

Relatório de SIO

Projeto - Vulnerabilidades

Universidade de Aveiro



DETI

Bruno Gomes
João Gonçalves
João Oliveira
Marco Almeida

16 de Novembro de 2022

Conteúdo

1	Introdução	2
2	Funcionamento do Site	3
2.1	Home	3
2.2	Forum	4
2.3	User Area	5
2.4	Login	6
2.5	Sign up	6
3	Lista de Vulnerabilidades	7
3.1	CWE-79 - Cross-site Scripting	8
3.2	CWE-89 - SQL Injection	10
3.3	CWE-284 - Improper Access Control	13
3.4	CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor	15
3.5	CWE-521 - Weak Password Requirements e CWE-20 - Improper Input Validation	17
3.6	CWE-425 - Direct Request ('Forced Browsing')	19
3.7	CWE-256 - Plaintext Storage of a Password	21
4	Conclusão	24
5	Contribuidores	25

Capítulo 1

Introdução

Este projeto consiste na criação de uma site, neste caso um fórum, com duas versões diferentes, tem-se o site funcional com diversas fraquezas na sua segurança, e também uma versão na qual estas fraquezas foram corrigidas, a versão segura do site.

Consegue-se testar as vulnerabilidades do site original devido a diversos teste que foram efetuados ao longo do projeto, todas as vulnerabilidades testadas estão presentes no relatório na lista de vulnerabilidades 3.

O conceito do site é simples, é um fórum no qual se pode fazer login ou criar conta, fazer publicações, agendar consultas, gerir consultas das quais já se encontravam marcadas e ver informações sobre os médicos participantes.

Capítulo 2

Funcionamento do Site

2.1 Home

Na página inicial do site, ou home page, pode-se ver diversas opções como efetuar a Login, o Sign up e procurar todos os médicos disponíveis, ou algum médico específico no site, pode-se também escolher contacta nos.

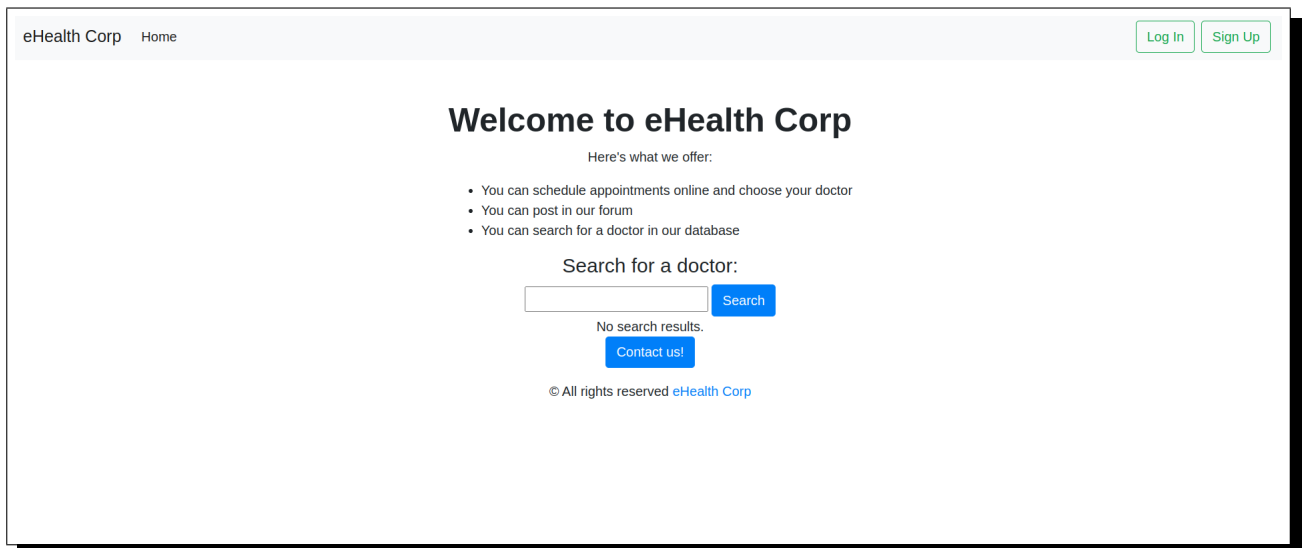


Figura 2.1: Home page

2.2 Forum

A página do forum é o local onde se podem encontrar todas as publicações feitas pelos utilizadores do site, nesta página também é possível fazer as suas próprias publicações.

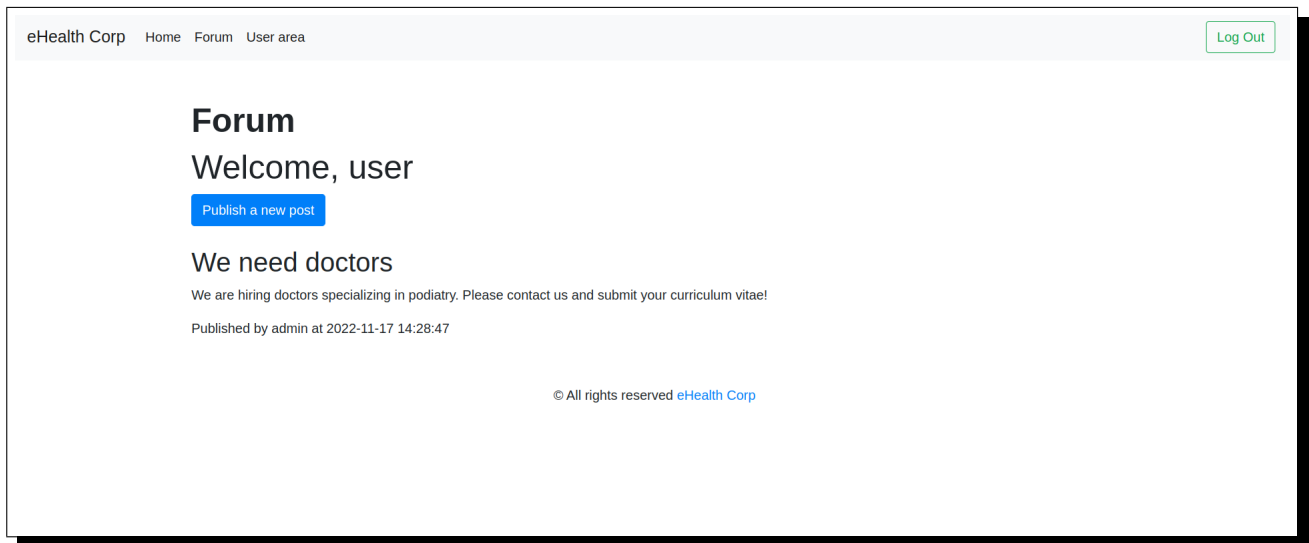


Figura 2.2: Forum page

Para efetuar uma publicação pressiona se o botão de "Publish a new post" e para efetuar a publicação têm de se inserir os dados como o titulo e o contexto da publicação.

