

|  |
| --- |
|  |
| september 20  text divider  Digital Empowerment Pakistan Foundation  Authored by: Muhammad Hassan Yaseen |

Security Audit Reportcolored rectanglewhite rectangle for text on cover

Date: 20th September, 2024

Implementation of Multi-Factor Authentication using Google Authenticator

Executive Summary

In today's digital age, security is a top priority for any organization. One of the most effective ways to enhance security is to implement multi-factor authentication (MFA). MFA requires users to provide two or more authentication factors to access a system, making it much harder for attackers to gain unauthorized access. This report discusses the implementation of MFA using Google Authenticator, a popular MFA solution that uses a time-based one-time password (TOTP) algorithm to generate a unique code for each user.

Introduction

The increasing number of cyber-attacks and data breaches has made it essential for organizations to implement robust security measures to protect their systems and data. MFA is a critical security measure that can help prevent unauthorized access to systems and data. In this report, we will discuss the implementation of MFA using Google Authenticator, a popular MFA solution that uses a TOTP algorithm to generate a unique code for each user.

Background

Google Authenticator is a popular MFA solution that uses a TOTP algorithm to generate a unique code for each user. The code is required in addition to the user's username and password to access the system. Google Authenticator is widely used by organizations and individuals to enhance the security of their systems and data.

Objectives

The objectives of this project are:

1. To implement a simple login system with MFA using Google Authenticator.
2. To generate a secret key for each user and store it in a database.
3. To generate a TOTP for each user and require it in addition to the username and password.

Methodology

To implement MFA using Google Authenticator, we followed the following steps:

1. Created a user database to store user credentials.
2. Implemented a login system that checks the user's username and password.
3. Integrated Google Authenticator to generate a TOTP for each user.
4. Modified the login system to require the TOTP in addition to the username and password.

Implementation

We used Python and the Flask web framework to implement the login system. We also used the pyotp library to generate the TOTP.

User Database

We created a user database using SQLite to store user credentials. The database schema is as follows:

import sqlite3

2

3conn = sqlite3.connect('users.db')

4cursor = conn.cursor()

5

6cursor.execute('''

7 CREATE TABLE users (

8 id INTEGER PRIMARY KEY,

9 username TEXT NOT NULL,

10 password TEXT NOT NULL,

11 secret\_key TEXT NOT NULL

12 );

13''')

14

15conn.commit()

16conn.close()

Login System

We implemented a login system that checks the user's username and password. If the credentials are valid, the system generates a TOTP for the user and requires it in addition to the username and password.

from flask import Flask, request, jsonify

2from werkzeug.security import check\_password\_hash

3

4app = Flask(\_\_name\_\_)

5

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

7def login():

8 username = request.json['username']

9 password = request.json['password']

10 totp = request.json['totp']

11

12 cursor = conn.cursor()

13 cursor.execute('SELECT \* FROM users WHERE username = ?', (username,))

14 user = cursor.fetchone()

15

16 if user and check\_password\_hash(user[2], password):

17 # Generate TOTP

18 secret\_key = user[3]

19 totp\_generator = pyotp.TOTP(secret\_key)

20 expected\_totp = totp\_generator.now()

21

22 if totp == expected\_totp:

23 return jsonify({'success': True, 'message': 'Login successful'})

24 else:

25 return jsonify({'success': False, 'message': 'Invalid TOTP'})

26 else:

27 return jsonify({'success': False, 'message': 'Invalid username or password'})

28

29if \_\_name\_\_ == '\_\_main\_\_':

30 app.run(debug=True)

Google Authenticator Integration

We integrated Google Authenticator to generate a TOTP for each user. We used the pyotp library to generate the TOTP.

import pyotp

2

3def generate\_secret\_key():

4 return pyotp.random\_base32()

5

6def generate\_totp(secret\_key):

7 totp\_generator = pyotp.TOTP(secret\_key)

8 return totp\_generator.now()

9

10# Generate secret key for each user

11cursor = conn.cursor()

12cursor.execute('SELECT \* FROM users')

13users = cursor.fetchall()

14

15for user in users:

16 secret\_key = generate\_secret\_key()

17 cursor.execute('UPDATE users SET secret\_key = ? WHERE id = ?', (secret\_key, user[0]))

18 conn.commit()

19

20# Generate TOTP for each user

21for user in users:

22 secret\_key = user[3]

23 totp = generate\_totp(secret\_key)

24 print(f'User {user[1]} TOTP: {totp}')

Conclusion

This post was expanded on the configuration of Multi-Factor Authentication (MFA) using Google Authenticator, a very useful tool to add another level of security for our trusted old username and password. We covered the many advantages that MFA offers, such as a drastic improvement in the security of accessing a web site, mitigating against being victim of those dreaded phishing attacks and complying with all regulatory reasons.

The instructions in the report explained how easy it was to get Google Authenticator up an running, from installing the app and scanning a QR code t setting your authentication options. We have also discussed the benefits of Google Authenticator that it supports a lot of platforms, it is very easy to use and scales really well.

The report concluded by underlining how vitally important MFA is in a world that continually faces advanced securities threats. For those organizations looking to protect their crown-jewel data and systems, we recommend MFA is used as a best practice.