



Projeto 1 – DetiShop

SEGURANÇA INFORMÁTICA E NAS ORGANIZAÇÕES



Alexandre Martins, 103552

Diogo Pires, 97889

Diogo Silva, 104341

Tomás Rodrigues, 104090

5/11/2023

Índice

1. Introdução.....	3
2. Funcionalidades	4
2.1 Gestão de utilizadores.....	4
2.3 Carrinho de compras e Wishlist.....	6
2.4 Processo de Checkout.....	7
2.5 Gestão de inventário.....	8
2.6 Histórico de Pedidos	9
2.7 Opiniões e Avaliações.....	10
3. Vulnerabilidades.....	11
3.1 CWE 89 – SQL Injection	11
3.1.1 Exploração da vulnerabilidade	11
3.1.2 Resolução da vulnerabilidade	11
3.2 CWE 79– Cross Site Scripting	12
3.2.1 Exploração da vulnerabilidade.....	12
3.2.2 Resolução da vulnerabilidade.....	12
3.2 CWE 256 – Plaintext storage of a password.....	13
3.2.1 Exploração da vulnerabilidade.....	13
3.2.2 Resolução da vulnerabilidade.....	14
3.3 CWE – 620 Unverified Password Change	15
3.3.1 Exploração da vulnerabilidade.....	15
3.4 CWE-311 Missing Encryption of sensitive data	16
3.4.1 Exploração da vulnerabilidade.....	16
3.4.2 Resolução da vulnerabilidade.....	17
3.5 CWE – 549 Missing password field masking.....	18
3.5.1 Exploração da vulnerabilidade.....	18
3.5.2 Resolução da vulnerabilidade.....	18
3.6 CWE - 756: Missing custom error page	19
3.6.1 Exploração da vulnerabilidade.....	19
3.6.2 Resolução da vulnerabilidade.....	19
3.7 CWE – 284 : Access Control Issues	20
3.7.1 Exploração de vulnerabilidade	20

3.7.2 <i>Resolução de vulnerabilidade</i>	20
4. Conclusão	21

1. Introdução

Este projeto e respetivo relatório foram efetuados no âmbito da disciplina de Segurança Informática e nas Organizações.

O objetivo deste projeto é criar uma loja relacionada com o Departamento de Telecomunicações e Informática da Universidade de Aveiro implementando algumas funcionalidades.

As principais tecnologias usadas foram Flask e SQLite 3.

Existem duas versões desta loja, uma versão segura e uma versão vulnerável, na versão vulnerável existem algumas vulnerabilidades que podem ser exploradas de modo a comprometer o sistema. Na versão segura, estas vulnerabilidades foram corrigidas tornando o sistema completamente seguro.

Em seguida apresentamos uma lista das vulnerabilidades apresentadas:

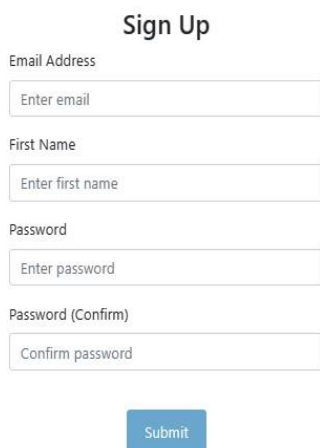
- CWE 79 - Cross Site Scripting
- CWE 89 – SQL Injection
- CWE 256 – Plaintext storage of a password
- CWE 620 – Unverified Password Change
- CWE 311 – Missing Encryption of sensitive data
- CWE 549 – Missing password field masking
- CWE 756 – Missing custom error page
- CWE 284 – Access Control Issues

2. Funcionalidades

Nesta secção iremos abordar de que forma implementámos as funcionalidades principais da nossa aplicação.

2.1 Gestão de utilizadores

Para efetuar a gestão de utilizadores implementámos uma página de registo, uma página de login, uma página onde é possível a um utilizador consultar os seus dados pessoais e mudar a sua password.



