Project 2 SIO

Application Security Verification Standard (ASVS)

03/01/2024

Group Members:

- Miguel Aido Miragaia | 108317 | miguelmiragaia@ua.pt
- Cristiano Antunes Nicolau | 108536 | cristianonoicolau@ua.pt
- Andre Lourenço Gomes | 97541 | alg@ua.pt
- Pedro Miguel Ribeiro Rei | 107463 | pedrorrei@ua.pt

Index

Introduction

In this project, we embark on an insightful journey, building upon our previous work to elevate our e-commerce web application to the coveted level 1 of the Application Security Verification Standard (ASVS). This endeavor not only signifies our commitment to enhanced security but also reflects our continuous evolution in the digital security landscape.

Our focus is twofold. Firstly, we delve into the implementation of Time-based One-Time Passwords (TOTP), a robust method that significantly elevates user authentication security. Secondly, we aim to strengthen password security, a critical aspect often overlooked yet vital in safeguarding user data and transactions. Interestingly, this project is not our first foray into advanced security measures. In our initial venture, we had successfully integrated an encrypted database, a foundational element that we continue to utilize and refine in this current undertaking.

A critical component of our strategic approach involved a meticulous analysis of an extensive list of security areas and Common Weakness Enumerations (CWEs) provided in an Excel format. This comprehensive list served as a valuable resource, allowing us to carefully select relevant CWEs applicable to our project while identifying those that were not pertinent. This process was instrumental in tailoring our security measures to be both effective and specific to the unique needs of our e-commerce platform.

Through this project, we aim not only to enhance the security of our web application but also to set a precedent in applying practical, advanced security standards in e-commerce environments. It is a journey of technical advancement, strategic decision-making, and continuous learning.

ASVS

Authentication

2.1.1

Area: Password Security Credentials

CWE 521

Emphasizes the need for robust password policies and tools that empower users to set strong, unique, and secure passwords, thereby enhancing the overall security posture of the application or system.

Verification Requirement: Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).

Additional Comments: In the past, user-set passwords were at least 8 characters in length. A thorough review of our password policies and tools highlighted the need for stricter criteria, particularly concerning the handling of multiple spaces within passwords.

Implementation: In response to the new verification requirement, the existing password checking mechanism was re-evaluated. A key observation was the potential inconsistency when dealing with passwords containing multiple spaces, and to the 12 characters length at least. To address this, the `password_checker` function was slightly modified.

```
def password_checker(s):
    size = (len(s)>7)
    digits = any(i.isdigit() for i in s)
    caps = any(i.isupper() for i in s)
    if size & digits & caps:
      return True
    else: return False
```

The updated function now includes a step to remove multiple spaces from the input password before performing the length check, ensuring that the length criteria are applied correctly, and have at least 12 characters length, unlike the 8 characters length implemented before.

```
def password_checker(s):
    s = ' '.join(s.split())
    size = len(s) >= 12 and len(s) <= 127
    if size:
       return True
    return False
```

This modification not only aligns with industry best practices but also ensures that the application remains resilient against potential vulnerabilities related to password length and formatting. By refining our password policies and tools, we continue to prioritize user security and maintain a strong security posture for our application.

2.1.2

Area: Password Security Credentials

CWE 521

Emphasizes the need for robust password policies and tools that empower users to set strong, unique, and secure passwords, thereby enhancing the overall security posture of the application or system.

Verification Requirement:

1. Passwords must be permitted to be 64 characters or longer.
2. Passwords must be no longer than 128 characters.

Additional Comments: In the past, user-set passwords were at least 8 characters in length and didn't have specific length restrictions beyond that.

Implementation: In response to the updated verification requirements, the `password_checker` function was re-evaluated and modified to ensure compliance with the specified password length criteria. In the past, user-set passwords were at least 8 characters in length and didn't have specific length restrictions beyond that.

```
def password_checker(s):
        size = (len(s)>7)
        digits = any(i.isdigit() for i in s)
        caps = any(i.isupper() for i in s)
        if size & digits & caps:
                return True
        else: return False
```

The revised function now accurately checks the length of the password, allowing passwords that are 64 characters or longer but ensuring they do not exceed 128 characters. Additionally, the function continues to remove multiple spaces from the input password to ensure consistent length verification.

```
def password_checker(s):
        s = ' '.join(s.split())
        size = len(s) >= 12 and len(s) <= 127
        if size:
                return True
        return False
```

This modification not only aligns with the specified requirements but also strengthens our password policies, ensuring that the application remains resilient against potential vulnerabilities related to password length and formatting. By refining our password policies and tools, we continue to prioritize user security and maintain a strong security posture for our application.

2.1.6

Area: Password Security Credentials

CWE 620

The "Unverified Password Change" vulnerability, involves a scenario where a software application allows a user to change their password without verifying the user's old or current password adequately.

