



universidade  
de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

**Curso:** 8321 - Licenciatura em Engenharia Eletrotécnica e de Computadores

**Disciplina:** 42573 - Segurança Informática e nas Organizações

**Ano letivo:** 2023/2024

# Vulnerabilities

107715 — Afonso Rodrigues

Universidade de Aveiro, 6 de setembro de 2024

---

## Conteúdo

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introdução</b>  | <b>2</b>  |
| <b>2</b> | <b>Vulnerabilities</b>   | <b>3</b>  |
| 2.1      | CWE - 79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') . . . . . | 3         |
| 2.2      | CWE - 352: Cross-Site Request Forgery (CSRF) . . . . .   | 5         |
| 2.3      | CWE - 89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') . . . . . | 7         |
| 2.4      | CWE - 759: Use of a One-Way Hash without a Salt & CWE - 328: Use of Weak Hash . . . . .                  | 9         |
| 2.5      | CWE - 836: Use of Password Hash Instead of Password for Authentication . . . . .                         | 10        |
| 2.6      | CWE - 308: Use of Single-factor Authentication . . . . .   | 10        |
| 2.7      | CWE - 613: Insufficient Session Expiration . . . . .   | 13        |
| <b>3</b> | <b>Conclusão</b>   | <b>14</b> |
| <b>4</b> | <b>Bibliografia</b>  | <b>14</b> |

---

## 1 Introdução

Tendo em conta o contínuo crescimento da transformação digital de serviços e do resto do mundo e dos perigos que aparecem com a má prática dessa transformação, no âmbito da unidade curricular de **SIO**, foi-nos apresentada a realização deste trabalho prático, sendo este relatório o resultado e análise do problema. O objetivo principal do mesmo, é a criação de uma Loja de *Merchandising* do DETI com vulnerabilidades (**/app**), e outro com as mesmas corrigidas (**/app\_sec**).

O presente relatório visa explicar alguns conceitos relativos à temática e explorar as vulnerabilidades encontradas bem como as ferramentas utilizadas.

---

## 2 Vulnerabilities

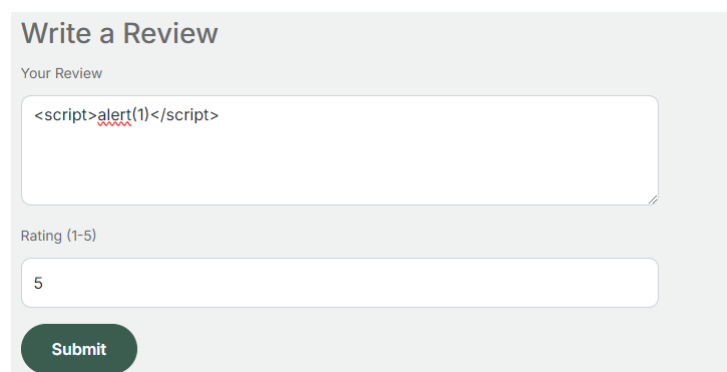
### 2.1 CWE - 79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

O CWE-79 refere-se a uma falha de segurança conhecida como Cross-site Scripting (XSS), que ocorre quando uma aplicação web não sanitiza corretamente a entrada fornecida pelo utilizador antes de a incluir numa página gerada. Isso permite a um atacante injetar scripts maliciosos, que podem ser executados no navegador de outros utilizadores. Estes ataques são usados para roubar cookies, manipular o conteúdo da página ou redirecionar os utilizadores para sites maliciosos.

Para explorar esta vulnerabilidade, escrevi uma *review* com um *script* (figura 2) de forma a que este fique armazenado na base de dados para quando um utilizador aceder ao mesmo produto, irá ficar exposto a esse *script* (figura 3). Isto tudo acontece porque no código do carrossel (figura 1) que demonstra as várias *reviews* de um produto, o texto do comentário aparece como seguro (**safe**) independente do conteúdo do mesmo, possibilitando, neste caso, um *script* de JavaScript.

```
<div class="col-lg-6">
  <h3>Reviews</h3>
  <div id="reviewsCarousel" class="carousel slide" data-bs-ride="carousel" style="top: 35px">
    <div class="carousel-inner">
      {% for review in reviews %}
        <div class="carousel-item {% if loop.first %}active{% endif %}">
          <p>{{ review.COMENTARIO | safe }}</p>
          <small>{{ review.NOME_U }} - {{review.N_STARS}}/5</small>
        </div>
      {% endfor %}
    </div>
    <button class="carousel-control-prev" type="button" data-bs-target="#reviewsCarousel" data-bs-slide="prev">
      <span class="carousel-control-prev-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Previous</span>
    </button>
    <button class="carousel-control-next" type="button" data-bs-target="#reviewsCarousel" data-bs-slide="next">
      <span class="carousel-control-next-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Next</span>
    </button>
  </div>
</div>
```

