Name: Muhammad Usman Ali Sadiq

Cybersecurity

Task 02

Implementing Multi-Factor Authentication

➢ Selecting an MFA solution compatible with the system

- **Evaluate MFA Providers**: Assess solutions like Google Authenticator, Microsoft Authenticator, Duo, Okta, or Authy, ensuring they support common protocols like TOTP (Time-based One-Time Password) and push notifications.
- **Compatibility**: Ensure the selected solution is compatible with the backend (Python) and the database (MongoDB) setup.
- For Python, libraries such as pyotp for TOTP or integration with third-party services via APIs may be appropriate.
- **Customization**: Consider options that allow customization for your users and ease of integration with existing login processes.

For simplicity, we'll use pyotp (a Python library) for generating TOTP codes, which can be verified using apps like Google Authenticator.

Install the Required Libraries:

- Install pyotp to handle OTP generation and validation:

```
pip install pyotp
```

➢Configuring MFA Settings and Options

Backend Integration:

- Configure MFA during user registration and login phases.
- Store relevant MFA details (e.g., secret keys for TOTP) in the MongoDB database, ensuring these are encrypted or hashed.

Enforce MFA:

- Make MFA optional initially, allowing users to adopt it voluntarily.
- Eventually, enforce MFA after a grace period or based on specific roles or user risk factors.

MFA Methods:

- Provide users with multiple options like authenticator apps, SMS-based codes (considering security trade-offs), and email-based codes.

Backend Integration (Python + MongoDB)

- **Generate Secret for TOTP**: Each user should have a unique secret key stored in the database. This secret will be used to generate time-based OTPs.

```python
import pyotp
import qrcode
from pymongo import MongoClient

# Database connection
client = MongoClient('mongodb://localhost:27017/')
db = client['user_database']
users_collection = db['users']

def generate_totp_secret():
    # Generate a TOTP secret key for the user
    secret = pyotp.random_base32()
    return secret

# Function to create a QR code for users to scan in their MFA app
def generate_qr_code(username, secret):
    totp = pyotp.TOTP(secret)
    provisioning_uri = totp.provisioning_uri(username, issuer_name="MyfirstApp")
    # Generate QR code
    qr_img = qrcode.make(provisioning_uri)
    qr_img.save(f"{username}_mfa_qr.png")
    # Save QR for the user
    print(f"QR code generated for {username}")
```

- **Saving the TOTP Secret in MongoDB**: When a user registers or opts for MFA, you can store their TOTP secret in the database.

```python
def save_user_mfa(username, secret):
    # Add the MFA secret to the user's record in MongoDB
    users_collection.update_one(
        {"username": username},
        {"$set": {"mfa_secret": secret}}
    )
    print(f"Secret saved for user {username}")

# Example usage
username = "testuser"
secret = generate_totp_secret()
save_user_mfa(username, secret)
generate_qr_code(username, secret)
```

Verifying OTP During Login: Once MFA is enabled for a user, they will need to provide a valid OTP during login. Here's how you can verify the OTP:

```python
def verify_otp(username, otp_code):
    user = users_collection.find_one({"username": username})
    if not user or "mfa_secret" not in user:
        return False  # No MFA set up for this user

    totp = pyotp.TOTP(user['mfa_secret'])
    return totp.verify(otp_code)  # Verify the OTP

# Example usage
otp_input = input("Enter OTP: ")
is_valid = verify_otp("testuser", otp_input)
if is_valid:
    print("Login successful")
else:
    print("Invalid OTP")
```

What's Happening Here:

1. **QR Code**: A QR code is generated for the user when they opt for MFA, which can be scanned using Google Authenticator.
2. **TOTP Secret**: This secret is saved in MongoDB and used for generating and verifying OTPs.
3. **OTP Validation**: During login, the user provides an OTP, which is validated against the current TOTP generated from the stored secret.

➢ Educating Users on How to Set Up and Use MFA

- **User Guide**:
  - Create a step-by-step guide (with screenshots) on how to enable MFA in their accounts.
  - The guide should include how to download and use authenticator apps.
- **Tooltips and Prompts**:

- Implement user prompts and tooltips within the user dashboard to encourage adoption.

- **Video Tutorials**:
  - Consider providing video tutorials for better user engagement.

➢ Monitoring MFA Adoption and Addressing Issues

- **Tracking Usage**:
  - Track how many users enable MFA via the database (e.g., a flag indicating MFA is enabled for an account).
- **Identify Roadblocks**:
  - Set up logging and user feedback mechanisms to capture and resolve any MFA-related issues.
- **Help Desk Support**:
  - Ensure the help desk or support system is equipped to troubleshoot common MFA issues like device syncing or lost authentication devices.

➢ Regularly Updating and Maintaining MFA Configurations

- **Regular Updates**:
  - Review MFA solution updates and apply security patches or upgrades as needed.
- **Audit MFA**:
  - Regularly audit the MFA system's performance and adoption rate.

    - Conduct penetration testing to ensure MFA configurations are robust and secure.
- **User Experience Enhancements**:
    - Periodically survey users to understand pain points or areas for improving the MFA process.