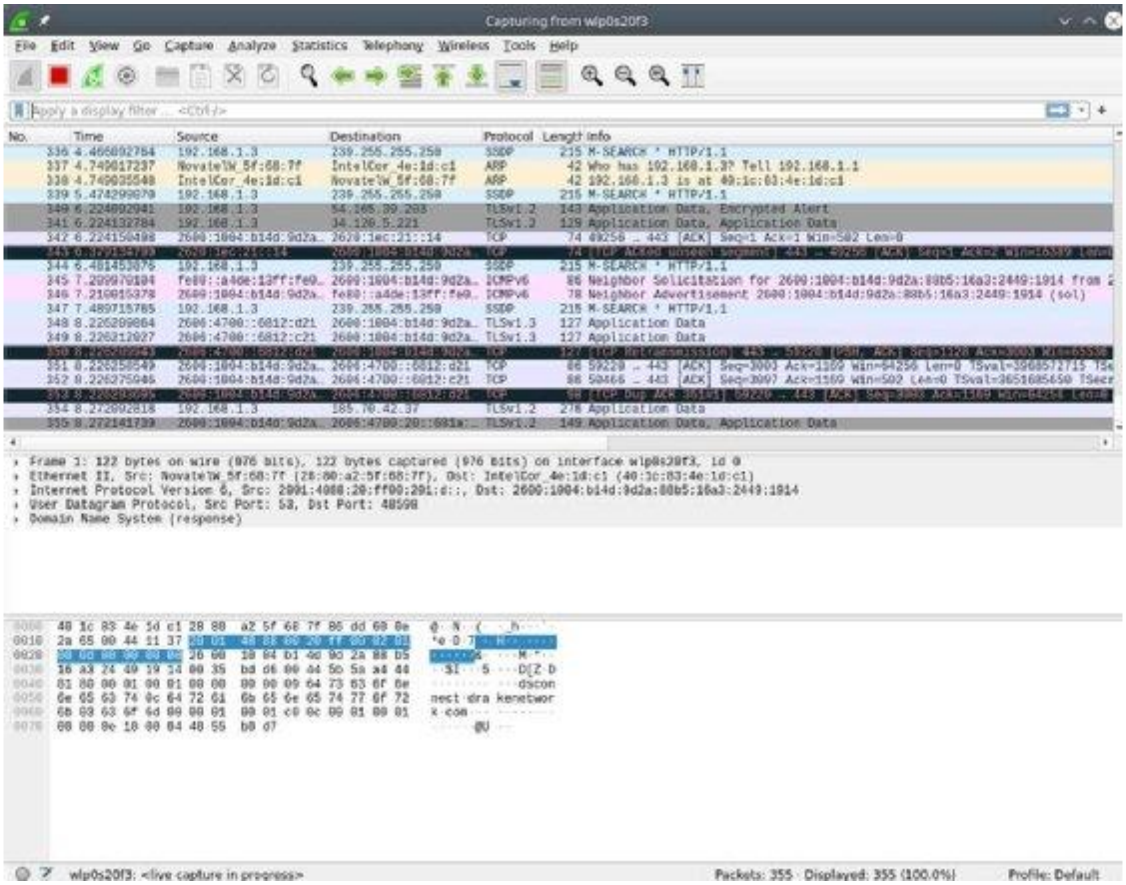
Once installed, launch Wireshark. One of the first things you see is a screen displaying the different network interfaces on the system, as well as a graph that indicates network activity on each network interface. Note that, in a Linux context specifically, low-level packet capture generally requires root access. Therefore, it may prompt you to elevate permissions to root on Linux.

The following screenshot shows quite a few network interfaces. Many are wired and internal interfaces that have no activity, as indicated by the flat lines. The top network interface -- a Wi-Fi interface -- shows activity, as indicated by the squiggly line.

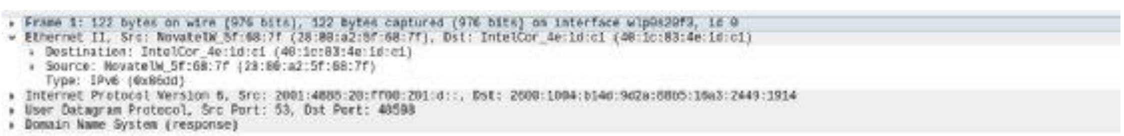


Double-click on the network interface that connects to the network you want to capture. Wireshark opens a window to show the packets being transmitted on the network. Wireshark offers many options for managing the display filters.