Figura 1: Código que mostra as várias *reviews* de um certo produto



Write a Review

Your Review

`<script>alert(1)</script>`

Rating (1-5)

5

Submit

Figura 2: *Script* malicioso

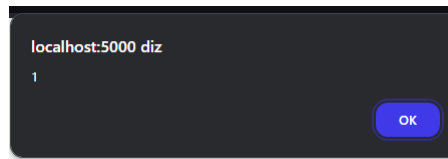


Figura 3: O que aparece cada vez que um utilizador carrega a página

Tendo em conta que o erro que origina esta vulnerabilidade está presente no código do carrossel, para corrigir este problema simplesmente removi o texto como seguro, onde assim esse mesmo texto é processado antes de exposto para verificar a veracidade do mesmo (figura 4 e figura 5).

```
<div class="col-lg-6">
  <h3>Reviews</h3>
  <div id="reviewsCarousel" class="carousel slide" data-bs-ride="carousel" style="top: 35px">
    <div class="carousel-inner">
      {% for review in reviews %}
      <div class="carousel-item {% if loop.first %}active{% endif %}">
        <p>{{ review.COMENTARIO }}</p>
        <small>{{ review.NOME_U }} - {{review.N_STARS}}/5</small>
      </div>
      {% endfor %}
    </div>
    <button class="carousel-control-prev" type="button" data-bs-target="#reviewsCarousel" data-bs-slide="prev">
      <span class="carousel-control-prev-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Previous</span>
    </button>
    <button class="carousel-control-next" type="button" data-bs-target="#reviewsCarousel" data-bs-slide="next">
      <span class="carousel-control-next-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Next</span>
    </button>
  </div>
</div>
```

Figura 4: Código corrigido do carrossel de *reviews*

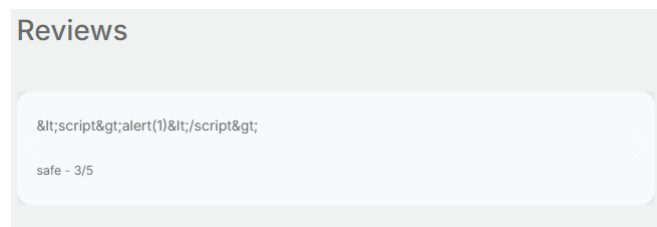


Figura 5: XSS - Resultado depois da correção

## 2.2 CWE - 352: Cross-Site Request Forgery (CSRF)

CWE-352, conhecido como Cross-Site Request Forgery (CSRF), é uma vulnerabilidade em que um atacante força um utilizador autenticado a executar ações indesejadas numa aplicação web sem o seu conhecimento. Isso ocorre porque a aplicação não verifica adequadamente a origem dos *requests*, permitindo que um invasor envie requisições maliciosas através do *browser* do utilizador.

