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 Assignment

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	Md Rasel Hossain Student Id: 180118 4th Year 2nd Semester Session: 2018-19 Dept. of Computer Science and Engineering Jashore University of Science and Technology.

Date of Submission: 12/01/2024

Contents

1. Implementation of Caesar Cipher Encryption and Decryption.

Introduction

The Caesar Cipher is a substitution cipher that shifts each letter in the plaintext by a specified number of positions down the alphabet. It's named after Julius Caesar, who reportedly used it for private communication. While relatively simple, it serves as a foundational concept in cryptography.

Objective

The primary objective of this lab was to implement the Caesar Cipher algorithm in C++, enabling both encryption and decryption of text messages.

Algorithm

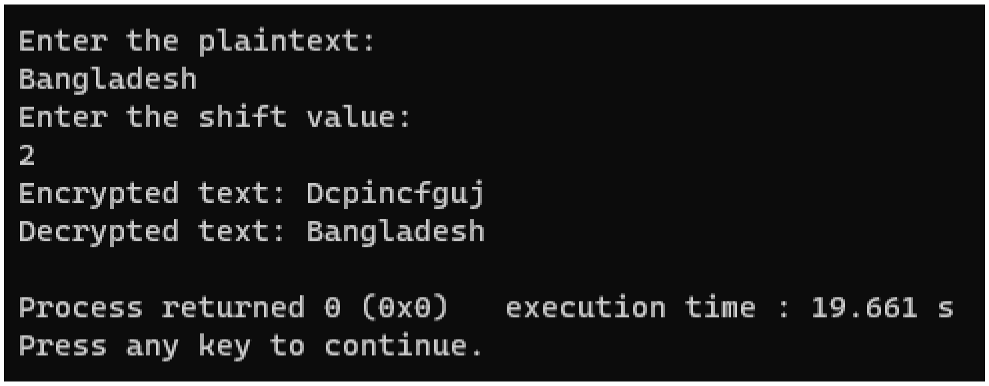
- Define a function for encryption and decryption, each taking plaintext and shift value as input.
- Iterate through each character in the plaintext.
- If it's a letter:
  - Shift its position in the alphabet by the specified value.
  - Wrap around to the beginning if it passes the end of the alphabet.
  - Append the shifted character to the ciphertext.
- Otherwise, append the character as is to the ciphertext.

Code Implementation

```
#include<iostream>
#include<string>
using namespace std;

string encrypt(string plaintext, int shift)
 {
    string ciphertext = "";

    for(char& ch: plaintext)
    {
       if(isalpha(ch))
       {
          char base = isupper(ch)? 'A':'a';
```

```
            ch = (ch - base + shift)%26 + base;
        }
        ciphertext += ch;
    }
    return ciphertext;
}

string decrypt(string ciphertext, int shift)
{
    return encrypt(ciphertext, 26 - shift);
}

int main()
{
    string plaintext, encrypted, decrypted;
    int shift;

    cout<<"Enter the plaintext: "<<endl;
    getline(cin, plaintext);

    cout<<"Enter the shift value: "<<endl;
    cin>> shift;

    encrypted = encrypt(plaintext, shift);
    cout<<"Encrypted text: "<< encrypted<<endl;

    decrypted = decrypt(encrypted, shift);
    cout<<"Decrypted text: "<< decrypted<<endl;

    return 0;
}
```

Output


```
Enter the plaintext:
Bangladesh
Enter the shift value:
2
Encrypted text: Dcpincfguj
Decrypted text: Bangladesh

Process returned 0 (0x0)   execution time : 19.661 s
Press any key to continue.
```

Discussion

- ✓ The Caesar Cipher is relatively weak due to its limited key space.
- ✓ It can be easily broken using frequency analysis or brute force techniques.
- ✓ It serves as a basic introduction to cryptography and highlights the importance of strong encryption methods for modern security needs.

Conclusion

The lab successfully implemented the Caesar Cipher in C++, demonstrating its encryption and decryption capabilities. While not suitable for modern secure communication, it provides a foundation for exploring more advanced cryptographic algorithms.

2. Implementation of Vigenere Cipher Encryption and Decryption.

Introduction

The Vigenere Cipher is a polyalphabetic substitution cipher that improves upon the Caesar Cipher by using a keyword to determine the shift for each letter. This lab aims to guide participants through the implementation of the Vigenere Cipher, explaining its algorithm, and demonstrating the encryption and decryption processes.

Objective

The primary objective of this lab was to implement the Vigenere Cipher algorithm in C++, enabling both encryption and decryption of text messages using a keyword.