In the top pane, shown in the screenshot below, Wireshark displays information from the headers of each packet, including, by default, a time index showing the elapsed time between the start of the capture and when the packet was scanned. You can adjust the time format and save the timer data with the capture to recover the actual time a scanned packet was sent. The packet's source and destination IP addresses, the protocol in use, the length of the packet and information about the packet are also displayed. You can drill down and obtain more information by clicking on a row to display details of the packet in question.

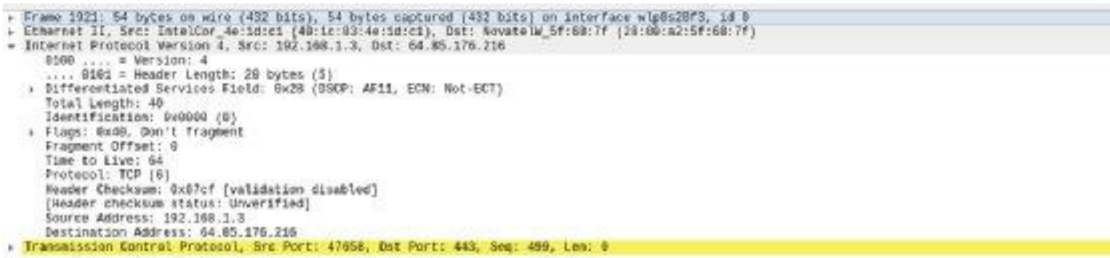


The middle pane contains drill-down details about the packet selected in the top frame. Select the > icons displayed on the left to reveal varying levels of detail about each layer of information contained within the packet. For example, here is the Ethernet header for an individual packet.



This header tells us the source and destination MAC addresses, as well as the identity of the next protocol in the stack: IPv6. We can then drill into the IPv6 header.

Here, we find the source and destination IP addresses, as well as IP-specific information. Like the layers of an onion, we can navigate up and down the stack observing detailed information about each layer along the way. Likewise, we can navigate into overlapping protocols. Pictured above is IPv6; here is a view of IPv4.

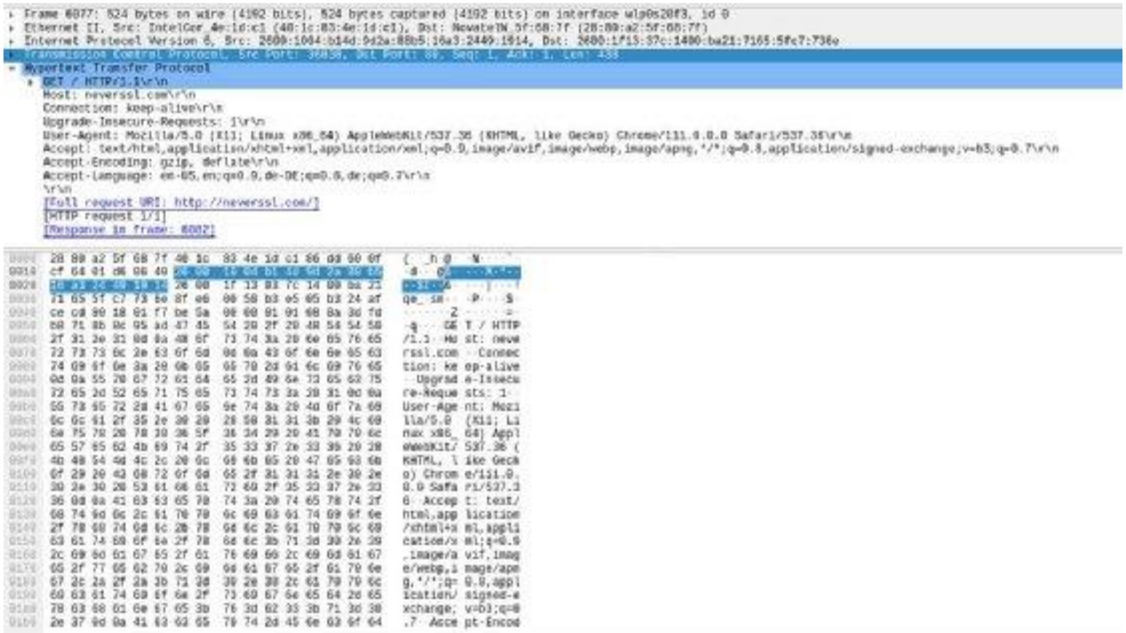


We can also view information about higher-level protocols, such as TCP.

