



universidade de aveiro  
theoria poiesis praxis

# Vulnerabilidades

## Delta Report

**Unidade Curricular: Segurança Informática nas Organizações (SIO)**

DETI

Ano Letivo 2022/23

### **Autores:**

- 102690, João Monteiro, P7
- 102382, Vânia Morais, P7
- 103415, João Sousa, P1

## Contexto e Metodologia

Este relatório tem como objetivo apresentar uma análise detalhada das vulnerabilidades CWE (Common Weakness Enumeration) existentes no nosso sistema. Após revisão do relatório anterior, constatou-se que havia uma falta de explicação clara e detalhada sobre cada vulnerabilidade. Diante disso, decidimos refazer o relatório, abordando cada vulnerabilidade individualmente, fazendo a sua análise, demonstração e explicando porque é que ela acontece. Além disso, serão apresentadas as mudanças feitas para preveni-las. Essas vulnerabilidades incluem:

1. CWE-256: Plaintext Storage of a Password
2. CWE-79: Cross Site Scripting(XSS)
3. 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

## Índice

<b>Contexto e Metodologia</b>	<b>1</b>
<b>Cálculo do CVSS</b>	<b>3</b>
<b>CWE-256: Plaintext Storage of a Password</b>	<b>5</b>
<b>CWE-79: Cross Site Scripting(XSS)</b>	<b>8</b>
<b>CWE-521: Weak Password Requirements</b>	<b>11</b>
<b>CWE-89: SQL Injection</b>	<b>14</b>
<b>CWE-20: Improper Input Validation</b>	<b>17</b>
<b>CWE-200: Exposure of Sensitive Information to an Unauthorized Actor</b>	<b>20</b>
<b>Conclusão</b>	<b>22</b>
<b>Referências</b>	<b>22</b>

## Cálculo do CVSS

Antes de tudo, achamos importante explicar como é feito o cálculo de cada CVSS, para uma posterior compreensão das nossas escolhas.

O processo de cálculo de um score começa primeiro com o cálculo das métricas base de acordo com a equação, onde é equacionado um valor que pode variar de zero a dez e cria um vetor. O vetor é representado por uma cadeia de caracteres que contém os valores atribuídos a cada métrica base, isto tem como objetivo comunicar exatamente como o cálculo é derivado para cada vulnerabilidade.

Existem seis métricas base que capturam as características mais fundamentais de uma vulnerabilidade:

1. Attack Vector (AV): Esta métrica reflete o contexto pelo qual a exploração da vulnerabilidade é possível. A pontuação aumenta quanto mais remoto (lógica e fisicamente) um invasor pode estar para explorar o componente vulnerável.

2. Attack Complexity (AC): Esta métrica descreve as condições além do controle do invasor que devem existir para explorar a vulnerabilidade. Tais condições podem exigir a coleta de mais informações sobre o destino, a presença de determinadas configurações do sistema ou exceções computacionais.

3. Privileges Required (PR): Essa métrica descreve o nível de privilégios que um invasor deve possuir antes de explorar a vulnerabilidade com êxito. Esta pontuação aumenta à medida que menos privilégios são necessários.

4. User Interaction (UI): Esta métrica captura o requisito de um usuário, que não seja o invasor, para participar do comprometimento bem-sucedido do componente vulnerável. Esta métrica determina se a vulnerabilidade pode ser explorada apenas pela vontade do invasor ou se um usuário separado (ou processo iniciado pelo usuário) deve participar de alguma maneira. A pontuação aumenta quando nenhuma interação do usuário é necessária.

5. Scope (S): se um ataque bem-sucedido afeta um componente diferente do componente vulnerável a pontuação aumenta e as métricas de

Confidencialidade, Integridade e Autenticação devem ser pontuadas em relação ao componente afetado.

6. Confidentiality (C): Esta métrica mede o impacto na confidencialidade dos recursos de informação gerenciados por um componente de software devido a uma vulnerabilidade explorada com sucesso. A confidencialidade refere-se a limitar o acesso e a divulgação de informações apenas a usuários autorizados, bem como impedir o acesso ou a divulgação para, não autorizados.