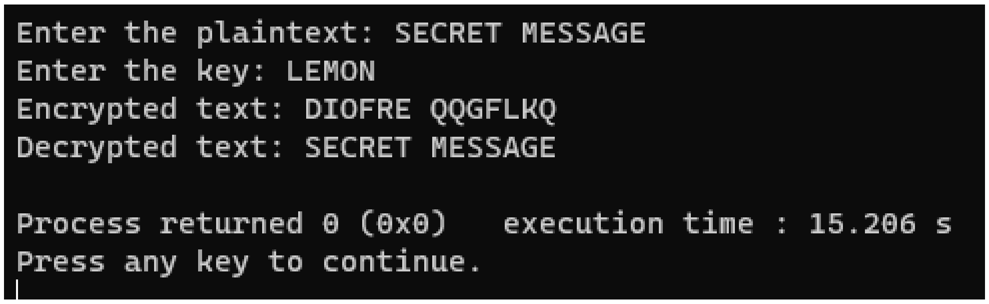
Algorithm

- Define functions for encryption and decryption, taking plaintext, keyword, and mode as input.
- Convert plaintext and keyword to uppercase for consistency.
- Iterate through each character in the plaintext:
  - Obtain the corresponding keyword character.
  - Determine the shift value based on the keyword character's position in the alphabet.
  - Shift the plaintext character by the calculated value.
  - Wrap around to the beginning of the alphabet if necessary.
  - Append the shifted character to the ciphertext.
- Repeat keyword characters if the plaintext is longer than the keyword.

Code Implementation

```
#include<iostream>
#include<string>
using namespace std;

string encrypt(string plaintext, string key)
{
    string ciphertext = "";
    int keyLength = key.length();

    for(int i=0, j=0; i< plaintext.length(); i++)
    {
        char currentChar = plaintext[i];

        if(isalpha(currentChar))
        {
            char base = isupper(currentChar)?'A':'a';
            char keyChar = key[j % keyLength];
            char encryptedChar;
            if(base =='A')
            {
                encryptedChar = ((currentChar - base + keyChar - 'A') %
26) + base;
            }
            else{
                encryptedChar = ((currentChar - base + keyChar - 'a') % 26)
+ base;
            }


            ciphertext += encryptedChar;
            j++;
        }
        else
        {
            ciphertext += currentChar;
        }
    }
    return ciphertext;
}

string decrypt(string ciphertext, string key)
{
    string plaintext = "";
    int keyLength = key.length();

    for(int i=0, j=0; i<ciphertext.length();i++)
    {
        char currentChar = ciphertext[i];

        if(isalpha(currentChar))
        {
            char base = isupper(currentChar)?'A':'a';
            char keyChar = key[j % keyLength];
            char decryptedChar;
            if(base == 'A')
            {
                decryptedChar = ((currentChar - base - (keyChar - 'A') + 26)
% 26) + base;
            }
            else{
                decryptedChar = ((currentChar - base - (keyChar - 'a') + 26)
% 26) + base;
            }

            plaintext += decryptedChar;
            j++;
        }
        else
        {
            plaintext += currentChar;
        }
    }
    return plaintext;
}

int main()
{
    string plaintext, key,encrypted,decrypted;

    cout<<"Enter the plaintext: ";
    getline(cin, plaintext);

    cout<<"Enter the key: ";
    getline(cin, key);

    encrypted = encrypt(plaintext, key);
    cout<<"Encrypted text: "<<encrypted<<endl;

    decrypted = decrypt(encrypted, key);
    cout<<"Decrypted text: "<<decrypted<<endl;
    return 0;
}
```

Output



```
Enter the plaintext: SECRET MESSAGE
Enter the key: LEMON
Encrypted text: DIOFRE QQGFLKQ
Decrypted text: SECRET MESSAGE

Process returned 0 (0x0)   execution time : 15.206 s
Press any key to continue.
```

Discussion

- ✓ The Vigenere Cipher is more secure than the Caesar Cipher due to its variable shifts.
- ✓ It resists simple frequency analysis attacks.
- ✓ However, it can be broken using techniques like Kasiski examination and Friedman's test.
- ✓ It highlights the importance of key length and complexity in encryption.

Conclusion

The lab successfully implemented the Vigenere Cipher in C++, demonstrating its encryption and decryption capabilities. While more secure than the Caesar Cipher, it's still vulnerable to advanced cryptanalysis. It serves as a stepping stone towards understanding modern cryptographic algorithms.

3. Implementation of Autokey Cipher Encryption and Decryption.

Introduction

The Autokey Cipher is a polyalphabetic substitution cipher that uses a keyword combined with the plaintext itself as the key for encryption. This lab aims to guide participants through the implementation of the Autokey Cipher, explaining its algorithm, and demonstrating the encryption and decryption processes.

Objective

The primary objective of this lab was to implement the Autokey Cipher algorithm in C++, enabling both encryption and decryption of text messages using an initial key.

Algorithm

- Define functions for encryption and decryption, taking plaintext, initial key, and mode as input.
- Convert plaintext and key to uppercase for consistency.
- Iterate through each character in the plaintext:
    - Obtain the corresponding key character.
    - If it's the first character, use the initial key.
    - Otherwise, use the previously encrypted character as the key character.
    - Determine the shift value based on the key character's position in the alphabet.
    - Shift the plaintext character by the calculated value.
    - Wrap around to the beginning of the alphabet if necessary.
    - Append the shifted character to the ciphertext and update the key.

