



universidade
de aveiro

Segurança Informática e nas Operações

Vulnerabilidades

Projeto nº 1

2022/2023

Diogo Pires, 97889, P7
João Monteiro, 102690, P7
João Sousa, 103415, P1
Vânia Morais, 102383, P7

Índice

Vulnerabilidades	2
CWE-256: Plaintext Storage of a Password	2
CWE-79: Cross Site Scripting(XSS)	2
CWE-521: Weak Password Requirements	4
CWE-89: SQL Injection	5
CWE-20: Improper Input Validation	6
CWE-200: Exposure of Sensitive Information to an Unauthorized Actor	7

Vulnerabilidades

CWE-256: Plaintext Storage of a Password

Nesta vulnerabilidade foi apenas necessário guardar as passwords na base de dados em hash. Para isso acrescentamos o código mostrado na figura 1.

```
if form.validate_on_submit():
    admin = User(username = form.username.data, password = generate_password_hash(form.password.data),
    db.session.add(admin)
```

figura 1 - Código responsável pela encriptação da palavra chave

5	vania05@email.com	pbkdf2:sha256:260000\$XBXPuHabB9tE9tOe\$ad06045a3a17691ae91e5288d415ad5eff8f1ac29deca9eaf7424b8a3e9983d1	Male	Vania Morais	False
---	-------------------	--	------	-----------------	-------

figura 2 - Exemplo de uma password encriptada

CWE-79: Cross Site Scripting(XSS)

Para testar esta vulnerabilidade, inserimos na barra de pesquisa da página da equipa de médicos, um script do tipo **<script>alert("You have been hacked!")</script>**.

Meet Our Team

<script>alert("You have been hacked!")</script>

figura 3- Exemplo de como foi feito o ataque de Cross Site Scripting

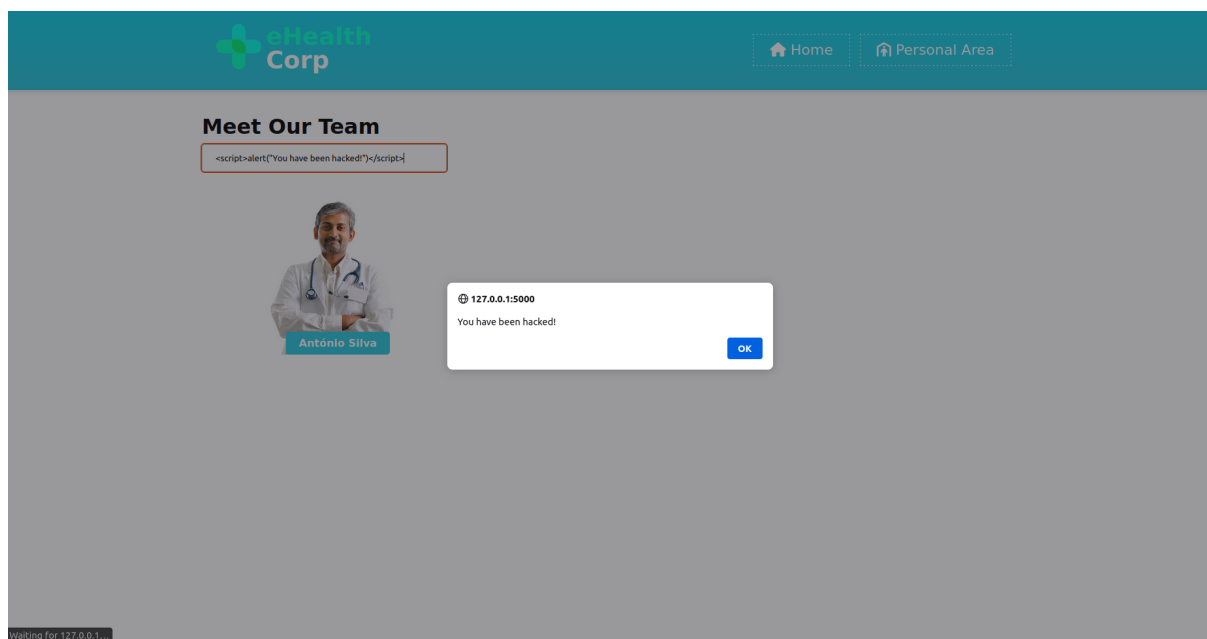


figura 4- resultado do ataque Cross Site Scripting

Para resolver esta situação, aproveitamos o facto de o *flask* já ser seguro o suficiente nestes casos, não dando como seguras todas as inserções feitas pelo utilizador.

```
<form action="/doctors" method="POST">
  {{ form.hidden_tag() }}
  {{ wtf.form_field(form.name) }}
</form>
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <h3 class=flashes style="color:rgb(216, 35, 35);">
      {% for message in messages %}
        {{ message }}
      {% endfor %}
    </h3>
  {% endif %}
{% endwith %}
```