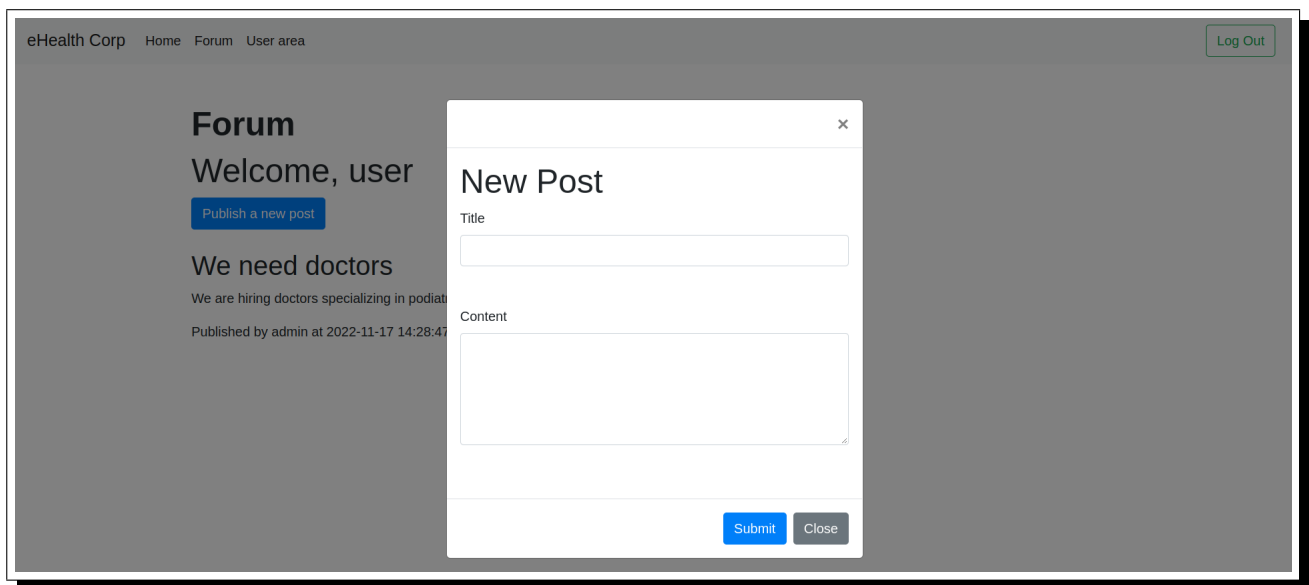


Figura 2.3: New Post Forum

2.3 User Area

Na página da area do utilizador tem-se as informações do utilizador que está com o login efetuado, consegue-se observar informações como o user, a palavra-passe do user(encriptada), o seu nome completo e o seu email.

Nesta página também se encontram a lista de consultas que o utilizador tem marcado, a opção de as marcar caso seja desejado e por último a opção de guardar um certificado com todos os dados da consulta que foi inserido o id.

eHealth Corp Home Forum User area Log Out

Logged in successfully!

user's personal space

My info

Username	Password	Full Name	Email
user	sha256\$8fyg9G527kzIKPjLS4fb5e31c2798b9b6cfbd48de51ea724181691bb0e1c64050b46fc251a74b0e9b	username	mail@gmail.com

My appointments

You have no appointments

Appointment ID

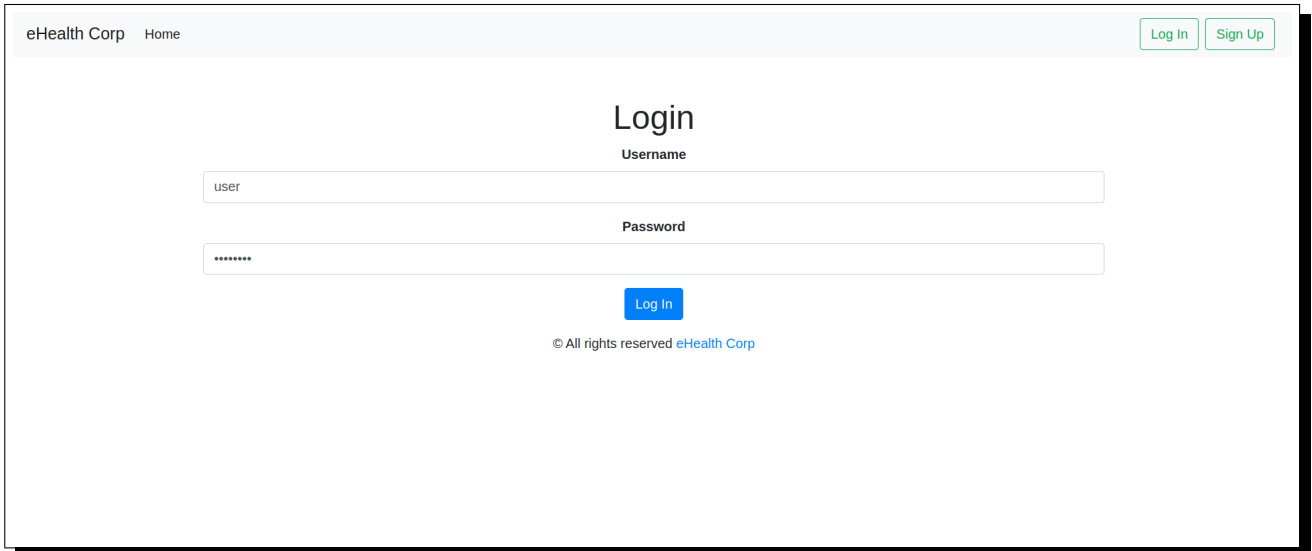
[Get Appointment Results](#)

[Schedule an appointment](#)

Figura 2.4: User area page

2.4 Login

Ao se pressionar o botão de Login presente na home page é se redirecionado para a página de login, na qual se apresenta o local para inserir o utilizador e a sua palavra-passe para aceder à conta, caso esta já tenha sido previamente criada.