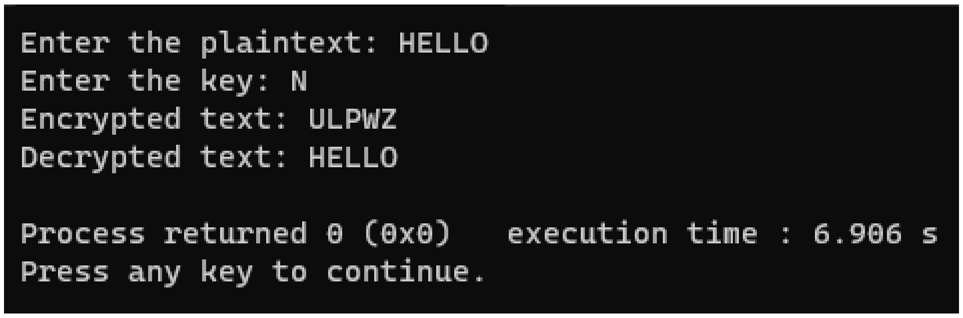
Code Implementation

```
#include<iostream>

#include<string>

using namespace std;

string encrypt(string plaintext, string key)

{

    string ciphertext = "";

    int keyLength = key.length();


    for(int i=0, j=0; i< plaintext.length();i++)

    {

        char currentChar = plaintext[i];
        if(isalpha(currentChar))
        {
            char base = isupper(currentChar)?'A':'a';
            char keyChar, encryptedChar;

            if(j<keyLength)
                keyChar = key[j++];
            else
                keyChar = plaintext[i - keyLength];

            if(base == 'A'){
                encryptedChar = ((currentChar - base + keyChar - 'A') %
26) + base;
            }
            else{
                encryptedChar = ((currentChar - base + keyChar - 'a') % 26)
+ base;
            }
            ciphertext += encryptedChar;
        }
        else{
            ciphertext += currentChar;
        }
    }
    return ciphertext;
```

```
}
string decrypt(string ciphertext, string key)
{
    string decryptedText = "";
    int keyLength = key.length();

    for(int i=0,j=0; i< ciphertext.length();i++)
    {
        char currentChar = ciphertext[i];
        if(isalpha(currentChar))
        {
            char base = isupper(currentChar)?'A':'a';
            char keyChar, decryptedChar;

            if(j<keyLength)
                keyChar = key[j++];
            else
                keyChar = decryptedText[i - keyLength];

            if(base == 'A'){
                decryptedChar = ((currentChar - base - (keyChar - 'A') + 26)
% 26) + base;
            }
            else{
                decryptedChar = ((currentChar - base - (keyChar - 'a') + 26)
% 26) + base;
```

```
            }
            decryptedText += decryptedChar;
        }
        else{
            decryptedText += currentChar;
        }
    }
    return decryptedText;
}

int main()
{
    string plaintext, key, encrypted, decrypted;
    cout<<"Enter the plaintext: ";
    getline(cin, plaintext);
    cout<<"Enter the key: ";
    getline(cin, key);
    encrypted = encrypt(plaintext, key);
    cout<<"Encrypted text: "<<encrypted<<endl;

    decrypted = decrypt(encrypted, key);
    cout<<"Decrypted text: "<<decrypted<<endl;
    return 0;
}
```

Output



```
Enter the plaintext: HELLO
Enter the key: N
Encrypted text: ULPWZ
Decrypted text: HELLO

Process returned 0 (0x0)   execution time : 6.906 s
Press any key to continue.
```

Discussion

- ✓ The Autokey Cipher is more secure than the Vigenère Cipher due to its dynamic key generation.
- ✓ It resists frequency analysis attacks more effectively.
- ✓ However, it's still vulnerable to advanced cryptanalysis techniques like the Friedman test.
- ✓ It highlights the importance of key management and the limitations of classical ciphers.

Conclusion

The lab successfully implemented the Autokey Cipher in C++, demonstrating its encryption and decryption capabilities. While more secure than the Vigenère Cipher, it's not unbreakable. It serves as an example of the evolution of cryptographic techniques and the importance of modern algorithms for secure communication.

4. Implementation of Rail Fence Cipher Encryption and Decryption.

Introduction

The Rail Fence Cipher is a transposition cipher that rearranges the characters of a plaintext by writing it in a zigzag pattern across a set number of "rails" or lines. This lab aims to guide participants through the implementation of the Rail Fence Cipher, explaining its algorithm, and demonstrating the encryption and decryption processes.

Objective

The primary objective of this lab was to implement the Rail Fence Cipher algorithm in C++, enabling both encryption and decryption of text messages using a specified number of rails.