7. Integrity (I): Esta métrica mede o impacto na integridade de uma vulnerabilidade explorada com sucesso. Integridade refere-se à confiabilidade e veracidade das informações.

8. Availability (A): Esta métrica mede o impacto na disponibilidade do componente afetado resultante de uma vulnerabilidade explorada com sucesso. Refere-se à perda de disponibilidade do próprio componente afetado, como um serviço de rede (por exemplo, web, banco de dados, e-mail). Como a disponibilidade se refere à acessibilidade dos recursos de informação, os ataques que consomem largura de banda da rede, ciclos do processador ou espaço em disco afetam a disponibilidade de um componente afetado.

Com base nas vulnerabilidades encontradas na nossa aplicação, através de uma análise dos problemas que podem causar, calculamos o CVSS de cada uma para analisar o nível de severidade e risco de uma vulnerabilidade no ambiente de computação.

## **CWE-256: Plaintext Storage of a Password**

CVSS - 9.6 (Critical)

Vector String - CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H

A vulnerabilidade CWE-256, conhecida como Plaintext Storage of a Password, ocorre quando as senhas dos usuários são armazenadas sem criptografia, o que significa que elas estão em formato de texto simples e podem ser facilmente lidas e acessadas por qualquer pessoa que tenha acesso ao banco de dados ou arquivo onde as senhas estão armazenadas.

Armazenar senhas em texto simples é perigoso pois coloca tanto o sistema quanto os usuários em risco. Se um hacker conseguir encontrar e ler as senhas usadas para acessar um sistema. Eles poderiam simplesmente encontrar um usuário com credenciais de administrador e comprometer todo o sistema ou site. Além disso, como os invasores estariam a utilizar nomes de usuários e senhas corretas, a segurança interna pode não detectar a invasão ou detectá-la muito tempo depois do dano já ter sido causado.

No contexto da nossa aplicação, como é um sistema relacionado à saúde, numa situação de uso real, este sistema conteria informações confidenciais dos pacientes, que poderiam ser vazadas (leaked), adulteradas, entre outros casos.

Para o cálculo do CVSS desta vulnerabilidade consideramos que o Attack Vector (AV): a vulnerabilidade pode ser explorada remotamente na network; Attack Complexity (AC): a complexidade de um ataque necessário para explorar esta vulnerabilidade foi avaliado como baixo; Privileges Required (PR): consideramos que não precisa de privilégios; User Interaction (UI): é necessária interação do utilizador; Scope (S): a vulnerabilidade afeta recursos além dos privilégios de autorização pretendidos pela componente vulnerável; Confidentiality (C): o impacto na confidencialidade da vulnerabilidade é alto; Integrity (I): o impacto na integridade da vulnerabilidade é grande; Availability (A): há perda total de disponibilidade, fazendo com que o invasor seja capaz de negar totalmente o acesso aos recursos do componente afetado.

Passando a falar de como os dados são armazenados, no momento de criação da conta de cada utilizador, são enviados para a base de dados os dados de cadastro de acordo com o que cada pessoa preenche (username, email, password, etc). Destes, na nossa aplicação, são utilizados o username e password para fazer o log in.

Na aplicação insegura, a password é armazenada na base de dados em Plaintext, isto é, sem qualquer tipo de processamento, da maneira como foram escritos na input box. Seriam então enviados para a base de dados com o formato:

email do usuário	password
joaomonteir5@gmail.com	joaozinho
toni.doc@ehealth.com	asjidjasdias

*Tabela 1:* esta tabela foi criada apenas para simular os dados em Plaintext que estão na base de dados

Numa situação em que o invasor conseguisse acessar a base de dados, obteria imediatamente acesso aos dados de todos os usuários já que, estes apresentar-se-iam na a base de dados como foram enviados para lá, sem hashing. Então, para contornar este problema decidimos implementar no nosso código uma função de hash que atua no momento em que se dá a submissão do formulário de cadastro e os dados são enviados para a base de dados. Esta função é do módulo utilities da livreria Werkzeug. Para aplicá-la para dar hash às passwords, passamos como parâmetros a password, o método de hashing (pbkdf2:sha256) e o tamanho das string que vamos utilizar como Salt (utilizamos 8).