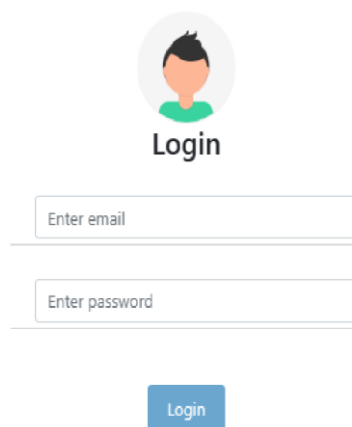
Sign Up

Email Address

First Name

Password

Password (Confirm)




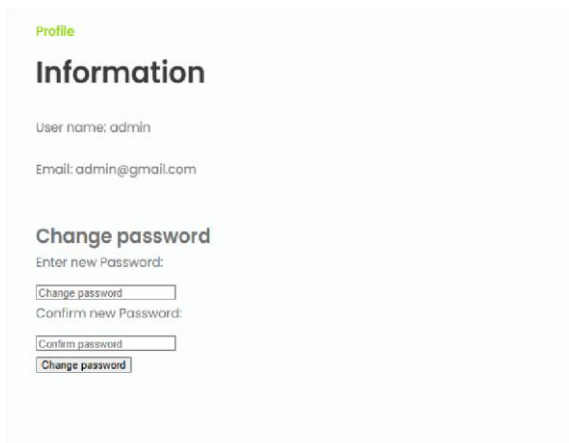

Login

Figura 1 - Página de registo

Figura 2 - Página de login



Profile

Information

User name: admin

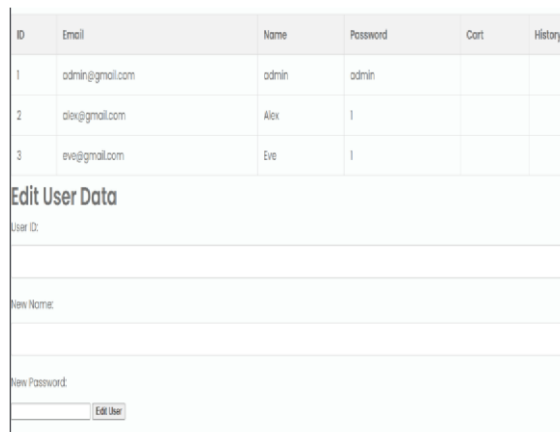
Email: admin@gmail.com

Change password

Enter new Password:

Confirm new Password:

Figura 3 - Gestão de perfil



ID	Email	Name	Password	Cart	History
1	admin@gmail.com	admin	admin		
2	alex@gmail.com	Alex	1		
3	eve@gmail.com	Eve	1		

Edit User Data

User ID:

New Name:

New Password:

Figura 4 - Gestão de utilizadores (Admin)

2.2 Catálogo de Produtos

O nosso catálogo permite aos utilizadores ver o título, descrição preço e disponibilidade de cada produto. Temos também uma barra de pesquisa, um botão que permite aceder a uma página de reviews e outro que permite adicionar o produto ao carrinho de compras.