This header includes information about the source and destination TCP ports, the flags set on the packet and other helpful troubleshooting details.

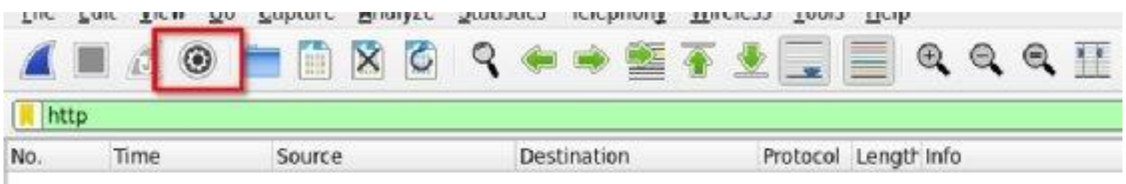
The bottom pane is a hexadecimal display that shows the digital contents of the packet itself. Highlighting any of the data displays the protocol details in the middle pane, as shown in the screenshot below.

One useful Wireshark feature is it helps point you to the most relevant information in the various protocols you examine using the tool. For example, the following screenshot illustrates the -- less common nowadays -- scenario of accessing an HTTP-only site, in this case neverssl.com, and the raw application-level traffic information presented in that case.

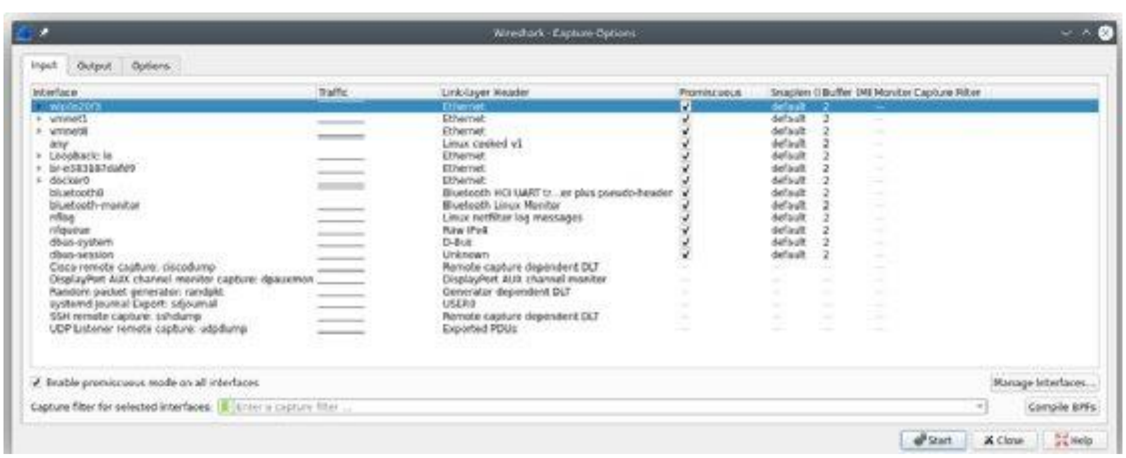


Modifying capture options

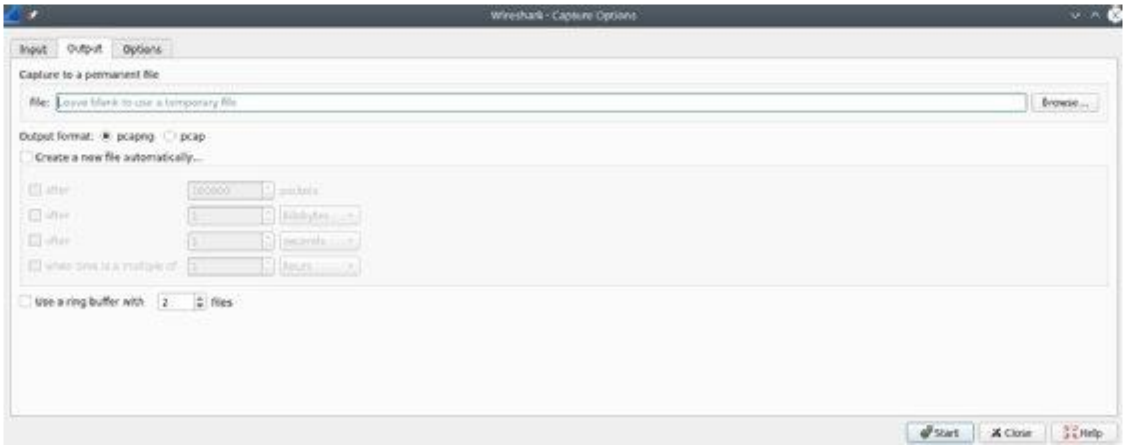
While it is simple to run a basic packet capture in Wireshark, the tool also enables users to modify several options to adjust their capture. You can access these options by clicking the gear-shaped Capture Options icon in the toolbar, as seen below.