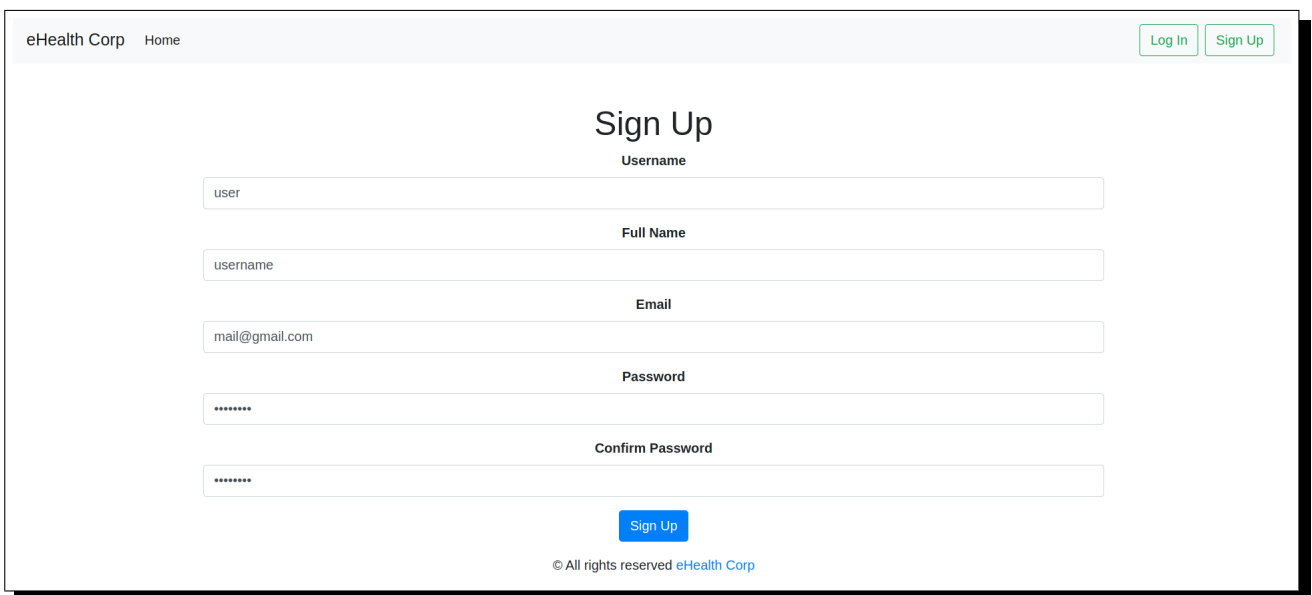


The screenshot shows the Login page for eHealth Corp. The header includes the company name and a 'Home' link, along with 'Log In' and 'Sign Up' buttons. The main content area features a 'Login' title, a 'Username' label, a text input field containing 'user', a 'Password' label, a password input field with masked characters, and a blue 'Log In' button. The footer contains the copyright notice '© All rights reserved eHealth Corp'.

Figura 2.5: Login page

2.5 Sign up

Ao se pressionar o botão de Sign up na home page é se redirecionado para a página Sign up, esta serve para o utilizador criar uma conta, para isso têm de ser inseridos os seguintes dados: nome de utilizador, nome completo, mail, palavra-passe e a sua confirmação. Após isto, o utilizador poderá efetuar o login na página normalmente.



The screenshot shows the Sign Up page for eHealth Corp. The header is identical to the login page. The main content area features a 'Sign Up' title and several input fields: 'Username' (containing 'user'), 'Full Name' (containing 'username'), 'Email' (containing 'mail@gmail.com'), 'Password' (masked), and 'Confirm Password' (masked). A blue 'Sign Up' button is at the bottom. The footer contains the copyright notice '© All rights reserved eHealth Corp'.

Figura 2.6: Sign up Page

Capítulo 3

Lista de Vulnerabilidades

3.1 CWE-79 - Cross-site Scripting

3.2 CWE-89 - SQL Injection

3.3 CWE-284 - Improper Access Control

3.4 CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor

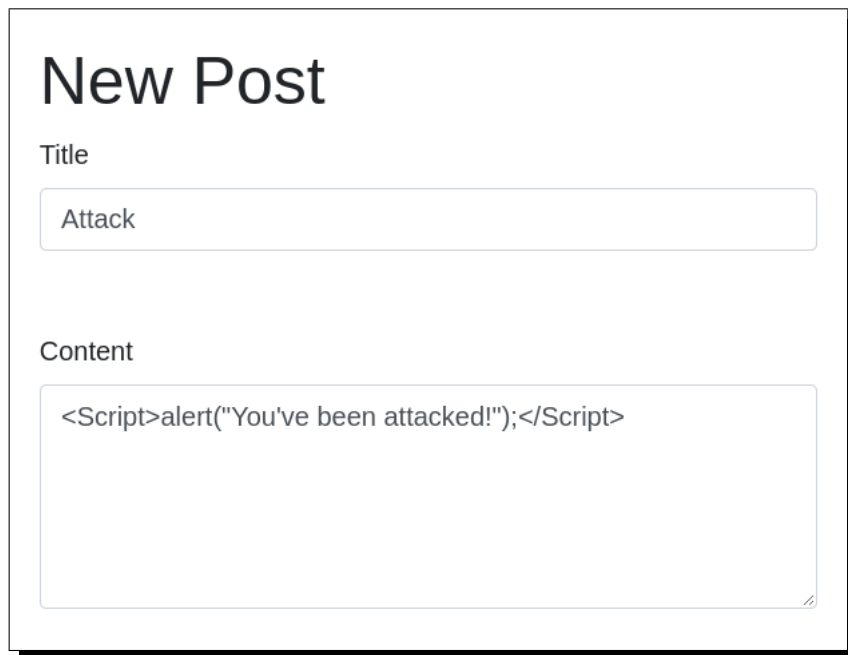
3.5 CWE-521 - Weak Password Requirements e CWE-20 - Improper Input Validation

3.6 CWE-425 - Direct Request ('Forced Browsing')

3.7 CWE-256 - Plaintext Storage of a Password

3.1 CWE-79 - Cross-site Scripting

A vulnerabilidade CWE - 79, ou Cross Site Scripting, consiste em inserir scripts maliciosos em sites ou aplicações web. Isto ocorre devido ao facto do servidor não verificar a integridade dos inputs antes de os enviar ao cliente.



The image shows a web form titled "New Post". It has two input fields: "Title" and "Content". The "Title" field contains the word "Attack". The "Content" field contains the HTML/JavaScript code: `<Script>alert("You've been attacked!");</Script>`. The form is styled with a light gray background and rounded corners.