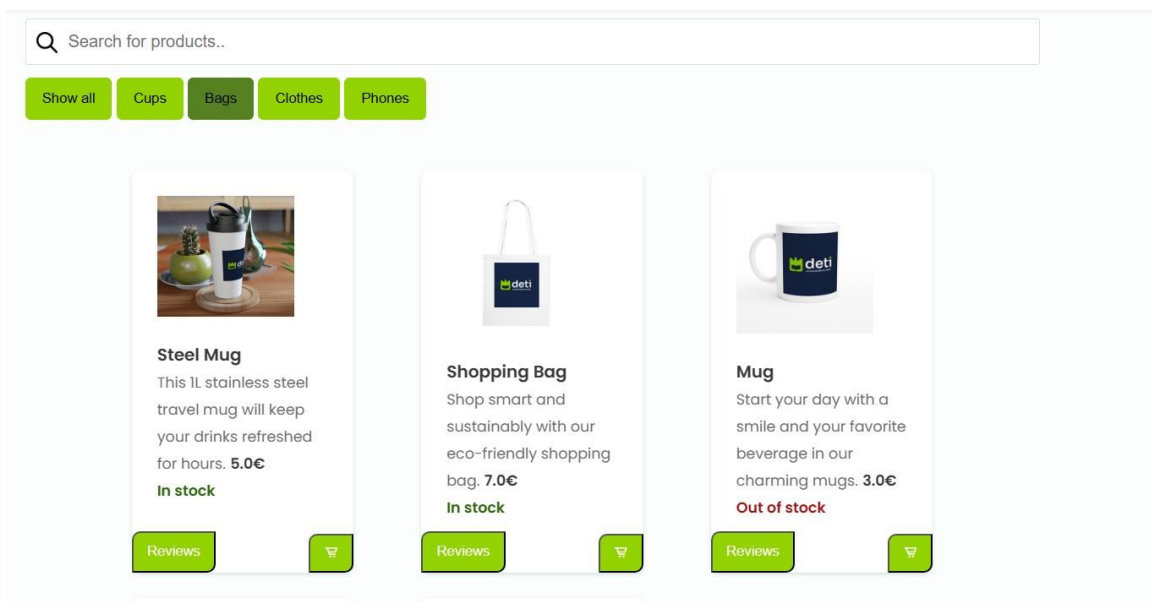


Figura 5 - Catálogo de produtos

2.3 Carrinho de compras e Wishlist

Para fazer a gestão do carrinho de compras e da Wishlist optámos por implementar duas tabelas para uma fácil gestão e visualização dos itens.

A partir daqui é possível remover itens do carro, da wishlist e mover itens do carro para a wishlist e vice-versa. Ao clicar no botão “Checkout” o utilizador é encaminhado para uma página de checkout com os produtos presentes no carrinho.

Cart

Product Name	Price	Quantity	Action
Steel Mug	5.0 €	1	One More Unit
			Add To Wishlist
			Remove From Cart

Total Price: \$5.0

Wishlist

Product Name	Price	Quantity	Action
T-shirt	25.0 €	1	Add To Cart
			Remove From Wishlist
Shopping Bag	14.0 €	2	Add To Cart
			Remove From Wishlist

Total Price: \$39.0

Continue Shopping

Checkout

Orders

Figura 6 - Carro e wishlist

2.4 Processo de Checkout

Criámos uma página de checkout acessível através da página do carrinho de compras, ao clicar no botão “Continue to checkout” o pedido que contém os itens presentes no carro é processado e adicionado ao histórico de pedidos. O carro é posteriormente limpo. Se a caixa “Save credit card information” estiver clicada, a informação do cartão de crédito é guardada.

Checkout







Billing Address		Payment	
 Full Name		Accepted Cards	
<input type="text" value="John M. Doe"/>		   	
 Email		Name on Card	
<input type="text" value="john@example.com"/>		<input type="text" value="John More Doe"/>	
Address		Credit card number	
<input type="text" value="542 W. 15th Street"/>		<input type="text" value="1234-5678-9012-1234"/>	
City		Exp Month	
<input type="text" value="New York"/>		<input type="text" value="January"/>	
Country	Zip	Exp Year	CVV
<input type="text" value="NY"/>	<input type="text" value="10001"/>	<input type="text" value="2018"/>	<input type="text" value="352"/>
<input checked="" type="checkbox"/> Save credit card information			
<input type="button" value="Continue to checkout"/>			

Figura 7 - Checkout

2.5 Gestão de inventário

A quantidade de itens no inventário é passível de ser vista pelo utilizador a partir do catálogo, há 3 estados possíveis “In stock” caso haja mais de 10 unidades de um mesmo produto, “Low stock” se houver entre 3 e 10 e “Out of Stock” caso haja 0.

O stock é atualizado a cada pedido.