Algorithm

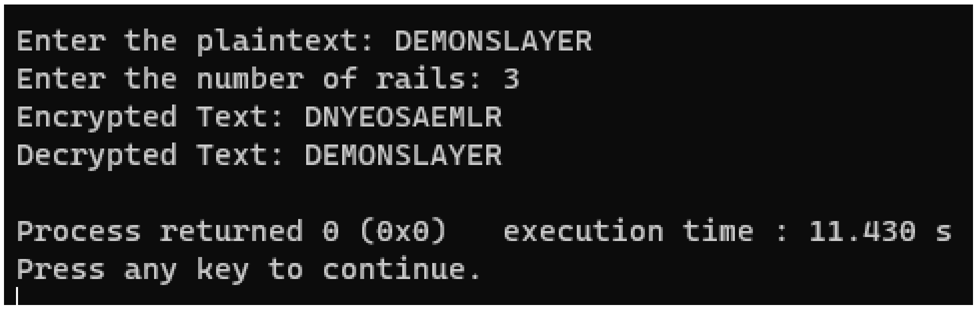
- Define functions for encryption and decryption, taking plaintext and the number of rails as input.
- Create an empty list of lists (rails) to hold the ciphertext.
- Iterate through the plaintext characters:
  - Move down a rail for each character, wrapping back to the top rail when reaching the bottom.
  - Place the character in the corresponding position on the current rail.
- Read the ciphertext by traversing the rails from top to bottom, left to right.

Code Implementation

```
#include <iostream>
#include <string>
using namespace std;

string encryptRailFence(string plaintext, int rails) {
    string ciphertext = "";
    int length = plaintext.length();

    for (int i = 0; i < rails; ++i) {
        for (int j = i; j < length; j += rails * 2 - 2) {
            ciphertext += plaintext[j];

            if (i > 0 && i < rails - 1 && j + (rails - i - 1) * 2 < length) {
                ciphertext += plaintext[j + (rails - i - 1) * 2];
            }
        }
    }

    return ciphertext;
}

string decryptRailFence(string ciphertext, int rails) {
    string decryptedText(ciphertext.length(), ' ');
    int length = ciphertext.length();
    int index = 0;

    for (int i = 0; i < rails; ++i) {
        for (int j = i; j < length; j += rails * 2 - 2) {
            decryptedText[j] = ciphertext[index++];

            if (i > 0 && i < rails - 1 && j + (rails - i - 1) * 2 < length) {
                decryptedText[j + (rails - i - 1) * 2] = ciphertext[index++];
            }
        }
    }

    return decryptedText;
}

int main() {
    string plaintext, encrypted, decrypted;
    int rails;

    cout << "Enter the plaintext: ";
    getline(cin, plaintext);

    cout << "Enter the number of rails: ";
```

```
        cin >> rails;

        encrypted = encryptRailFence(plaintext, rails);
        cout << "Encrypted Text: " << encrypted << endl;

        decrypted = decryptRailFence(encrypted, rails);
        cout << "Decrypted Text: " << decrypted << endl;

        return 0;
    }
```

Output



```
Enter the plaintext: DEMONSLAYER
Enter the number of rails: 3
Encrypted Text: DNYEOSAEMLR
Decrypted Text: DEMONSLAYER

Process returned 0 (0x0)   execution time : 11.430 s
Press any key to continue.
```

Discussion

- ✓ The Rail Fence Cipher is relatively weak compared to modern ciphers.
- ✓ It can be broken using frequency analysis and pattern recognition techniques.
- ✓ Its strength depends on the number of rails and message length.
- ✓ It serves as an example of classical cryptography and highlights the need for more complex algorithms.

Conclusion

The lab successfully implemented the Rail Fence Cipher in C++, demonstrating its encryption and decryption capabilities. While not secure for modern communications, it provides insights into the evolution of cryptography and the importance of strong algorithms.

5. Wireshark Data Capture and Steal login information.

Introduction

Wireshark is a powerful tool for capturing and analyzing network traffic. It allows users to observe the details of communication exchanges between devices on a network, including protocols used, IP addresses, application data, and more. Network traffic capture and analysis are valuable for various purposes, including troubleshooting network issues, detecting security vulnerabilities, monitoring network activity, and investigating network performance.

Objective

The main objectives of this lab were:

- ✓ Learn how to set up and configure Wireshark for capturing network traffic.
- ✓ Capture live network traffic on a specified interface.
- ✓ Analyze captured packets using Wireshark's features.
- ✓ Export the captured data for further analysis or documentation.

Equipment and Software

- ✓ Wireshark software installed on a computer.
- ✓ A computer connected to a network.

Methodology

Step – 1: Installing Wireshark

Ensure Wireshark is installed on the computer used for data capture.

Step – 2: Launching Wireshark

Open Wireshark and select the network interface to capture data from. And connect to the network.