Figura 3.1: Criação de um script malicioso

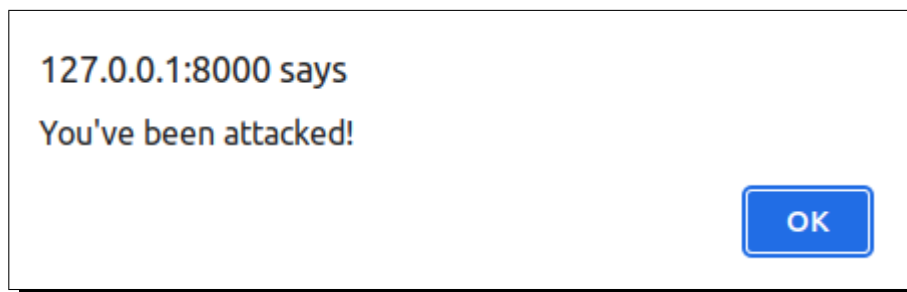


Figura 3.2: Execução do script

Na versão segura da app esta vulnerabilidade é corrigida através do uso do filtro *safe* do flask/jinja2.

```
{% for post in posts %}
<h2 class="pagination-centered">{{post[1] | safe}}</h2>
<p class="pagination-centered">{{post[2] | safe}}</p>
```

Figura 3.3: Versão segura

A esta vulnerabilidade atribuímos o valor CVSS de 9.4. Para tal, considerámos o vetor de ataque network (pois os scripts podem ser executados através da rede), complexidade de ataque baixa (os scripts são relativamente simples), não requer privilégios por parte do atacante nem

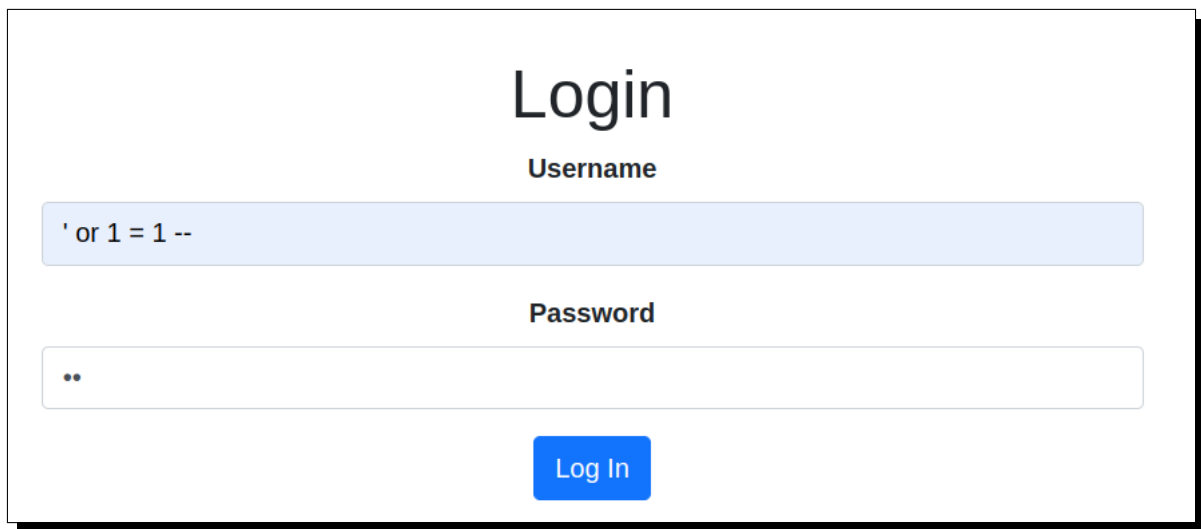
interação por parte de outro usuário para funcionar, não altera a scope (pois não afeta outras funcionalidades do servidor), existe alguma perda de confidencialidade, dependendo do script realizado pelo atacante, há uma alta perda de integridade e existe possibilidade do atacante bloquear completamente o acesso ao serviço.

CVSS score:

- Attack vector: Network
- Attack Complexity: Low
- Privileges Required: None
- User Interaction: None
- Scope: None
- Confidentiality: Low
- Integrity: High
- Availability: High

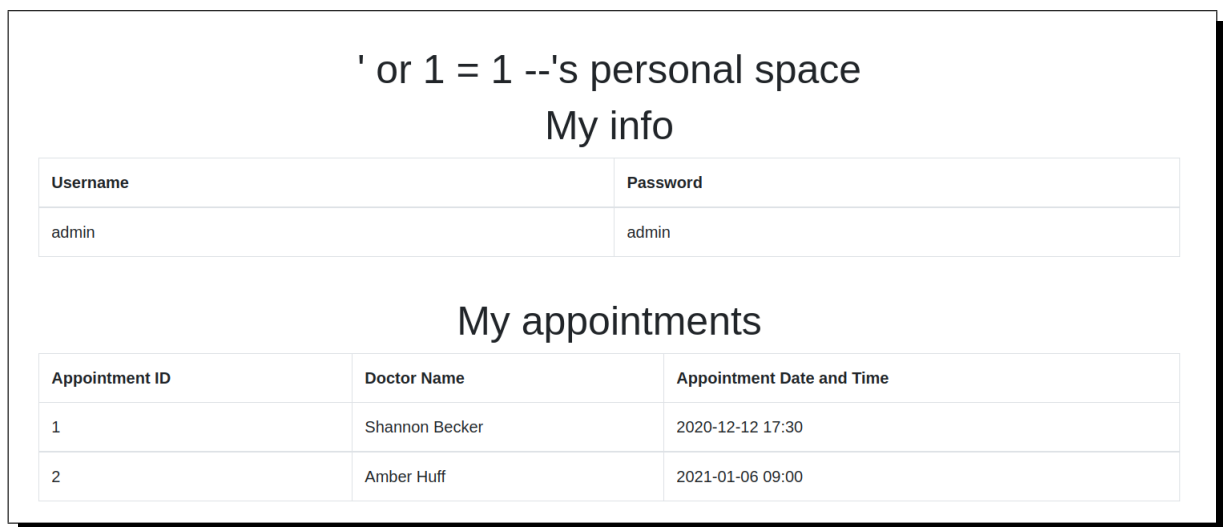
3.2 CWE-89 - SQL Injection

A vulnerabilidade CWE - 89, ou SQL Injection, consiste em inserir scripts maliciosos que irão ter acesso a bases de dados associadas aos campos de input onde foi inserido o script.



The screenshot shows a web application interface with the title "Login". Below the title, there are two input fields: "Username" and "Password". The "Username" field contains the text "' or 1 = 1 --". The "Password" field is empty. Below the input fields, there is a blue button labeled "Log In".

Figura 3.4: Criação do script



The screenshot shows a web application interface with the title "' or 1 = 1 --'s personal space". Below the title, there is a section titled "My info" which contains a table with two columns: "Username" and "Password". The table has one row with the values "admin" and "admin". Below this, there is a section titled "My appointments" which contains a table with three columns: "Appointment ID", "Doctor Name", and "Appointment Date and Time". The table has two rows: one with "1", "Shannon Becker", and "2020-12-12 17:30", and another with "2", "Amber Huff", and "2021-01-06 09:00".