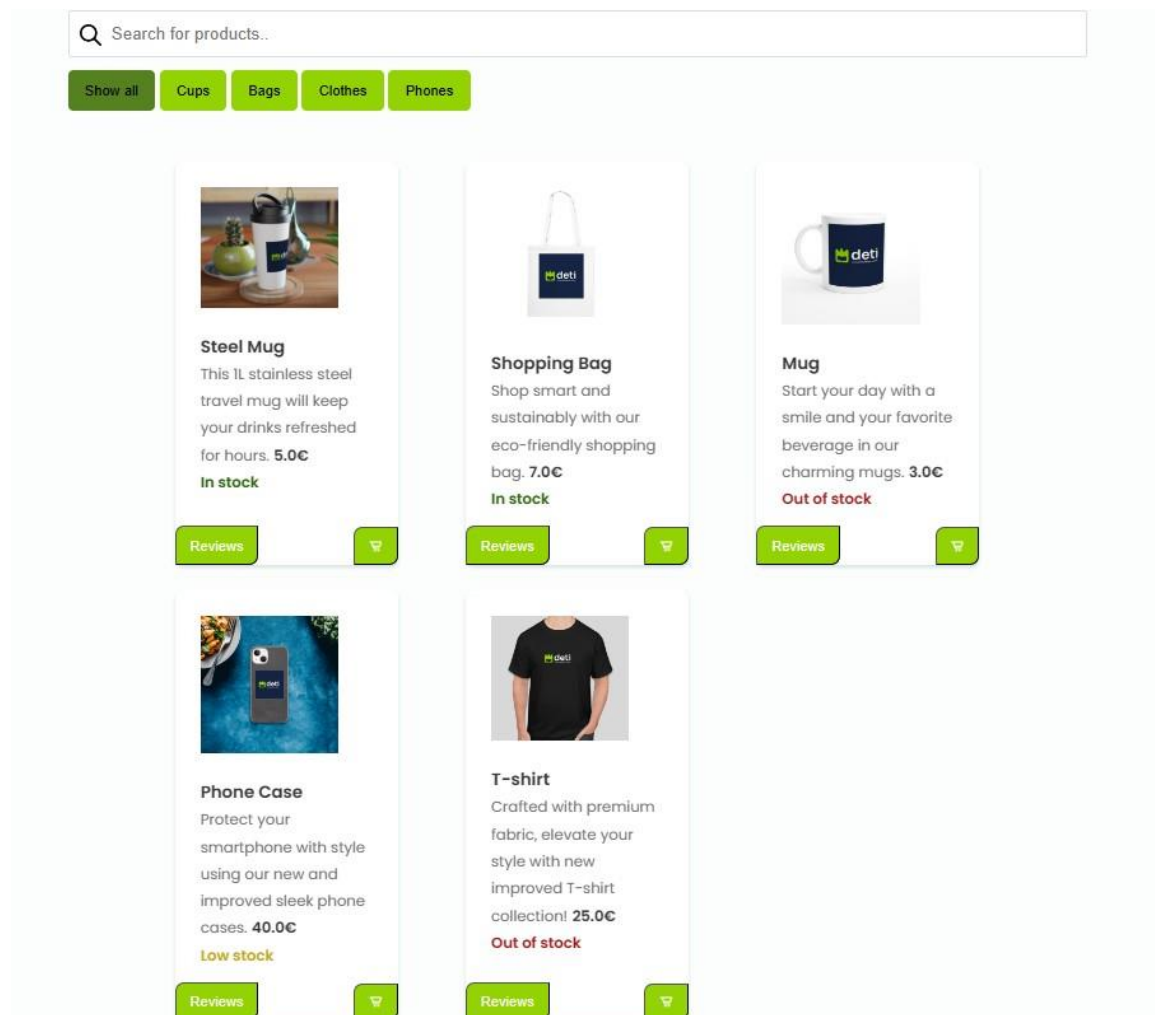


Figura 8 - Gestão de stock

2.6 Histórico de Pedidos

Implementámos uma página que permite ao utilizador gerir o seu histórico de pedidos. Nesta página o utilizador pode a data, os pedidos e o preço total dos seus pedidos anteriores. Ao clicar no botão “Reorder” é reencaminhado para a página do Carrinho de compras com os itens presentes no respetivo pedido adicionados ao mesmo.