Clicking this button opens the Capture Interfaces window, which has three tabs. The Input tab lets you modify Wireshark interfaces and enable promiscuous mode. This mode is what enables the interface to capture network traffic that is not directed specifically to your capture system.



The Output tab controls where Wireshark stores the packets it captures. You can automatically store captured packets in a file and modify the format of that file, or you can create a file based on the amount of data captured or the amount of time elapsed.



The Options tab offers choices for how to display the packets and options for MAC and DNS names, as well to limit the size of packet captures. Some of these options can help improve the performance of Wireshark. For example, you can adjust settings to prevent name issues, as they otherwise slow down your capture system and generate many name queries. Time and size limits can also place limitations on unattended captures.

3. Interpret and analyze packet contents

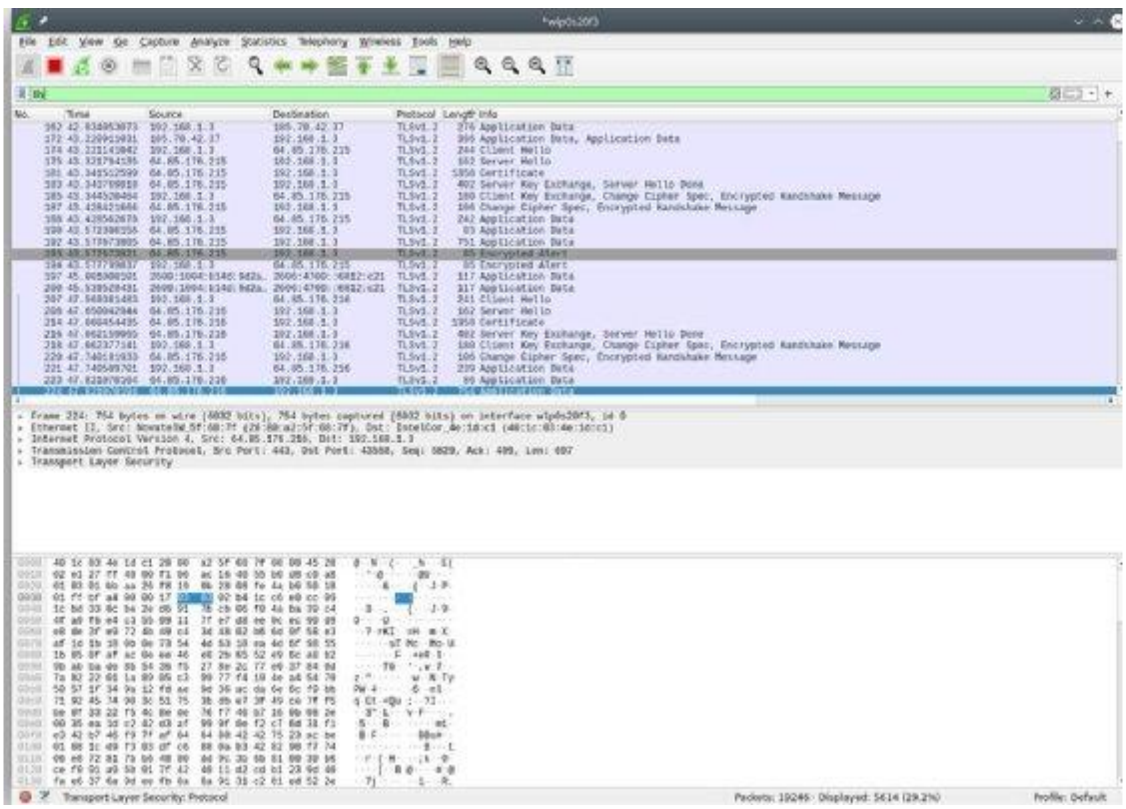
The single most useful analysis feature of Wireshark, in my opinion, is filters. Every day, more and more data traverses the network. This makes the proverbial haystack (data) much larger and the proverbial needles (information germane to what we're trying to discover) much harder to find.