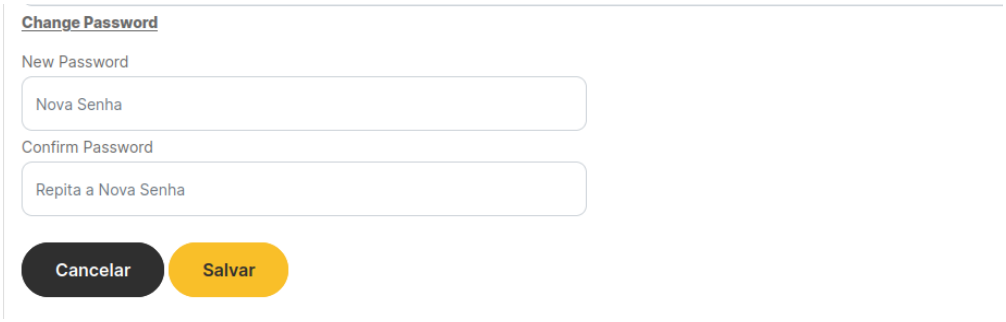
Verification Requirement: Verify that password change functionality requires the user's current and new password.

Additional Comments: The updated password change functionality ensures that users must provide their current password (`old_password`) along with the new password. This two-step verification process strengthens the security of password changes and helps prevent unauthorized modifications to user accounts.

Implementation: The enhanced password change functionality has been implemented both in the front-end and at the endpoint '/changepswd'. In the past, to update the password user only needed to put the new password and repeat the password. Another problem is that the new password did not need to meet the credentials requirements.

```
@app.route('/changepswd', methods=['POST'])
def changepswd():
    data = request.get_json()
    user_id = data.get('id')
    password = data.get('password')
    if user_id is None or not user_id.isdigit():
        return jsonify({'error': 'ID de usuário inválido'})
    user = users.query.get(int(user_id))
    if user is None:
        return jsonify({'error': 'Usuário não encontrado'})
    if password is None:
        return jsonify({'error': 'Por favor, preencha todos os campos'})
.....
```

```
function savePasswordChanges() {
  const newPassword = document.getElementById('new-password-field').value;
  const repeatNewPassword = document.getElementById('repeat-new-password-field').value;

  if (newPassword !== repeatNewPassword) {
    alert('As senhas não coincidem!');
    return;}

  const editedData = { id: user_id, password: newPassword };
  fetch('http://127.0.0.1:5000/changepswd', {method: 'POST',headers: {'Content-
Type':'application/json'  },body: JSON.stringify(editedData)}).then(response => response.json())

.then(data => {
     if (data.success) {
          alert('Senha alterada com sucesso!');
          window.location.href = "userdashboard.html";
      } else {alert('Erro ao atualizar senha: ' + data.error);}
}) .catch(error => {console.error('Erro na solicitação:', error); }); }
```



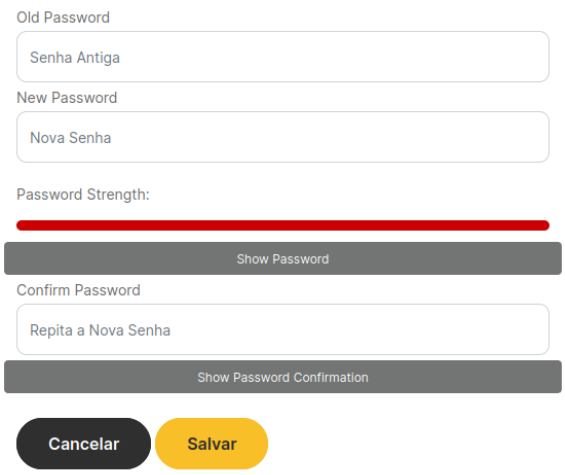
When a user attempts to change their password, the system prompts them to provide their current password. If the user fails to provide the correct `old_password` or omits it altogether, the password change process is not permitted, thereby ensuring that only authorized users can modify their account passwords.

```
@app.route('/changepswd', methods=['POST'])
def changepswd():
    data = request.get_json()
    user_id = data.get('id')
    new_password = data.get('password')
    old_password = data.get('old_password')

    def password_checker(s):
        s = ' '.join(s.split())
    if user is None:
    return jsonify({'error': 'Usuário não encontrado'}), 404
    if not new_password or not old_password:
        return jsonify({'error': 'Por favor, preencha todos os campos'}), 400

    if old_password == new_password:
        return jsonify({'error': 'A nova senha não
```

```
        size = len(s) >= 12 and len(s) <= 127
        if size:
            return True
        return False

    if user_id is None or not user_id.isdigit():
        return jsonify({'error': 'ID de usuário inválido'}), 400

    user = users.query.get(int(user_id))
```

```
pode ser igual à antiga'}), 400

    if not password_checker(new_password):
        return jsonify({'error': 'A senha precisa ter pelo menos 12 elementos'}), 400

    if not bcrypt.check_password_hash(user.password, old_password):
        return jsonify({'error': 'A senha antiga não confere'}), 401
.....
```

```
function savePasswordChanges() {

  const oldPassword = document.etElementById(old-password-field').value;
  const newPassword = document.getElementById('new-password-field').value;
  const repeatNewPassword = document.getElementById('repeat-new-password-field').value;

  if (newPassword !== repeatNewPassword) {
    alert('As senhas não coincidem!');
    return;}

  const editedData = { id: user_id, password: newPassword, old_password: oldPassword};
  fetch('http://127.0.0.1:5000/changepswd', {method: 'POST',headers: {'Content-
Type':'application/json'  },body: JSON.stringify(editedData)}).then(response => response.json())

.then(data => {
    if (data.success) {
        alert('Senha alterada com sucesso!');
        window.location.href = "userdashboard.html";
     } else {alert('Erro ao atualizar senha: ' + data.error);}
}) .catch(error => {console.error('Erro na solicitação:', error); }); }
```



2.1.8

Area: Password Security Credentials

CWE 521

Emphasizes the need for robust password policies and tools that empower users to set strong, unique, and secure passwords, thereby enhancing the overall security posture of the application or system.