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    form = RegisterForm()
    if form.validate_on_submit():
        new_user = User(username = form.username.data, gender =
form.gender.data, full_name = form.full_name.data, email =
form.email.data, password = generate_password_hash
(form.password.data, method='pbkdf2:sha256', salt_length=8)
        db.session.add(new_user)
        db.session.commit()
        return '<h1>new user has been created</h1>'

    return render_template('signup.html', form=form)
```

Consequentemente, na base de dados as passwords irão apresentar-se do seguinte modo:

email do usuário	password
userteste22@gmail.com	pbkdf2:sha256:260000\$QMJqKUZiqYd gTs5Q\$5508bad3dc6764e3f2891e72a6 6134a2994b47b9bc2e04f77982e84730 a18fe7
userteste23@gmail.com	pbkdf2:sha256:260000\$VM5EyMXdPre 9LEYd\$409173d47b9456d69f7a14307e 815e81800c7ae2a5052010b0dc969aca 7f6689

Tabela 2: esta tabela foi criada apenas para simular os dados pós-hash que estão na base de dados.

Vale ainda ressaltar que as passwords apresentadas para o userteste22 e userteste23 são exatamente as mesmas, mas apresentadas na base de dados de maneiras diferentes, pois foi utilizado um Salt distinto na hora de dar hash a cada uma das passwords.

Para log in, utilizamos a função `check_password_hash` também do módulo `utilities` livreria da `Werkzeug`. Esta função recebe como entrada a password hashed que a `generate_password_hash` retorna e a password introduzida no forms em Plaintext. Retorna Verdadeiro se a senha corresponder, Falso caso contrário.