Getting the right data without the right filters is nearly impossible on a network of even moderate size, let alone a busy or large one.

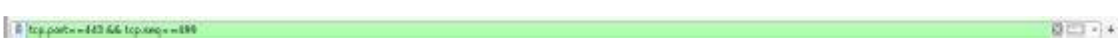
How to set up a Wireshark display filter

Wireshark has multiple types of filters. You can sort through captured data using a display filter. As the name suggests, this filter limits what is shown on the screen. This small, innocuous-seeming edit box is arguably the most powerful control in the entire tool. The purpose of a display filter is to assist with analysis.

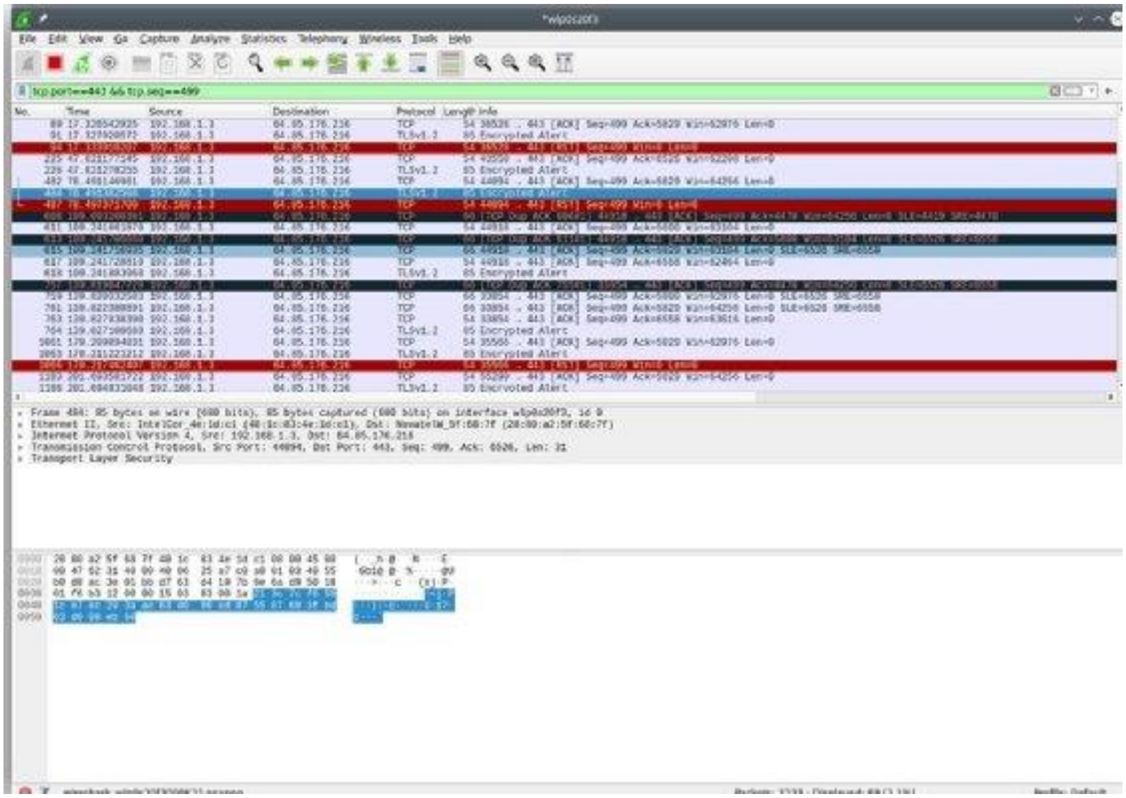
You can specify what you want to see -- no more and no less. The simplest filter is to limit the protocol(s) shown. For example, putting the statement "tls" into the filter bar limits results to only TLS traffic.



