**Sir Syed University of Engineering & Technology (SSUET)**

**Software Engineering Department**

**Program: Cyber Security**

***Operating Systems (SE-204L)***

***Semester: 4th***

***Batch: 2023S***

***Section: A***

**PROJECT REPORT**

***A circular logo with text and symbols

Description automatically generated***

***Submitted by:***

**MUHAMMAD BILAL IMRAN**

**2023S-BCYS-037**

**Teacher Name:**

**Muhammad Waqar**

**Project Report: Cryptographic Failures in a Web Application**

**1. Introduction**

This project focuses on building a simple web application to demonstrate cryptographic vulnerabilities and their exploitation. The primary goal is to showcase how insecure cryptographic practices can lead to data breaches and how to improve such systems. We use Flask to develop web applications and employ Kali Linux tools to exploit weaknesses.

**2. Objectives**

1. **Build a web application** with intentional cryptographic flaws.
2. **Demonstrate vulnerabilities** and exploit them using Kali Linux.
3. **Fix the identified issues** using secure cryptographic practices.
4. Compare weak cryptography with secure implementations.

**3. Tools and Technologies**

* **Programming Language**: Python 3
* **Web Framework**: Flask
* **Cryptography Library**: base64 (insecure implementation) and cryptography (secure implementation)
* **Operating System**: Kali Linux
* **Utilities**: Curl, Python scripts
* **Testing Environment**: Localhost (http://127.0.0.1:5000)

**4. Implementation**

**4.1 Weak Cryptographic Web Application**

The following code demonstrates a Flask web application with intentional cryptographic weaknesses.

**Code: app.py**

from flask import Flask, request, render\_template\_string

import base64

app = Flask(\_\_name\_\_)

# Secret key for encryption (this is a bad practice)

SECRET\_KEY = "supersecretkey"

def encrypt\_data(data):

# Simple XOR encryption (this is a very weak form of encryption)

encrypted = ''.join(chr(ord(c) ^ ord(k)) for c, k in zip(data, SECRET\_KEY))

return base64.b64encode(encrypted.encode()).decode()

def decrypt\_data(encrypted\_data):

encrypted = base64.b64decode(encrypted\_data).decode()

decrypted = ''.join(chr(ord(c) ^ ord(k)) for c, k in zip(encrypted, SECRET\_KEY))

return decrypted

@app.route('/')

def home():

return render\_template\_string('''

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Crypto Failure App</title>

</head>

<body>

<h1>Encrypt and Decrypt Data</h1>

<form action="/encrypt" method="post">

<label for="data">Data to Encrypt:</label><br>

<input type="text" id="data" name="data"><br><br>

<input type="submit" value="Encrypt">

</form>

<form action="/decrypt" method="post">

<label for="encrypted\_data">Encrypted Data to Decrypt:</label><br>

<input type="text" id="encrypted\_data" name="encrypted\_data"><br><br>

<input type="submit" value="Decrypt">

</form>

</body>

</html>

''')

@app.route('/encrypt', methods=['POST'])

def encrypt():

data = request.form['data']

encrypted\_data = encrypt\_data(data)

return f"Encrypted Data: {encrypted\_data}"

@app.route('/decrypt', methods=['POST'])

def decrypt():

encrypted\_data = request.form['encrypted\_data']

decrypted\_data = decrypt\_data(encrypted\_data)

return f"Decrypted Data: {decrypted\_data}"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**4.2 Exploiting the Vulnerabilities**

**Steps for Exploitation**

1. Start the web application:
2. python app.py
3. Encrypt data using curl:
4. curl -X POST -d "data=secretmessage" http://127.0.0.1:5000/encrypt

Response:

Encrypted Data: YOUR\_ENCRYPTED\_DATA

1. Use the following Python script to decrypt the data manually using the weak XOR encryption:

**Decryption Script: decrypt.py**

import base64

SECRET\_KEY = "supersecretkey"

def decrypt\_data(encrypted\_data):

encrypted = base64.b64decode(encrypted\_data).decode()

decrypted = ''.join(chr(ord(c) ^ ord(k)) for c, k in zip(encrypted, SECRET\_KEY))

return decrypted

encrypted\_data = "YOUR\_ENCRYPTED\_DATA\_HERE"

decrypted\_data = decrypt\_data(encrypted\_data)

print(f"Decrypted Data: {decrypted\_data}")

1. Replace "YOUR\_ENCRYPTED\_DATA\_HERE" with the data from the curl response and run the script:
2. python3 decrypt.py

**4.3 Secure Cryptographic Web Application**

To address the weaknesses, the updated code leverages the cryptography library for secure encryption and decryption.

**Improved Code: app.py**

from flask import Flask, request, render\_template\_string

from cryptography.fernet import Fernet

app = Flask(\_\_name\_\_)

# Generate a key for encryption

key = Fernet.generate\_key()

cipher\_suite = Fernet(key)

def encrypt\_data(data):

encrypted\_data = cipher\_suite.encrypt(data.encode())

return encrypted\_data.decode()

def decrypt\_data(encrypted\_data):

decrypted\_data = cipher\_suite.decrypt(encrypted\_data.encode())

return decrypted\_data.decode()

@app.route('/')

def home():

return render\_template\_string('''

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Crypto Failure App</title>

</head>

<body>

<h1>Encrypt and Decrypt Data</h1>

<form action="/encrypt" method="post">

<label for="data">Data to Encrypt:</label><br>

<input type="text" id="data" name="data"><br><br>

<input type="submit" value="Encrypt">

</form>

<form action="/decrypt" method="post">

<label for="encrypted\_data">Encrypted Data to Decrypt:</label><br>

<input type="text" id="encrypted\_data" name="encrypted\_data"><br><br>

<input type="submit" value="Decrypt">

</form>

</body>

</html>

''')

@app.route('/encrypt', methods=['POST'])

def encrypt():

data = request.form['data']

encrypted\_data = encrypt\_data(data)

return f"Encrypted Data: {encrypted\_data}"

@app.route('/decrypt', methods=['POST'])

def decrypt():

encrypted\_data = request.form['encrypted\_data']

decrypted\_data = decrypt\_data(encrypted\_data)

return f"Decrypted Data: {decrypted\_data}"

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**5. Results**

**Weak Cryptography Results**

* Data encrypted using XOR can be easily reversed with the secret key, demonstrating poor security practices.

**Improved Cryptography Results**

* By using the cryptography library, the encryption is stronger, avoiding hardcoded keys and weak algorithms.

**6. Conclusion**

This project highlights the importance of using secure cryptographic practices in software development. It demonstrates how vulnerabilities can be exploited and provides practical solutions to mitigate such issues. Proper encryption methods and secure key management are critical for robust application security.

**MUHAMMAD BILAL IMRAN (PROFILES):**

**LINKEDIN PROFILE:**

<https://www.linkedin.com/in/muhammad-bilal-imran-353510287/>

**GITHUB PROFILE:**

<https://github.com/232-MUHAMMAD>

**CERTIFICATIONS:**

INTODUCTION TO CYBER SECURITY

**BADGE:**

A green and white label with white text

Description automatically generated

**CERTIFICATE:**

A certificate of course completion

Description automatically generated