

Project 1 - Vulnerabilidades

Universidade de Aveiro

Segurança Informática e nas Organizações

Francisco Gonçalves nº Mec: 1008538



universidade
de aveiro

Conteúdos

1. *Resumo*
2. **Estrutura**
3. **CWE 79**
4. **CWE 89**
5. **CWE 521**
6. **CWE 798**
7. **CWE 307**
8. **CWE 601**
9. **CWE 620**
10. **CWE 352**

Resumo

Este projeto foi desenvolvido no âmbito da cadeira de Segurança Informática e nas Organizações. A aplicação web foi desenvolvida em Python, utilizando a micro web framework Flask. Consiste em duas versões de uma loja online, uma insegura com vulnerabilidades que comprometem a segurança geral da aplicação, e uma versão segura, que implementa soluções para estas vulnerabilidades.

Foram explorados um total de 8 vulnerabilidades:

- CWE-79 (Cross-Site Scripting)
- CWE-89 (SQL Injection)
- CWE-521: Weak Password Requirements
- CWE-798: Use of Hard-Coded Credentials
- CWE-307: Improper Restriction of Excessive Authentication Attempts
- CWE-601: URL Redirection to Untrusted Site ('Open Redirect')
- CWE-620: Unverified Password Change
- CWE-352 Cross-Site Request Forgery (CSRF)

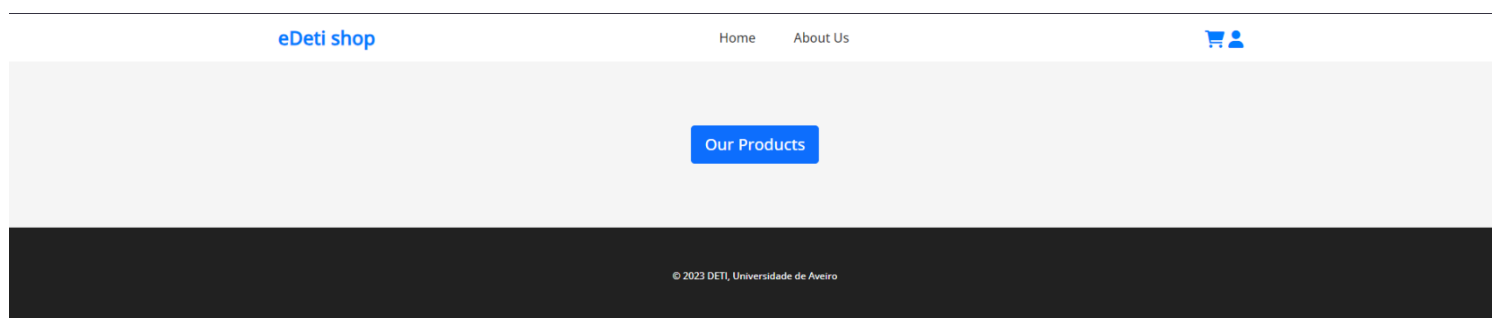
Estas vulnerabilidades, e as suas soluções, serão detalhadamente abordadas nos próximos conteúdos.

Estrutura

A aplicação web foi desenvolvida para ser uma loja de venda de produtos do Departamento de Electrónica, Telecomunicações e Informática.

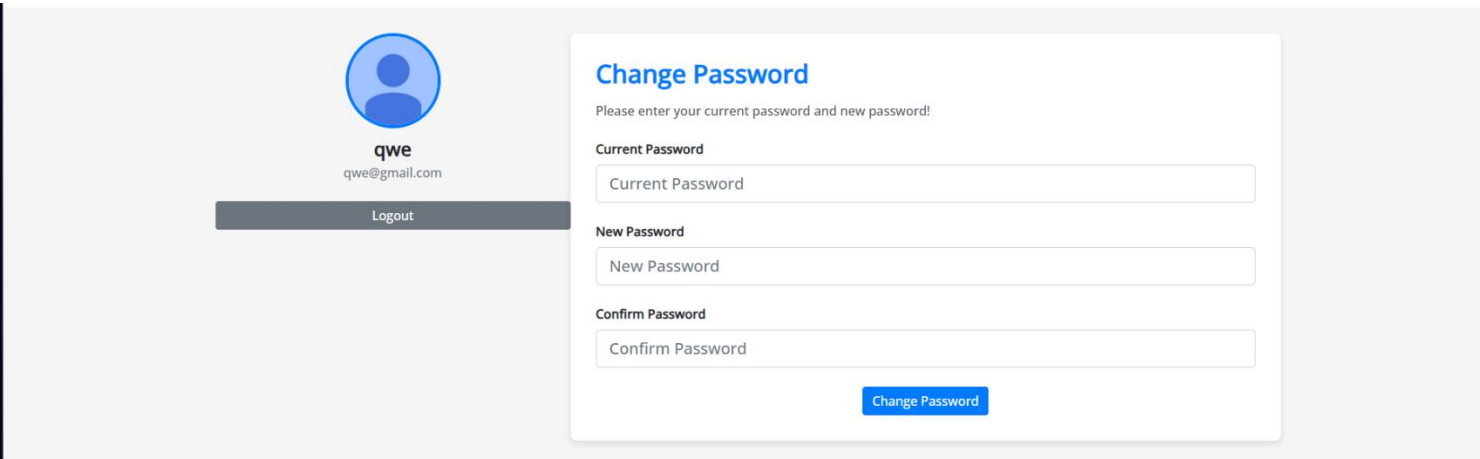
Home Page & Base:

Como base o website tem uma *navbar* com ligações à home, perfil do utilizador, sobre nós. A home page é constituída por um botão que direciona para os produtos.