Clicando no botão “Download” é fornecido um recibo da compra.

Order History

Order ID	Order Date	Total Price	Reorder	Receipt
<div><div></div><div>Order ID: ORD_TEST_ORDER</div></div>	2002-10-31 00:00:00	\$500.0	<div>Reorder</div>	<div>Download</div>
<div><div></div><div>Order ID: ORD_1699123188_7079</div></div>	2023-11-04 18:39:48	\$10.0	<div>Reorder</div>	<div>Download</div>

Continue Shopping

Checkout

Cart

Figura 9 - Histórico de pedidos

2.7 Opiniões e Avaliações

Na secção 2.5, após clicar no botão 'Reviews', será apresentada a página das reviews relativa ao item seleccionado, sendo possível dar rating de 1 a 5 e inserir um comentário

Add review

Author: Alex

Rating:

Comment:

Write your comment here

Add review

Other Reviews

User: Alex

Rating: 4/5

Pretty nice. Liked it fr fr

User: Eve

Rating: 3/5

mid

Figura 10 – Página de Reviews

3. Vulnerabilidades

Nesta secção iremos listar as vulnerabilidades presentes na versão vulnerável da aplicação e mostrar de que forma as corrigimos na versão segura.

3.1 CWE 89 – SQL Injection

3.1.1 Exploração da vulnerabilidade

```
def authenticate_user(email, password):
    query = "SELECT email,password,name FROM users WHERE email= '" + \
        email + "' AND password='" + password + "'"
    connection = sqlite3.connect('data.db', check_same_thread=False)
    cursor = connection.cursor()

    try:
        cursor.execute(query)
        results = cursor.fetchall()
        if len(results) == 0:
            return False, "Incorrect email or password", None, None, None

        return True, "", results[0][0], results[0][1], results[0][2]

    except Exception as e:
        print(e)
        return False, "There was an error validating your input, please check your data and try again.", None, None, None
```

Figura 11 – Query suscetível a SQL Injections

O seguinte código é suscetível a SQL Injections porque os dados introduzidos pelo utilizador não são “sanitized”.

3.1.2 Resolução da vulnerabilidade

```
def authenticate_user(email, password):
    if '/' in email:
        return False, "email can't contain character '/'"
    connection = sqlite3.connect('data.db', check_same_thread=False)
    cursor = connection.cursor()

    try:
        cursor.execute( "SELECT email, password, name FROM users WHERE email= ? AND password = ?", (email, password))
        results = cursor.fetchall()
        if len(results) == 0:
            return False, "Incorrect email or password", None, None, None

        return True, "", results[0][0], results[0][1], results[0][2]

    except Exception as e:
        print(e)
        return False, "There was an error validating your input, please check your data and try again.", None, None, None
```

Figura 12 - Versão segura

A resolução consiste em dar “sanitize” aos valores lidos pelo utilizador

3.2 CWE 79– Cross Site Scripting

3.2.1 Exploração da vulnerabilidade

Na seção dos comentários podemos fazer o site vulnerável a XSS aplicando o filtro 'safe' fazendo assim com que o conteúdo seja gerado como HTML sem exceções.

```
<ul class="review-list">
  {% for review in reviews %}
  <li class="review-item">
    <strong class="user">User: {{ review[2] }}</strong><br>
    <strong class="rating">Rating: {{ review[4] }}/5</strong><br>
    <span class="comment">{{ review[3] | safe}}</span>
  </li>
  <br>
  {% endfor %}
</ul>
```