```
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>'
```



## CWE-79: Cross Site Scripting(XSS)

CVSS - 7.1(High)

Vector String - CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:L

Cross-Site Scripting (XSS) é uma vulnerabilidade de segurança comum em aplicativos web que permite que um invasor injete código malicioso (geralmente JavaScript). Isso pode permitir que o invasor roube informações sensíveis dos usuários, como senhas ou dados bancários, ou execute ações maliciosas em nome dos usuários, como transferir fundos de suas contas bancárias. Existem dois tipos de XSS: refletido e armazenado. O XSS refletido ocorre quando o código malicioso é injetado numa página web através de uma entrada do usuário, enquanto o XSS armazenado ocorre quando o código malicioso é armazenado em uma página web e é executado sempre que alguém acessa a página.

Ao analisar os problemas que esta vulnerabilidade pode causar, calculamos o CVSS desta forma: Attack Vector (AV): esta vulnerabilidade é explorável remotamente; Attack Complexity (AC): não existem condições de acesso especializadas ou circunstâncias atenuantes; Privileges Required (PR): não requer acesso a configurações ou arquivos para realizar um ataque; User Interaction (UI): exige que o usuário execute alguma ação antes que a vulnerabilidade possa ser explorada; Scope (S): o componente vulnerável e o componente impactado são diferentes; Confidentiality (C): há alguma perda de confidencialidade; Integrity (I): a modificação dos dados é possível, mas o invasor não tem controle sobre as consequências de uma modificação; Availability (A): há desempenho reduzido ou interrupções na disponibilidade de recursos.

Para testar esta vulnerabilidade, decidimos fazer um XSS refletido onde inserimos na barra de pesquisa 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 1 - Exemplo de como foi feito o ataque de Cross Site Scripting

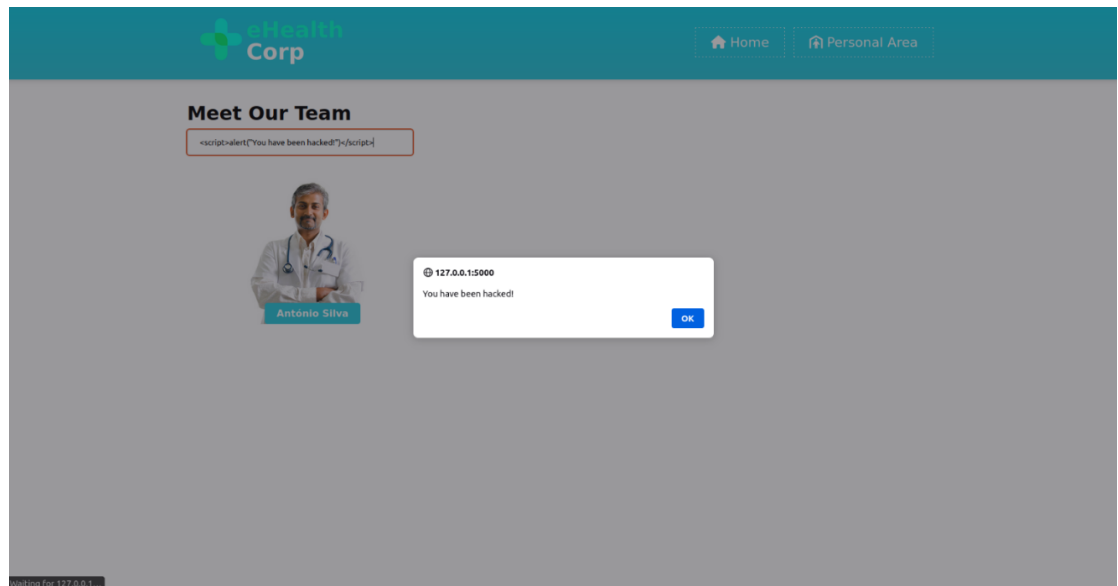


figura 2 - resultado do ataque Cross Site Scripting

A biblioteca Jinja2, que é usada como mecanismo de modelagem de template em Flask, tem suporte built-in para fazer *escaping* automático de caracteres especiais em variáveis exibidas em templates. Isso significa que, ao usar a sintaxe correta para exibir uma variável num template, os caracteres que podem ser interpretados como código malicioso serão automaticamente convertidos em entidades HTML, o que os torna inofensivos.

Para evitar o XSS, é importante garantir que todas as entradas do usuário sejam *escaped* corretamente antes de serem exibidas em uma página web. Isso inclui tanto dados enviados através de formulários quanto dados enviados via query strings ou dados de cabeçalho HTTP.

Em Flask, pode-se usar o objeto request para obter informações sobre a solicitação do usuário. Para evitar o XSS, é importante usar os métodos fornecidos pelo objeto request para acessar os dados da solicitação e garantir que eles sejam *escaped* corretamente antes de serem usados.

Além disso, é importante usar a biblioteca de validação de input, como WTForms, para sanitizar inputs e evitar a injeção de códigos maliciosos.

Em resumo, ao usar Flask, deve-se sempre fazer *escaping* corretamente das entradas do usuário e usar métodos fornecidos pelo objeto request para acessar os dados da solicitação, e sempre usar biblioteca de validação de inputs para sanitizar dados.

Tendo em conta toda a informação acima, escrevemos o código:

```
<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 %}
```

A segunda linha é uma chamada para o método `hidden_tag()` do objeto `form`. Ele cria uma tag oculta que é usada para proteger o formulário contra ataques.

A terceira linha é uma chamada para o método `form_field()` da biblioteca WTForms. Ele renderiza um campo de formulário específico, nesse caso, o campo "name" do formulário.

## CWE-521: Weak Password Requirements

CVSS - 7.5(High)

Vector String - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

CWE-521 é uma vulnerabilidade conhecida como "Weak Password Requirements" que ocorre quando um sistema permite que os usuários escolham *passwords* fracas ou fáceis de adivinhar. Isso pode permitir que os atacantes obtenham acesso não autorizado aos dados ou sistemas protegidos por *passwords* fracas, comprometendo a segurança do sistema.

Existem várias maneiras de uma *password* ser considerada fraca, incluindo:

- *Passwords* curtas: *passwords* com menos caracteres são mais fáceis de adivinhar, especialmente se usadas com técnicas de *Brute Force*.
- *Passwords* comuns: *passwords* comuns, como "pass123" ou "123456", são muito fáceis de adivinhar.
- *passwords* sem complexidade: *passwords* sem caracteres especiais, números ou letras maiúsculas são mais fáceis de adivinhar.
- *Passwords* baseadas em informações pessoais: *passwords* baseadas em informações pessoais, como nomes de familiares, datas de nascimento, etc., são mais fáceis de adivinhar.

Com isto, calculamos a CSS desta vulnerabilidade da seguinte forma: Attack Vector (AV): esta vulnerabilidade é explorável remotamente; Attack Complexity (AC): não existem condições de acesso especializadas ou circunstâncias atenuantes; Privileges Required (PR): não requer acesso a configurações ou arquivos para realizar um ataque; User Interaction (UI): o sistema pode ser explorado sem qualquer interação de qualquer usuário; Scope (S): o componente vulnerável e o componente impactado são os mesmos; Confidentiality (C): o invasor é capaz de negar totalmente o acesso aos recursos do componente afetado; Integrity (I): não há perda de integridade com um ataque; Availability (A): não há impacto na disponibilidade do componente impactado.

Para prevenir a vulnerabilidade CWE-521, é importante estabelecer políticas de *passwords* seguras e garantir que os usuários sejam forçados a cumprir essas políticas. Algumas dicas para políticas de *passwords* seguras incluem:

- Exigir *passwords* com um comprimento mínimo de, pelo menos, 8 caracteres.
- Incentivar o uso de caracteres especiais, números e letras maiúsculas.
- Proibir *passwords* comuns e senhas baseadas em informações pessoais.
- Proibir o reutilização de *passwords* antigas.
- Usar ferramentas de verificação de *passwords* para avaliar a força de uma senha em tempo real quando o usuário cria ou altera a senha.
- Usar técnicas de criptografia de *password* segura para armazenar *passwords* no sistema.

É importante lembrar que, mesmo que as políticas de *password* sejam seguras, os usuários podem ainda escolher senhas fracas. Portanto, é importante monitorar e detectar tentativas de adivinhação de *passwords* para detectar e bloquear ataques de *Brute Force*.

Na nossa aplicação, definimos as políticas de *passwords* seguras: mínimo de 8 caracteres, deve conter pelo menos um número e deve conter pelo menos uma letra maiúscula. Todas estas políticas foram implementadas nas linhas de código abaixo:

```
def password_check(form, field):
    password = form.password.data
    if len(password) < 4:
        raise ValidationError('Password must be at least 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 say')))
    email = StringField('Email', validators = [InputRequired(),
Email(message = 'Invalid email'), Length(max = 50)])
```