Verification Requirement: Verify that a password strength meter is provided to help users set a stronger password.

Additional Comments: No password strength meter was present, and for a more correct and secure way of defining passwords, users should be able to visualize the strength of their passwords as they are creating them.

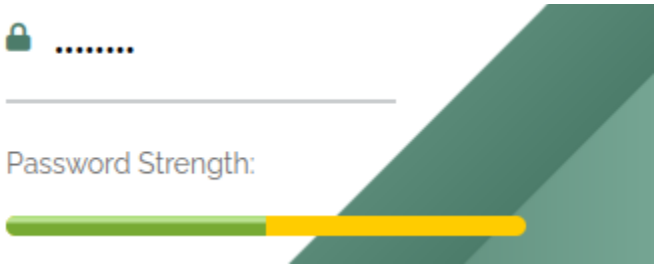
Implementation: To address the absence of a password strength meter, a dynamic and interactive password strength meter was implemented within the registration or password-setting interface. The meter provided real-time feedback to users as they input their chosen passwords.

This implementation aimed to enhance user awareness regarding the strength of their chosen passwords, guiding them towards creating more robust and secure credentials. It empowered users to make informed decisions about their password choices, aligning with industry best practices for password security.

The introduced password strength meter played a crucial role in improving the overall security posture of the application by encouraging users to adopt stronger and more resilient passwords during the account creation process and password redefinition.

```
...
const passwordInput = document.getElementById("new-password-field");
const passwordStrengthMeter = document.getElementById("password-strength-meter");
passwordInput.addEventListener("input", function () {
   const password = passwordInput.value;
   const result = zxcvbn(password);
   passwordStrengthMeter.value = result.score;
   passwordStrengthMeter.style.background = getColor(result.score);
});
function getColor(score) {
        switch (score) {
                case 0: return "#cc0000";
                case 1: return "#ff6600";
                case 2: return "#ffcc00";
                case 3: return "#99cc00";
                case 4: return "#00cc00";
                default: return "#cccccc";  }  };
...
```

Prior to implementing a password strength meter, the registration page lacked a crucial element that assesses and communicates the strength of user passwords. This absence presented several limitations and potential security risks (Limited User Guidance, Weak Passwords, Dependency on User Awareness).



Following the implementation of the dynamic password strength meter, significant improvements were observed (Improved User Engagement, Informed Decision-Making, Elevated Security Posture).

2.1.9

Area: Password Security Credentials

CWE 521

Emphasizes the need for robust password policies and tools that empower users to set strong, unique, and secure passwords, thereby enhancing the overall security posture of the application or system.

Verification Requirement: Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.

Additional Comments: In the past, our password composition rules mandated a mix of uppercase letters, lowercase letters, and digits, with a minimum length of 8 characters. However, recognizing the evolving landscape of password security and the desire for user flexibility, these composition rules have been re-evaluated and relaxed. The new approach encourages longer, more diverse passwords without mandating specific character types.

Implementation: To accommodate the updated verification requirements, the `password_checker` function underwent significant revisions. Previously, it enforced a combination of uppercase letters, lowercase letters, and digits, with a length greater than 7 characters.

```
def password_checker(s):
        size = (len(s)>7)
        digits = any(i.isdigit() for i in s)
        caps = any(i.isupper() for i in s)
        if size & digits & caps:
                return True
        else: return False
```

The function has been re-engineered to remove these character type constraints while ensuring that passwords are between 12 and 127 characters in length. By removing unnecessary composition rules, the application now provides users with greater flexibility in creating passwords, without compromising security. This change reflects our commitment to user-centric security practices and fosters a more positive user experience.

```
def password_checker(s):
        s = ' '.join(s.split())
        size = len(s) >= 12 and len(s) <= 127
        if size:
                return True
        return False
```

2.1.12

Area: Password Security Credentials

CWE 521

Emphasizes the need for robust password policies and tools that empower users to set strong, unique, and secure passwords, thereby enhancing the overall security posture of the application or system.

Verification Requirement: Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.

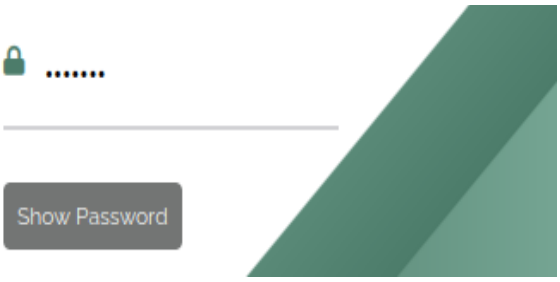
Additional Comments: Users couldn't choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password. The only option was to visualize the password masked every time. Adding an option is essential so Users can momently visualize if their password is written the way they want.