Figura 13 - Código suscetível a XSS

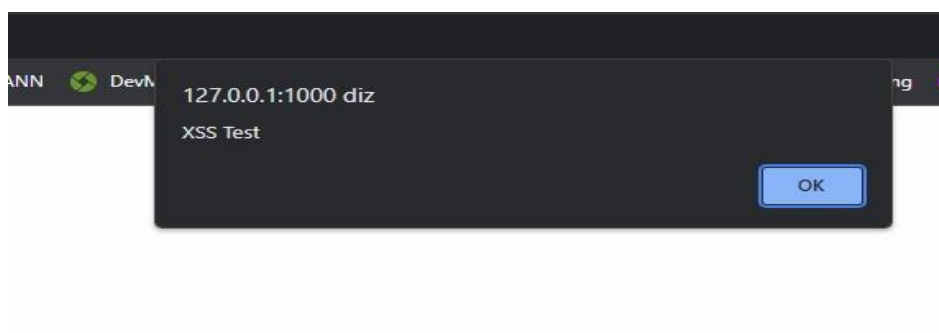


Figura 14 - Exemplo de um ataque XSS

3.2.2 Resolução da vulnerabilidade

Para resolver esta vulnerabilidade, simplesmente tiramos o filtro 'safe' de forma a dar 'sanitize' ao conteúdo

```
-->
<ul class="review-list">
  {% for review in reviews %}
  <li class="review-item">
    <strong class="user">User: {{ review[2] }}</strong><br>
    <strong class="rating">Rating: {{ review[4] }}/5</strong><br>
    <span class="comment">{{ review[3] }}</span>
  </li>
  <br>
  {% endfor %}
</ul>
```

Figura 15 - Versão segura para XSS

3.2 CWE 256 – Plaintext storage of a password

3.2.1 Exploração da vulnerabilidade

A seguinte função envia a password para a base de dados de forma não encriptada

```
def sign_up(name, password, password2, email):
    if password != password2:
        return False, "passwords don't match"

    connection = sqlite3.connect('data.db', check_same_thread=False)
    cursor = connection.cursor()
    # check if there already exists a user with that name
    query = "SELECT EXISTS(SELECT 1 FROM users WHERE email ='" + \
        email + "');"
    try:
        cursor.execute(query)
        results = cursor.fetchall()
        if results[0][0] == 1:
            return False, "email already exists"
    except:
        return False, "There was an error validating your input, please check your data and try again."
    # if we've reached this point, we can add the user to the database
    try:
        cursor.execute("INSERT INTO users VALUES (NULL, ?, ?, ?, NULL, NULL)",
            (email, name, password))
        connection.commit()
        return True, 'Account created successfully. Click <a href="/login">here</a> to login'
    except:
        return False, "There was an error validating your input, please check your data and try again."
```

Figura 16 - Password inserida na base de dados sem encriptação

Assim, a password fica armazenada na base de dados da seguinte forma, o que é uma vulnerabilidade.

password
admin
1
1

Figura 17 – Password armazenada de forma vulnerável na base de dados

3.2.2 Resolução da vulnerabilidade

De forma a resolver esta vulnerabilidade foi criada uma função de hash e utilizá-la para e utilizá-la para encriptar a password de forma a torná-la ilegível.

```
def hash(input):  
    input=bytes(input, 'utf-8')  
    digest = hashes.Hash(hashes.SHA256())  
    digest.update(input)  
    return digest.finalize()
```

Figura 18 - Função de Hash criada

```
password = hash(password)
```

Figura 19 - Envio seguro de password para a base de dados

password
K[?P?B?N?s@
F?K4?K?n4iD"DI
?y?P?_P8[?<?jlk?vũ"
?iv?A?M?g?s?K?O*?H?

Figura 20 - Password armazenada de forma segura na base de dados

3.3 CWE – 620 Unverified Password Change

3.3.1 Exploração da vulnerabilidade

A mudança da password é feita através de apenas dois campos “Change Password” e “Confirm Password”. Isto é uma vulnerabilidade pois não requer que quem está a mudar a password necessite de saber a password atual.

Change password

Enter new Password:

Confirm new Password:

Figura 21 - Campos para alteração de password na versão vulnerável

3.3.2 Resolução da vulnerabilidade

Uma solução possível é pedir a password atual e confirmar que corresponde.