De forma a testar esta vulnerabilidade, foi criado um servidor *test\_server* que envia para a aplicação (/app) uma *review* maliciosa de um produto, apresentado na figura 6. Como as figuras 8 e 7 mostram, a aplicação está vulnerável a este tipo de ataque onde cada vez que um utilizador entra na página de um produto está exposto a possíveis ataques externos.

```
1  # SERVIDOR DE TESTE PARA ATAQUES DE CSRF
2
3  from flask import Flask, render_template_string
4  # from flask_cors import CORS
5
6  app = Flask(__name__)
7
8  @app.route('/')
9  def csrf_attack():
10     html_content = '''
11     <!DOCTYPE html>
12     <html lang="en">
13     <head>
14         <meta charset="UTF-8">
15         <title>CSRF Attack Test</title>
16     </head>
17     <body>
18         <h1>CSRF Attack Test</h1>
19         <form id="csrfForm">
20             <input type="hidden" id="reviewText" name="reviewText" value="Foste Atacado!!!!!!!!!!">
21             <input type="hidden" id="reviewStars" name="reviewStars" value="5">
22             <input type="hidden" id="productName" name="productName" value="Porta-Chaves DETI">
23             <input type="hidden" id="username" name="username" value="admin">
24             <button type="submit">Submit</button>
25         </form>
26
27         <script>
28             document.getElementById('csrfForm').addEventListener('submit', function (e) {
29                 e.preventDefault();
30
31                 const reviewText = document.getElementById('reviewText').value;
32                 const reviewStars = document.getElementById('reviewStars').value;
33                 const productName = document.getElementById('productName').value;
34                 const username = document.getElementById('username').value;
35
36                 fetch('http://192.168.150.228:5000/submit_review', {
37                     method: 'POST',
38                     headers: {
39                         'Content-Type': 'application/json'
40                     },
41                 })
42             });
43         </script>
44     </body>
45     </html>
46     '''
47     return render_template_string(html_content)
```

Figura 6: Código do servidor de ataque

### CSRF Attack Test

Submit

192.168.1.112:5001 diz  
Review submitted successfully!

OK

Figura 7: Resultado do envio da *review* maliciosa

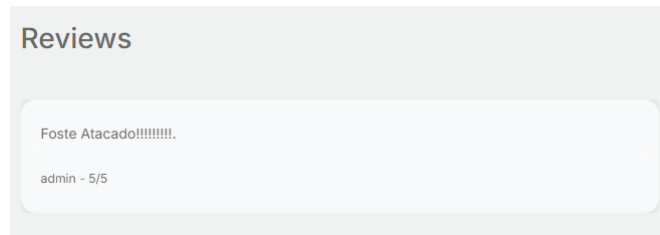


Figura 8: A *review* maliciosa

Para combater esta forma de ataque é necessário fazer alterações na API que suporta a aplicação. Desta forma, usando a biblioteca *flask\_wtf.cors*, ativando a proteção de CSRF e garantindo que todos *requests* têm um adequado CSRF *token*, garantimos a proteção da aplicação contra este tipo de ataques.

```
app = Flask(__name__)
CORS(app)
csrf = CSRFProtect(app)

app.register_blueprint(rotas)

app.secret_key='log key'
```

Figura 9: Ativação da proteção CSRF na API

```
<div class="col-1g-6">
  <h3>Write a Review</h3>
  <form id="reviewForm">
    <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
    {% if 'logged_in' in session and session['logged_in'] %}
    <div class="mb-3">
      <label for="reviewText" class="form-label">Your Review</label>
      <textarea class="form-control" id="reviewText" name="reviewText" rows="4" required></textarea>
    </div>
    <div class="mb-3">
      <label for="reviewStars" class="form-label">Rating (1-5)</label>
      <select class="form-control" id="reviewStars" name="reviewStars" required>
        <option value="5">5</option>
        <option value="4">4</option>
        <option value="3">3</option>
        <option value="2">2</option>
        <option value="1">1</option>
      </select>
    </div>
    <input type="hidden" id="productName" name="productName" value="{{ product.NOME }}">
    <input type="hidden" id="username" name="username" value="{{ session['username'] }}">
    <button type="submit" class="btn btn-primary">Submit</button>
  </form>
</div>
```

Figura 10: Alteração no formulário de envio de *reviews*

## Method Not Allowed

The method is not allowed for the requested URL.

Figura 11: CSRF - Resultado depois da correção

## 2.3 CWE - 89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

O CWE-89, conhecido como SQL Injection, acontece quando uma aplicação web permite que comandos SQL maliciosos sejam injetados através da entrada do utilizador. Sem o devido cuidado, um invasor pode manipular consultas SQL, acessando, modificando ou apagando dados sensíveis da base de dados.