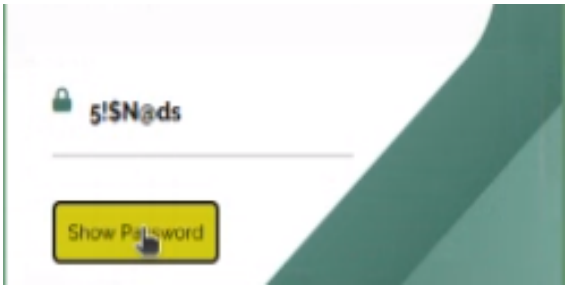
Implementation: To address the limitation of not allowing users to temporarily view the entire masked password or the last typed character, an enhancement was made to the password input functionality. Users can now either view the entire masked password or momentarily revealing the password while pressing the button. This implementation provides users with flexibility and convenience during the password input process.

This enhancement aligns with best practices for user experience and accessibility, allowing users to verify the accuracy of their entered password without compromising security. By offering these options, the application enhances user satisfaction and usability.

```
togglePassword = function (show) {
        const passwordInput = loginForm.querySelector('input[name="password"]');
        passwordInput.type = show ? "text" : "password";
};
```

The implemented features provided several benefits that significantly enhanced the user experience (Improved User Satisfaction, Reduced Chances of Input Errors, Enhanced Usability of Password Input Functionality).


.....
Show Password


5!$N@ds
Show Password

2.2.1

Area: General Authenticator Requirements

CWE 307

"Improper Restriction of Excessive Authentication Attempts," refers to a security weakness where an application does not properly limit the number of authentications attempts a user can make, making it susceptible to brute force attacks.

Verification Requirement: Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.

Additional Comments: There weren't any anti-automation controls because there was nothing to control. Creating a CAPTCHA for verifying that no more than 100 failed attempts per hour seems to be the better option to control the app flow with anti-automation.

Implementation: The absence of anti-automation controls in the application left it vulnerable to various security threats, including breached credential testing, brute force attacks, and account lockout attempts. Without effective controls in place, the application lacked mechanisms to prevent malicious actors from systematically attempting to gain unauthorized access. This posed a significant security risk as attackers could exploit the absence of protective measures, potentially leading to compromised accounts, unauthorized access, or denial-of-service attacks.

In recognizing the importance of mitigating these threats, it became evident that implementing robust anti-automation controls was crucial. The team identified the need for measures such as introducing CAPTCHA challenges. These controls aimed to establish a multi-layered defense against automated attacks and enhance the overall security posture of the application.

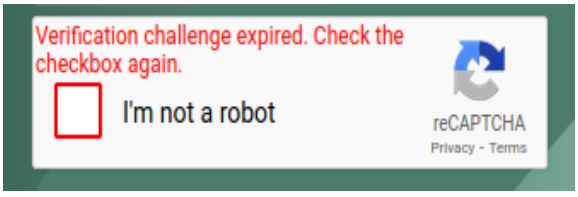
```
...

# Validate reCAPTCHA
   recaptcha_response = request.form.get('g-recaptcha-response')
   secret_key = '6LeO8EApAAAAAEuL6cVVl4l9Dsoin_NUeyngcpiD'  # Replace with your secret key
   recaptcha_data = {
      'secret': secret_key,
      'response': recaptcha_response,
   }
   recaptcha_verification = requests.post('https://www.google.com/recaptcha/api/siteverify',
data=recaptcha_data)
   recaptcha_result = recaptcha_verification.json()
};

...
```