Profile Page:

Na profile page o utilizador pode ver e modificar informação do seu perfil.



About Us:

Esta página exhibe informações acerca dos contribuidor deste projeto.

About Us Page

This is the about us page of the project of Segurança Informática e nas Organizações

This project was done by:

Francisco Gonçalves Nmec: 1085381

Project description

The assignment focuses on vulnerabilities in software projects, their exploration, and prevention. We were required to develop a small online shop application selling memorabilia for the Department of Electronics, Telecommunications, and Informatics (DETI) at the University of Aveiro. The online shop features items like mugs, cups, t-shirts, hoodies, and similar memorabilia.

Common Weakness Enumeration (CWE)


The following are the CWEs that were addressed in this project:

1. CWE 89 3. CWE 79 4. CWE 89 5. CWE 521 6. CWE 798 7. CWE 307 8. CWE 601 9. CWE 620 10. CWE 352

Products::

Esta página mostra os vários produtos disponíveis e contem botões para mais informação ou para adicionar ao carrinho.

Products




T-Shirt Black
A great product

10€

[See Details](#)

[Add to Cart](#)

In Stock




T-Shirt White
Another great product

20€

[See Details](#)

[Add to Cart](#)

In Stock



T-Shirt Grey
A fantastic product

30€

[See Details](#)

[Add to Cart](#)

In Stock

Product:

Esta página contém informações mais detalhadas sobre o produto selecionado e também inclui uma seção para adicionar avaliações do produto.



Hoodie Grey

60€

1

Add to cart

Product Details

A superb product

Post a Review

Product Evaluation*

1

Write your Review*

Send Review



qwe@gmail.com

Rating: 1

24/07/23

Desiludido com a qualidade. Esperava mais

Cart:

Esta página mostra os artigos seleccionados para compra disponibilizando ao utilizador a opção de proceder ao *checkout*.

Your Shopping Cart

Item Name	Quantity	Price	Action
Hoodie Grey	1	60 x 1	Remove

[Continue Shopping](#)[Proceed to Checkout](#)

Checkout:

Esta página é usada pelo utilizador para finalizar a sua compra. Nesta versão do projeto esta funcionalidade ainda não está completamente finalizada.

Checkout

Your basic information

First name*

Last name*

Email*

Your Payment Information

Credit Card*

Expiry date*

CVC/CVV*

[Proceed](#)

Product Name

€99/ shipping included

✓ 14 Day Delivery

✓ 30 Day Returns

✓ 24 Hour Support

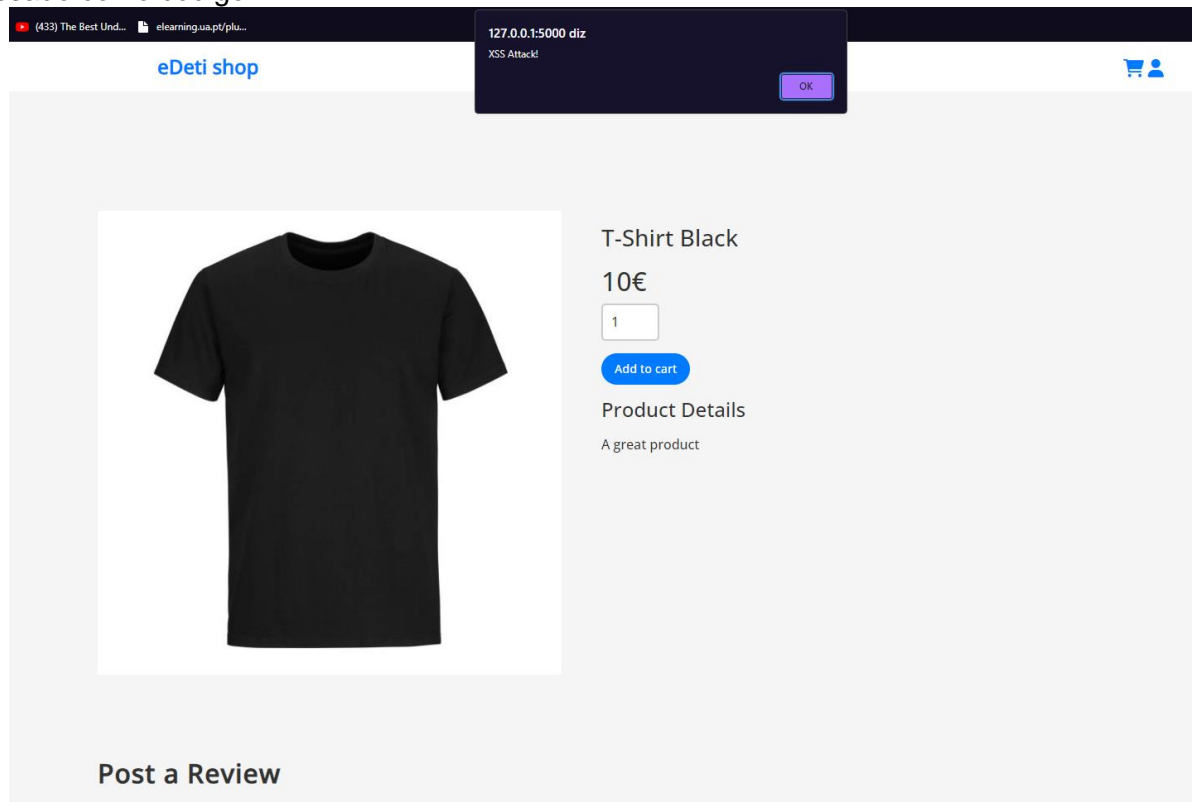
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-Site Scripting')

Esta vulnerabilidade está presente na funcionalidade que lida com as reviews.