Por exemplo, na parte insegura da aplicação (/app), na página de um produto (produto.html), ao inserir na campo da review, "(Comentário)'; DROP TABLE Carrinho; -- //", e posteriormente clicar no botão "Submit" o review irá ser válida e, neste caso, cada vez que a página é recarregada, irá aparecer o comentário feito e a tabela **Carrinho** será apagada.

```
@app.route('/submit_review', methods=['POST'])
def submit_review():
    data = request.json
    review_text = data.get('reviewText')
    review_stars = data.get('reviewStars')
    product_name = data.get('productName')
    username = data.get('username')

    try:
        (function) def get_connection() -> Connection
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute(f"INSERT INTO Review (NOME_U, NOME_P, N_STARS, COMENTARIO) VALUES ('{username}', '{product_name}', {review_stars}, '{review_text}')"
        conn.commit()

        cursor.close()
        conn.close()

        return jsonify({'success': True, 'message': 'Review submitted successfully!'})
    except Exception as e:
        print(f"Error inserting review: {e}")
        return jsonify({'success': False, 'message': 'An error occurred while submitting your review.'})
```

Figura 12: Código da submissão de uma *review* para a base de dados

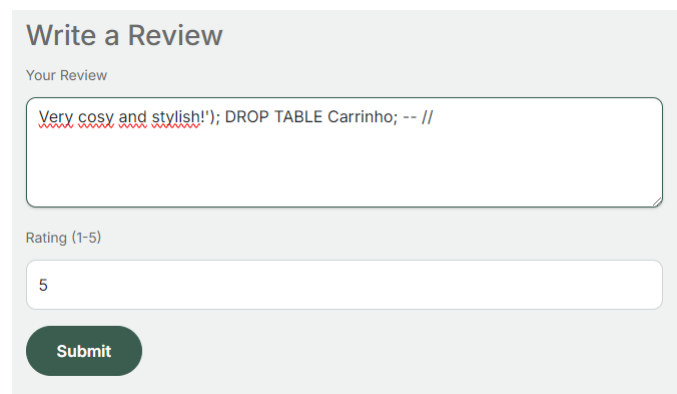


Figura 13: Comentário com comando SQL



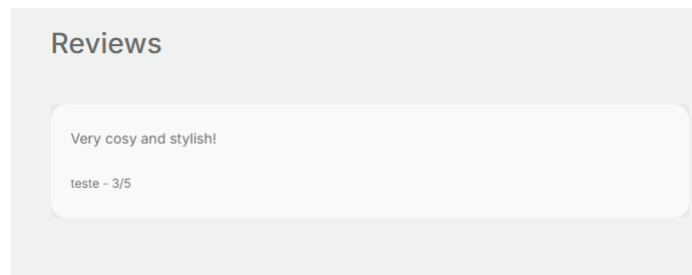


Figura 14: Como um utilizador vê a *Review* maliciosa

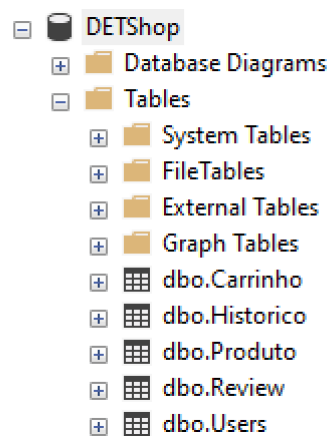


Figura 15: Base de Dados antes do ataque

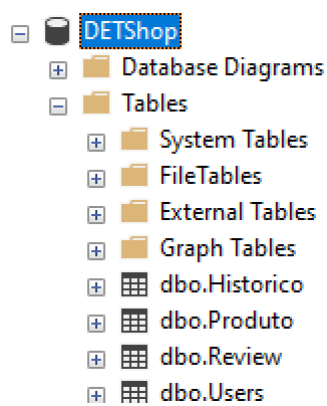


Figura 16: Base de Dados depois do ataque

Para resolver esta vulnerabilidade, acedemos à base de dados através de *parameterized queries*, de forma a que o *review text input* não altere o comando sql, o que origina a vulnerabilidade.

```
@app.route('/submit_review', methods=['POST'])
def submit_review():
    data = request.json
    review_text = escape(data.get('reviewText'))
    review_stars = data.get('reviewStars')
    product_name = escape(data.get('productName'))
    username = escape(data.get('username'))

    try:
        conn = get_connection()
        cursor = conn.cursor()
        cursor.execute(f"INSERT INTO Review (NOME_U, NOME_P, N_STARS, COMENTARIO) VALUES (?, ?, ?, ?)", username, product_name, review_stars,
            review_text)
        conn.commit()

        cursor.close()
        conn.close()

        return jsonify({'success': True, 'message': 'Review submitted successfully!'})
    except Exception as e:
        print(f"Error inserting review: {e}")
        return jsonify({'success': False, 'message': 'An error occurred while submitting your review.'})
```