Figura 3.5: Execução do script

Para solucionar este problema, tivemos de alterar a maneira como utilizávamos as bases de dados, passando a criá-las e utilizá-las de acordo com a API.

```
with app.app_context():
    db.drop_all()
    db.create_all()
```

Figura 3.6: Criação correta da base de dados

```

cursor.execute("""DROP TABLE IF EXISTS users""")
cursor.execute("""CREATE TABLE IF NOT EXISTS users (
id INTEGER PRIMARY KEY,
name TEXT NOT NULL,
password TEXT NOT NULL,
fullname TEXT NOT NULL,
email TEXT NOT NULL,
UNIQUE(name)
);""")
cursor.execute("""INSERT OR IGNORE INTO users VALUES (NULL,'admin', 'admin', 'Admin', 'adminmail@asd.com')""")
cursor.execute(
    """INSERT OR IGNORE INTO users VALUES (NULL,'zeman1', 'zeman123', 'José Manuel Figueiras', 'zeman123@hotmail.com')""")
cursor.execute(
    """INSERT OR IGNORE INTO users VALUES (NULL,'antonio', 'qwertyops', 'António Costa Silva', 'toniocosta@gmail.com')""")
connection.commit()

```

Figura 3.7: Inserção de dados de maneira incorreta na base de dados

```

new_user = User(email=email, fullname=fullname, password=generate_password_hash(
    password1, method='sha256'), username=username)
db.session.add(new_user)
db.session.commit()

```

Figura 3.8: Inserção de dados de maneira correta na base de dados

```

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True)
    username = db.Column(db.String(150), unique=True)
    fullname = db.Column(db.String(150))
    password = db.Column(db.String(150))

```

Figura 3.9: Criação da tabela de utilizadores da maneira correta

Estes scripts podem ter vários efeitos, dependendo do código SQL injetado.

A esta vulnerabilidade atribuímos o valor CVSS de 9.8. Considerámos o vetor de ataque network, complexidade de ataque baixa (basta apenas um pouco de código SQL), não requer privilégios por parte do atacante nem interação por parte de outro usuário para funcionar, não altera a scope, existe uma alta perda de confidencialidade (pois o atacante irá ter acesso às credenciais de todos os utilizadores, há uma alta perda de integridade e existe possibilidade do atacante bloquear completamente o acesso ao serviço.

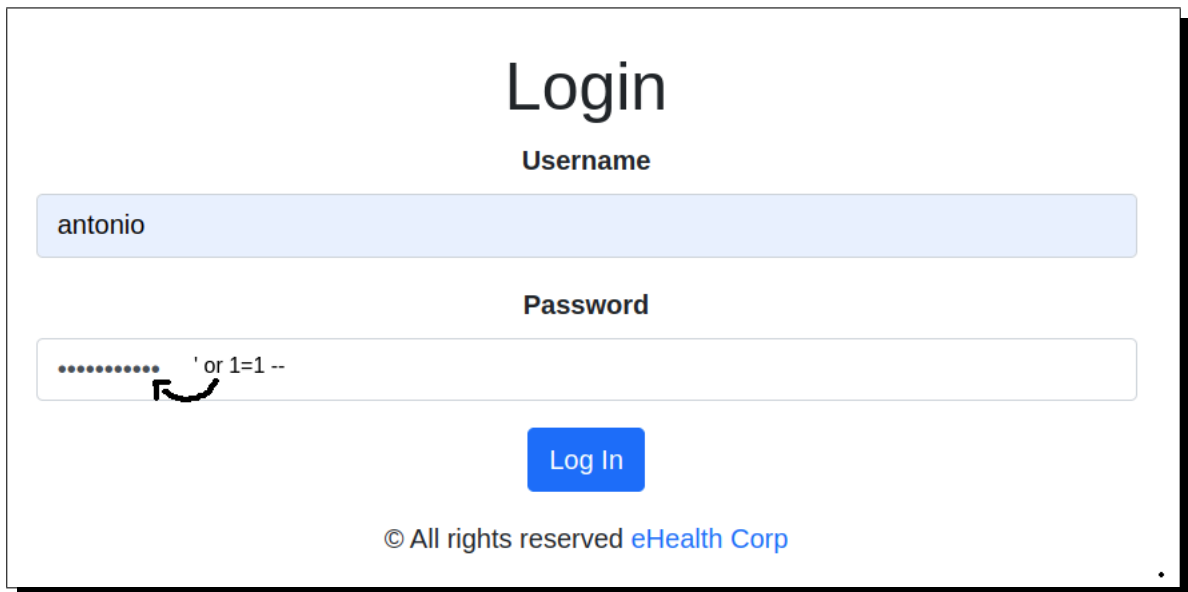
CVSS score:

- Attack vector: Network
- Attack Complexity: Low
- Privileges Required: None
- User Interaction: None
- Scope: None

- Confidentiality: High
- Integrity: High
- Availability: High

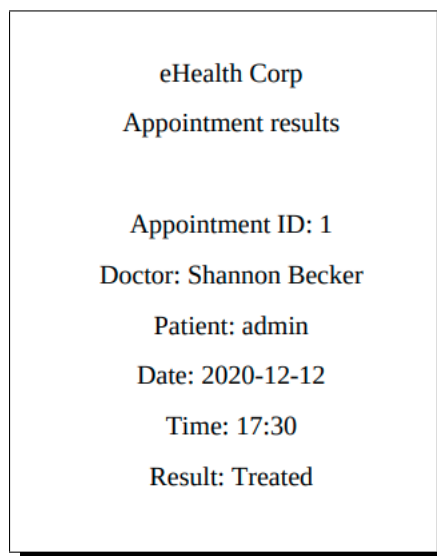
3.3 CWE-284 - Improper Access Control

Esta vulnerabilidade, de maneira semelhante à CWE - 89, pode ser explorada colocando o nome de um usuário já existente e colocando o script anterior no campo password, isto irá validar o login do usuário sem ser necessário obter a sua password.



The screenshot shows a web application login interface. At the top, the word "Login" is displayed in a large, bold font. Below it, the label "Username" is positioned above a text input field containing the name "antonio". Further down, the label "Password" is positioned above another text input field. This field contains a series of dots followed by the payload "' or 1=1 --". A curved arrow points from the text "Colocar o script no campo password" (Place the script in the password field) to the password input field. Below the password field is a blue button labeled "Log In". At the bottom of the form, the text "© All rights reserved eHealth Corp" is visible.

Figura 3.10: Colocar o script no campo password



The screenshot displays a page titled "eHealth Corp" with the subtitle "Appointment results". The main content area lists the following information: "Appointment ID: 1", "Doctor: Shannon Becker", "Patient: admin", "Date: 2020-12-12", "Time: 17:30", and "Result: Treated".