figura 5- Código Flask usado para combater a vulnerabilidade de Cross Scripting

CWE-521: Weak Password Requirements

Passwords fracas são mais fáceis de descobrir pelo método *Brute Force*. Para evitar que os utilizadores não usem passwords fortes por não terem noção do perigo, acrescentamos o código apresentado na figura 6.

```
def password_check(form,field):
    password = form.password.data
    if len(password)< 4:
        raise ValidationError('Password must be at lest 8 letters long')
    elif re.search('[0-9]',password) is None:
        raise ValidationError('Password must contain a number')
    elif re.search('[A-Z]',password) is None:
        raise ValidationError('Password must have one uppercase letter')

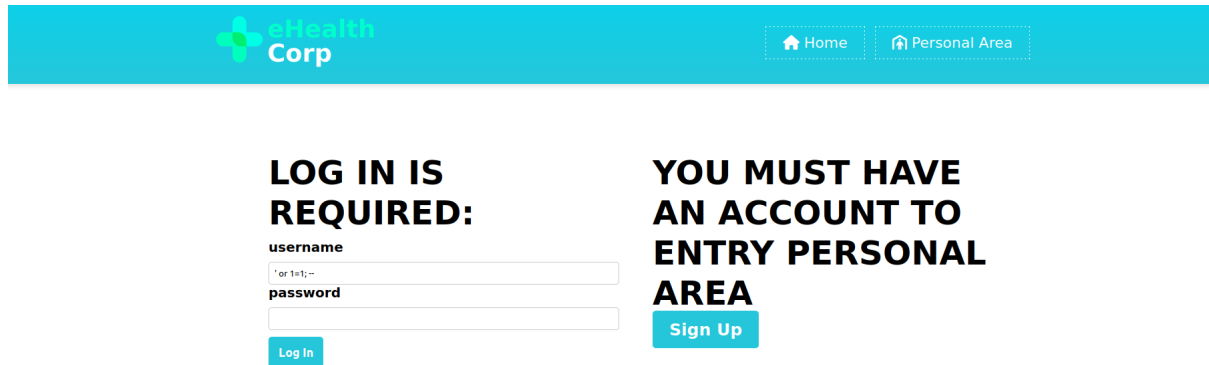
class RegisterForm(FlaskForm):
    gender = SelectField('Gender', choices= (('Male'),('Female'),('Other'),('Prefer not to
    email = StringField('Email', validators = [InputRequired(), Email(message = 'Invalid e
    full_name = StringField('Full name', validators = [InputRequired(), Length(min = 3, ma
    username = StringField('Username', validators = [InputRequired(), Length(min = 4, max
    password = PasswordField('Password', validators = [[InputRequired(), password_check]]))
```

figura 6- Código que torna a password mais forte

Neste pedaço de código, podemos ver que foi feita uma função (*password_check*) onde são definidos todos os requisitos que achamos necessários para uma password ser forte.

CWE-89: SQL Injection

Nesta vulnerabilidade, conseguimos fazer Login sem ter uma conta. Para isso, basta, no campo do username, inserir código SQL do tipo (“ ‘ or 1=1; – “):



The screenshot shows the eHealth Corp login interface. At the top, there is a navigation bar with the eHealth Corp logo and links for Home and Personal Area. The main content area is split into two columns. The left column, titled 'LOG IN IS REQUIRED:', contains a login form with fields for 'username' and 'password', and a 'Log In' button. The right column, titled 'YOU MUST HAVE AN ACCOUNT TO ENTRY PERSONAL AREA', contains a 'Sign Up' button. The 'username' field contains the SQL injection payload: ' or 1=1; – '.

figura 7- Inserção de código SQL

Para evitar situações destas no futuro, foi necessário alterar o código de forma a: primeiro, não ser possível deixar campos vazios e segundo, verificar se o user existe. Mais uma vez tiramos proveito do facto de *Flask* ser seguro neste tipo de situações.