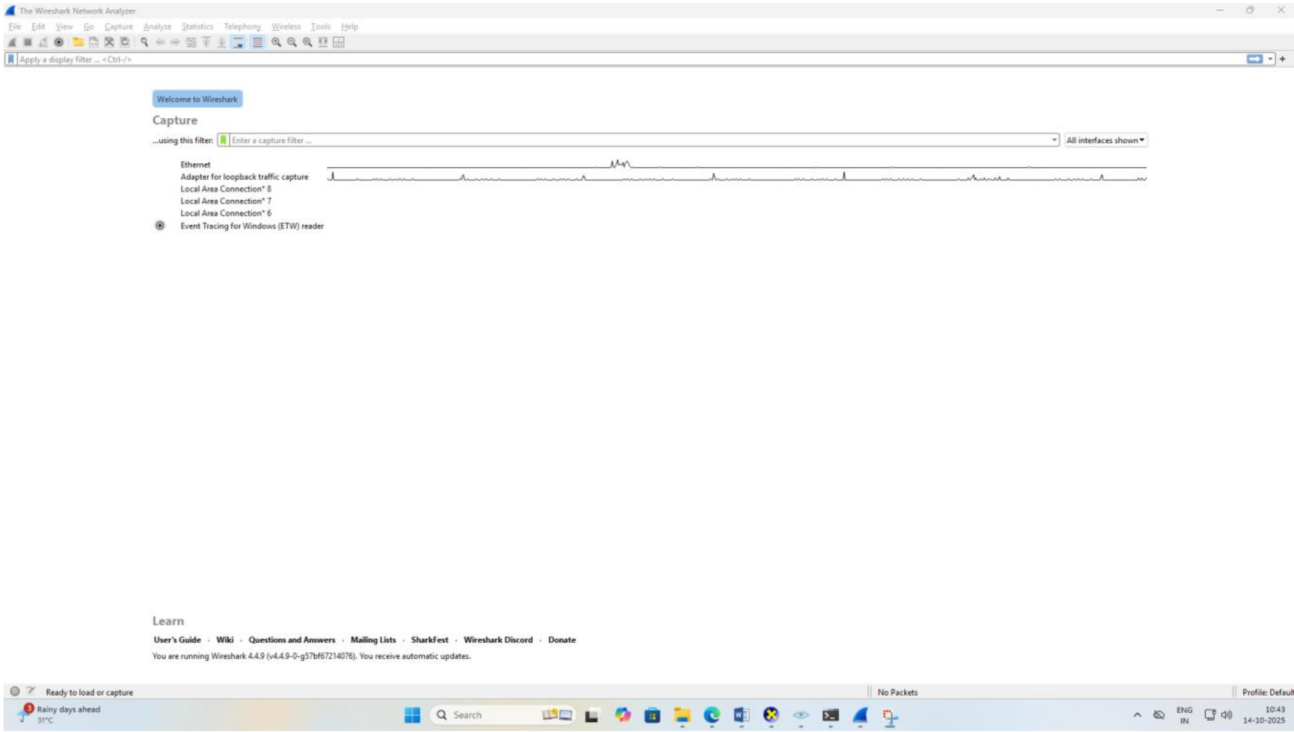
Filters can be much more complicated than this. The example below limits the display to results where the TCP port is 443 and the TCP sequence number is 499.



Why is this useful? Beats me. But the important thing is that you can filter for it if you need to. The results of the filter are below.