Figura 3.11: Acesso aos dados do utilizador

admin's personal space

My info

Username	Password	Full Name	Email
admin	admin	Admin	adminmail@asd.com

My appointments

Appointment ID	Doctor Name	Appointment Date and Time
1	Shannon Becker	2020-12-12 17:30
2	Amber Huff	2021-01-06 09:00

Get Appointment Results

There are no contact requests

Schedule an appointment

Figura 3.12: Acesso à conta do utilizador

A esta vulnerabilidade atribuímos o valor CVSS de 8.6. Considerámos o vetor de ataque `network`, complexidade de ataque baixa (basta apenas um pouco de código SQL), não requer privilégios por parte do atacante nem interação por parte de outro usuário para funcionar, não altera a scope, existe uma alta perda de confidencialidade (pois o atacante irá ter acesso à conta de qualquer utilizador), há alguma perda de integridade e o atacante pode cortar o acesso ao serviço apenas ao utilizador da conta comprometida (alterando a password).

CVSS score:

- Attack vector: Network
- Attack Complexity: Low
- Privileges Required: None
- User Interaction: None
- Scope: None
- Confidentiality: High
- Integrity: Low
- Availability: Low

3.4 CWE-200 - Exposure of Sensitive Information to an Unauthorized Actor

A vulnerabilidade CWE - 200 consiste na exposição de informação a utilizadores que não lhe deviam ter acesso.

No nosso caso, esta vulnerabilidade está evidente no url do site, onde qualquer utilizador pode alterar o url de modo a ter acesso a páginas de outros utilizadores.

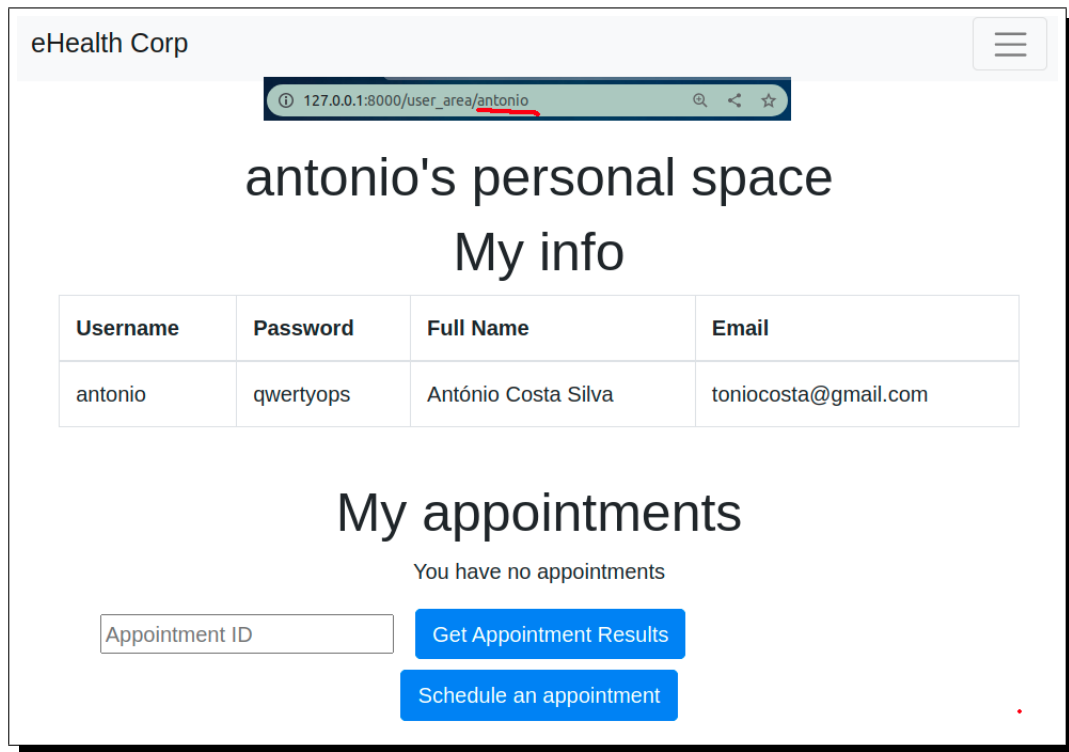


Figura 3.13: Acesso à página de um utilizador alterando o url

Na versão segura do site esta vulnerabilidade foi corrigida ao utilizar a função *login_required* presente no flask. Esta função apenas vai permitir a visualização do conteúdo da página de um utilizador apenas se for esse utilizador que tenha feito login.

```
@app.route('/user_area/<string:uname>', methods=['POST', 'GET'])
def user_area(uname):
    contact_requests = get_contact_requests()
    info = get_user_info(uname)
    if info != []:
        info = info[0]

    appointments = get_patient_appointments(uname)
    doctors = search_for_doctor('')
    print(uname, info, appointments, doctors)
    return render_template('user_area.html', uname=uname, info=info, appointments=appointments, doctors=doctors, contact_requests=contact_requests)
```

Figura 3.14: Versão insegura do código


```

@auth.route('/user_area/<string:uname>')
@login_required
def user_area(uname):
    if uname != current_user.username:
        flash("Restricted access.", category="error")
        return redirect(url_for('auth.login'))
    appointments = Appointment.query.filter_by(patient=current_user.username).all()
    contact_requests = ContactForm.query.all() if current_user.username == 'admin' else []
    return render_template('user_area.html', user=current_user, doctors=Doctor.query.all(), contact_requests=contact_requests, appointments=appointments)

```

Figura 3.15: Versão segura do código

A esta vulnerabilidade atribuímos o valor CVSS de 8.6 pois os efeitos que esta causa no servidor irão ser muito semelhantes à vulnerabilidade CWE - 284, logo os seus parâmetros CVSS irão ser idênticos.

CVSS score:

- Attack vector: Network
- Attack Complexity: Low
- Privileges Required: None
- User Interaction: None
- Scope: None
- Confidentiality: High
- Integrity: Low
- Availability: Low

3.5 CWE-521 - Weak Password Requirements e CWE-20 - Improper Input Validation

As vulnerabilidades CWE - 521 e CWE - 20 são bastante semelhantes, por isso iremos analisá-las em conjunto.

A vulnerabilidade CWE - 521 consiste em não aplicar critérios rigorosos na criação da password de um utilizador, o que fará com que esta esteja mais vulnerável a ataques.

Para resolver este problema foram implementadas na versão segura da app uma série de medidas para aumentar a complexidade e segurança da password de um utilizador.

A vulnerabilidade CWE - 20 ocorre quando os dados introduzidos pelo utilizador não são devidamente tratados, o que pode levar a erros e conflitos, visto que muitos desses dados estarão presentes no url.