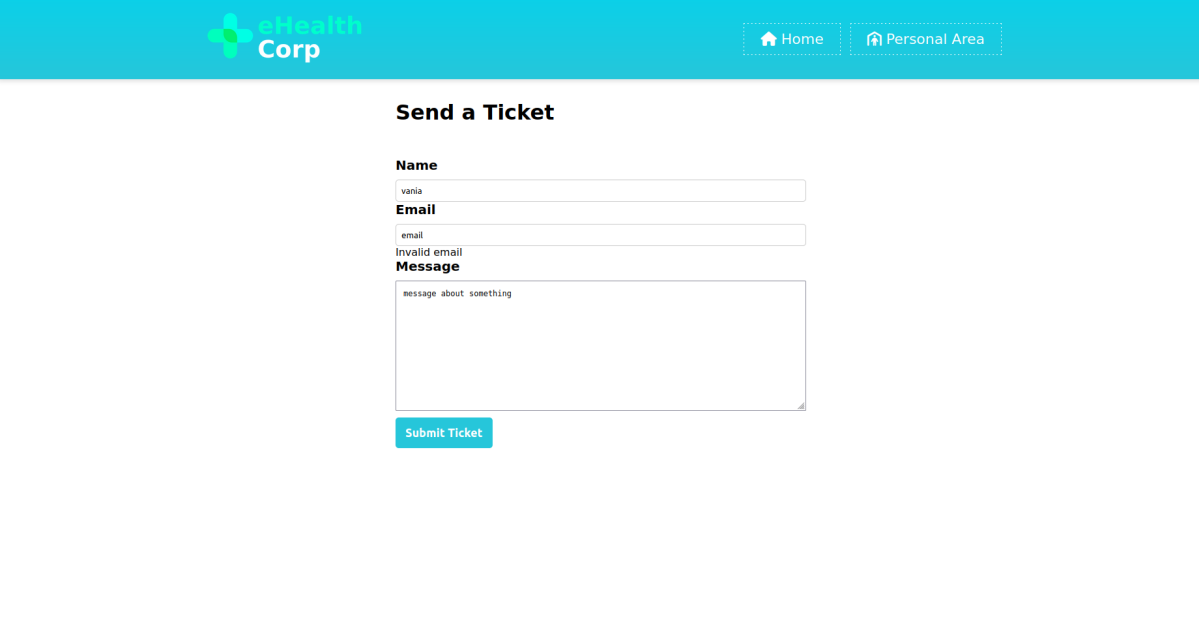
```
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()

    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).first()
        if user:
            password = user.password
            if check_password_hash(password, form.password.data):
                login_user(user, remember = form.remember.data)
                return redirect(url_for('dashboard'))
            return '<h1>Invalid username or password</h1>'
    return render_template('login.html', form=form)
```

Figura 8- Código que previne uma SQL Injection

CWE-20: Improper Input Validation

Para testar esta vulnerabilidade, tentamos inserir um email não válido. Na app não segura isto não tem qualquer tipo de problema, mas na segura, o utilizador não o consegue fazer, como podemos ver na figura abaixo. Isto acontece porque foi feita uma validação no campo do email.



The screenshot shows a web application interface for 'eHealth Corp'. At the top, there is a blue header with the company logo and two navigation links: 'Home' and 'Personal Area'. Below the header, the main content area is titled 'Send a Ticket'. It contains a form with three fields: 'Name' (containing 'vania'), 'Email' (containing 'email'), and 'Message' (containing 'message about something'). The 'Email' field has a red error message 'Invalid email' displayed below it. A 'Submit Ticket' button is located at the bottom of the form.

figura 9- Resultado de uma Validation

```
class TicketForm(FlaskForm):  
    name = StringField('Name', validators = [InputRequired(), Length(min = 4, max = 30)])  
    email = StringField('Email', validators = [InputRequired(), Email(message = 'Invalid email'), Length(max = 50)])  
    message = TextAreaField('Message', validators = [InputRequired(), Length(min = 4, max = 300)], )
```

Figura 10- Validações

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

De modo a vermos esta vulnerabilidade, depois de ficarmos registados, ao fazer Login, se nos enganarmos na password, é nos dado um alerta que a password está errada, dando a entender que o user existe.

Para prevenir isto, deixamos de dar mensagens de erro específicas e passamos a dar mensagens mais gerais, do tipo:

Invalid username or password

Figura 11- Mensagem de erro