Project 2

Universidade de Aveiro

João Ferreira

Project 2

Segurança Informática e nas Organizações

Universidade de Aveiro

João Ferreira

(103625) joaop.ferreira@ua.pt

2024

Contents

Chapter 1

Introduction

This report aims to provide a critical analysis of the application, which consists of an online store for memorabilia products for the Department of Electronics, Telecommunications, and Informatics (DETI) at the University of Aveiro. The main objective of this evaluation is to ensure that the application runs without errors, has consistent behavior, and features that fulfill the purpose of the online store. The scope of this analysis follows the requirements established by Level 1 of the Application Security Verification Standard (ASVS), representing the minimum level of verification required for all web applications.

The evaluation will be conducted using the OWASP ASVS checklist, which encompasses controls that are fully testable by automated methods, along with some dynamic manual techniques.

This report will highlight the main findings, prioritizing identified security issues, with the aim of providing valuable insights for enhancing the security of the developed web application.

Chapter 2

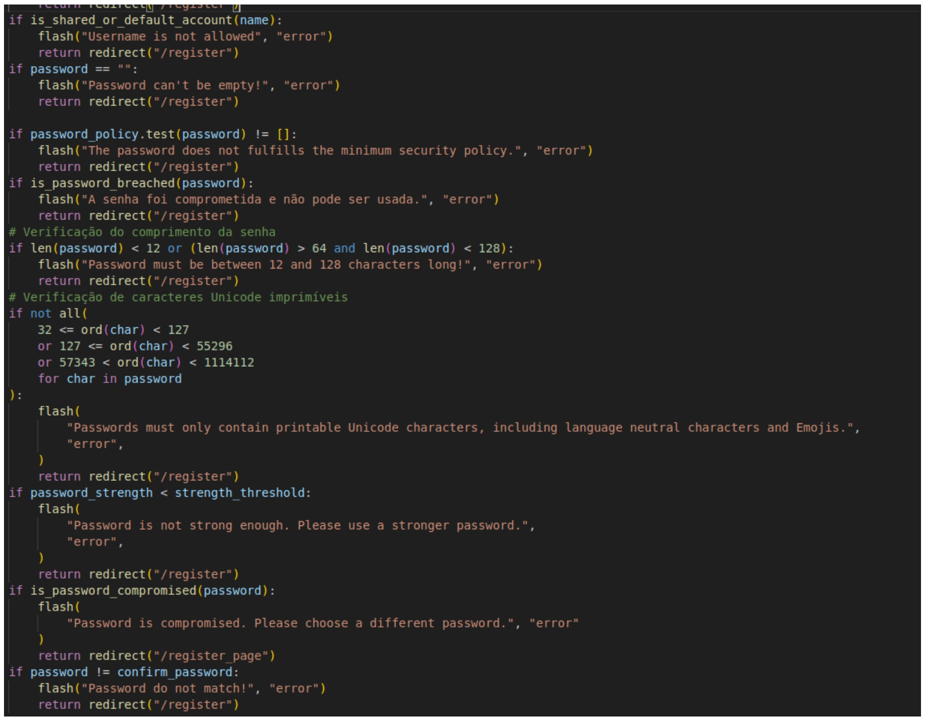
ASVS List - Key Issues

To conduct a security audit based on the OWASP Application Security Verification Standard (ASVS) Level 1, the focus is on identifying common vulnerabilities that could be exploited in web applications. Level 1 includes a basic security verification, which can be conducted through automated testing and some manual methods.

2.1 Password Security Credentials (V2.1)

Discussing the authentication section, more precisely in the "Password Security Credentials" area, I was able to fulfill all the requirements.

To demonstrate the implementation, I will present code snippets as well as some images of the app.