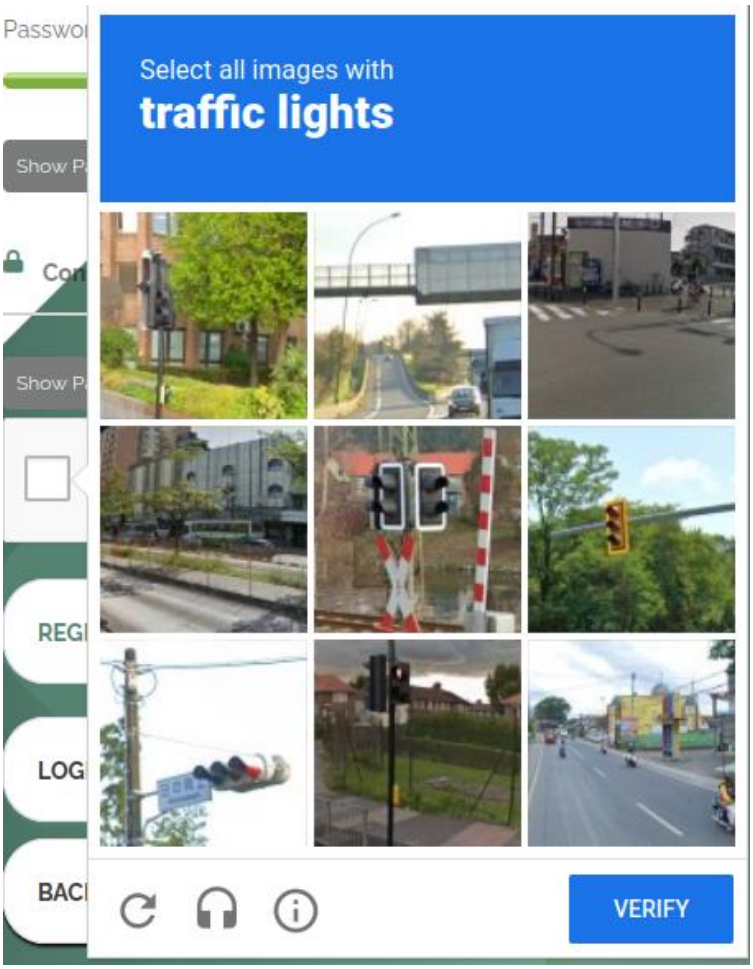
---

register.html:

```
<link href="https://fonts.googleapis.com/css2?family=Jost:wght@500&display=swap" rel="stylesheet">
...
<div class="g-recaptcha" data-sitekey="6LeO8EApAAAAALs9BZLVOYP5f8z6YkEeeImArfZr"></div>
...
```

The newly integrated CAPTCHA challenge, which played a pivotal role in verifying that no more than 100 failed attempts per hour were possible on a single account. The CAPTCHA mechanism acted as a deterrent to automated attacks, ensuring that malicious actors were thwarted in their attempts to exploit vulnerabilities.





The implementation of anti-automation controls brought forth several benefits, contributing to a more secure and resilient application (Mitigation of Automated Threats, Optimized Application Flow with Anti-Automation).

2.2.2

Area: General Authenticator Requirements

CWE 304

Also known as "Missing Critical Step in Authentication," refers to situations where an application relies on weak authenticators, such as SMS or email, without implementing stronger methods for user verification.

Verification Requirement: Verify that the use of stronger methods is offered instead, or before, any of the weaker ones, such as SMS or email.

Additional Comments: To ensure that CWE 304 isn't present in our project, we chose not to implement any verification using weak authenticators. Instead, we chose to implement, as a stronger method, a TOTP two factor authentication, using Google Authenticator.

As for our previous iteration of this project, the authentication process was simply based on checking whether the user had provided the right credentials to access our website. Although the password was kept in a cyphered database, in a hashed format, implementing 2FA added the needed extra layer of security.

Implementation: In this new iteration, we made a few changes to our application, which will be addressed here:

REGISTER:

When a user registers to our website, we make use of the PYOTP library and generate a random secret key, using Base32 encoding, for that user. That key is then retrieved to the 'secret_key' variable, which is stored in our database, on a new field in the 'users' table, called totp_secret. This value will later be used to authenticate our user when he tries to log in.

```
totp = pyotp.TOTP(pyotp.random_base32())
secret_key = totp.secret
```

LOG IN:

When a user tries to log in, if the credentials provided check out to be right, then the user is redirected to a new page in our app, '2fa.hmtl'.

Note: We couldn't attach the demonstration image to the Word document because it includes a QR code, and Word does not allow this. The image can be accessed in the 'analysis/ASVS/images/#2.2.2' directory.

In this new page, a QR Code is generated based on the user's secret key (totp_secret) and user ID:

```
var totpUri = 'otpauth://totp/DETIStore?secret=' + secretKey +
'&issuer=SIO - DETIStore - ' + id + ' user';
var qrcodeContainer = document.getElementById('qrcode');
var qrcode = new QRCode(qrcodeContainer, {
     text: totpUri,
     width: 200,
     height: 200
 });
```

The user now has to have a second device to continue the log-in process. Once Google Authenticator is installed on this device, the user will scan the QR Code, and the authenticator app will start generating TOTPs, which the user will need to submit in order to successfully log in.

VERIFICATION:

After the user has completed all the necessary steps to log in, all that's left is to verify is if the provided code was correct. To do this, we create a new TOTP object using the user's secret key stored in the database. Then, by using the 'verify' method of this TOTP object, it's possible to verify if the correct code was provided. If that's the case, a token is generated, the user is logged in, and then redirected based on the type of user he is (admin or normal user); otherwise, the log in fails.

```
user = users.query.filter_by(id=user_id).first()
  totp = pyotp.TOTP(user.totp_secret)
   if totp.verify(user_provided_code):
          twofactorauth = True
   else:
          twofactorauth = False
```

2.2.3

Area: General Authenticator Requirements

CWE 620

The "Unverified Password Change" vulnerability, involves a scenario where a software application allows a user to change their password without verifying the user's old or current password adequately.

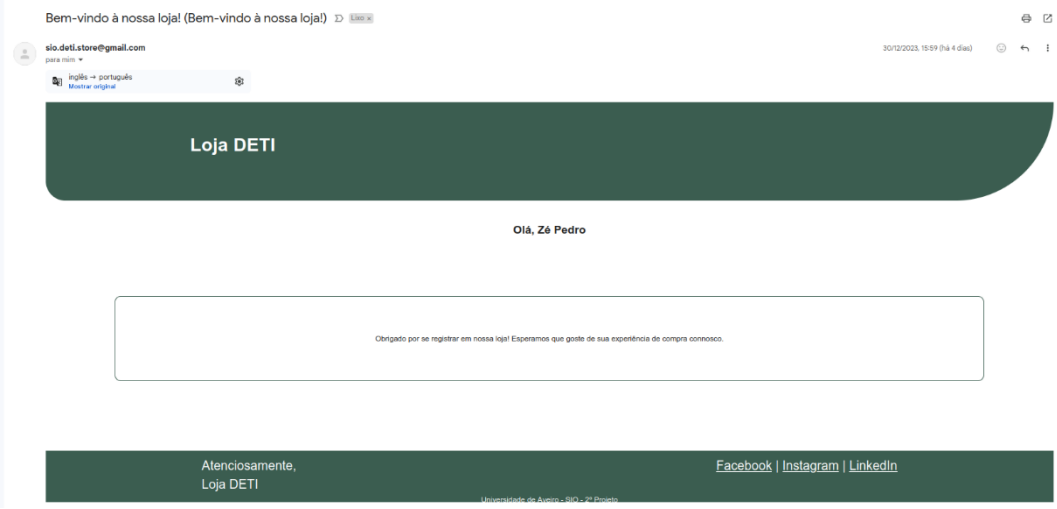
Verification Requirement: Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.