Como aplicar?

Primeiramente, de forma a verificar se a vulnerabilidade é aplicável executamos um script básico.

Como é possível observar, o trecho “<script>alert('XSS Attack')</script>” foi enviado pela review e processado como código.



Este exemplo pode ser muito mais perigoso. Imagine que um usuário autenticado está navegando no site e visualiza uma página onde os comentários de outros usuários (como as avaliações de produtos) são carregados. Se o site for vulnerável a XSS, um atacante pode injetar um script malicioso que rouba o cookie de sessão do usuário autenticado. Esse cookie pode ser usado para se passar pelo usuário, permitindo ao atacante acessar a conta da vítima.

Impacto e Criticidade:

O exemplo mostrado só afeta o utilizador a ser atacado, porém, um exemplo de um ataque comprometedor seria mostrar a página de login inicial da loja ao administrador após x segundos de inatividade, de forma a simular a expiração da sessão atual, quando na verdade trata-se de uma página falsa que irá roubar o login de administrador. A partir daqui toda a aplicação está comprometida.

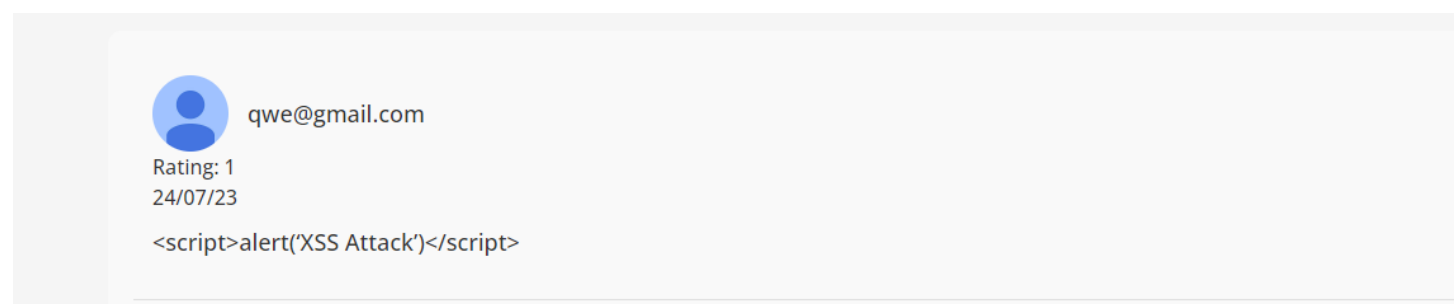
Como Mitigar:

Como o site que estou a desenvolver em Flask já escapa automaticamente as variáveis, tive de introduzir esta vulnerabilidade ao adicionar “|safe” nas variáveis no template HTML, como se pode ver na imagem seguinte. Portanto, ao remover esta parte do código, o site ficará protegido contra ataques XSS.

```
<div class="reviews">
  {% for review in reviews %}
    <div class="review">
      <ul class="userID">
        <li></li>
        <li>{{ review[1] |safe}}</li> <!-- Assuming review[1] is the email/username -->
      </ul>
      <ul>
        <li>Rating: {{ review[2] |safe}}</li>
        <li>24/07/23</li>
      </ul>
      <p class="rtxt">{{ review[3] |safe }}</p>
    </div>
  {% endfor %}
</div>
```

Mas em outra situação a forma de fazer o website ficar seguro seria interpretar todos os dados que são inseridos dinamicamente no HTML como uma string literal (isto deve ser feito no lado do cliente porque cada motor de navegador interpreta e renderiza HTML de maneira diferente, o que significa que algum código pode ser seguro para um navegador, mas não para outro).

Como resultado fica assim as reviews:



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

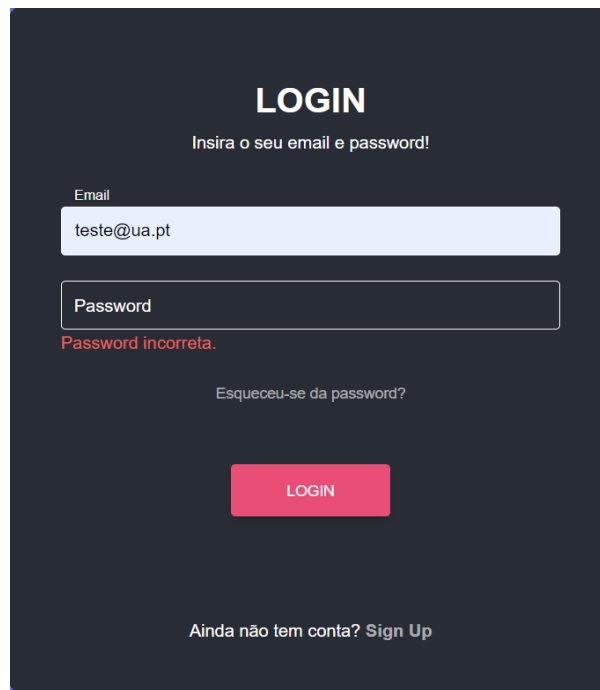
A CWE-89, conhecida como SQL Injection, é uma das vulnerabilidades de segurança mais críticas e comuns em aplicações web. Ela ocorre quando entradas fornecidas pelo usuário são inseridas diretamente em consultas SQL sem a devida sanitização, permitindo que um atacante manipule essas consultas. Isso pode resultar em acesso não autorizado a dados, modificação ou exclusão de registros, e até mesmo o comprometimento total do banco de dados da aplicação. A exploração dessa vulnerabilidade pode ter consequências graves, desde a perda de dados sensíveis até a interrupção completa dos serviços da aplicação.

Como aplicar:

A vulnerabilidade CWE-89 (SQL Injection) está localizada na página de autenticação.

Neste contexto, um ataque básico de SQL Injection ocorre quando um invasor altera um dos valores utilizados na consulta SQL original, resultando em uma consulta modificada que permite ao invasor obter acesso a uma conta que normalmente estaria fora de seu alcance. Esse ataque explora uma falha na validação das entradas, contornando as medidas de segurança e permitindo o acesso não autorizado a dados sensíveis ou contas protegidas.

Como mostrado na figura, o login não foi bem-sucedido devido a um erro na palavra-passe. No entanto, um atacante poderia explorar a vulnerabilidade CWE-89 (SQL Injection) para conseguir acessar o site.



LOGIN

Insira o seu email e password!

Email

teste@ua.pt

Password

Password incorreta.

Esqueceu-se da password?

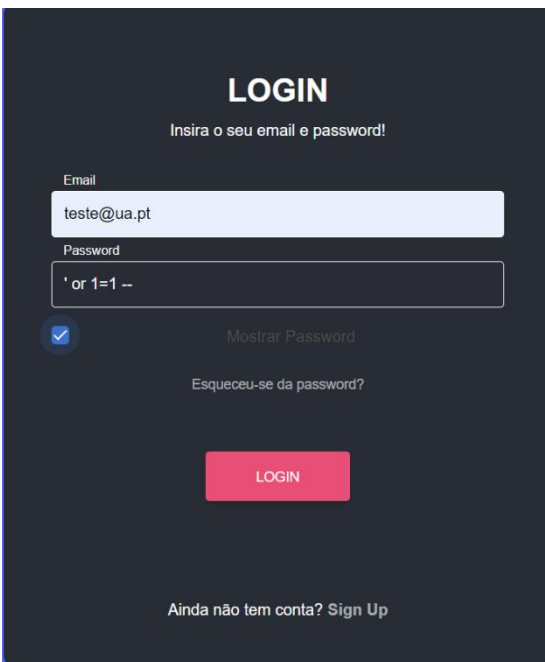
LOGIN

Ainda não tem conta? [Sign Up](#)

O atacante para conseguir executar o sql injection com sucesso pode tentar vários comandos na zona de login para que na base de dados retorne True e assim ter acesso ao site.

O comando que utilizamos pode ser observado na imagem seguinte (“ OR 1=1 - -”), como também pode ser observado os pacotes (POST,GET) do utilizador com o servidor.

Podemos ver através do terminal a autorização do email: (“ OR 1=1 - -”) pela conta do teste.



```
{'email': 'teste@ua.pt', 'username': 'teste'}
127.0.0.1 - - [10/Aug/2024 23:20:02] "POST /login HTTP/1.1" 302 -
127.0.0.1 - - [10/Aug/2024 23:20:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [10/Aug/2024 23:20:02] "GET /static/css/bootstrap.css HTTP/1.1" 304 -
127.0.0.1 - - [10/Aug/2024 23:20:02] "GET /static/css/base.css HTTP/1.1" 304 -
127.0.0.1 - - [10/Aug/2024 23:20:02] "GET /static/css/products.css HTTP/1.1" 304 -
127.0.0.1 - - [10/Aug/2024 23:20:02] "GET /static/css/style.css HTTP/1.1" 304 -
```

Impacto e Criticidade:

A vulnerabilidade CWE-89 (SQL Injection) é grave devido à sua facilidade de exploração e ao impacto potencial. Pode permitir a execução de comandos SQL arbitrários, comprometendo dados sensíveis e a integridade da aplicação. Apesar de frameworks e boas práticas de codificação ajudarem a mitigar esses riscos, a SQL Injection continua sendo uma ameaça crítica que pode ter consequências severas, incluindo vazamento e manipulação de dados.

Como Mitigar:

Para mitigar a vulnerabilidade CWE-89, Injeção de SQL, foi utilizada a estratégia de Consultas Parametrizadas, assegurando a separação entre os dados do usuário e o código SQL.

Consultas Parametrizadas: Esta é a defesa central contra SQL Injection. Ao usar parâmetros nas consultas SQL e passar os valores como tuplas ou listas no método execute, o banco de dados trata esses valores como dados, não como código, impedindo a execução de comandos maliciosos.