Change password

Enter Old Password:

Enter New Password:

Confirm New Password:

Figura 22 - Pedido de password atual

3.4 CWE-311 Missing Encryption of sensitive data

3.4.1 Exploração da vulnerabilidade

O nome, email e a password são armazenados na base de dados de forma não encriptada o que torna fácil um atacante que tenha acesso à base de dados expor todos estes dados sensíveis

```
def sign_up(name, password, password2, email):
    if password != password2:
        return False, "passwords don't match"

    connection = sqlite3.connect('data.db', check_same_thread=False)
    cursor = connection.cursor()
    # check if there already exists a user with that name
    query = "SELECT EXISTS(SELECT 1 FROM users WHERE email ='" + \
        email + "');"
    try:
        cursor.execute(query)
        results = cursor.fetchall()
        if results[0][0] == 1:
            return False, "email already exists"
    except:
        return False, "There was an error validating your input, please check your data and try again."
    # if we've reached this point, we can add the user to the database
    try:
        cursor.execute("INSERT INTO users VALUES (NULL, ?, ?, ?, NULL, NULL)",
            (email, name, password))
        connection.commit()
        return True, 'Account created successfully. Click <a href="/login">here</a> to login'
    except:
        return False, "There was an error validating your input, please check your data and try again."
```

Figura 23 - Nome, password e email enviados de forma não encriptada para a base de dados

id	email	name	password
1	admin@gmail.com	admin	admin
2	alex@gmail.com	Alex	1
3	eve@gmail.com	Eve	1

Figura 24 - Nome, password e email armazenados de forma legível

3.4.2 Resolução da vulnerabilidade

À semelhança do que aconteceu com a password decidimos usar a função de hash criada anteriormente para resolver esta vulnerabilidade.

```
password = hash(password)
```

Figura 25- Password enviada de forma encriptada para a base de dados

Assim temos a seguinte base de dados, completamente ilegível.

password
K[ZZPBBB\NBs@
FBK4K4n4idDI
ByB_P8[<jlkv0"
ivABBNMgSKBo"H

Figura 26 - Password armazenada de forma totalmente ilegível

3.5 CWE – 549 Missing password field masking

3.5.1 Exploração da vulnerabilidade

Esta vulnerabilidade acontece quando o utilizador, acede ao seu perfil e tenta mudar a password. Pode ser uma grave falha de segurança caso o utilizador se encontre em locais públicos ou perto de pessoas com más intenções.

Change password

Enter new Password:

Confirm new Password:

Figura 27 - Password sem máscara

3.5.2 Resolução da vulnerabilidade

Para resolver a vulnerabilidade é preciso usar uma máscara para proteger a password

Change password

Enter new Password:

Confirm new Password:

Figura 28 - Password com máscara

3.6 CWE - 756: Missing custom error page

3.6.1 Exploração da vulnerabilidade

Esta vulnerabilidade acontece quando o utilizador se depara com uma página de erro que expõe a forma como a base de dados está construída. Isto pode comprometer gravemente a integridade dos dados.