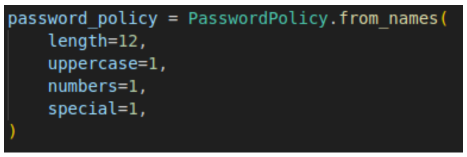
```python
if is_shared_or_default_account(name):
    flash("Username is not allowed", "error")
    return redirect("/register")
if password == "":
    flash("Password can't be empty!", "error")
    return redirect("/register")

if password_policy.test(password) != []:
    flash("The password does not fulfills the minimum security policy.", "error")
    return redirect("/register")
if is_password_breached(password):
    flash("A senha foi comprometida e não pode ser usada.", "error")
    return redirect("/register")
# Verificação do comprimento da senha
if len(password) < 12 or (len(password) > 64 and len(password) < 128):
    flash("Password must be between 12 and 128 characters long!", "error")
    return redirect("/register")
# Verificação de caracteres Unicode imprimíveis
if not all(
    32 <= ord(char) < 127
    or 127 <= ord(char) < 55296
    or 57343 < ord(char) < 1114112
    for char in password
):
    flash(
        "Passwords must only contain printable Unicode characters, including language neutral characters and Emojis.",
        "error",
    )
    return redirect("/register")
if password_strength < strength_threshold:
    flash(
        "Password is not strong enough. Please use a stronger password.",
        "error",
    )
    return redirect("/register")
if is_password_compromised(password):
    flash(
        "Password is compromised. Please choose a different password.", "error"
    )
    return redirect("/register_page")
if password != confirm_password:
    flash("Password do not match!", "error")
    return redirect("/register")
```

Figure 2.1: if statements to restrict and enhance password security.

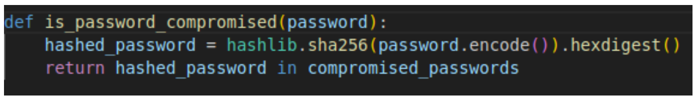
In this code, it is possible to verify the restrictions applied to the registration of a new user on our web page.

Some of these restrictions make use of the following auxiliary functions:



```python
password_policy = PasswordPolicy.from_names(
    length=12,
    uppercase=1,
    numbers=1,
    special=1,
)
```

Figure 2.2: Password Policy



```python
def is_password_compromised(password):
    hashed_password = hashlib.sha256(password.encode()).hexdigest()
    return hashed_password in compromised_passwords
```

Figure 2.3: Password Verification

This line is used to remove any extra spaces from the password string.

Code Block 2.1: Multiple spaces replaced by a single space

```python

password = " ".join(password.split())
```

2.2 General Authenticator Requirements (V2.2)

2.2.1 V2.2.1

The code includes a mechanism to lock the account after multiple failed login attempts. Specifically, if a user fails 5 times within a 300-second (5-minute) period, the account is temporarily locked.

Code Block 2.2: Login Block

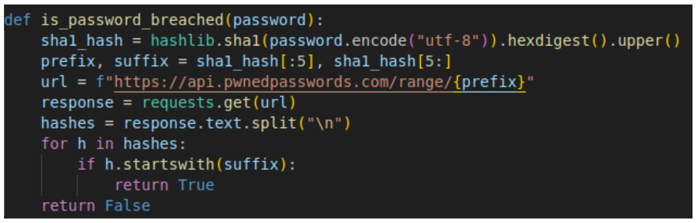
```python

login_attempts = {}

@app.route("/login", methods=["POST"])
def login():
    if request.method == "POST":
        name = request.form["name"].strip()
        password = request.form["password"].strip()

        if name in login_attempts:
            elapsed_time = time.time() - login_attempts[name
                ]["last_attempt_time"]
            if login_attempts[name]["failed_attempts"] >= 5
                and elapsed_time < 300:
                flash(
                    "Conta bloqueada devido a muitas
                        tentativas falhadas. Tente novamente
                        mais tarde.",
                    "error",
                )
                return redirect(url_for("login_page"))
```

There is a check for compromised passwords using the pwnedpasswords API. This helps prevent the use of known compromised passwords



```python
def is_password_breached(password):
    sha1_hash = hashlib.sha1(password.encode("utf-8")).hexdigest().upper()
    prefix, suffix = sha1_hash[:5], sha1_hash[5:]
    url = f"https://api.pwnedpasswords.com/range/{prefix}"
    response = requests.get(url)
    hashes = response.text.split("\n")
    for h in hashes:
        if h.startswith(suffix):
            return True
    return False
```

Figure 2.4: Checks if a password has appeared in leaked password databases

2.2.2 V2.2.2

The code uses email to send a verification code as part of the two-factor authentication (2FA) process. This method is used to add an additional layer of security after the initial authentication.