Additional Comments: To enhance user security and mitigate the risks, a notification system has been implemented. This system ensures that users receive timely notifications via email when significant authentication-related events occur. These events include account creation, password changes, and any administrative actions that modify user credentials.

Implementation: A dedicated function, `send_email`, has been developed to facilitate the sending of notification emails to users. This function is invoked during key authentication-related events, ensuring that users are promptly informed about changes to their accounts. The function utilizes a template-based approach to generate the content of the emails, providing a consistent and user-friendly notification experience. By implementing this notification system, the application enhances transparency and accountability, thereby bolstering overall security measures and user trust.

```
send_email(email, 'Bem-vindo à nossa loja!', name, 'Obrigado por se registar na nossa loja! Esperamos que goste da sua experiência de compra connosco.')
```

```
def send_email(recipient_email, subject, name, message):
        recipient_email = recipient_email
        subject = subject
        template_vars = {
                'name': name,
                'message': message
        }
        html_content = render_template('email_template.html', **template_vars)
        try:
                msg = Message(subject=subject, recipients=[recipient_email], html=html_content,
              sender=app.config['MAIL_USERNAME'])
                mail.send(msg)
        except Exception as e:
                return str(e)
        return True
```



2.5.4

Area: Credential Recovery Requirements

CWE 16

Titled "Configuration," refers to vulnerabilities related to insecure or improper configurations that could lead to security issues. This includes situations where default settings are not secure, configurations expose sensitive information, or where misconfigurations create security weaknesses.

Verification Requirement: Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").

Additional Comments: In the past, the application utilized default account names such as "user" and "admin" for user and administrative roles, respectively. While these accounts were initially established for testing and development purposes, they present a security risk if not properly managed or removed from the production environment. To address this vulnerability, a comprehensive review of user accounts has been conducted, and all shared or default accounts, including "user" and "admin", have been identified and deactivated or renamed to enhance security.

Implementation: To mitigate the risk associated with improper authentication, a series of corrective actions have been implemented. Firstly, shared or default accounts, such as "user" and "admin", have been identified and securely deactivated or renamed.

Past Users Credentials:

User name: user email: usersec@example.com password: Password123!	Administrator name: admin email: adminsec@example.com password: Password123!

Secondly, the application's authentication mechanism has been enhanced to prevent the use of default or commonly used account names. These changes ensure that each user is uniquely identified within the system, thereby reducing the likelihood of unauthorized access and enhancing overall security posture.
Present Users Credentials:

User name: Ze Pedro email: zepedro@email.com password: 123456789pass	Administrator name: Joao Paulo email: joaopaulo@detistore.com password: 123456789pass

2.7.2

Area: Out of Band Verifier Requirements

CWE 287

Titled "Improper Authentication," it is a vulnerability that occurs when an application does not properly authenticate users before granting access to sensitive information or functionality.

Verification Requirement: Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.

Additional Comments: In the past, tokens were set to expire after 1 hour. Currently, the token expiration has been enhanced to a more secure timeframe, reducing the window of vulnerability to 10 minutes.

Implementation: To address the vulnerability, the token expiration period for out-of-band authentication requests, codes, or tokens was adjusted to adhere to the 10-minute timeframe. This implementation strengthens the application's authentication mechanism, reducing the risk of unauthorized access and improving overall security.

Following the implementation, the application now enforces a more stringent token expiration policy, significantly reducing the window during which authentication tokens remain valid. This improvement enhances the overall security posture by promptly invalidating authentication tokens, minimizing the risk of improper authentication.

2.7.3

Area: Out of Band Verifier Requirements

CWE 287

Titled "Improper Authentication," it is a vulnerability that occurs when an application does not properly authenticate users before granting access to sensitive information or functionality.

Verification Requirement: Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.

Additional Comments: In the past, tokens were generated each time a user logged into their account. However, after the enhancement, tokens are now generated and associated with a unique user session. When a user logs out, the token is securely stored to ensure it is used only once, maintaining the principle of one-time usability and enhancing security.

Implementation: To address the vulnerability, a modification was made to the token generation process. Tokens are now tied to specific authentication requests and user sessions. Upon user logout, the token is securely stored, preventing unauthorized reuse.

```
invalid_tokens = set()

@app.route('/logout', methods=['POST'])
def logout():
    token = request.headers.get('Authorization')
    payload = verify_token(token)
    if not payload:
        return jsonify({'success': False, 'error': 'Invalid token'}), 401
    invalidate_token(token)
    return jsonify({'success': True}), 200
```

Following the implementation, the application now ensures that tokens are generated and utilized only once, enhancing the overall authentication security by preventing misuse and unauthorized access.