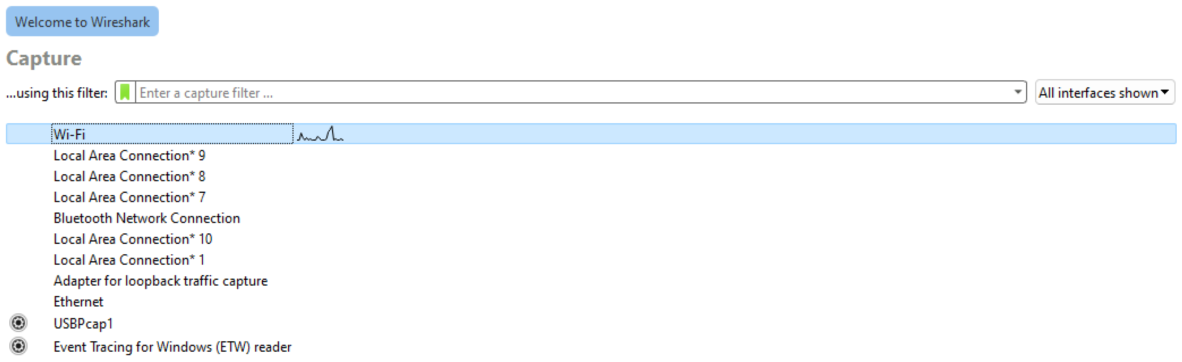


Fig-1: Launching Wireshark and connect to the network

Step – 3: Starting Capture

Initiate the packet capture by clicking the "Start" button in Wireshark.

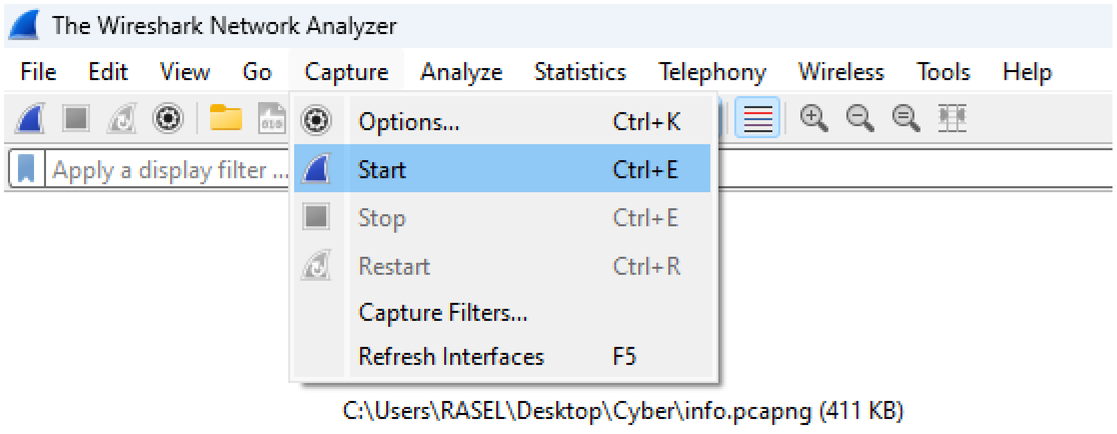
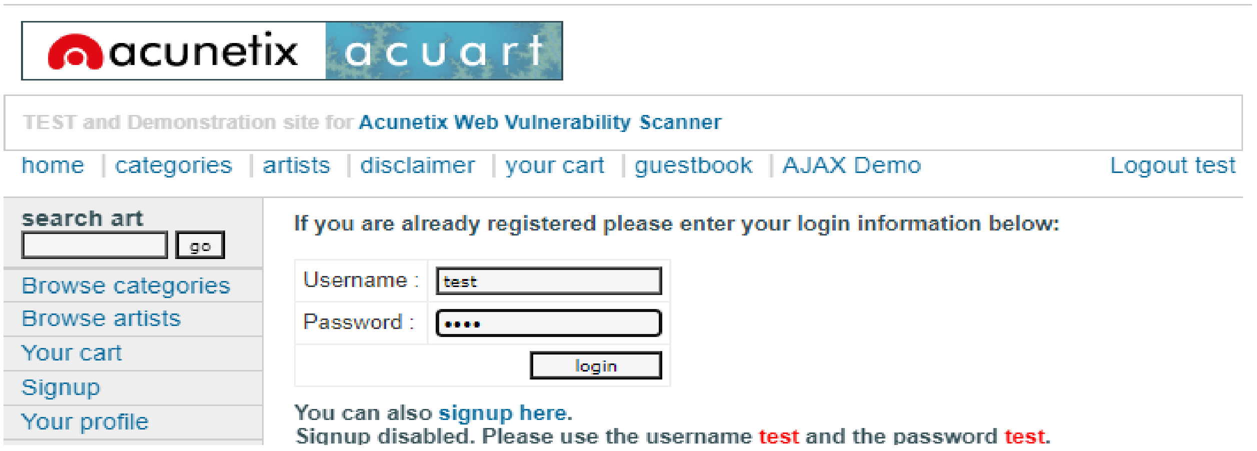