TemplateNotFound

jinja2.exceptions.TemplateNotFound: checkout.html

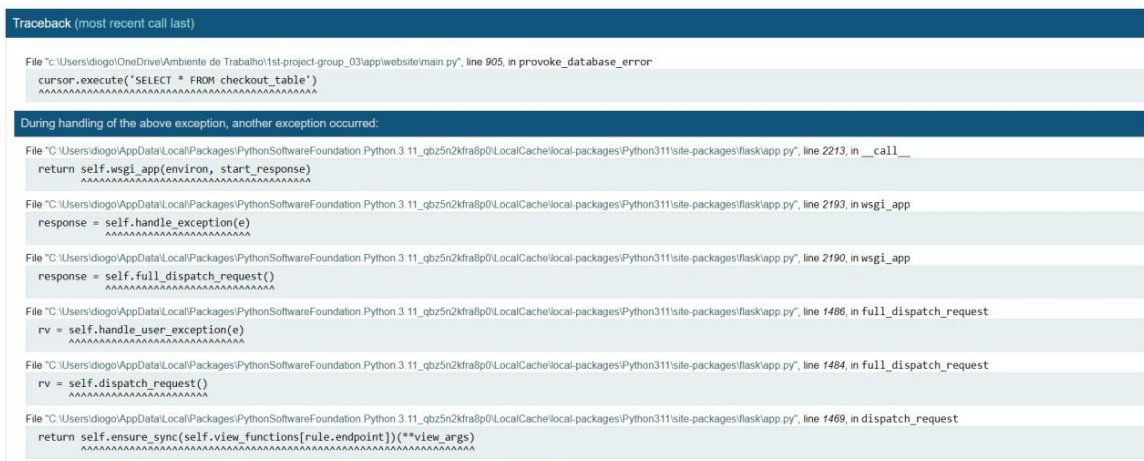


Figura 29- Página de erro que expõe a base de dados

3.6.2 Resolução da vulnerabilidade

Para resolver esta vulnerabilidade o utilizador é encaminhado para uma página de erro padrão e segura.

```
except Exception as e:
    return render_template('error.html', error_message=str(e)), 500
```

Figura 30- Utilizador reencaminhado para página de erro padrão

Ups there was an error :/

Please go back.

Figura 31- Página de erro padrão

3.7 CWE – 284 : Access Control Issues

3.7.1 Exploração de vulnerabilidade

Quando na página profile.html na sessão de admin existe a opção “User Manager” no header que permite entrar numa página na qual é possível alterar várias informações de qualquer usuário.

O problema surge quando outro perfil tenta acessar a página inserindo <http://127.0.0.1:2000/usermanager> (endereço do seguro) na barra de pesquisa do navegador. Dessa forma ganhará acesso a essa página e poder de visualizar e alterar informações que não lhe são autorizadas.

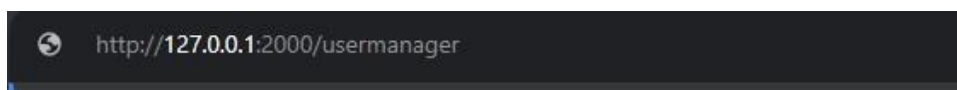


Figura 32 – Barra do navegador com o link, acesso indevido

3.7.2 Resolução de vulnerabilidade

Esta vulnerabilidade em específico é resolvida de uma forma bastante simples:

```
@app.route('/usermanager', methods=['GET', 'POST'])
def edit_user():

    connection = sqlite3.connect('data.db', check_same_thread=False)
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM users")
    users_data = cursor.fetchall()

    if session['user_name'] == "admin":
        if request.method == 'POST':
            id = request.form['user_id']
            name = request.form['new_name']
            password = request.form['new_password']

            result, msg = user_editor(id, name, password)

            if not result:
                flash(msg, category='error')
            else:
                flash(msg, category='success')

            cursor.execute("SELECT * FROM users")
            users_data = cursor.fetchall()
            return render_template('usermanager.html', users_data=users_data)
        else:
            return render_template('usermanager.html', users_data=users_data)
    else:
        return 'Acesso negado'
```

Figura 33 - Resolução da vulnerabilidade

Com a adição deste *if* é verificado qual o perfil está a tentar aceder à página. Caso não seja o perfil do administrador ser-lhe-á negado o acesso

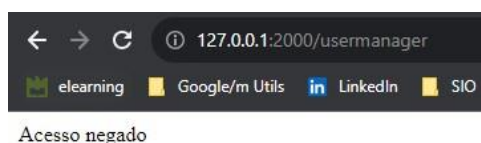


Figura 34 – Resultado na versão segura

4. Conclusão

É importante salientar que antes de ser disponibilizado [este](#) repositório nós já tínhamos começado o projeto noutro repositório pessoal cujos *screenshots* dos *commits* se encontram no novo repositório num ficheiro chamado `commitHistoryFromPreviousGitRep.png`.

Todos os elementos trabalharam de forma igual ao longo do projeto tanto na divisão da implementação do site como na criação e correção de vulnerabilidades.