2.8.1

Area: Single or Multi Factor One Time Verifier Requirements

CWE 613

Titled "Insufficient Session Expiration," CWE 613 is a vulnerability that arises when a web application does not properly invalidate or expire session tokens after a user logs out or when a session reaches its designated timeout.

Verification Requirement: Verify that time-based OTPs have a defined lifetime before expiring.

Additional Comments: As previously mentioned, the TOTP login was implemented with Google Authenticator. Upon checking this application's documentation, we learn that the lifetime for Google Authenticator's code is 1 minute. This value is hardcoded and cannot be changed. With this said, by using Google Authenticator, we have the verification that time-based OTPs have a defined lifetime before expiring.

Implementation: The implementation of the 2FA using Google Authenticator is documented above, on **#2.2.2.**

Session Management

3.7.1

Area: Defenses Against Session Management Exploits

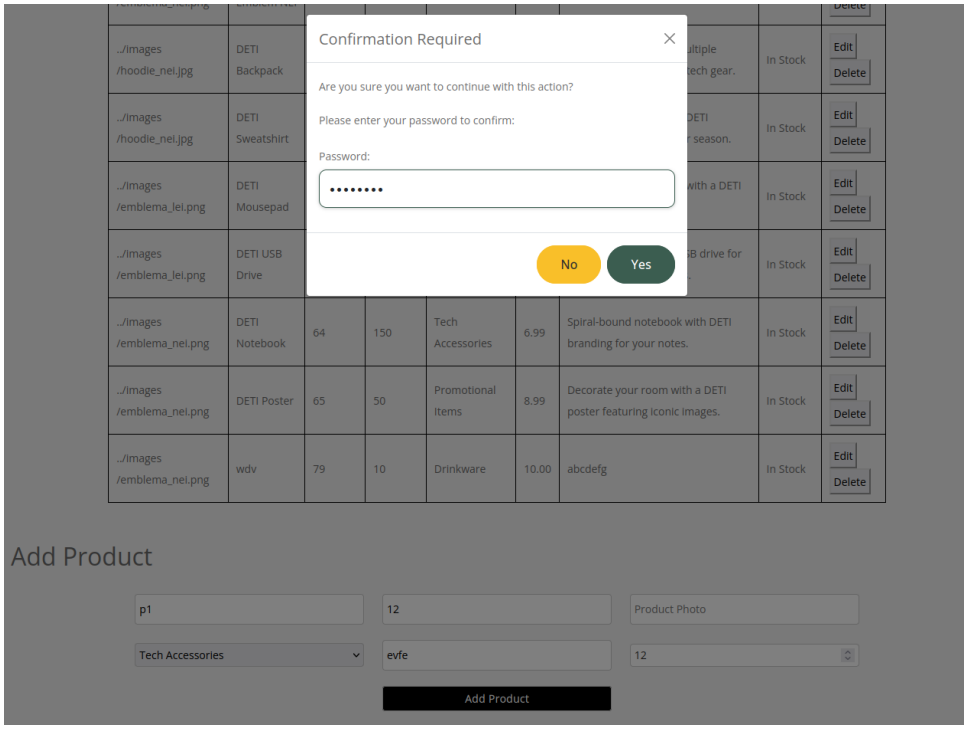
CWE 778

Titled "Insufficient Logging," refers to a security weakness where an application does not generate adequate log entries for relevant security events. The application was vulnerable to insufficient logging, which could potentially compromise sensitive transactions or account modifications.

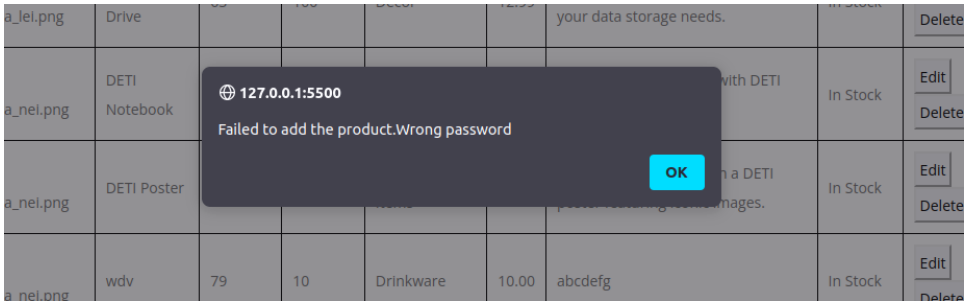
Verification Requirement: Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.

Additional Comments: To mitigate the risk of sensitive data exposure and unauthorized actions, the application has implemented a robust defense mechanism. Specifically, when performing operations related to sensitive information, such as adding, editing, or deleting products, which are functions solely accessible to administrators, the system requires users to re-enter their password. If the correct password is not provided or if no re-authentication occurs, the operation cannot be completed. This additional layer of security ensures that only authorized administrators can execute critical actions, enhancing the overall defense against session management exploits.