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ...
Options... Ctrl+K
Start Ctrl+E
Stop Ctrl+E
Restart Ctrl+R
Capture Filters...
Refresh Interfaces F5
C:\Users\RASEL\Desktop\Cyber\info.pcapng (411 KB)

Fig-2: Starting Capture

Step – 4: Give the login details


acunetix acuart
TEST and Demonstration site for Acunetix Web Vulnerability Scanner
home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo Logout test
search art go
Browse categories
Browse artists
Your cart
Signup
Your profile

If you are already registered please enter your login information below:

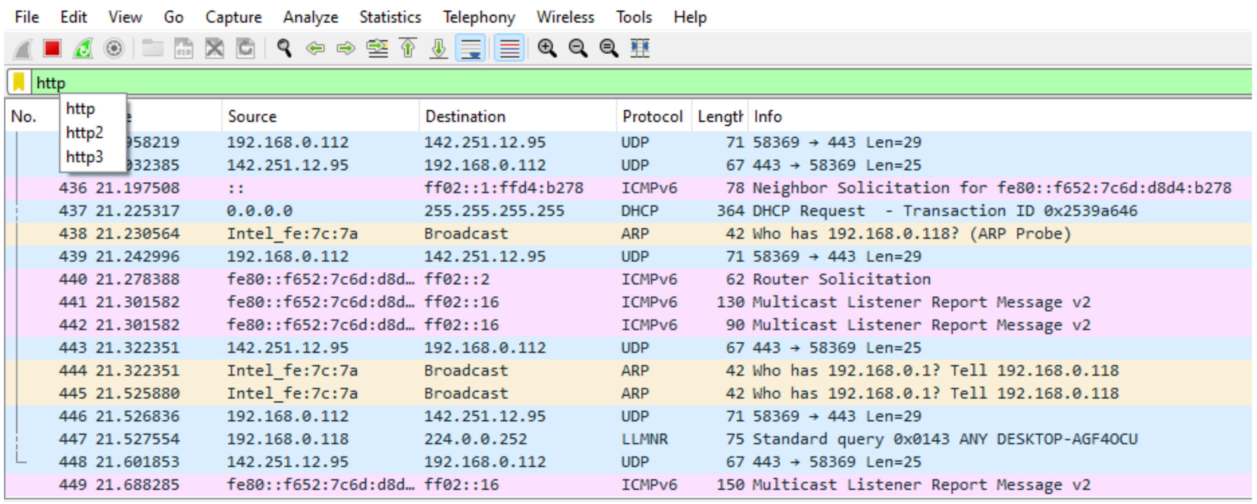
Username : test
Password : ••••

login

You can also signup here.
Signup disabled. Please use the username test and the password test.

Fig-3: User Login

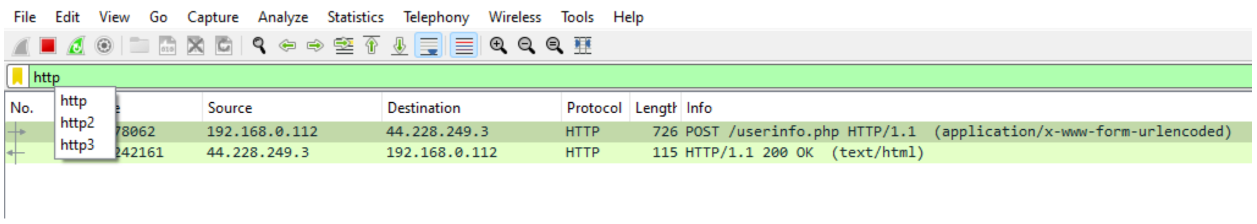
Step – 5: Filtering site


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http
http
http2
http3

```
No.                  Source                  Destination          Protocol  Length Info
     958219          192.168.0.112           142.251.12.95        UDP         71 58369 → 443 Len=29
     932385          142.251.12.95           192.168.0.112        UDP         67 443 → 58369 Len=25
 436 21.197508       ::                      ff02::1:ffd4:b278    ICMPv6      78 Neighbor Solicitation for fe80::f652:7c6d:d8d4:b278
 437 21.225317       0.0.0.0                 255.255.255.255      DHCP       364 DHCP Request  - Transaction ID 0x2539a646
 438 21.230564       Intel_fe:7c:7a          Broadcast            ARP         42 Who has 192.168.0.118? (ARP Probe)
 439 21.242996       192.168.0.112           142.251.12.95        UDP         71 58369 → 443 Len=29
 440 21.278388       fe80::f652:7c6d:d8d…    ff02::2              ICMPv6      62 Router Solicitation
 441 21.301582       fe80::f652:7c6d:d8d…    ff02::16             ICMPv6     130 Multicast Listener Report Message v2
 442 21.301582       fe80::f652:7c6d:d8d…    ff02::16             ICMPv6      90 Multicast Listener Report Message v2
 443 21.322351       142.251.12.95           192.168.0.112        UDP         67 443 → 58369 Len=25
 444 21.322351       Intel_fe:7c:7a          Broadcast            ARP         42 Who has 192.168.0.1? Tell 192.168.0.118
 445 21.525880       Intel_fe:7c:7a          Broadcast            ARP         42 Who has 192.168.0.1? Tell 192.168.0.118
 446 21.526836       192.168.0.112           142.251.12.95        UDP         71 58369 → 443 Len=29
 447 21.527554       192.168.0.118           224.0.0.252          LLMNR       75 Standard query 0x0143 ANY DESKTOP-AGF4OCU
 448 21.601853       142.251.12.95           192.168.0.112        UDP         67 443 → 58369 Len=25
 449 21.688285       fe80::f652:7c6d:d8d…    ff02::16             ICMPv6     150 Multicast Listener Report Message v2
```