Separação de Código e Dados: Com essa abordagem, o código SQL permanece separado dos dados de entrada do usuário, garantindo que qualquer entrada seja tratada apenas como valor e não como código, protegendo assim a base de dados contra-ataques.

```
def add_product(quantity, stock, description, name, img_path, price):
    conn = sqlite3.connect(db_path)
    c = conn.cursor()
    c.execute("INSERT INTO Products (quantity, stock, description, name, img_path, price) VALUES (?, ?, ?, ?, ?, ?)", (quantity, stock, description, name, img_path, price))
    conn.commit()
    conn.close()
```

CWE-521: Weak Password Requirements

CWE-521, intitulada "Weak Password Requirements" (Requisitos Fracos de Senha), refere-se à falha em estabelecer requisitos de senha suficientemente fortes em uma aplicação ou sistema. Isso significa que as políticas de senha permitidas são fracas e, como resultado, deixam o sistema vulnerável a ataques, como força bruta ou engenharia social.

Como aplicar:

A forma de explorar esta CWE é simples. Ao definir um limite de tamanho de 8 caracteres para a senha, sem mais nenhuma restrição, é possível descobrir a senha de um utilizador com relativa facilidade, seja através de um ataque de força bruta ou de engenharia social.

```
@auth.route('/sign-up', methods=['POST', 'GET'])
def sign_up():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password1']
        password_confirm = request.form['password2']
        username = request.form['username']
        errors = {}
        error_regist = 0
        if(len(password) < 8):
            errors["length"] = 1
            error_regist = 1
        elif(password != password_confirm):
            errors["confirm"] = 1
            error_regist = 1
        if(error_regist == 0):
            exists = verify_user(email)
            if(exists == False):
                add_user(email, password, username)
                return redirect(url_for('auth.login'))
            else:
                return render_template('sign_up.html', email_in_use = True)
        return render_template('sign_up.html', errors=errors, email=email, username=username)
    else:
        return render_template('sign_up.html')
```

Impacto e Criticidade:

- **Aumento do Risco de Quebra de Senha:** Senhas fracas são mais suscetíveis a ataques de força bruta, onde um invasor tenta todas as combinações possíveis de caracteres.
- **Facilitação de Ataques de Engenharia Social:** Senhas simples podem ser mais fáceis de adivinhar através de ataques de engenharia social.
- **Comprometimento de Contas:** Uma vez que uma senha fraca é descoberta, a conta associada pode ser comprometida, possivelmente levando a acessos não autorizados a informações sensíveis ou recursos críticos.

Como Mitigar:

Para resolver este problema, é essencial impor restrições na criação de senhas. Regras como incluir um caractere especial, aumentar o comprimento mínimo para 12 caracteres, exigir ao menos uma letra maiúscula, uma minúscula e números, tornam os ataques de força bruta mais difíceis de serem bem-sucedidos e complicam a execução de técnicas de engenharia social.

```
@auth.route('/sign-up', methods=['POST', 'GET'])
def sign_up():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password1']
        password_confirm = request.form['password2']
        username = request.form['username']
        errors = {}

        # Stronger password requirements
        if len(password) <= 12:
            errors["length"] = "Password must be at least 12 characters long."
        if not re.search("[a-z]", password):
            errors["lowercase"] = "Password must contain at least one lowercase letter."
        if not re.search("[A-Z]", password):
            errors["uppercase"] = "Password must contain at least one uppercase letter."
        if not re.search("[0-9]", password):
            errors["number"] = "Password must contain at least one number."
        if not re.search("[@#$$%^&+_.!]", password):
            errors["special"] = "Password must contain at least one special character (e.g., @, #, $, etc.)."
        if password != password_confirm:
            errors["confirm"] = "Passwords do not match."
```

CWE-798: Use of Hard-coded Credentials:

A vulnerabilidade CWE-798 refere-se ao uso de credenciais embutidas no código, como uma chave secreta, em vez de as armazenar de forma segura. No teu código, a chave secreta está definida diretamente no código-fonte, o que representa um risco de segurança.

Se um invasor obtiver acesso ao código-fonte da tua aplicação, poderá facilmente descobrir a chave secreta. Isso pode permitir que o invasor falsifique cookies de sessão, o que pode resultar em vários problemas de segurança, incluindo acesso não autorizado a contas de utilizador e execução de código arbitrário.

Como aplicar:

As credenciais hardcoded no código representam um risco de segurança sério, pois podem permitir que um atacante bypass a autenticação estabelecida pelo administrador do sistema. Neste caso específico, estamos lidando com uma variante de entrada (inbound), uma vez que a aplicação Flask contém um mecanismo de autenticação que valida as credenciais de entrada em comparação com um conjunto de credenciais fixas no código.

No código em questão, a chave secreta da aplicação Flask está embutida diretamente, o que configura uma exposição à vulnerabilidade CWE-798.

```
app.config['SECRET_KEY'] = b'KXEHT'
```

Impacto e Criticidade:

O comprometimento de segurança ocorre quando credenciais embutidas, como chaves ou senhas, são facilmente extraídas por um invasor que obteve acesso ao código-fonte, permitindo autenticação como um usuário legítimo e acesso não autorizado a sistemas sensíveis. A escalabilidade do ataque aumenta se as credenciais comprometidas forem usadas em vários sistemas, afetando rapidamente muitos usuários. Mitigar essa falha é difícil, pois exige a alteração de credenciais no código e a redistribuição do software, o que pode ser demorado, deixando o sistema vulnerável durante esse período. Além disso, versões antigas com as credenciais embutidas podem continuar em uso, mantendo a exposição ao risco. Devido à facilidade de exploração e ao impacto potencial, a criticidade do CWE-798 é considerada alta.