Figura 17: Código de submissão corrigido

Assim, ao inserir “(Comentario)’); DROP TABLE Carrinho; – //”, a aplicação já não aceita a ação devido ao mesmo não aceitar como o texto está escrito e como resultado, o segundo comando para apagar a tabela não corre.

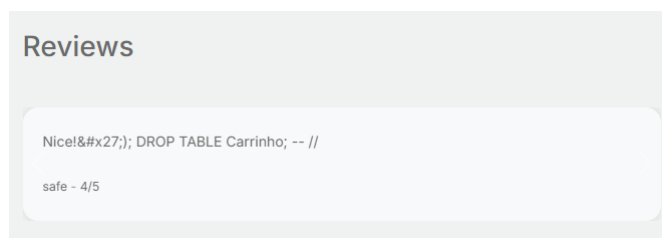
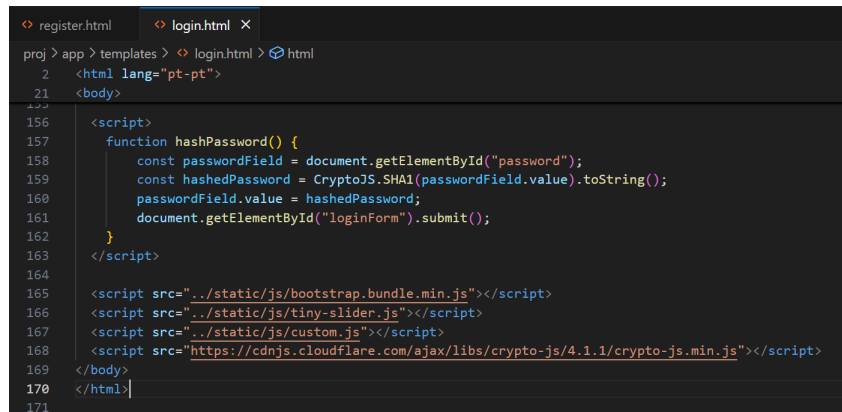


Figura 18: SQL Injection - Resultado depois da correção

## 2.4 CWE - 759: Use of a One-Way Hash without a Salt & CWE - 328: Use of Weak Hash

O CWE-759 e o CWE-328 referem-se ao uso de hashes criptográficos inadequados para proteger senhas. No caso de um “hash sem salt”, a ausência de um valor único associado ao hash torna as senhas mais vulneráveis a ataques de rainbow table. O uso de algoritmos de hash fracos também facilita a quebra de senhas, comprometendo a segurança do sistema.

Na aplicação (/app) tanto nos processos de registo de utilizador como no login, as passwords estão expostas a estes perigos, como mostra a figura 19, devido ao facto de usar o algoritmo SHA-1, que apesar a sua rápida performance, é debilitada para esta função e de não ocorrer o processo de *salting*.



```

proj > app > templates > login.html > html
2 <html lang="pt-pt">
21 <body>
156
157 <script>
158   function hashPassword() {
159     const passwordField = document.getElementById("password");
160     const hashedPassword = CryptoJS.SHA1(passwordField.value).toString();
161     passwordField.value = hashedPassword;
162     document.getElementById("loginForm").submit();
163   }
164 </script>
165 <script src="../../static/js/bootstrap.bundle.min.js"></script>
166 <script src="../../static/js/tiny-slider.js"></script>
167 <script src="../../static/js/custom.js"></script>
168 <script src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.1.1/crypto-js.min.js"></script>
169 </body>
170 </html>
171