Fig-4: Filtering the site

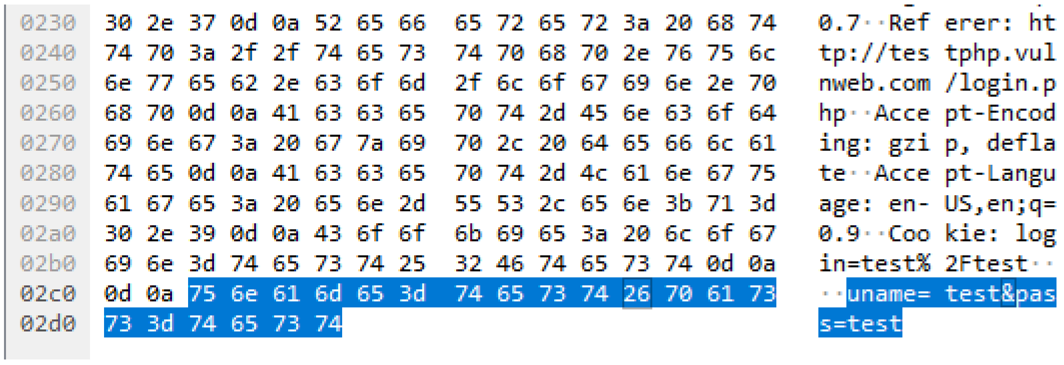
Step – 6: Desired website


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http
http
http2
http3

```
No.            Source           Destination      Protocol  Length Info
     78062     192.168.0.112    44.228.249.3     HTTP        726 POST /userinfo.php HTTP/1.1  (application/x-www-form-urlencoded)
     242161    44.228.249.3     192.168.0.112    HTTP        115 HTTP/1.1 200 OK  (text/html)
```

Fig-5: Desired website

Step – 7: Steal Login Information



```
0230  30 2e 37 0d 0a 52 65 66  65 72 65 72 3a 20 68 74   0.7··Ref erer: ht
0240  74 70 3a 2f 2f 74 65 73  74 70 68 70 2e 76 75 6c   tp://tes tphp.vul
0250  6e 77 65 62 2e 63 6f 6d  2f 6c 6f 67 69 6e 2e 70   nweb.com /login.p
0260  68 70 0d 0a 41 63 63 65  70 74 2d 45 6e 63 6f 64   hp··Acce pt-Encod
0270  69 6e 67 3a 20 67 7a 69  70 2c 20 64 65 66 6c 61   ing: gzi p, defla
0280  74 65 0d 0a 41 63 63 65  70 74 2d 4c 61 6e 67 75   te··Acce pt-Langu
0290  61 67 65 3a 20 65 6e 2d  55 53 2c 65 6e 3b 71 3d   age: en- US,en;q=
02a0  30 2e 39 0d 0a 43 6f 6f  6b 69 65 3a 20 6c 6f 67   0.9··Coo kie: log
02b0  69 6e 3d 74 65 73 74 25  32 46 74 65 73 74 0d 0a   in=test% 2Ftest··
02c0  0d 0a 75 6e 61 6d 65 3d  74 65 73 74 26 70 61 73   ··uname= test&pas
02d0  73 3d 74 65 73 74                                  s=test
```

Fig-6: Steal login information

Results

Wireshark captures a variety of packets, including those related to HTTP, DNS, and other protocols. Analyzing the captured data helps identify network issues or irregularities.

Discussion

Capturing network traffic with Wireshark offers valuable insights into network behavior. It can be used for troubleshooting network issues, detecting security vulnerabilities, monitoring network performance, and investigating network attacks. However, it is crucial to ensure ethical considerations and respect for privacy when capturing and analyzing network traffic.

Conclusion

This lab explored the basic functionalities of Wireshark for capturing and analyzing network traffic. Understanding the captured data opens doors to various potential applications in network troubleshooting, security analysis, and performance monitoring. While limitations exist, Wireshark remains a powerful tool for gaining valuable insights into the often-invisible world of network communication.