Como Mitigar:

No meu caso, a implementação de um ficheiro .env dedicado provou ser uma solução eficaz para mitigar as vulnerabilidades identificadas. Este arquivo tem a função de armazenar a session_key de forma segura, isolando credenciais sensíveis e protegendo-as contra acessos indevidos.

CWE-307: Improper Restriction of Excessive Authentication Attempts:

A CWE-307, conhecida como "Falha na Implementação de Limite de Tentativas de Autenticação", refere-se a uma vulnerabilidade em sistemas que não aplicam um número máximo de tentativas de login ou que implementam inadequadamente esse limite. Isso permite que um atacante realize ataques de força bruta ou de enumeração de credenciais, onde ele tenta várias combinações de nome de usuário e senha até encontrar uma combinação válida. A ausência de um mecanismo de bloqueio ou de aumento progressivo de atraso nas tentativas falhadas torna o sistema suscetível a acessos não autorizados, comprometendo a segurança dos dados.

Como aplicar:

A CWE-307 pode ser explorada por atacantes para realizar ataques de força bruta, onde várias combinações de nome de usuário e senha são testadas sem restrição. A falta de limite nas tentativas de autenticação permite que o invasor descubra credenciais válidas, obtenha acesso não autorizado ao sistema ou cause degradação de desempenho por sobrecarregar os recursos do servidor.

```
@auth.route('/login', methods=['POST','GET'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        if check_password(email, password):
            username = get_username(email)
            session.permanent = True
            session['user'] = {'email': email, 'username': username}
            return redirect(url_for('views.home'))
        else:
            return render_template('login.html', incorrect_password = True, email=email)
    else:
        return render_template('login.html')
```

Como podemos ver na imagem não existe nenhuma restrição para o número de tentativas de login.

Impacto e Criticidade:

A exploração da CWE-307 pode ter um impacto severo, pois permite que atacantes realizem ataques de força bruta para obter acesso não autorizado a contas, comprometendo a confidencialidade e a integridade dos dados. A criticidade dessa vulnerabilidade é alta, especialmente em sistemas que armazenam informações sensíveis ou que controlam o acesso a recursos críticos, pois a falha pode resultar em vazamento de dados, manipulação de informações, ou até mesmo em interrupções no serviço.

Como Mitigar:

Para mitigar a CWE-307, limitei o número de tentativas de login e bloqueei a conta temporariamente após várias falhas. Aumentei progressivamente o tempo de bloqueio a cada sequência de tentativas incorretas. Essas medidas reduzem o risco de ataques de força bruta e melhoram a segurança do sistema.


```

def check_login_attempts(email):
    try:
        conn = sqlite3.connect(db_path)
        c = conn.cursor()
        c.execute("SELECT attempts, last_attempt, locked_until FROM LoginAttempts WHERE email = ?", (email,))
        record = c.fetchone()
        conn.close()

        if record:
            attempts, last_attempt, locked_until = record
            print(f"Attempts: {attempts}, Last Attempt: {last_attempt}, Locked Until: {locked_until}")
            if locked_until and datetime.strptime(locked_until, '%Y-%m-%d %H:%M:%S') > datetime.now():
                print(f"Account is temporarily locked until: {locked_until}")
                return False, "Account is temporarily locked. Try again later."
            if attempts >= 5:
                locked_until = datetime.now() + timedelta(minutes=15)
                conn = sqlite3.connect(db_path)
                c = conn.cursor()
                c.execute("UPDATE LoginAttempts SET locked_until = ? WHERE email = ?",
                        (locked_until.strftime('%Y-%m-%d %H:%M:%S'), email))
                conn.commit()
                conn.close()
                print(f"Account locked due to too many attempts. Locked until: {locked_until}")
                return False, "Too many attempts. Account is temporarily locked."
            return True, None
        return True, None
    except Exception as e:
        print(f"Error in check_login_attempts: {e}")
        return False, "An error occurred. Please try again later."

```

```

def increment_login_attempts(email):
    try:
        print(f"Incrementing login attempts for email: {email}")
        conn = sqlite3.connect(db_path)
        c = conn.cursor()
        current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        c.execute("""
            INSERT INTO LoginAttempts (email, attempts, last_attempt)
            VALUES (?, 1, ?)
            ON CONFLICT(email)
            DO UPDATE SET attempts = attempts + 1, last_attempt = ?
        """, (email, current_time, current_time))

        conn.commit()
        conn.close()
        print(f"Login attempts incremented for email: {email}")

    except Exception as e:
        print(f"Error in increment_login_attempts: {e}")