```

Figura 19: Código de *hashing* da password

Para corrigir isto, alterei o algoritmo para o SHA-256 devido à sua resistência a colisões e por ser mais segura e, como a figura 20 mostra, o *salting* da password.



```

function hashPassword() {
  const passwordField = document.getElementById("password");
  const salt = CryptoJS.lib.WordArray.random(128 / 8).toString();
  const saltedPassword = salt + passwordField.value;
  const hashedPassword = CryptoJS.SHA256(passwordField.value).toString();
  passwordField.value = salt + ":" + hashedPassword;
  document.getElementById("registerForm").submit();
}

```

Figura 20: Código de *hashing* com as correções

## 2.5 CWE - 836: Use of Password Hash Instead of Password for Authentication

CWE-836 ocorre quando uma aplicação utiliza diretamente o valor hash de uma senha como mecanismo de autenticação, ao invés de autenticar usando a senha em si. Isso enfraquece a segurança, já que o hash pode ser exposto ou utilizado de forma inadequada para contornar o processo de autenticação.

Como apresentado na secção anterior, a aplicação possui um mecanismo de autenticação armazenando a password do utilizador em *hash*.

Para assegurar maior proteção na área de autenticação, o processo de *hashing* foi removido, ou seja, a password escolhida pelo utilizador é a que fica guardada na base de dados.

## 2.6 CWE - 308: Use of Single-factor Authentication

CWE-308 refere-se ao uso de autenticação com um único fator, como apenas uma senha. Esta abordagem é considerada fraca, pois depende de apenas um elemento de

---

segurança, o que facilita ataques de força bruta ou de phishing. A utilização de múltiplos fatores de autenticação (MFA) é recomendada para aumentar a segurança.

Na aplicação insegura (/app), a autenticação é feita da forma "tradicional" onde o utilizador, depois de criar uma conta, insere as suas credenciais e inicia uma sessão (figura 21).

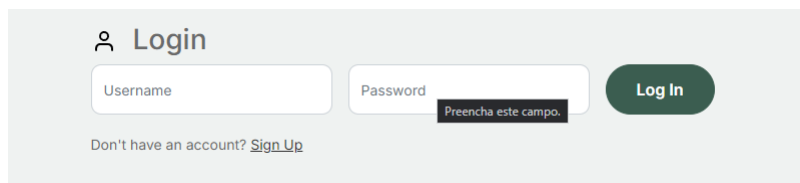


Figura 21: Login na aplicação insegura (/app)

Para garantir um MFA, foi implementado Google Authenticator, mais propriamente o mecanismo de *time-based one-time password (TOTP)*. Para isto, foi alterado o código do registo (figura 22) para associar um TOTP ao utilizador, quando esta fase é concluída o utilizador é redirecionado para uma página com o QR Code gerado (figura 24). Este tem de ser lido através da aplicação móvel da Google Authenticator para o utilizador puder aceder ao seus TOTP's (figura 25).

Estando novamente na página de login (figura 26), para iniciar uma sessão o utilizador apenas tem de inserir as suas credenciais e o código gerado pelo autenticador.

```
@app.route('/register', methods=['POST'])
@csrf.exempt
def register():
    username = request.form['username']
    password = request.form['password']

    otp_secret = pyotp.random_base32()

    uri = pyotp.totp.TOTP(otp_secret).provisioning_uri(name=username, issuer_name='DETSshop')

    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute(f"INSERT INTO Users (USERNAME, PASS, OTP) VALUES (?, ?, ?)", username, password, otp_secret)
    conn.commit()
    cursor.close()
    conn.close()

    session['otp_uri'] = uri

    img = qrcode.make(uri)
    img.save('./static/images/qrcode.png')

    return redirect('loginCode')
```

Figura 22: Código de registo

```

@app.route('/login', methods=['POST'])
@csrf.exempt
def login():
    username = request.form['username']
    password = request.form['password']
    otp_code = request.form['otp_code']
    conn = get_connection()
    cursor = conn.cursor()

    cursor.execute(f"SELECT * FROM Users WHERE USERNAME = ? AND PASS = ?", username, password)
    result = cursor.fetchone()

    cursor.close()
    conn.close()

    if result:
        user_id, stored_password, otp_secret = result

        totp = pyotp.TOTP(otp_secret)
        if not totp.verify(otp_code):
            return render_template('login.html', alert_message="Invalid OTP code")

        if stored_password == password:
            session['logged_in'] = True
            session['username'] = username
            return render_template('profile.html', username=username)
        else:
            return render_template('login.html', alert_message="Invalid username or password")
    else:
        return render_template('login.html', alert_message="Invalid username or password")