Code Block 2.3: Email Message

```python

@app.route("/two_factor_page")
def two_factor_page():
    session["user_id"].append(generate_verification_code())
    conn = sqlite3.connect("database/shop_sec.db")
    c = conn.cursor()
    res = c.execute("SELECT * FROM users WHERE user_id=?", (
        session["user_id"][0],))
    data = res.fetchone()
    conn.close()

    send_email(data[2], session["user_id"][-1])

    return render_template("two_factor.html")
```

The code implements two-factor authentication, which is a more secure method than just using a password. However, the code only offers email verification as the 2FA method.

Code Block 2.4: 2FA

```python

def send_email(to:str, verification_code:int):
    subject = "Verification Code for Two-Factor
        Authentication"
    body = f"Your verification code is: {verification_code}"
    msg = Message(subject, recipients=[to], body=body)

    mail.send(msg)
```

2.3 Credential Recovery Requirements (V2.5)

2.3.1 V2.5.4

There is no possibility to register accounts with usernames such as "admin", "root", or "sa".

Code Block 2.5: Default Accounts Check

```python

def is_shared_or_default_account(username):
    shared_default_accounts = ["root", "admin", "sa"]

    return username.lower() in shared_default_accounts
```

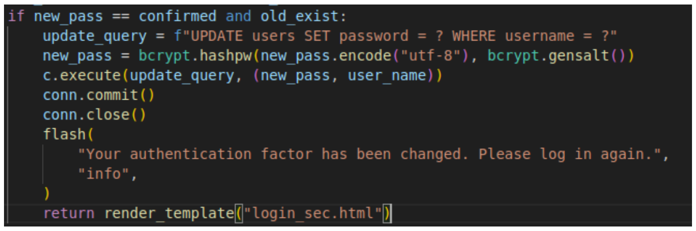
2.3.2 V2.5.5

Additionally, during both the registration and login processes or passwords replacements, the user will always be notified of the steps they are taking, as well as any errors or failures they may encounter during these procedures.

Code Block 2.6: Notifications Events

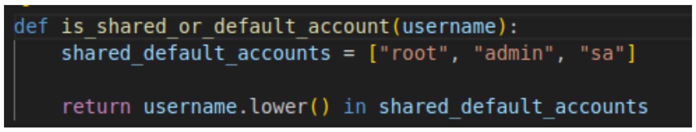
```python

if new_pass == confirmed and old_exist:
            update_query = f"UPDATE users SET password = ? WHERE username = ?"
            new_pass = bcrypt.hashpw(new_pass.encode("utf-8"), bcrypt.gensalt())
            c.execute(update_query, (new_pass, user_name))
            conn.commit()
            conn.close()
            flash(
                "Your authentication factor has been changed. Please log in again.",
                "info",
            )
```



```python
if new_pass == confirmed and old_exist:
    update_query = f"UPDATE users SET password = ? WHERE username = ?"
    new_pass = bcrypt.hashpw(new_pass.encode("utf-8"), bcrypt.gensalt())
    c.execute(update_query, (new_pass, user_name))
    conn.commit()
    conn.close()
    flash(
        "Your authentication factor has been changed. Please log in again.",
        "info",
    )
    return render_template("login_sec.html")
```

Figure 2.5: Credentials has been changed notification



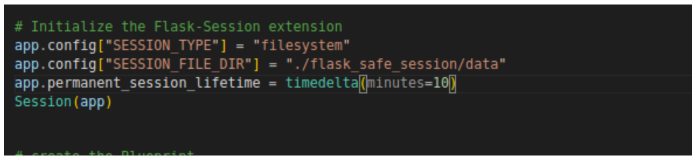
```
def is_shared_or_default_account(username):
    shared_default_accounts = ["root", "admin", "sa"]

    return username.lower() in shared_default_accounts
```

Figure 2.6: Invalid admin usernames

2.4 Out of Band Verifier Requirements (V2.7.2)

In this authentication parameter, the application allows for a timeout after a certain period, ensuring that each authentication request is unique.



```
# Initialize the Flask-Session extension
app.config["SESSION_TYPE"] = "filesystem"
app.config["SESSION_FILE_DIR"] = "./flask_safe_session/data"
app.permanent_session_lifetime = timedelta(minutes=10)
Session(app)
```

Figure 2.7: Time out for all sessions, set to 10 min

2.5 Session Termination (V3.3)

2.5.1 V3.3.1

The logout method removes the user_id from the session. The line session["user_id"].pop() effectively invalidates the user's session. The redirection after logout is to the home page with return redirect("/"), which ensures that the user is not authenticated after logout.