```

```

@auth.route('/login', methods=['POST', 'GET'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        errors = {}

        print(f"Received login request for email: {email}")

        can_attempt, error_message = check_login_attempts(email)
        if not can_attempt:
            print(f"Login denied for email: {email} - {error_message}")
            errors["attempts"] = error_message
            return render_template('login.html', email=email, errors=errors)

        user_password = get_user_password(email)
        if user_password and check_password_hash(user_password, password):

            session.permanent = True
            session.clear()
            session['user'] = {'email': email, 'username': get_username(email)}
            session.modified = True
            clear_login_attempts(email)
            print(f"Login successful for email: {email}")
            return redirect(url_for('views.home'))

        else:
            print(f"Invalid login attempt for email: {email}")
            increment_login_attempts(email)
            errors["invalid"] = "Invalid email or password."
            return render_template('login.html', email=email, errors=errors)

    else:
        print("Rendering login page.")
        return render_template('login.html')

```

CWE-601: URL Redirection to Untrusted Site ('Open Redirect'):

A CWE-601, conhecida como "Redirecionamento Aberto", ocorre quando uma aplicação web permite que usuários redirecionem para URLs externos sem validação adequada. Isso pode ser explorado por atacantes para direcionar vítimas a sites maliciosos, facilitando ataques de phishing e comprometendo a confiança na aplicação.

Como aplicar:

No caso deste website, a vulnerabilidade pode ser explorada da seguinte forma: um atacante identifica um ponto no site onde pode controlar o destino de um redirecionamento, como a URL `/product?value=123`, que direciona o usuário para uma página específica de produto com o ID 123. O atacante então manipula o parâmetro de redirecionamento (`value`) para apontar para um URL externo malicioso, como por exemplo `https://example.com/product?value=https://malicious-site.com`. Se o servidor não validar corretamente o parâmetro `value`, o usuário pode ser redirecionado para o site malicioso, expondo-se a phishing ou outros ataques.

```

<div class="mt-3 d-flex">
  <a href="/product?value={{ product[0] }}" class="btn btn-secondary flex-fill me-2">See Details</a>
  <form action="/add_cart" method="POST" class="flex-fill">
    <input type="hidden" name="product_id" value="{{ product[0] }}">
    <input type="number" name="quantity" value="1" min="1" max="{{ product[1] | int }}" class="form-control">
    <button type="submit" class="btn btn-primary w-100">Add to Cart</button>
  </form>
</div>

```

Impacto e Criticidade:

Pode ter um impacto significativo ao permitir que um atacante redirecione usuários para sites maliciosos. Isso pode resultar em ataques de phishing, instalação de malware e comprometimento da segurança do usuário, além de causar uma perda de confiança no site afetado. A criticidade dessa vulnerabilidade é geralmente alta, especialmente em sites com grande número de usuários ou que lidam com informações sensíveis, pois pode levar a sérias consequências para a segurança e reputação da organização.

Como Mitigar:

A vulnerabilidade CWE-601 (Redirecionamento Aberto) foi mitigada ao substituir o link de redirecionamento direto () por um formulário que faz uma submissão POST para o servidor (<form action="/product" method="POST">). Esta alteração elimina a possibilidade de um utilizador malicioso manipular o URL de redirecionamento, garantindo que o redirecionamento é gerido de forma segura no backend. Ao processar a requisição no servidor, é possível validar o product_id antes de redirecionar, evitando potenciais redirecionamentos para URLs não confiáveis.

CWE-620: Unverified Password Change:

A CWE-620, conhecida como "Mudança de Senha Não Verificada", ocorre quando um sistema permite que um usuário altere sua senha sem validar adequadamente sua identidade, geralmente por meio da confirmação da senha atual. Essa falha pode permitir que atacantes que já tenham acesso à sessão do usuário modifiquem a senha sem a devida autorização, comprometendo a segurança da conta e facilitando o controle indevido da mesma. A verificação adequada é crucial para garantir que somente o legítimo proprietário da conta possa alterar suas credenciais.

Como aplicar:

Tomar proveito da CWE-620 envolve explorar a ausência de verificação adequada durante a mudança de senha para assumir o controle de uma conta. Um atacante que já tenha acesso à sessão de um usuário (por meio de ataques como sequestro de sessão ou roubo de cookies) pode, sem precisar saber a senha atual, alterar a senha para uma de sua escolha. Com isso, ele pode bloquear o legítimo proprietário da conta e obter controle total sobre os recursos e dados associados àquela conta, potencialmente causando danos significativos ao usuário ou à organização.

```

@views.route('/profile', methods=['GET', 'POST'])
def profile():
    if request.method == 'POST':
        email = session['user']['email']
        username = session['user']['username']
        errors = {}

        old_password = request.form['old-password']
        new_password = request.form['new-password']
        if len(get_user_password(email)) != len(old_password):
            errors['old-password'] = 'Invalid password.'
            return render_template('profile.html', email = email, username = username, errors=errors)

        for i in range(len(get_user_password(email))):
            if get_user_password(email)[i] != old_password[i]:
                errors['old-password'] = 'Invalid password.'
                return render_template('profile.html', email = email, username = username, errors=errors)

        update_password(email, new_password)
        errors['success'] = 'Password updated successfully!'
        return render_template('profile.html', email = email, username = username, errors=errors)
    else:
        if('user' in session):
            email = session['user']['email']
            username = session['user']['username']
            return render_template('profile.html', email = email, username = username)
        else:
            return redirect(url_for('auth.login'))

```

Como se pode ver neste caso a mudança de password ocorre verificando o tamanho das passwords o que não é necessário e a real comparação das passwords é feito num for loop que é vulnerável a ataques de timing.