```

Figura 23: Código de login



Figura 24: Página do QR Code do Google Authenticator

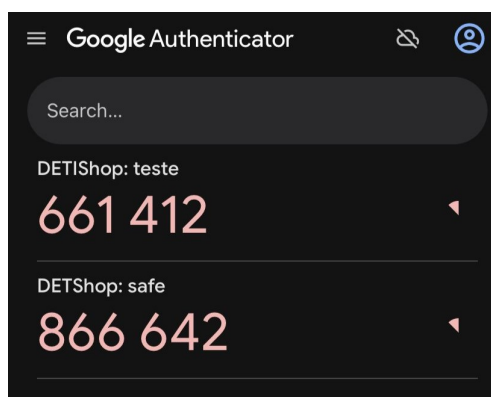


Figura 25: TOTP da aplicação Google Authenticator

Figura 26: Página de login na aplicação segura (/app\_sec)

## 2.7 CWE - 613: Insufficient Session Expiration

CWE-613 ocorre quando as sessões de utilizadores não são encerradas adequadamente após um certo período de inatividade ou logout. Isso permite que um atacante que consiga acesso a uma sessão inativa ainda possa agir em nome do utilizador legítimo, comprometendo a segurança do sistema.

Na (/app), a sessão de um utilizador só termina se o mesmo fizer o *logout*, o que, por exemplo, se uma sessão iniciada em espaço público não for encerrada vai estar completamente exposta por esta não acabar após um intervalo de tempo.

Figura 27: Página de logout

| Name    | Value            | Domain    | Path | Expires / Max-Age | Size | HttpOnly | Secure | Sam... | Partiti... | Cross ... | Prio... |
|---------|------------------|-----------|------|-------------------|------|----------|--------|--------|------------|-----------|---------|
| session | eyJsb2dnZWRFa... | localhost | /    | Session           | 92   | ✓        |        |        |            |           | Me...   |

Figura 28: Sessão depois das alterações

Para a aplicação segura (/app\_sec), a cookie da sessão foi alterada de forma a que passado 2 minutos de inatividade a mesma terminasse.

```
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(minutes=2)

@app.before_request
def make_session_permanent():
    session.permanent = True
```

Figura 29: Alterações no comportamento da sessão

| Name    | Value               | Domain        | Path | Expires / Max-Age        | Size | HttpOnly | Secure | SameSite | Partition K... | Cross Site | Priority |
|---------|---------------------|---------------|------|--------------------------|------|----------|--------|----------|----------------|------------|----------|
| session | eyJlcGVybW50p0cn... | 192.168.1.112 | /    | 2024-09-06T02:32:19.343Z | 68   | ✓        |        |          |                |            | Medium   |

Figura 30: Sessão depois das alterações

---

### 3 Conclusão

Neste relatório, identifiquei e corriji várias vulnerabilidades de segurança numa aplicação web. Abordei questões como *SQL Injections*, ataques de XSS e CSRF, com o uso de password hash a importância de usar *Salt* e uma hash com maiores capacidades para aumentar a segurança, o porque de usar múltiplos fatores de autenticação e como uma sessão ilimitada pode corromper uma aplicação.

Ao analisar e resolver essas vulnerabilidades, aprendi práticas importantes de segurança no desenvolvimento de software. Transformei uma aplicação de insegura em segura, protegendo-a contra possíveis ameaças.

### 4 Bibliografia

- <https://cwe.mitre.org/data/definitions/79.html>
- <https://cwe.mitre.org/data/definitions/89.html>
- <https://cwe.mitre.org/data/definitions/613.html>
- <https://cwe.mitre.org/data/definitions/328.html>
- <https://cwe.mitre.org/data/definitions/759.html>
- <https://cwe.mitre.org/data/definitions/836.html>
- <https://cwe.mitre.org/data/definitions/308.html>
- <https://cwe.mitre.org/data/definitions/352.html>