```
full_name = StringField('Full name', validators =
[InputRequired(), Length(min = 3, max = 50)])
username = StringField('Username', validators =
[InputRequired(), Length(min = 4, max = 15)])
password = PasswordField('Password', validators =
[InputRequired(), password_check])
```

Este código define uma função "password\_check" que é usada como um validador para o campo "password" num formulário de registo. A função verifica se a *password* atende a certos requisitos de segurança, como ter pelo menos 8 caracteres, conter pelo menos um número e ter pelo menos uma letra maiúscula. Se qualquer um desses requisitos não for atendido, a função lança uma exceção "ValidationError" com uma mensagem de erro específica.

A classe "RegisterForm" é então definida, que herda de "FlaskForm" e tem vários campos, incluindo "gender", "email", "full\_name", "username" e "password". O campo "password" tem a função "password\_check" como um dos seus validadores, o que significa que a função será chamada sempre que o formulário for validado e, se algum erro de validação for encontrado, será exibida a mensagem de erro correspondente.

## CWE-89: SQL Injection

CVSS - 9.8(Critical)

Vector String - CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CWE-89 é uma vulnerabilidade de injeção de SQL, que ocorre quando um atacante consegue inserir comandos SQL maliciosos numa aplicação vulnerável, permitindo que eles alterem a base de dados de forma não autorizada. Isso pode permitir ao atacante acessar, modificar ou excluir dados confidenciais, ou até mesmo assumir o controle total da base de dados.

Para o cálculo do CSS desta vulnerabilidade foi analisado da seguinte forma: Attack Vector (AV): esta vulnerabilidade é explorável remotamente; Attack Complexity (AC): não existem condições de acesso especializadas ou circunstâncias atenuantes; Privileges Required (PR): não requer acesso a configurações ou arquivos para realizar um ataque; User Interaction (UI): o sistema pode ser explorado sem qualquer interação de qualquer usuário; Scope (S): o componente vulnerável e o componente impactado são os mesmos; Confidentiality (C): pode haver perda total de confidencialidade, resultando na divulgação de todos os recursos do componente afetado ao invasor; Integrity (I): o invasor pode modificar qualquer/todos os arquivos protegidos pelo componente afetado; Availability (A): o invasor é capaz de negar totalmente o acesso aos recursos do componente afetado.

A injeção de SQL geralmente ocorre quando dados não confiáveis são inseridos em uma consulta SQL sem uma validação adequada ou *escaping*. Por exemplo, se uma aplicação pede ao usuário para digitar o nome de um usuário e usa esses dados sem fazer *escaping* como parte de uma consulta SQL, um atacante pode inserir comandos maliciosos, como `''; DROP TABLE users; --`, para excluir a tabela de usuários ou `'' or 1=1; --`, para, numa página de *Login*, conseguir entrar sem ter uma conta.

**eHealth Corp**

[Home](#) [Personal Area](#)

**LOG IN IS REQUIRED:**

username

\* or test: -

password

[Log In](#)

**YOU MUST HAVE AN ACCOUNT TO ENTRY PERSONAL AREA**

[Sign Up](#)

figura 3 - Exemplo de uma SQL Injection

Para evitar a injeção de SQL, é importante usar bibliotecas de acesso a bases de dados que forneçam mecanismos de *escaping* automático e evitar a construção de consultas SQL dinamicamente com dados não confiáveis. Também é importante validar todos os dados de entrada e limitar os privilégios de acesso ao banco de dados de acordo com as necessidades da aplicação.

O código apresentado abaixo mostra o que foi feito para combater a vulnerabilidade. A função `login()` verifica se o formulário foi submetido e validado usando o método `"validate_on_submit()"`. Se for válido, é procurado um usuário na base de dados com o nome de usuário especificado no formulário, senão a página de login é renderizada novamente com o formulário preenchido.

Se um usuário for encontrado, ele armazena a *password* do mesmo numa variável `"password"`. Esta é verificada através da função `check_password_hash(password, form.password.data)` e se for correta, o usuário é logado usando a função `login_user(user, remember =`



`form.remember.data`) e é redirecionado para a página "dashboard". Caso contrário, é mostrada uma mensagem de erro "*Invalid username or password*".

A função `check_password_hash` é provida pelo pacote "werkzeug" que é usado pelo Flask, a função é utilizada para comparar a *password* criptografada com a *password* digitada pelo usuário, se as duas *passwords* forem iguais, a função retorna True, caso contrário, retorna False.

A função `login_user` é provida pelo pacote "*flask\_login*" e é usada para logar o usuário, e o parâmetro "remember" permite que o usuário seja lembrado na próxima vez que acessar o sistema.