Implementation: The implementation involves incorporating a password re-authentication step for sensitive operations within the admin functionalities. For tasks like adding, editing, or deleting products, users with administrative privileges are prompted to provide their password again.



If the correct password is not entered, or if re-authentication fails, the system prevents the completion of the sensitive transaction, thereby safeguarding against unauthorized access and ensuring that only authenticated administrators can perform these critical actions.



Access Control

4.3.1

Area: Other Access Control Considerations
CWE 419

Administrative interfaces are crucial components of applications or systems that manage sensitive functions and data. To enhance security and prevent unauthorized use, it is essential to implement appropriate multi-factor authentication (MFA) mechanisms.

Verification Requirement: Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.

Additional Comments: To ensure that we prevent unauthorized access to admin pages and functions, we made sure that no user, neither normal or admin, can log in without going through the TOTP two factor authentication.

Implementation: The implementation of the 2FA using Google Authenticator is documented above, on **#2.2.2.**

Error Handling and Logging

#7.1.1

Area: Log Content Requirements

CWE 532

Titled "Inclusion of Sensitive Information in Log Files," this weakness relates to the improper handling or logging of sensitive information, leading to potential exposure.

Verification Requirement: Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.

Additional Comments: In our application, we have configured a log file that only logs information that excludes log credentials and/or payment details. This log file only logs generic information such as user IDs and date and time of events, such as: a user has registered himself; a user is trying to log in (hasn't gone through 2FA yet); a user has logged in; a user has added a review; a user has made a purchase. We also have logs for when an admin adds, deletes or edits products.

Implementation: The code below establishes a logging configuration with a file handler, a specific log format, and a logger associated with a particular log file. The logger is set to record log entries of INFO level or higher. This configuration allows controlled logging to the "app_logs.log" file with a customized format.

```
file_handler = logging.FileHandler("app_logs.log")
formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
file_handler.setFormatter(formatter)

logger = logging.getLogger("app_logs.log")
logger.setLevel(logging.INFO)
logger.addHandler(file_handler)
```

With the log file configurated, all that's left is to define what and when to write to it. We do this, for example, when a user logs in:

```
logger.info(f"User {user_id} logged in successfully")
```

#7.1.2

Area: Log Content Requirements

CWE 532

Titled "Inclusion of Sensitive Information in Log Files," this weakness relates to the improper handling or logging of sensitive information, leading to potential exposure.

Verification Requirement: Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy. ([C9](https://owasp.org/www-project-proactive-controls/#div-numbering)).

Additional Comments: As mentioned above, in our application, we have configured a log file that only logs information that excludes sensitive data. This log file only logs generic information such as user IDs and date and time of events.

Implementation: The implementation of the log file is documented above, at **#7.1.1**.

Configuration

14.2.2

Area: Dependency

CWE 307

Titled "Sensitive Data Discovery," it refers to a weakness where an application or system unintentionally exposes sensitive information. This could include the presence of default accounts, sample configurations, documentation, or unnecessary features that could potentially lead to security vulnerabilities.

Verification Requirement: Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.

Additional Comments: Some old features were dependent on some functions. As these features were substituted their functions, flask calls, etc. weren't removed and are in need to be removed.

Implementation: To address the presence of unneeded features and outdated functions:

1. Conducted a thorough review of the application codebase, configurations, documentation, and sample applications to identify any remnants of old or unnecessary features.
2. Removed deprecated functions, unused configurations, and any sample applications that were no longer required for the application's functionality.

This implementation successfully ensured that only essential and secure components remained within the application, aligning with best practices for minimizing the risk of sensitive data exposure.

The benefits of the implemented changes, such as: Reducing attack surface, Improvement of maintainability and the Enhancement of the security posture.

This implementation aims to ensure that only essential and secure components remain within the application, aligning with best practices for minimizing the risk of sensitive data exposure.

Conclusion

This project, aimed at bolstering the security of our e-commerce web application, has reached its culmination. It has been a path marked by significant learning, challenges, and ultimately, rewarding achievements in web security.

In concluding our project on securing an e-commerce web application, we have journeyed through a landscape rich in learning and practical applications. Our approach, grounded in the utilization of Common Weakness Enumeration (CWE) standards, has been instrumental in identifying and addressing various security vulnerabilities. While specific aspects such as password security credentials served as examples, our scope was much more extensive, covering a broad range of security measures.

The integration of CWE standards into our security strategy was not just a compliance exercise, but a significant educational experience. It allowed us to understand the intricacies of web application vulnerabilities and the best practices for mitigating them. This approach has led to the development of a more robust, secure, and reliable e-commerce platform.

Perhaps the most significant takeaway from this project is the depth of understanding and expertise we have gained. This knowledge transcends the immediate scope of our current project. It equips us with a versatile toolkit of security strategies and practices that can be applied to future projects, ensuring that the lessons learned here extend far beyond the confines of this single application.

Our project stands as a testament to the fact that applying comprehensive security standards like CWE can immensely benefit web applications, especially in sensitive domains like e-commerce. It is a clear indication that prioritizing security in web development not only protects the application and its users but also enhances the overall quality and credibility of the platform.

In essence, our journey in this project has been one of growth, learning, and successful application. We have not only fortified our e-commerce web application against potential threats but have also prepared ourselves to face and address the security challenges in the fast-evolving digital world.