6. Record the keyboard strokes using key logger tools.

Introduction

Key loggers are implanted on a machine to intentionally monitor the user activity by logging keystrokes and eventually delivering them to a third party. While they are seldom used for legitimate purposes key loggers are often maliciously exploited by attackers to steal confidential information. Many credit card numbers and passwords have been stolen using key loggers which make them one of the most dangerous types of spyware known to date.
Key loggers can be implemented as tiny hardware devices or more conveniently in software. Software-based key loggers can be further classified based on the privileges they require to execute like privileged key logger with full privileges in kernel space and unprivileged key logger without any privileges in user space.

Objective
Keyloggers are built for the act of keystroke logging — creating records of everything you type on a computer or mobile keyboard. These are used to quietly monitor your computer activity while you use your devices as normal. Keyloggers are used for legitimate purposes like feedback for software development but can be misused by criminals to steal your data.

Hardware Keylogger
Hardware keyloggers are physical components built-in or connected to your device. Some hardware methods may be able to track keystrokes without even being connected to your device. For brevity, we'll include the keyloggers you are most likely to fend against:

Keyboard hardware keyloggers can be placed in line with your keyboard's connection cable or built into the keyboard itself. This is the most direct form of interception of your typing signals.

Hidden camera keyloggers may be placed in public spaces like libraries to visually track keystrokes.

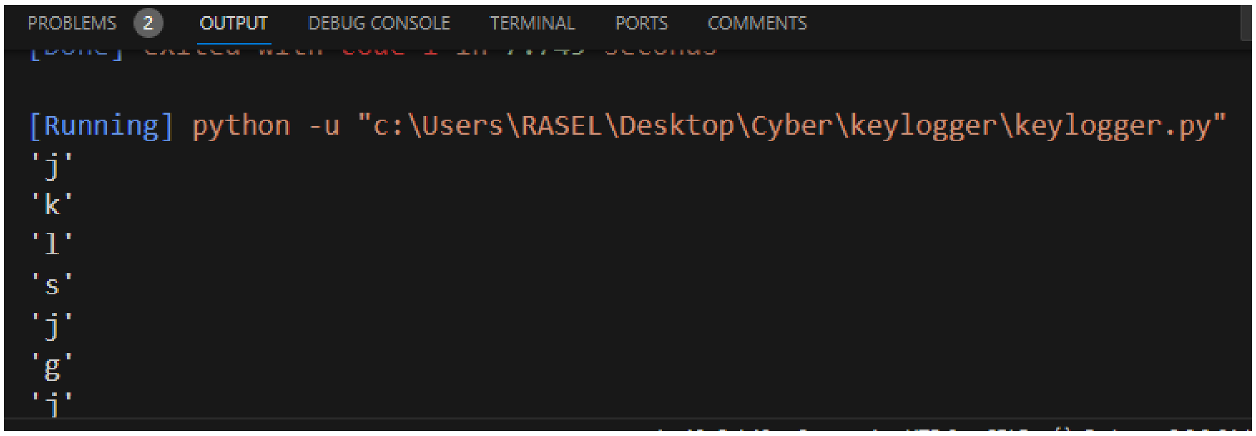
USB disk-loaded keyloggers can be a physical Trojan horse that delivers the keystroke logger malware once connected to your device.

Code Implementation

```
from pynput import keyboard

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

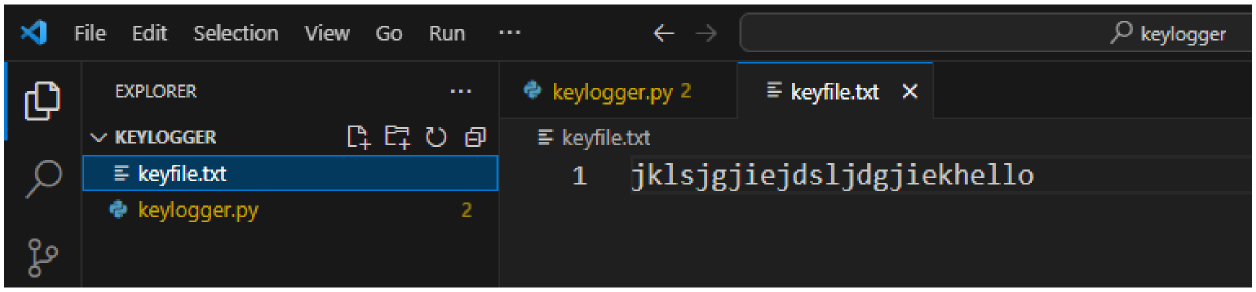
```
if __name__ == "__main__":
    listener = keyboard.Listener(on_press=keyPressed)
    listener.start()
    input()
```

Output



```
[Running] python -u "c:\Users\RASEL\Desktop\Cyber\keylogger\keylogger.py"
'j'
'k'
'l'
's'
'j'
'g'
'j'
```

Fig-7: Keylogger output



```
jklsjgjiejdsljdgjiekhello
```

Fig-8: Keylogger text file

Conclusion

Keyloggers are marketed as legitimate software and most of them can be used to steal personal user data. At present, Keyloggers are used in combination with phishing and social engineering to commit cyber fraud.