```
@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)
```

A função acima combate uma SQL injection utilizando o ORM (Object-Relational Mapping) do Flask-SQLAlchemy. Em vez de construir uma string SQL manualmente e passá-la para a base de dados, ela utiliza o método `filter_by()` do SQLAlchemy para filtrar os dados. Este cria uma cláusula WHERE que compara uma coluna com um valor específico, sem permitir que dados maliciosos sejam inseridos na string SQL. Dessa forma, evita-se a possibilidade de injeção de código malicioso na base de dados.

## CWE-20: Improper Input Validation

CVSS - 5.5(Moderate)

Vector String - CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H

CWE-20 é uma categoria de vulnerabilidade de segurança conhecida como "Improper Input Validation". Significa que a aplicação não valida adequadamente as entradas que recebe de fontes externas, como o usuário ou outras aplicações. Isso permite que os invasores possam fornecer dados maliciosos como entrada, com o objetivo de comprometer a segurança da aplicação.

Estes ataques podem incluir, entre outros:

- Injeção SQL: onde os invasores fornecem entradas maliciosas que são usadas para construir comandos SQL, permitindo que eles acessem ou alterem dados na base de dados
- Injeção de script: onde os invasores fornecem entradas maliciosas que são executadas como código no lado do cliente, permitindo que eles acessem informações confidenciais ou realizem ações maliciosas no navegador do usuário
- Injeção de comandos: onde os invasores fornecem entradas maliciosas que são executadas como comandos no sistema operacional, permitindo que eles obtenham acesso de administrador ou alterem configurações críticas.

O vetor do cálculo do CVSS foi feito desta forma: Attack Vector (AV):o componente vulnerável não está vinculado à pilha de rede e o caminho do invasor é por meio de recursos de gravação/execução; Attack Complexity (AC): não existem condições de acesso especializadas ou circunstâncias atenuantes; Privileges Required (PR): o invasor é autorizado com privilégios que fornecem recursos básicos de usuário que normalmente podem afetar apenas configurações e arquivos pertencentes a um do utilizador; User Interaction (UI): o sistema pode ser explorado sem qualquer interação de qualquer usuário; Scope (S): o componente vulnerável e o componente impactado são os mesmos; Confidentiality (C): não há perda de

confidencialidade dentro do componente afetado; Integrity (I): não há perda de integridade no componente afetado; Availability (A): o invasor é capaz de negar totalmente o acesso aos recursos do componente afetado.

Para combater a vulnerabilidade CWE-20, é necessário validar todas as entradas, garantindo que elas estejam no formato esperado e que não contenham caracteres ou sequências maliciosas. Além disso, deve-se sanitizar as entradas, removendo ou codificando caracteres maliciosos, antes de usá-las.

As funções abaixo estão a utilizar validações de *input* para prevenir a vulnerabilidade descrita acima. Essas validações são aplicadas nos campos "name", "email", "message", "username" e "password" do formulário. Algumas das validações incluem:

- *InputRequired()*: garante que o campo não está vazio
- *Length(min = 4, max = 30)*: verifica se o tamanho do *input* está dentro do limite especificado
- *Email(message = 'Invalid email')*: verifica se o input é um email válido
- *BooleanField('remember me')*: verifica se o checkbox está marcado ou não.

Essas validações ajudam a garantir que os inputs fornecidos pelo usuário são válidos e estão dentro do limite esperado, o que diminui a possibilidade de ataques como SQL Injection, XSS e outros ataques que podem ser causados por inputs maliciosos.

```
class TicketForm(FlaskForm):
    name = StringField('Name', validators = [InputRequired(),
Length(min = 4, max = 30)])

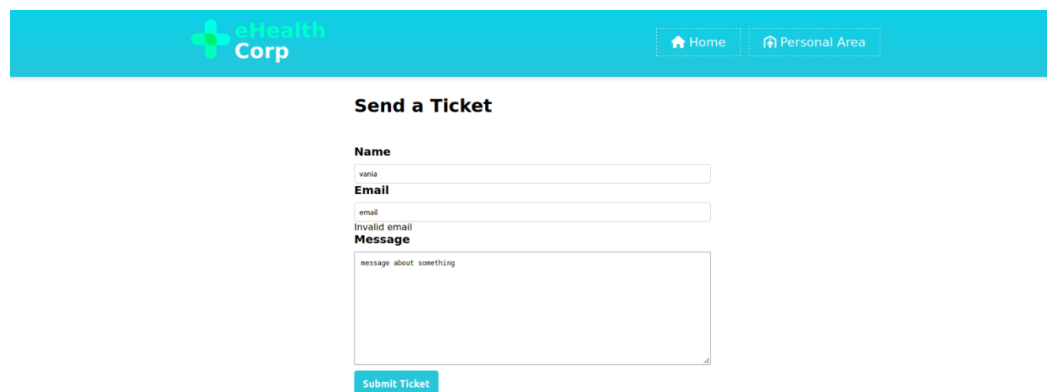
    email = StringField('Email', validators = [InputRequired(),
Email(message = 'Invalid email'), Length(max = 50)])

    message = TextAreaField('Message', validators =
[InputRequired(), Length(min = 4, max = 300)], )