Para resolver esta vulnerabilidade foi criada a função *replace_all()* para substituir todos os caracteres que poderiam gerar problemas por versões seguras dos mesmos.

```
@app.route('/signup/', methods=['POST', 'GET'])
def signup():
    if request.method == 'POST':
        uname = request.form['uname']
        psw = request.form['psw']
        confirmpsw = request.form['confirmpsw']
        email = request.form['email']
        fullname = request.form['fullname']
        result, msg = insert_user(uname, psw, confirmpsw, fullname, email)
        if not result:
            flash(msg, category='error')
        else:
            flash(msg, category='success')
            return render_template('signup.html')
    else:
        return render_template('signup.html')
```

Figura 3.16: Versão insegura do código

```

@auth.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('uname')
        email = request.form.get('email')
        fullname = request.form.get('fullname')
        password1 = request.form.get('psw')
        password2 = request.form.get('confirmpsw')

        user = User.query.filter_by(username=username).first()
        user2 = User.query.filter_by(email=email).first()
        if user:
            flash('Username already exists.', category='error')
        if user2:
            flash('Email already exists.', category='error')

        if len(email) < 5:
            flash("Email must be greater than 4 characters.", category='error')
        elif password1 != password2:
            flash("Passwords don't match.", category='error')
        elif len(password1) < 7:
            flash("Password must be at least 7 characters.", category='error')
        else:
            username = replace_all(username)

            new_user = User(email=email, fullname=fullname, password=generate_password_hash(password1, method='sha256'), username=username)
            db.session.add(new_user)
            db.session.commit()
            login_user(new_user, remember=True)
            flash("Account created!", category='success')
            return redirect(url_for('views.index'))

    return render_template("signup.html", user=current_user)

```

Figura 3.17: Versão segura do código

A estas vulnerabilidades atribuímos o valor CVSS de 7. Para tal, considerámos o vetor de ataque network, complexidade de ataque alta (pois o atacante ainda terá de investir em outro software para aceder às bases de dados), não requer privilégios por parte do atacante nem interação por parte de outro usuário para funcionar, não altera a scope, existe uma alta perda de confidencialidade (caso o atacante consiga aceder à conta de outro utilizador), ocorre alguma perda de integridade e o atacante consegue cortar o acesso aos recursos do site apenas ao utilizador que atacou.

CVSS score:

- Attack vector: Network
- Attack Complexity: High
- Privileges Required: None
- User Interaction: None
- Scope: None
- Confidentiality: High
- Integrity: Low
- Availability: Low

3.6 CWE-425 - Direct Request ('Forced Browsing')

Esta vulnerabilidade ocorre quando o site ou aplicação não aplica restrições em url's, scripts ou ficheiros restritos, como páginas que iriam precisar de login para serem acessadas.

```
@app.route('/forum/<string:uname>', methods=['POST', 'GET'])
def forum(uname):
    posts = get_posts()
    if request.method == "POST":
        return render_template("forum.html", uname=uname, posts=posts)
    else:
        return render_template('forum.html', uname=uname, posts=posts)
```

Figura 3.18: Versão insegura do código

Para resolver a vulnerabilidade foi adicionado um *decorator* a páginas restritas para efetivamente verificar se foi efetuado o login e se o login foi efetuado pelo utilizador que está a tentar acessar a página.

```
@auth.route('/forum/<string:uname>', methods=['POST', 'GET'])
@login_required
def forum(uname):
    if uname != current_user.username:
        flash("Restricted access.", category="error")
        return redirect(url_for('auth.login'))
    return render_template("forum.html", user=current_user, posts=Post.query.all())
```

Figura 3.19: Versão segura do código

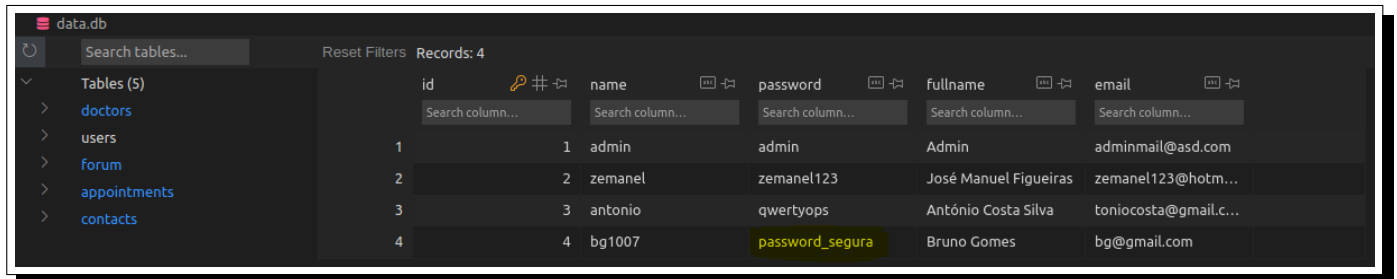
A esta vulnerabilidade atribuímos o valor CVSS de 5.3. Para tal, considerámos o vetor de ataque network, complexidade de ataque baixa, não requer privilégios por parte do atacante nem interação por parte de outro usuário para funcionar, não altera a scope, existe alguma perda de confidencialidade (uma vez que o atacante irá ter acesso a conteúdos restritos), não ocorre qualquer perda de integridade nem qualquer perda de acesso aos serviços.

CVSS score:

- Attack vector: Network
- Attack Complexity: Low
- Privileges Required: None
- User Interaction: None
- Scope: None
- Confidentiality: Low
- Integrity: None
- Availability: None

3.7 CWE-256 - Plaintext Storage of a Password

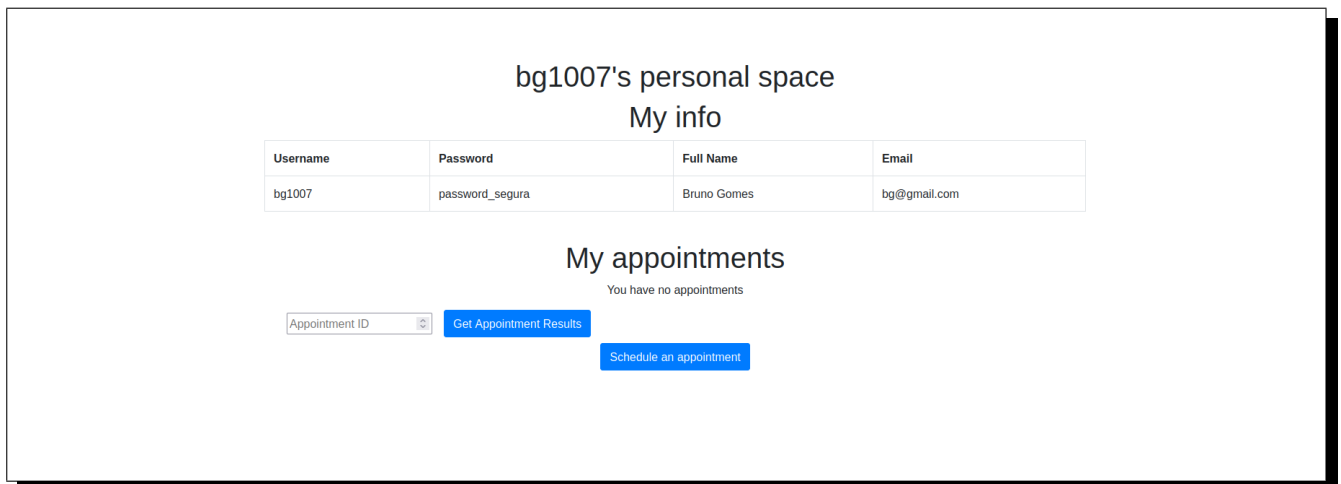
Esta vulnerabilidade ocorre quando as passwords criadas pelos utilizadores são guardadas numa base de dados em plain text, ou seja, texto não encriptado. Com isto surgem vários problemas caso uma base de dados seja vítima de um ataque, pois o atacante terá acesso a todas as passwords imediatamente.