Impacto e Criticidade:

O impacto da CWE-620 pode ser grave, pois permite que um atacante altere senhas sem a devida verificação, levando ao controle não autorizado de contas e ao bloqueio de usuários legítimos. Isso pode resultar em violação de dados sensíveis, perda de acesso a serviços críticos e danos à reputação da organização. A criticidade da CWE-620 está na sua facilidade de exploração, especialmente se combinada com outras vulnerabilidades, tornando essencial a implementação de mecanismos robustos de reautenticação para proteger a integridade das contas dos usuários.

Como Mitigar:

Este código mitiga a vulnerabilidade CWE-620 ao exigir que o usuário forneça sua senha atual correta antes de permitir a alteração para uma nova senha. Ele valida a força da nova senha, verificando se ela atende a requisitos de segurança, como comprimento mínimo e a presença de letras maiúsculas, minúsculas, números e caracteres especiais. A nova senha é protegida por hashing antes de ser armazenada, garantindo a segurança dos dados. Essas medidas evitam que um invasor altere a senha sem autorização adequada, protegendo a conta do usuário.

```

@auth.route('/profile', methods=['GET', 'POST'])
def profile():
    if request.method == 'POST':
        email = session['user']['email']
        old_password = request.form['old-password']
        new_password = request.form['new-password']
        new_password_confirm = request.form['confirm-password']
        errors = {}

        # Validate passwords and other fields
        if not check_password_hash(get_user_password(email), old_password):
            errors['old-password'] = 'Incorrect password.'
        if new_password != new_password_confirm:
            errors['new-password'] = 'Passwords do not match.'
        if len(new_password) < 12:
            errors['length'] = 'Password must be at least 12 characters long.'
        if not re.search(r"[a-z]", new_password):
            errors['lowercase'] = 'Password must contain at least one lowercase letter.'
        if not re.search(r"[A-Z]", new_password):
            errors['uppercase'] = 'Password must contain at least one uppercase letter.'
        if not re.search(r"[0-9]", new_password):
            errors['number'] = 'Password must contain at least one number.'
        if not re.search(r"[@#%&+_-!]", new_password):
            errors['special'] = 'Password must contain at least one special character (e.g., @, #, $, etc.).'

        if errors:
            return jsonify(errors=errors), 400

        # If no errors, proceed to update the password
        new_hashed_password = generate_password_hash(new_password)
        update_password(email, new_hashed_password)
        return jsonify(success=True), 200

```

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

A CWE-352, também conhecida como Cross-Site Request Forgery (CSRF), é uma vulnerabilidade de segurança web onde um atacante induz um usuário autenticado a executar ações indesejadas em um site onde está autenticado. Isso é possível porque o navegador do usuário automaticamente envia cookies e outros tokens de autenticação com cada solicitação, independentemente da origem da solicitação. Quando não há proteção adequada, como tokens de CSRF, o atacante pode explorar essa confiança implícita para realizar operações maliciosas, como mudanças de senha ou transferências de fundos, sem o conhecimento ou consentimento do usuário.

Como aplicar:

A CWE-352, conhecida como Cross-Site Request Forgery (CSRF), pode ser explorada por um atacante ao induzir um usuário autenticado a executar ações indesejadas em um site. O atacante cria uma página maliciosa que, ao ser visitada pela vítima, envia uma solicitação para a aplicação web onde a vítima está autenticada. O navegador da vítima envia a solicitação com os cookies de sessão, permitindo que o atacante realize ações como transferências de dinheiro ou mudanças de senha, sem o conhecimento do usuário.

Impacto e Criticidade:

O impacto da CWE-352, ou Cross-Site Request Forgery (CSRF), pode ser significativo, pois permite que um atacante execute ações maliciosas em nome de um usuário autenticado, comprometendo a integridade e a confidencialidade dos dados. A criticidade dessa vulnerabilidade é elevada, pois um ataque bem-sucedido pode levar à perda de dados, transferências financeiras não

autorizadas, ou até mesmo à exposição de informações sensíveis. A falta de proteção contra CSRF pode resultar em sérios prejuízos tanto para os usuários quanto para a reputação da aplicação.

Como Mitigar:

A forma como mitiguei a CWE-352 foi implementando tokens CSRF nas requisições sensíveis. Esses tokens são gerados de forma única para cada sessão do usuário e incluídos em formulários e cabeçalhos de requisições. Ao receber uma solicitação, o servidor verifica se o token fornecido corresponde ao token armazenado na sessão do usuário, garantindo que a solicitação seja legítima e originada de uma fonte confiável. Essa abordagem impede que um atacante consiga forjar requisições em nome do usuário, protegendo a aplicação contra ataques CSRF.

```
from flask_wtf.csrf import CSRFProtect
csrf = CSRFProtect()
def create_app():
    app = Flask(__name__)
    csrf.init_app(app)
    # Import blueprints
```

```
<form method="POST" action="{{ url_for('auth.login') }}">
  <input type="hidden" name="csrf_token" value="{{ csrf_token() }}">
  <h2 class="fw-bold mb-2 text-uppercase">login</h2>
  <p class="alpha-60 mb-5">Insira o seu email e password!</p>
  <div class="form-outline form-white mb-4">
```