This implementation helps to ensure that, after logout or session expiration, the session cannot be resumed by using the browser's back button or by downstream relying parties.

Code Block 2.7: Session Pop

```

@app.route("/logout")
def logout():
    if "user_id" in session:
        session["user_id"].pop()
    return redirect("/")
```

2.6 Encrypted database Storage

For password encryption in the database, I am using a symmetric encryption method. When a new user registers, the plaintext password they provide is encrypted before being stored in the database. This process involves applying an AES encryption algorithm with a CFB mode of operation, which uses a secret key and an initialization vector (nonce) to ensure security.

During the login process, the plaintext password provided is encrypted using the same secret key and the stored initialization vector. The resulting encrypted password is then compared with the encrypted password stored in the database. If the two encrypted passwords match, the login is successful; otherwise, access is denied.

In summary, symmetric encryption with AES provides a robust layer of security, ensuring that passwords are effectively protected even if the encrypted data is compromised. The secret key and initialization vector are crucial for ensuring that encryption and decryption are performed correctly.

Code Block 2.8: Password Encryption

```python

def load_key():
    return open('secret.key', 'rb').read()

def encrypt_message(message):
    key = load_key()
    cipher = Cipher(algorithms.AES(key), modes.CFB(nonce),
        backend=default_backend())
    encryptor = cipher.encryptor()
    try:
        encrypted = encryptor.update(message.encode()) +
            encryptor.finalize()
        encrypted_message = base64.urlsafe_b64encode(nonce +
             encrypted).decode()
        return encrypted_message
    except Exception as e:
        print(f"Erro durante a criptografia: {e}")
        raise

def decrypt_message(encrypted_message):
    try:
        key = load_key()
        encrypted_message = base64.urlsafe_b64decode(
            encrypted_message)
        nonce = encrypted_message[:16]
        encrypted = encrypted_message[16:]
        cipher = Cipher(algorithms.AES(key), modes.CFB(nonce
            ), backend=default_backend())
        decryptor = cipher.decryptor()
        decrypted = decryptor.update(encrypted) + decryptor.
```

```
            finalize()
        return decrypted.decode('utf-8')
    except Exception as e:
        print(f"Erro durante a descriptografia: {e}")
        return None
```

2.7 File Upload (V12.1)

2.7.1 V12.1.1

To ensure the application remains efficient and secure, stringent measures are in place regarding file uploads. Users can include images in their comments; however, several critical restrictions are enforced to prevent issues such as excessive storage use or denial of service attacks.

- **File Size Limitation:** The application enforces a strict maximum file size limit of 1 MB per image. This cap prevents large files from overwhelming the server or consuming excessive storage space.
- **File Type Validation:** Only certain file types, such as images in formats like PNG, JPG, JPEG, and GIF, are permitted. This validation reduces the risk of unauthorized file types and potential security threats.

Code Block 2.9: File Upload Restrictions

```

MAX_FILE_SIZE = 1 * 1024 * 1024
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
```

These measures collectively ensure that the application maintains optimal performance and reliability, while still providing users with the capability to include visual content in their comments.

2.7.2 V12.1.2 (Level 2 Requirement)

To address the requirement of controlling individual user file uploads and preventing storage overflow, the application limits the number of files a single user can upload. These measure ensure that no single user can monopolize storage resources with excessive or large files.

- **Maximum Number of Files:** A user is restricted to a maximum of 10 files per comment. This limit ensures that users cannot excessively increase the number of files they upload, thus protecting the system from potential abuse and maintaining equitable resource distribution among users.

Code Block 2.10: File Upload Counter

```

MAX_FILES_PER_USER = 10

file_count = result[0]

                    if file_count >= MAX_FILES_PER_USER:
                        flash(f"You have reached the maximum number of files ({MAX_FILES_PER_USER})", "error")
                        return redirect(url_for('checkOut', index=post_id))


                    file_count =+ 1

                    # Update the file count for the existing record
                    c.execute("""
                        UPDATE fileComments
                        SET file_count=?
                        WHERE post_id=? AND user_name=?
                    """, (file_count, post_id, user_name))
```

These constraint is designed to maintain a balanced and efficient storage environment, preventing any individual user from adversely impacting the system's performance or storage availability.

Chapter 3

Workload and General Information

- João Ferreira (100%)