class LoginForm(FlaskForm):
```

```
username = StringField('username', validators =  
[InputRequired(), Length(min = 4, max = 20)])  
  
password = PasswordField('password', validators =  
[InputRequired(), Length(min = 8, max = 80)])  
  
remember = BooleanField('remember me')
```

Para testar se a validação era feita de forma correta, entramos na página “*Send a Ticket*” e tentamos enviar uma mensagem com um email não valido.



The screenshot shows the 'Send a Ticket' form in the eHealth Corp system. The form is located on a page with a blue header containing the eHealth Corp logo and navigation links for 'Home' and 'Personal Area'. The form itself has a title 'Send a Ticket' and three main input fields: 'Name', 'Email', and 'Message'. The 'Name' field contains the text 'vanto'. The 'Email' field contains the text 'email' and displays an error message 'Invalid email' below it. The 'Message' field is a large text area containing the text 'message about something'. At the bottom of the form is a blue button labeled 'Submit Ticket'.

figura 3 - Exemplo de uma SQL Injection

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

CVSS - 3.5(Low)

Vector String - CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:N/A:N

CWE-200 é uma vulnerabilidade de segurança que ocorre quando informações confidenciais ou sensíveis são expostas a atores não autorizados. Essa vulnerabilidade pode ocorrer devido a uma variedade de razões, como configurações de segurança inadequadas, falhas de design de software, erros de programação, ou ausência de medidas de segurança para proteger as informações. O utilizador pode tirar proveito desta vulnerabilidade para obter dados relativos à base de dados, o estado atual da aplicação e muitas outras finalidades.

Um dos exemplos mais comuns é ,que através das mensagens, o atacante pode aproveitar um email válido de alguma conta e, por exemplo, fazer um ataque *Brute Force* usando uma abordagem de tentativa e erro e esperando que, em algum momento, seja possível descobrir a password da conta em específico.

Calculamos o CVSS desta vulnerabilidade com as seguintes considerações: Attack Vector (AV): a vulnerabilidade explorável tem acesso à rede; Attack Complexity (AC): não existem condições de acesso especializadas ou circunstâncias atenuantes; Privileges Required (PR): o invasor é autorizado com privilégios que fornecem recursos básicos de usuário que normalmente podem afetar apenas configurações e arquivos pertencentes a um do utilizador; User Interaction (UI): exige que o usuário execute alguma ação antes que a vulnerabilidade possa ser explorada; Scope (S): o componente vulnerável e o componente impactado são os mesmos; Confidentiality (C): há alguma perda de confidencialidade,o acesso a algumas informações restritas é obtido, mas o invasor não tem controle sobre quais informações é obtido; Integrity (I): não há perda de integridade no componente afetado; Availability (A): não há impacto na disponibilidade no componente afetado.

Na nossa aplicação, ao fazer o login, quando o email está errado, aparece a mensagem de erro “Invalid username or password” ao utilizador, mas quando o utilizador erra apenas a password, aparece uma mensagem de erro a dizer “Invalid password”.

Para prevenir esta vulnerabilidade, alterou-se no código para a mensagem de erro da password ser a mesma que do email, mesmo estando só a password errada (“Invalid username or password”).

Esta é uma vulnerabilidade que está presente em muitos sites e app, não sendo dada muita importância à mesma.

## Conclusão

Este relatório apresentou uma análise detalhada das vulnerabilidades CWE (Common Weakness Enumeration) presentes no nosso sistema, abordando cada uma individualmente, fazendo a sua análise, demonstração, explicando a razão da sua existência e mostrando os seus CVSS e como chegamos a cada pontuação.

## Referências

- Common Vulnerability Scoring System Version 3.1 Calculator: <https://www.first.org/cvss/calculator/3.1>
- Common Weakness Enumeration: <https://cwe.mitre.org/index.html>
- Utilities - Werkzeug: <https://werkzeug.palletsprojects.com/en/1.0.x/utils/>
- Flask Documentation: <https://flask.palletsprojects.com/en/2.2.x/>
- WTForms Documentation: <https://wtforms.readthedocs.io/en/3.0.x/>
- Jinja Documentation: <https://jinja.palletsprojects.com/en/3.1.x/>