There's a bit of an art to setting up a filter. Wireshark attempts to help you find what you're looking for by suggesting how to complete your filter expression. For example, if you type "ip" into the filter bar, Wireshark pre-populates possible properties or subelements of IP that could be relevant. Likewise, typing "tcp" causes Wireshark to pre-populate the subelements of TCP.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark
Capture
...using this filter:
Enter a capture filter ...
All interfaces shown
Ethernet
Adapter for loopback traffic capture
Local Area Connection* 8
Local Area Connection* 7
Local Area Connection* 6
Event Tracing for Windows (ETW) reader
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists · SharkFest · Wireshark Discord · Donate
You are running Wireshark 4.4.9 (v4.4.9-0-g57bf67214076). You receive automatic updates.
Ready to load or capture
No Packets
Profile: Default
Rainy days ahead
31°C
Search
ENG
IN
10:43
14-10-2025

RESULT:

Ex. No: 13 Date:	**Vigenere Cipher**

AIM:

ALGORITHM:

Step 1:Create a 26×26 table with **A** to **Z** as the row heading and column heading This table is usually referred to as the **Vigenère Tableau**, **Vigenère Table** or **Vigenère Square**.

Step 2:Enter Plain Text and Keyword.

Step 3: Repeat a keyword, so that the total length is equal to that of the plaintext.

Step 4:To Encrypt pick a letter in the plaintext and its corresponding letter in the keyword, use the keyword letter and the plaintext letter as the row index and column index respectively , and the entry at the row-column intersection is the letter in the cipher text.

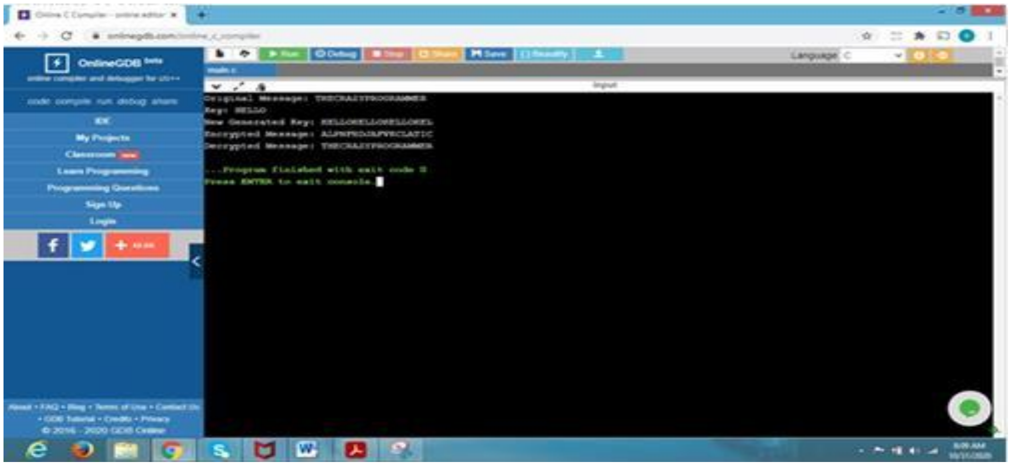
Step 5: Repeat the Step 4 until all plaintext letters are processed and converted in to cipher text. **Step 6:** To decrypt, pick a letter in the cipher text and its corresponding letter in the keyword, use the keyword letter to find the corresponding row, and the letter heading of the column that contains the cipher text letter is the needed plaintext letter.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int main(){
   char msg[] = "THECRAZYPROGRAMMER";
   char key[] = "HELLO";
   int msgLen = strlen(msg), keyLen = strlen(key), i, j;
   char newKey[msgLen], encryptedMsg[msgLen], decryptedMsg[msgLen];
   for(i = 0, j = 0; i < msgLen; ++i, ++j){
      if(j ==
         keyLen) j =
         0;
      newKey[i] = key[j];}
   newKey[i] = '\0';
   for(i = 0; i < msgLen; ++i)
      encryptedMsg[i] = ((msg[i] + newKey[i]) % 26) + 'A';
   for(i = 0; i < msgLen; ++i)
      decryptedMsg[i] = (((encryptedMsg[i] - newKey[i]) + 26) % 26) + 'A';
```