The screenshot shows a database interface with a table containing user information. The password field for the user 'bg1007' is highlighted, showing the plaintext value 'password_segura'.

id	name	password	fullname	email
1	admin	admin	Admin	adminmail@asd.com
2	zemanel	zemanel123	José Manuel Figueiras	zemanel123@hotmail.com
3	antonio	qwertyops	António Costa Silva	toniocosta@gmail.com
4	bg1007	password_segura	Bruno Gomes	bg@gmail.com

Figura 3.20: Passwords em plain text na base de dados insegura



The screenshot shows a user profile page for 'bg1007's personal space'. Under the 'My info' section, there is a table displaying user details. The password field shows the plaintext value 'password_segura'.

Username	Password	Full Name	Email
bg1007	password_segura	Bruno Gomes	bg@gmail.com

Below the 'My info' section, there is a 'My appointments' section with a message 'You have no appointments'. It includes a search bar for 'Appointment ID', a 'Get Appointment Results' button, and a 'Schedule an appointment' button.

Figura 3.21: Display da password

Como podemos ver no código abaixo, a password é guardada em texto normal, sem qualquer tipo de encriptação

```
@app.route('/signup/', methods=['POST', 'GET'])
def signup():
    if request.method == 'POST':
        uname = request.form['uname']
        psw = request.form['psw']
        confirmpsw = request.form['confirmpsw']
        email = request.form['email']
        fullname = request.form['fullname']
        result, msg = insert_user(uname, psw, confirmpsw, fullname, email)
        if not result:
            flash(msg, category='error')
        else:
            flash(msg, category='success')
            return render_template('signup.html')
    else:
        return render_template('signup.html')
```

Figura 3.22: Versão não segura do código

Para resolver a vulnerabilidade, as passwords foram encriptadas utilizando o algoritmo SHA-256, para que as passwords não pudessem ser lidas diretamente.

instance > database.db

Search tables... Reset Filters Records: 4

Tables (5)	id	email	username	fullname	password
user	Search column...	Search column...	Search column...	Search column...	Search column...
doctor	1	adminmail@asd.com	admin	admin	sha256\$kehsecK3...
contact_form	2	zemanel123@hotmail...	zemanel	José Manuel Figueiras	sha256\$jYyuP1n4H...
appointment	3	toniostaca@gmail.c...	antonio	António Costa Silva	sha256\$5ASSY1jHZ...
post	4	bg@gmail.com	bg34567	Bruno Gomes	sha256\$E2wSOIOhp...

Figura 3.23: Passwords encriptadas na base de dados segura

bg34567's personal space

My info

Username	Password	Full Name	Email
bg34567	sha256\$E2wSOIOhp6zvOlnq\$04d742a8316c56d34ac37c160bc5880ef22d6dc321b0b746763eb121f7737b1c	Bruno Gomes	bg@gmail.com

My appointments

You have no appointments

Appointment ID

Figura 3.24: Display da password

```

@auth.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('uname')
        email = request.form.get('email')
        fullname = request.form.get('fullname')
        password1 = request.form.get('psw')
        password2 = request.form.get('confirmpsw')

        user = User.query.filter_by(username=username).first()
        user2 = User.query.filter_by(email=email).first()
        if user:
            flash('Username already exists.', category='error')
        if user2:
            flash('Email already exists.', category='error')

        if len(email) < 5:
            flash("Email must be greater than 4 characters.", category='error')
        elif password1 != password2:
            flash("Passwords don't match.", category='error')
        elif len(password1) < 7:
            flash("Password must be at least 7 characters.", category='error')
        else:
            username = replace_all(username)

            new_user = User(email=email, fullname=fullname, password=generate_password_hash(password1, method='sha256'), username=username)
            db.session.add(new_user)
            db.session.commit()
            login_user(new_user, remember=True)
            flash("Account created!", category='success')
            return redirect(url_for('views.index'))
    return render_template("signup.html", user=current_user)

```

Figura 3.25: Versão segura do código

A esta vulnerabilidade atribuímos o valor CVSS de 7. Considerámos o vetor de ataque network, complexidade de ataque alta (pois o atacante ainda terá de investir em outro software para aceder às bases de dados), não requer privilégios por parte do atacante nem interação por parte de outro usuário para funcionar, não altera a scope, existe uma alta perda de confidencialidade (caso o atacante consiga aceder à conta de outro utilizador), ocorre alguma perda de integridade e o atacante consegue cortar o acesso aos recursos do site apenas ao utilizador que atacou.

CVSS score:

- Attack vector: Network
- Attack Complexity: High
- Privileges Required: None
- User Interaction: None
- Scope: None
- Confidentiality: High
- Integrity: Low
- Availability: Low

Capítulo 4

Conclusão

Ao longo da realização deste projeto, viemos a descobrir muitos dos mecanismos que permitem o funcionamento dos sites e aplicações web que usamos tão frequentemente. Viemos também a descobrir muitas das falhas e vulnerabilidades existentes nestes serviços, pois embora o projeto fosse focado em tais vulnerabilidades, enquanto implementávamos uma, surgiam outras pelas quais não esperávamos, o que nos deu a conhecer um pouco do processo de desenvolvimento deste tipo de serviços.

Ao corrigirmos as vulnerabilidades implementadas e corrigindo outras das quais não estávamos à espera, acabámos o projeto com um score cvss de 55.7, bastante superior ao esperado, mas que mesmo assim conseguimos corrigir por completo.

Capítulo 5

Contribuidores

O Trabalho foi analisado, discutido e realizado por

- Bruno Gomes 103320
- João Gonçalves 98287
- João Oliveira 102631
- Marco Almeida 103440