```
decryptedMsg[i] = '\0';printf("Original Message: %s", msg);
printf("\nKey: %s", key);printf("\nNew Generated Key: %s", newKey);
printf("\nEncrypted Message: %s", encryptedMsg); printf("\nDecrypted
Message: %s", decryptedMsg);return 0;}
```

OUTPUT:



RESULT:

EX. NO: 14 DATE:	EUCLID ALGORITHM WITH TIME

AIM:

ALGORITHM:

For two given numbers a and b, such that a >=b:
if b | a, then gcd(a, b) = b,
otherwise gcd(a, b) = gcd(b, a mod b)
gcd(a, b) = gcd(b, r), where r = a mod b and a = b · t + r:

- Firstly, let d = gcd(a, b). We get d | (b · t + r) and d | b, so d | r. Therefore we get gcd(a, b) | gcd(b, r).
- Secondly, let c = gcd(b, r). We get c | b and c | r, so c | a. Therefore we get gcd(b, r) | gcd(a, b).

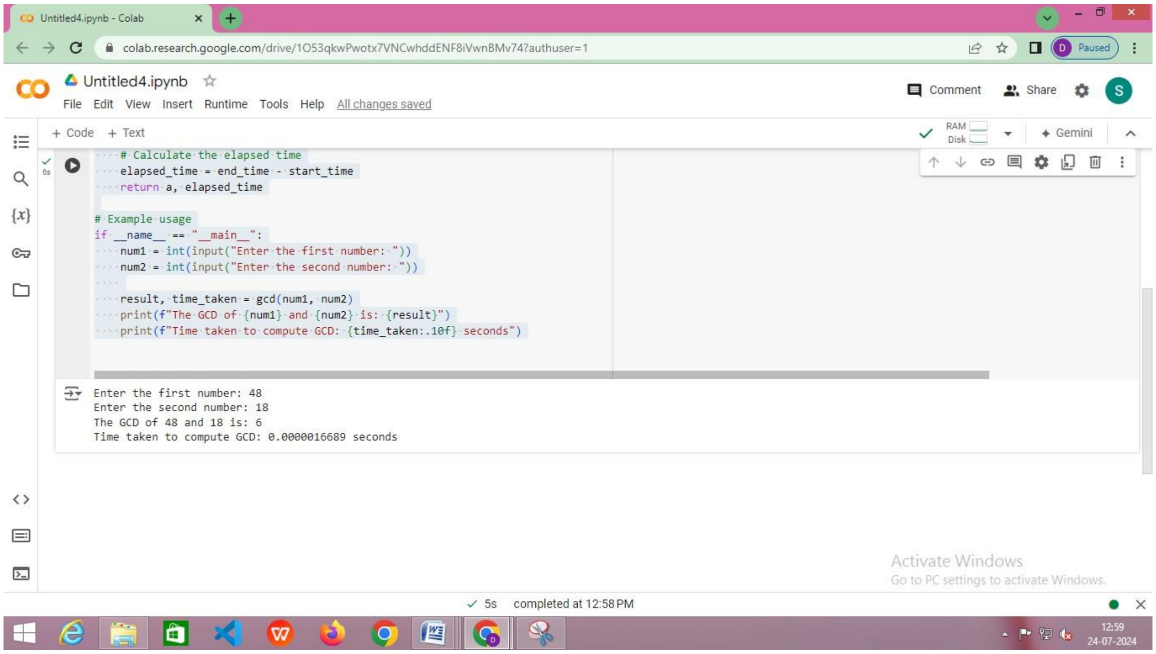
Hence gcd(a, b)

1. Start timing the execution
2. Euclidean algorithm to find GCD
3. End timing the execution
4. Calculate the elapsed time

PROGRAM:

```
import time
def gcd(a, b):
    # Start timing the execution
    start_time = time.time()
    # Euclidean algorithm to find GCD
    while b != 0:
        a, b = b, a % b
        # End timing the execution
    end_time = time.time()
        # Calculate the elapsed time
    elapsed_time = end_time - start_time
    return a, elapsed_time
# Example usage
if __name__== "__main__":
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number:
    ")) result, time_taken = gcd(num1, num2)
    print(f"The GCD of {num1} and {num2} is: {result}")
    print(f"Time taken to compute GCD: {time_taken:.10f}
    seconds")
```

OUTPUT:



RESULT: