



universidade de aveiro  
theoria poiesis praxis

# Project 1 – eHealthCorp

## Segurança Informática e nas Organizações

2022/2023

### Licenciatura em Engenharia Informática

Prof. João Paulo Barraca

Prof. André Zúquete

Prof. Paulo Bartolomeu

Prof. Alfredo Matos

Entregue em 17 de novembro de 2022

#### Trabalho Elaborado Por:

André Butuc (103530)

Artur Correia (102477)

Bruna Simões (103453)

Daniel Carvalho (77036)

Grupo 25

## Índice

Índice .....	2
1. Introdução .....	3
2. CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') .....	4
2.1. Explorações da Vulnerabilidade .....	4
2.2. Resolução da Vulnerabilidade: .....	7
3. CWE- 79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') .....	9
3.1. Explorações da Vulnerabilidade .....	9
3.2. Resolução da Vulnerabilidade .....	12
4. CWE-20: Improper Input Validation .....	14
4.1. Explorações da Vulnerabilidade .....	14
4.2. Resolução da Vulnerabilidade .....	15
5. CWE-311: Missing Encryption of Sensitive Data .....	17
5.1. Explorações da Vulnerabilidade .....	17
5.2. Resolução da Vulnerabilidade: .....	18
6. CWE-598: Use of GET Request Method With Sensitive Query Strings + CWE-200: Exposure of Sensitive Information to an Unauthorized Actor .....	20
6.1. Explorações das Vulnerabilidades .....	20
6.2. Resolução das Vulnerabilidades: .....	22
7. CWE-307: Improper Restriction of Excessive Authentication Attempts .....	26
7.1. Exploração da Vulnerabilidade .....	26
7.2. Resolução da Vulnerabilidade: .....	27
8. CWE-522: Insufficiently Protected Credentials .....	31
8.1. Exploração da Vulnerabilidade .....	31
8.2. Resolução da Vulnerabilidade .....	32
9. CWE-434: Unrestricted Upload of File with Dangerous Type .....	36
9.1. Exploração da Vulnerabilidade .....	36
9.2. Resolução da Vulnerabilidade .....	37
10. CWE-552: Files or Directories Accessible to External Parties .....	39
10.1. Exploração da Vulnerabilidade .....	39
10.2. Resolução da Vulnerabilidade .....	41
11. Conclusão .....	43
12. Referências Bibliográficas .....	44

## 1. Introdução

Neste trabalho foi proposta a exploração de várias vulnerabilidades num website de uma empresa de gerenciamento de pacientes e médicos, ehealthCorp, com o objetivo de explorar as causas e os efeitos que estas inseguranças têm no bom funcionamento e desenvolvimento de uma aplicação, bem como as metodologias e as boas práticas que devem ser aplicadas para evitar a sua ocorrência.

Com estes objetivos em mente, foram exploradas 10 vulnerabilidades diferentes numa versão insegura do website, sendo essas mesmo corrigidas numa versão semelhante, mas mais segura, do projeto. Ambas as aplicações foram desenvolvidas com recurso à framework web Flask, escrita em Python, utilizada tanto para a implementação do frontend como backend, com a base de dados relacional a ser implementada com recurso ao MySQL.

Neste relatório serão então, posteriormente, explicadas as vulnerabilidades encontradas, bem como as soluções implementadas para as corrigir.

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

### CVSS Estimado: 7.4

Esta vulnerabilidade ocorre quando a aplicação desenvolvida apresenta inputs específicos que são processados pela base de dados como um comando SQL, mas sem neutralizar ou parametrizar elementos especiais que podem modificar o intuito original do comando SQL, podendo portanto devolver um resultado anormal que poderá comprometer a integridade e estabilidade da base de dados, dando acesso impróprio aos dados da mesma (por exemplo, no caso da aplicação desenvolvida, o acesso indevido à página do utilizador/médico).

Deste modo, esta vulnerabilidade poderá ter várias consequências nefastas para a normal execução do sistema, permitindo, por exemplo, aceder aos dados de um utilizador, inclusive a muitas das suas informações confidenciais que poderão colocar em questão a integridade da identidade do utilizador, e aceder a outras plataformas através das informações encontradas na página de entrada do utilizador.

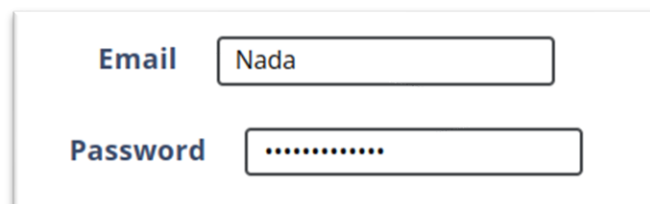
É também possível, através da exploração desta vulnerabilidade, executar comandos que revelem a estrutura da base de dados utilizada na aplicação, permitindo a rutura desta.

Torna-se assim extremamente necessário a remoção desta vulnerabilidade, uma vez que é bastante simples para um individuo com um conhecimento básico de bases de dados relacionais aceder aos dados de uma aplicação que não lida com a mesma.

### 2.1. Explorações da Vulnerabilidade

Esta vulnerabilidade pode ser encontrada:

→ Na página de login:



The image shows a login form with two input fields. The first field is labeled 'Email' and contains the text 'Nada'. The second field is labeled 'Password' and contains a series of dots, indicating a masked password.

Pode ser feita SQL Injection sendo o e-mail uma string qualquer (neste caso, equivale a “Nada”) e a password equivalente à string `' OR 1=1-- //`.

```
cursor.execute("SELECT ID, Email, Password FROM Utilizador WHERE Email ='" \
+ str(params_dict["email"]) + \
"' AND Password='" \
+ str(params_dict["password"]) + "'")
```

Assim sendo, o query SQL predominante será sobreposto, sendo interpretado ao invés o comando:

**SELECT ID, Email, Password FROM Utilizador WHERE Email = 'Nada' AND Password = '' OR 1=1 --//**

Deste modo, todo o código SQL que virá a seguir será comentado (através do uso de `--//` que corresponde a notação que permite comentar em MySQL). Adicionalmente, o utilizador encontrado pelo query corresponderá ao utilizador que consta em primeiro na tabela de Utilizador.

```
Nada
' OR 1=1-- //
(1, 'art.afo@ua.pt', '1904')
We are in.
1do (através do uso
```

Da mesma forma, será possível efetuar SQL injection através da atribuição da string `' OR 1=1-- //` no input do email, sendo a password uma string aleatória (não é relevante o seu valor pois todo o query depois do input do email será ignorado devido à inserção do comentário SQL `--//`).

Email

Password

Query executado na base de dados corresponde a:

**SELECT ID, Email, Password FROM Utilizador WHERE Email = '' OR 1=1-- // AND Password= 'N'**

Onde todo o resto (incluindo a string `AND Password= 'N'`) será ignorado.

```
' OR 1=1-- //
N
(1, 'art.afo@ua.pt', '1904')
We are in.
```

Resultado para ambos os casos anteriores:

Welcome, Artur  
Here's everything you need to know:

**Your information:**  
 Name: Artur Correia  
 Age: 20  
 E-mail: art.afo@ua.pt  
 NIF: 262190185  
 Nr utente: 273134778  
 Phone number: 930577403  
 Address: Quinta do Bosque, Lote 110, 2ªEsq

**Check Prescription:**  
 Please insert the prescription code:  
 Prescription code:   
 SUBMIT

**Check Exam:**  
 Please insert the exam code:  
 Exam code:   
 SUBMIT

**Upcoming Appointments**

DATE	HOUR	DOCTOR	ESPECIALIDADE
2022-11-15	00:00:00	Ana Filipa Gomes	Medicina Interna
2022-11-25	16:30:00	Antero Lobo	Ortopedia

Para ganhar **acesso a um paciente/médico em específico**, basta colocar no email o input `<email na base de dados> ' -- //`

Email

Password

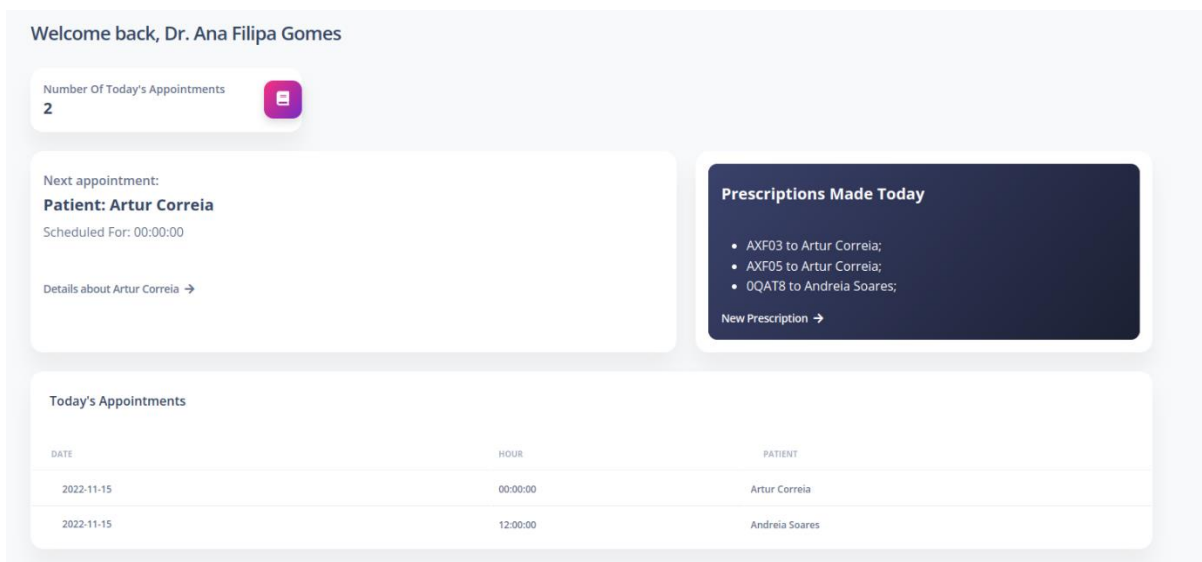
Onde a password será um input qualquer, uma vez que o script MySQL irá ignorar tudo o que está a seguir da notação `--//`. O query então será executado da seguinte maneira:

**SELECT ID, Email, Password FROM Utilizador WHERE Email = 'afgomes@mail.pt' -- //**

O que irá procurar na tabela de utilizadores o utilizador com o email correspondente sem necessitar de mais requisitos (o input da password não terá qualquer efeito na query pois está comentado).

```
afgomes@mail.pt' -- //
n
(2, 'afgomes@mail.pt', '1234')
We are in.
```

Resultado do query inserido:



Concluindo, através do email da vítima é possível ultrapassar a barreira de login, conseguindo facilmente aceder à página do utilizador pretendido sem o seu consento.

## 2.2. Resolução da Vulnerabilidade:

→ App insegura:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    params_dict = {"email": "", "password": "", "id": ""}

    if request.method == 'GET' and len(request.args) > 0:
        params_dict["email"] = request.args['email']
        params_dict["password"] = request.args['password']
        print(params_dict)
        # Buscar email e pass à base de dados
        cursor = db.cursor(buffered=True)
        print(params_dict["email"])
        print(params_dict["password"])
        cursor.execute("SELECT ID, Email, Password FROM Utilizador WHERE Email ='" \
            + str(params_dict["email"]) + \
            "' AND Password='" \
            + str(params_dict["password"]) + "'")
        user_data = cursor.fetchone()
```

Neste caso, a execução da *query* através do cursor da base de dados é feita com concatenação de strings, sendo concatenado diretamente nesta informação proveniente dos formulários recebidos. Esta metodologia para execução das *queries* é aplicada em todos os acessos à base de dados realizados na aplicação, o que implica que várias funcionalidades desta estejam vulneráveis à execução de SQL injection.

Como exemplo, apresenta-se o código para o login, na qual se verifica a inserção direta do email e password, sendo por isso possível dar override ao código e explorar dados que deveriam ser inacessíveis.

→ App segura:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    params_dict = {"email": "", "password": "", "id": ""}

    if request.method == 'POST':
        params_dict["email"] = request.form['email']
        params_dict["password"] = request.form['password']

        # Buscar email e pass à base de dados
        cursor = db.cursor()

        cursor.execute("SELECT ID, Email, Password FROM Utilizador WHERE Email = %s", (params_dict["email"],))
        user_data = cursor.fetchone()
```

Na versão segura da aplicação, todas as *queries* de acesso à base de dados são realizadas com parametrização, ao invés da concatenação direta de strings. Para tal, os campos recebidos do formulário são passados dentro de um tuplo, como segundo parâmetro do método `.execute()` do cursor do pacote `mysql.connector`.

Como exemplo, apresenta-se na figura anterior a realização de `login()`, na qual a aplicação irá procurar primeiro na base de dados o utilizador com um dado email inserido e procura o ID e a password desse mesmo email. Para além disso, o método utilizado na recolha de informação da versão segura corresponde ao método POST, que não armazena os dados em memória (contrário do que ocorre no método GET).

Caso não tenha encontrado nenhum utilizador com esse email, ou caso a palavra-passe esteja errada, é desencadeado um método de limitação do número de tentativas de autenticação, discutido mais à frente no relatório (**Página X**). É apresentada uma mensagem ao utilizador de que as credenciais de acesso não estão corretas.

Assim sendo, a tentativa de executar SQL Injection na página segura, ao contrário da aplicação insegura, é não sucedida, pelo que é contabilizada como uma tentativa de login incorreta.

✖ Email or password incorrect

### Login

Email

Password

Email or password incorrect

[Forgot password?](#)

or



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

#### CVSS Estimado: 7.0

Ocorre quando se torna possível a um indivíduo sem privilégios modificar a composição do sistema, devido ao facto do software não neutralizar corretamente o input disponível ao utilizador, inserindo código impróprio que corre de forma não desejada no código HTML, sendo, portanto, sempre executado em qualquer dispositivo e apresentado a outros utilizadores quando a página web é carregada, enquanto não for removido.

Existem 3 tipos de XSS: não persistente, persistente ou DOM-based.

No caso de XSS não persistente o servidor lê a informação do request HTTP e reflete o mesmo na HTTP response. Como consequência um atacante pode fazer com que uma vítima execute, sem o seu conhecimento, conteúdo perigoso. Deste modo, é possível ser alvo de phishing pois ao executar um site perigoso pode fornecer ao mesmo informações que depois poderão ser utilizadas pelos mesmos.

No caso de Stored XSS (Persistente), a aplicação guarda informação perigosa na base de dados e a mesma é incluída na página onde é consultada. Assim sendo, cada vez que os dados corrompidos forem lidos da base de dados e integrados numa página HTML, serão executados todos os parâmetros HTML que estiverem inscritos na mesma.

Já em DOM-based XSS o cliente é que cria a injeção de XSS para a página. Existem também outros tipos em que o servidor é responsável por fazer essa injeção. Se o servidor enviar um script com informação e voltar a injetar a mesma na página web, então XSS DOM-based torna-se possível.

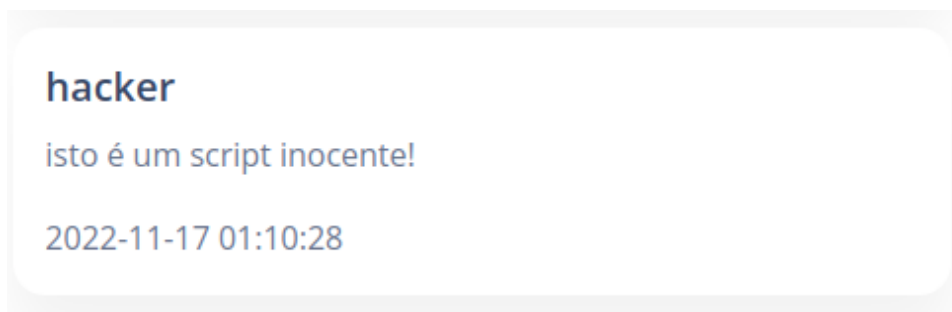
Desta forma, é fundamental tratar a ocorrência de vulnerabilidades deste tipo, de modo a ser impossível para qualquer indivíduo modificar a estrutura do HTML de modo a incorporar scripts ou outros elementos HTML indesejados.

#### 3.1. Explorações da Vulnerabilidade

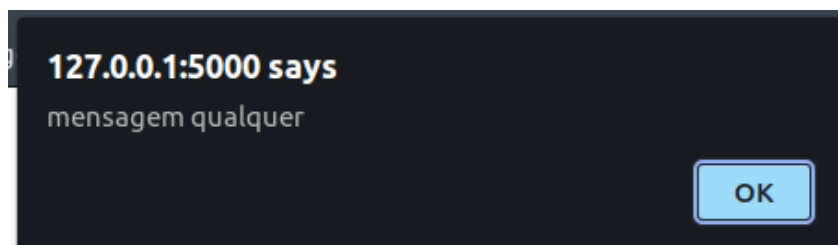
Esta vulnerabilidade pode ser encontrada:

→ Na página de reviews:

Ao criar um comentário na página de reviews, com um nome qualquer e com o corpo do comentário igual a um comentário qualquer, pode-se adicionar qualquer script, que este será automaticamente incorporado no HTML final. Por exemplo, adicionando a uma mensagem normal a seguinte frase: **<script>alert("mensagem qualquer")</script>**, a publicação do comentário implica a execução deste pequeno script, sendo que no lado do utilizador só será visível, o seguinte comentário:



No entanto, sempre que der reload à página irá ser executado o script inserido pelo atacante:



Isto ocorre porque o comentário deixa de ser interpretado como texto e passa a ser interpretado como um script HTML. Com o auxílio do Object Inspector no navegador utilizado, é permitido observar com mais facilidade o que ocorre em termos de código:

```
<div class="row"> flex == $0
  <div class="col-12">
    <h5 class="card-title">hacker</h5>
    <p class="card-text">
      " isto é um script inocente! "
      <script>alert("mensagem qualquer")</script>
    </p>
    <p class="card-footer p-0 m-0"> 2022-11-17 01:10:28</p>
  </div>
```

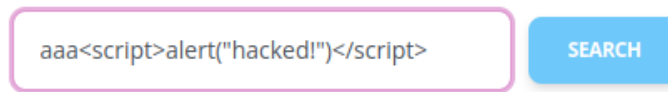
Assim, é notável a inserção do script criado dentro da mensagem escrita pelo utilizador com o título de "hacker".

Neste caso, os scripts são guardados entro da base de dados, pelo que cada vez que a página for carregada, será introduzido a qualquer utilizador o alerta injetado, tratando-se, portanto, de **XSS persistente**.

Um caso de XSS não persistente ocorre, por exemplo, quando os dados não são armazenados na base de dados:

→ Na página do doctor-dashboard (appointments):

Ao inserir o script na barra de pesquisas de consultas na página do médico, é possível executar automaticamente um script HTML que irá ser executado apenas uma vez nessa página, quando é inserido na barra de pesquisas:

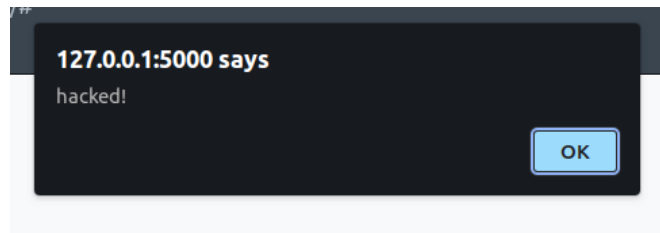


aaa<script>alert("hacked!")</script> SEARCH

Neste caso, o query executado será do tipo:

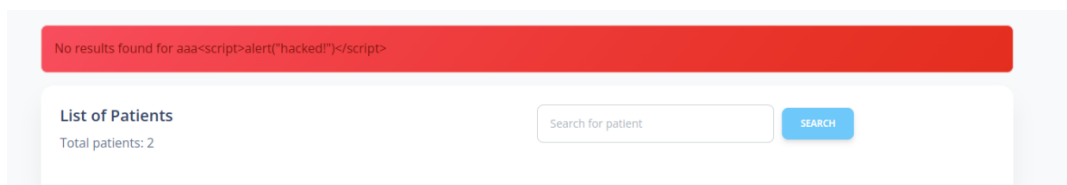
**SELECT ID\_Pac, Nome FROM Med\_Pac JOIN Pac\_User\_View ON ID\_Pac=ID WHERE ID\_Med=" + str(doctor\_id) + " AND Nome LIKE " + str(filter) + ""**

Onde *doctor\_id* não pode ser alterado, e *filter* corresponde ao input recebido pelo programa. Neste caso, a versão insegura irá apresentar o seguinte alerta:



Que será executado apenas uma vez, pois não é guardado na base de dados e corresponde apenas a um parametro de procura de dados.

A versão segura já não apresenta este problema, onde é procurado na base de dados todo o query, sem este ser interpretado como código:



No results found for aaa<script>alert("hacked!")</script>

List of Patients  
Total patients: 2

Search for patient SEARCH

## 3.2. Resolução da Vulnerabilidade

→ App insegura

```
@app.route('/reviews', methods=["GET", "POST"])
def reviews():
    params_dict = {"reviews": [], "total_reviews": 0}
    if request.method == "POST":
        name = request.form.get('name')
        review = request.form.get('review')
        print(name, review)

        if not name or not review:
            flash("Please fill all the fields")
            return redirect(url_for("reviews"))

        else:
            cursor = db.cursor()
            cursor.execute("INSERT INTO Comentario (Autor, Texto) VALUES ('" + str(name) + "', '" + str(review) + "')")
            db.commit()
            cursor.close()
            flash("Review posted successfully")
            return redirect(url_for("reviews"))

    elif request.method == "GET":
        params_dict['reviews'] = []

        cursor = db.cursor()
        cursor.execute("SELECT * FROM Comentario")
        reviews = cursor.fetchall()

        for (ID, Autor, Texto, Data) in reviews:
            params_dict["reviews"].append({"date": Data,
                                            "author": Autor, "id": ID,
                                            "text": Texto})
            params_dict["total_reviews"] += 1

    return render_template('reviews.html', params=params_dict)
```

No código da versão insegura, não é efetuada parametrização das queries SQL, conforme descrito anteriormente para a CWE-89. Assim, o query SQL é feito com a passagem direta de parâmetros, pelo que o que é gravado poderá ser interpretado como código HTML (incluído os ditos scripts) ou até mesmo como queries.

Além disto, nos documentos HTML da versão insegura é utilizado o parâmetro `|safe` na leitura dos comandos recebidos aquando da criação da página pela sua respetiva função no Flask, o que implica que não ocorra *escape* de símbolos perigosos nas entidades HTML geradas, sendo apresentado literalmente o input do utilizador.

```
{% for review in params.reviews %}

    <div class="card card-body mb-3">
        <div class="row">
            <div class="col-12">
                <h5 class="card-title">{{ review.author }}</h5>
                <p class="card-text"> {{ review.text|safe }}</p>
                <p class="card-footer p-0 m-0"> {{ review.date }}</p>
            </div>
        </div>
    </div>

{% endfor %}
```

## → App segura

```

@app.route('/reviews', methods=["GET", "POST"])
def reviews():
    params_dict = {"reviews": [], "total_reviews": 0}
    if request.method == "POST":
        name = request.form.get('name')
        review = request.form.get('review')
        print(name, review)

        if not name or not review:
            flash("Please fill all the fields")
            return redirect(url_for("reviews"))
        else:
            cursor = db.cursor()
            cursor.execute('''
                INSERT INTO Comentario (Autor, Texto)
                VALUES (%s, %s)''',
                            (name, review))

            db.commit()
            cursor.close()
            flash("Review posted successfully")
            return redirect(url_for("reviews"))

    elif request.method == "GET":
        params_dict['reviews'] = []

        cursor = db.cursor()
        cursor.execute("SELECT * FROM Comentario")
        reviews = cursor.fetchall()

        for (ID, Autor, Texto, Data) in reviews:
            params_dict["reviews"].append({"date": Data,
                                           "author": Autor, "id": ID,
                                           "text": Texto})
            params_dict["total_reviews"] += 1

    return render_template('reviews.html', params=params_dict)

```

Já na versão segura, o query SQL é parametrizado de modo que os valores introduzidos não possam modificar o query SQL de modo a introduzir. Assim, não são introduzidos quaisquer elementos HTML (e, por isso, não conseguem introduzir scripts), tornando o site invulnerável a este tipo de ataques.

Assim, aquando da inserção do mesmo input apresentado anteriormente como exemplo para o website inseguro, no website seguro, obtemos o seguinte resultado:

**hacker**

isto é um script inocente!

<script>alert("mensagem qualquer")</script>

2022-11-17 01:10:28

```

<div class="col-12">
  <h5 class="card-title">hacker</h5>
  <p class="card-text"> isto é um script inocente! <script>alert("mensagem
qualquer")</script></p>
  <p class="card-footer p-0 m-0"> 2022-11-17 01:10:28</p>

```

## 4. CWE-20: Improper Input Validation

### CVSS Estimado: 3.5

Esta CWE verifica-se nos casos em que a aplicação recebe input ou dados do utilizador, não validando corretamente se este input tem as propriedades requeridas para processar os dados de forma segura e correta.

Desta forma, um atacante poderá explorar campos de input não protegidos de forma a criar um input não esperado pelo resto da aplicação, que lhe permita alterar o fluxo de controlo desta, controlar arbitrariamente um recurso.

### 4.1. Explorações da Vulnerabilidade

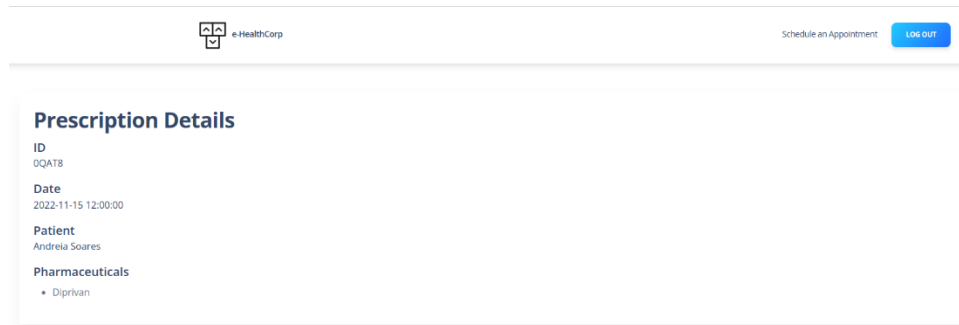
No sistema desenvolvido, verificam-se dois exemplos da ocorrência desta vulnerabilidade na página principal do Paciente. Nesta página, é permitido ao Paciente inserir códigos para aceder a uma prescrição a si passada, ou então para aceder aos resultados das suas análises.

Na versão insegura da aplicação, não é feita a validação do campo input para a obtenção destes códigos, não sendo então verificado se o código corresponde efetivamente a uma prescrição/exame do utilizador em causa, sendo verificado apenas se a prescrição se encontra na base de dados. Deste modo, se o utilizador colocar um código pertencente a outro utilizador, ele vai conseguir aceder às suas prescrições e/ou resultados de exames, estando portanto a aceder a informações sensíveis de outras pessoas, á qual o seu acesso não é autorizado, e podendo eventualmente usar estes códigos para levantar prescrições que não lhe foram a si receitadas.

Como exemplo, o utilizador “Artur” passa um código de prescrição pertencente à paciente “Andreia Soares”, sendo no entanto validada na mesma o input do código.

The screenshot displays the patient dashboard for 'Artur'. It features a 'Your Information' sidebar, a 'Check Prescription' section, and a 'Check Exam' section. The 'Check Prescription' section contains a text input field with the value 'SQAT8' and a 'Submit' button. The 'Check Exam' section contains an empty text input field and a 'Submit' button. The 'Your Information' sidebar lists details for Artur Correia, including age, email, NIF, and address.

Section	Field Label	Field Value	Action
Your Information	Name	Artur Correia	
	Age	20	
Check Prescription	Prescription code	SQAT8	Submit
	Please insert the prescription code:		
Check Exam	Exam code		Submit
	Please insert the exam code:		



## 4.2. Resolução da Vulnerabilidade

### → App insegura

Conforme anteriormente descrito, na função referente ao tratamento da submissão dos formulários anteriores (`patient_prescription_details()` para o caso das Prescrições), apenas é efetuada uma *query* SQL para verificar se o código passado pelo utilizador corresponde a uma entrada da tabela Prescrição na base de dados. Caso corresponda, a informação referente a esta Prescrição é então passada ao template HTML correspondente, e apresentada na página carregada.

```
@app.route('/patient-prescription-details', methods=['GET', 'POST'])
def patient_prescription_details():
    if session.get('user_id') is None:
        flash("You must login to access this page!")
        return redirect(url_for('login'))

    if request.method == 'POST':
        prescription_code = request.form["prescription_code"]
        pharmaceuticals = []

        cursor = db.cursor()
        cursor.execute("SELECT Code, ID_Pac, Data, Cod_Medic FROM Prescricao JOIN Consulta C on C.Num_Cons = Prescricao.Num_Consulta"
                        " WHERE Code = " + str(prescription_code) + ";")

        prescription = cursor.fetchall()

        if len(prescription) == 0:
            flash("There is no prescription with that code")
            cursor.close()
            return redirect(url_for("logged"))

        for (Code, ID_Pac, Data, Cod_Medic) in prescription:
            # Buscar o nome do Paciente
            cursor.execute("SELECT Nome FROM Pac_User_View WHERE ID = " + str(ID_Pac))
            name = cursor.fetchone()

            # Buscar os Farmacêuticos
            cursor.execute("SELECT Nome FROM Medicamento WHERE Codigo = " + str(Cod_Medic))
            pharma = cursor.fetchone()

            pharmaceuticals.append(pharma[0])

        params_dict = {"date": Data,
                      "patient": name[0], "id": prescription_code,
                      "pharmaceuticals": pharmaceuticals}

        cursor.close()
        return render_template('patient-prescription-details.html', params=params_dict)
```

## → App Segura

Por sua vez, na versão segura da aplicação é também verificado na *query* SQL referida se o ID do utilizador associado à prescrição passada é igual ao ID do utilizador autenticado, com recurso ao dicionário Session (populado com o ID do utilizador aquando do login).

```
1 #afartur+1*
2 @app.route('/patient-prescription-details', methods=['GET', 'POST'])
3 def patient_prescription_details():
4     if session.get('user_id') is None:
5         flash("You must login to access this page!")
6         return redirect(url_for('login'))
7
8     if request.method == 'POST':
9         prescription_code = request.form["prescription_code"]
10        pharmaceuticals = []
11
12        cursor = db.cursor()
13        cursor.execute("SELECT Code, ID_Pac, Data, Cod_Medic FROM Prescricao JOIN Consulta C ON C.Num_Cons = Prescricao.Num_Consulta "
14                       "WHERE Code = %s AND ID_Pac = %s", (prescription_code, session['user_id']))
15
16        prescription = cursor.fetchall()
```

Caso se verifique que este não é o caso, é negada a apresentação da prescrição ao utilizador, sendo este notificado através de um aviso que o código por ele introduzido não corresponde a uma Prescrição a ele passada.

The screenshot displays the user interface of the eHealth Corp application. At the top, a red banner contains a "Go Back" link and a message: "You don't have access to a prescription with that code!". Below this, the user is greeted with "Welcome, Artur" and "Here's everything you need to know:". On the left, a dark blue sidebar titled "Your information" lists user details: Name: Artur Correia, Age: 20, E-mail: art.afc@ua.pt, NIF: 262190185, Nr utente: 273134778, Phone number: 930577403, and Address: Quinta do Bosque, Lote 110, 2º Esq. On the right, there are two main sections. The first is "Check Prescription:", which prompts the user to "Please insert the prescription code:" and features a text input field containing "00478" and a blue "Submit" button. The second is "Check Exam:", which prompts the user to "Please insert the exam code:" and features an empty text input field and a blue "Submit" button. At the top right, there is a button labeled "Worried About Your Health?" with a sub-link "Schedule an appointment →".



## 5. CWE-311: Missing Encryption of Sensitive Data

### CVSS Estimado: 7.6

Esta vulnerabilidade deve-se à falta de procedimentos que encriptem dados sensíveis antes do seu armazenamento no sistema, como é, por exemplo, o caso de passwords.

Assim, esta vulnerabilidade em conjunção com a CWE-89 (SQL Injection) permite aceder aos dados sensíveis de um utilizador num formato humanamente legível, o que por sua vez, pode implicar a utilização desses dados para aceder de forma intrusiva a outras contas que o utilizador em questão possui.

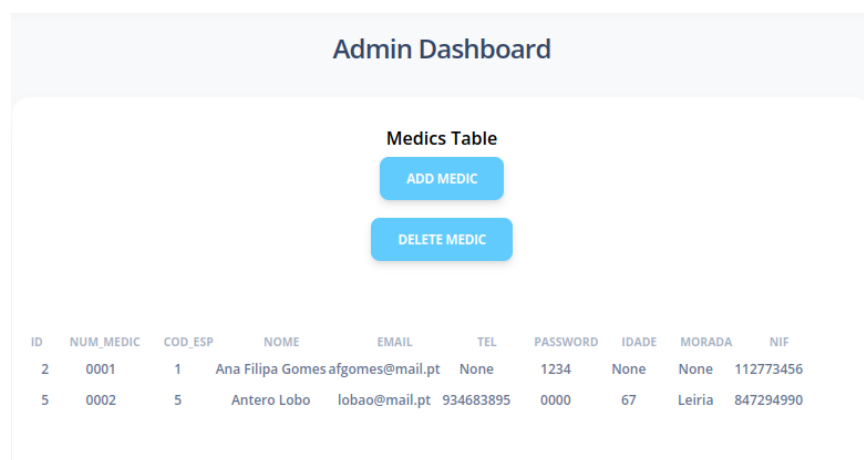
É comum um utilizador não utilizar sempre diferentes passwords para diferentes acessos às suas contas, o que torna crucial arranjar formas de minimizar os possíveis estragos que um acesso intrusivo de uma chave-mestre pode causar.

Uma possível solução a esta vulnerabilidade é codificar qualquer dado sensível que pode ser explorado.

### 5.1. Explorações da Vulnerabilidade

Esta vulnerabilidade pode ser visivelmente encontrada:

→ Na página do Dashboard Admin



Através de comandos de SQL Injection é possível aceder à admin dashboard que apresenta funcionalidades de superutilizador podendo manipular a base de dados e ver o conteúdo das suas tabelas.

Como é possível observar, a coluna de "Password" da tabela dos médicos apresenta as passwords de forma humanamente legível. Face a isto o atacante consegue obter o email do Utilizador e a sua password, podendo depois entrar na conta de um Utilizador específico da eHealthCorp ou então mesmo tentar aceder a outras contas do Utilizador em outros sites.

## 5.2. Resolução da Vulnerabilidade:

### → App insegura:

A vulnerabilidade encontra-se no código da admin-dashboard e na criação de uma nova conta de Paciente, na página Create Account. Tal verifica-se quando é aceite o campo da password do formulário, que é colocado diretamente na base de dados, sem processamento adicional.

```
def createacc():
    if request.method == 'POST':
        form_input = {
            "firstname": request.form['firstname'],
            "lastname": request.form['lastname'],
            "email": request.form['email'],
            "nutente": request.form['nutente'],
            "nif": request.form['nif'],
            "tel": request.form['tel'],
            "morada": request.form['morada'],
            "password": request.form['psw'],
            "confirm_password": request.form['pswc']
        }

        # Verificar se o email já existe
        cursor = db.cursor()
        cursor.execute("SELECT Email FROM Utilizador WHERE Email = '" + str(form_input["email"]) + "'")
        email_data = cursor.fetchone()
        if email_data is not None:
            flash("Email already exists")
            return redirect(url_for('createacc'))

        for key, value in form_input.items():
            if value == "":
                form_input[key] = None

        if form_input["password"] != form_input["confirm_password"]:
            flash("Passwords don't match")
            return redirect(url_for('createacc'))

        cursor.execute(" INSERT INTO Utilizador (Nome, Email, Tel, Password, Idade, Morada, NIF) VALUES ('" + str(form_input["firstname"]) +
            " " + str(form_input["lastname"]) + "', '" + str(form_input["email"]) + "', '" +
            str(form_input["tel"]) + "', '" + str(form_input["password"]) + "', " + str("Null") + "', '" +
            str(form_input["morada"]) + "', '" + str(form_input["nif"]) + "')")
```

### → App segura:

De forma a retificar esta vulnerabilidade foram utilizadas as funções `generate_password_hash()` e `check_password_hash()` do módulo `werkzeug.securit`. A primeira função gera uma hash da string passada como argumento (a password), apresentando como estratégia default “sha256” e a segunda aceita dois argumentos, a hash e uma string e retorna um verdadeiro se a hash fornecida é igual à hash da string fornecida e falso caso contrário.

Logo, aplicando as funções mencionadas, nomeadamente a `generate_password_hash` antes de inserir os dados na base de dados e a função `check_password_hash` aquando da autenticação na página de login, conseguimos codificar os dados e remover a vulnerabilidade.

Na função `create_new_account()`:

```

hashed_pass = generate_password_hash(form_input["password"])
cursor.execute('''
    INSERT INTO Utilizador (Nome, Email, Tel, Password, Idade, Morada, NIF)
    VALUES (%s, %s, %s, %s, %s, %s, %s)'''
    , (
        form_input["firstname"] + " " + form_input["lastname"], form_input["email"], form_input["tel"],
        hashed_pass, None, form_input["morada"], form_input["nif"])

```

Na função login():

```

if user_data is None or not check_password_hash(user_data[2], params_dict["password"]) :
    # Caso o E-Mail ou a Pass estejam errados
    if attempts == 0:
        flash(f'Password is incorrect! You have 4 attempts remaining.')

        cursor.execute('INSERT INTO Login_Attempts (IP, Num_Tentativas, Ult_tentativa) VALUES
            (user_IP, current_time)')
        db.commit()

        cursor.close()
        return redirect(url_for('login'))
    elif attempts < 3:

```

Agora, mesmo com a vulnerabilidade de SQL Injection para aceder à página de administração, o atacante não irá conseguir aceder à informação sensível do Utilizador.

Medics Table											ADD MEDIC	DELETE MEDIC
ID	NUM_MEDIC	COD_ESP	NOME	EMAIL	TEL	PASSWORD	IDADE	MORADA	NIF			
			Ana									
2	0001	1	Filipa	afgomes@mail.pt	None	pbkdf2:sha256:260000\$NK60HwDwYykX1ERw\$8bab0036dfca0d2899bd40f3fcb5e58f98c8979368bbaeb07ebe5d9d53a0a753	None	None	112773456			
			Gomes									
5	0002	5	Antero	lobao@mail.pt	934683895	pbkdf2:sha256:260000\$j0O9XNOsj2qhZt7D\$14ec859cd33c3f77a16070ace280fb783df007c7aa10c2729ec143a5a68149de	67	Leiria	847294990			
			Lobo									

## 6. CWE-598: Use of GET Request Method With Sensitive Query Strings + CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

### CVSS Estimado: 4.3

A CWE-598 ocorre quando uma aplicação web utiliza o método HTTP GET para processar um pedido, incluindo informação sensível na *query* desse pedido.

Por sua vez, a CWE-200 representa situações em que o produto expõe informação sensível a um ator que não está explicitamente autorizado a aceder a essa informação.

As duas vulnerabilidades indicadas coincidiram numa só vulnerabilidade exposta numa funcionalidade da aplicação insegura, o que nos fez ponderar juntá-las.

A utilização de métodos HTTP GET para o processamento de um pedido requer a presença de informação importante nos parâmetros da URL. Isto permite a modificação livre dos parâmetros da URL e consequentemente influencia o pedido HTTP GET.

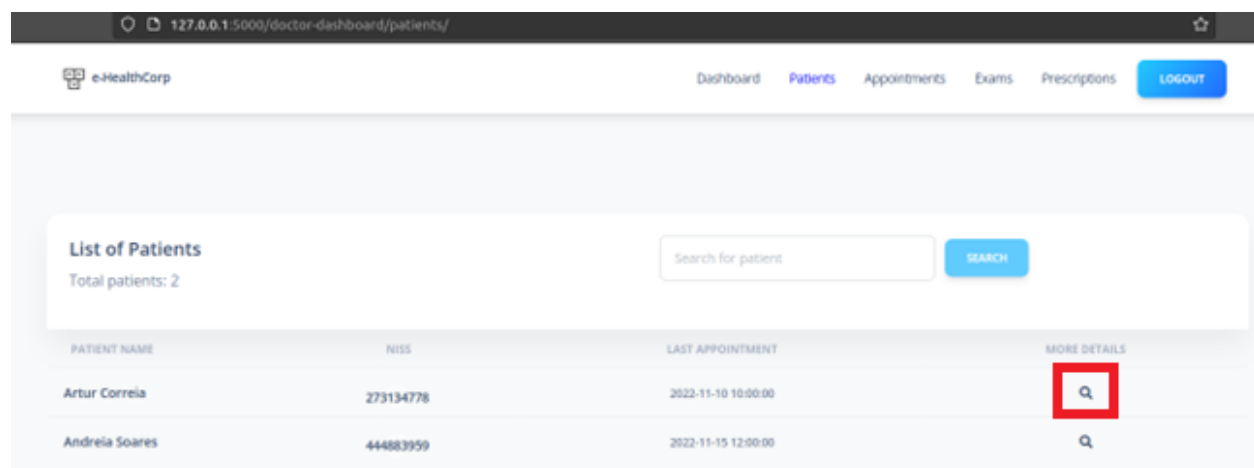
Deste modo, as vulnerabilidades em questão permite o acesso não supervisionado de informação presente na base de dados do sistema, podendo não ter permissões para o fazer, levando a uma clara violação dos direitos de privacidade dos dados de um Utilizador do sistema.

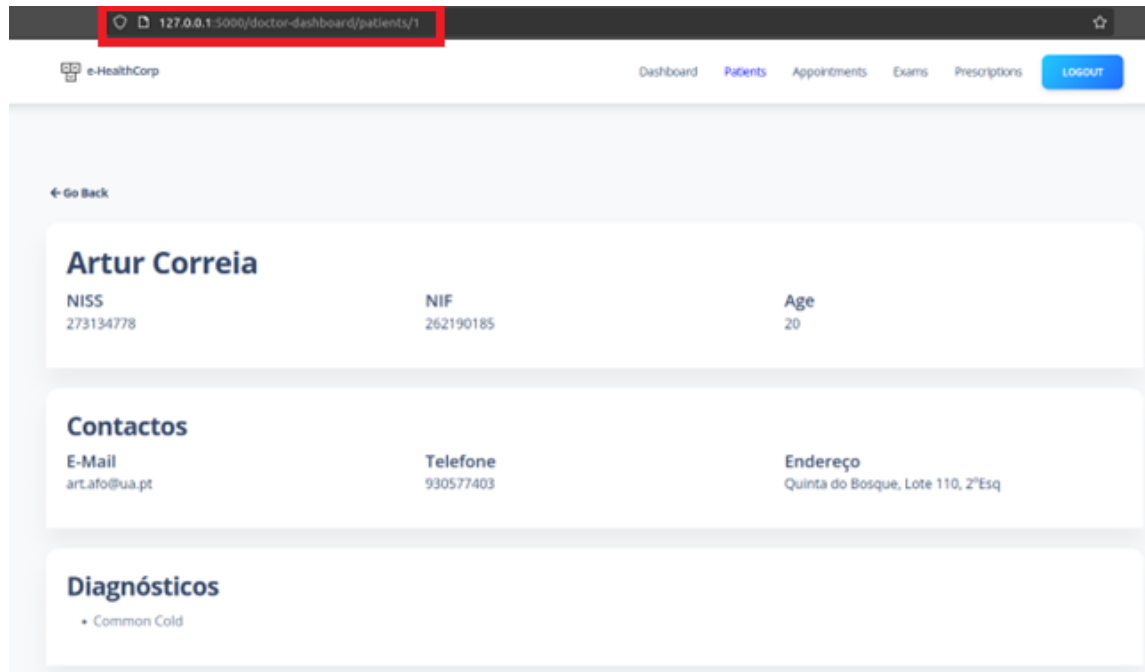
Uma possível solução a esta vulnerabilidade é remover o processamento de dados através de pedidos HTTP GET e usar pedidos HTTP POST.

### 6.1. Explorações das Vulnerabilidades

Esta vulnerabilidade pode ser visivelmente encontrada, por exemplo:

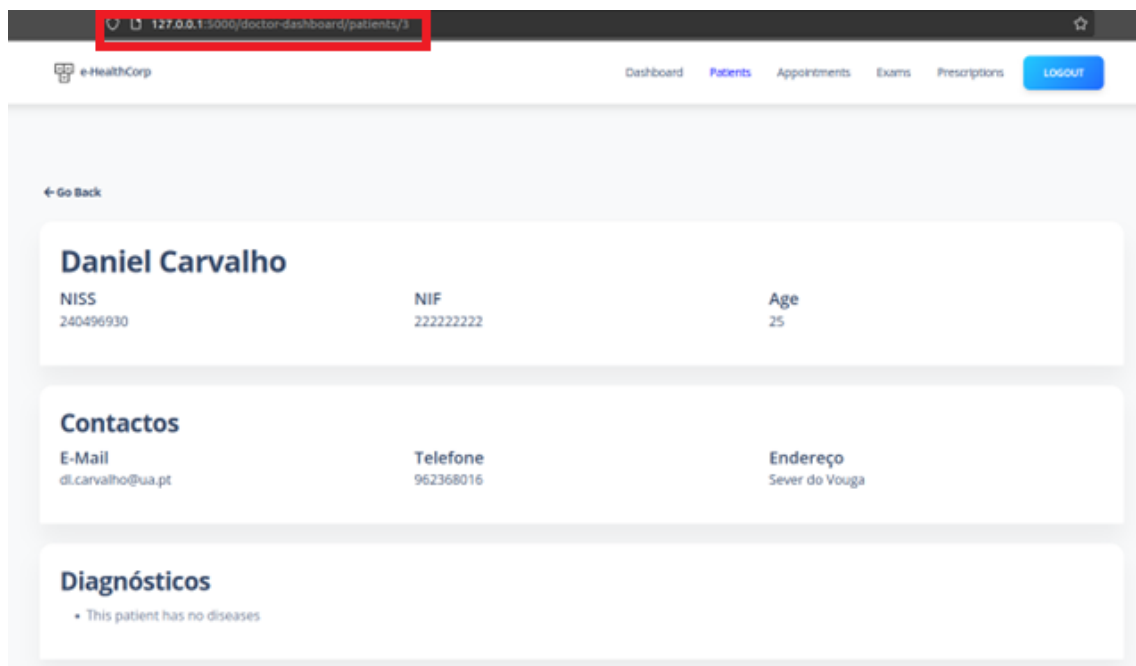
→ Na página patients da dashboard do médico





Podemos observar que ao URL foi adicionado o ID do paciente.

Caso o Médico em questão queira aceder a detalhes de um paciente que não lhe pertence só tem que alterar o ID no URL:



Ao colocar o número 3 no fim do URL, o “Médico” conseguiu aceder ao Paciente “Daniel Carvalho” que não se encontrava inicialmente na sua lista de pacientes (que se pode ver na primeira foto deste capítulo).

## 6.2. Resolução das Vulnerabilidades:

A vulnerabilidade em questão encontra-se presente em todas as páginas de informação ou detalhes sobre um paciente, consulta, exame ou prescrição, presentes na dashboard do médico, pois todas fazem o processamento do pedido através do método HTTP GET.

→ App insegura:

```
def doctor_dashboard_patients():
    if session.get('user_id') is None:
        flash("You must login to access this page!")
        return redirect(url_for('login'))

    params_dict = {"patients": [], "total_patients": 0}
    doctor_id = session["user_id"]

    if request.method == "GET":
        cursor = db.cursor()
        cursor.execute("SELECT ID_Pac FROM Med_Pac WHERE ID_Med=" + str(doctor_id))

        patients_id = cursor.fetchall()

        for ID_Pac in patients_id:
            print("SELECT U.ID, Nome, Num_Utente FROM (Paciente JOIN Utilizador U on U.ID = Paciente.ID) WHERE U.ID=" + str(ID_Pac[0]) + " LIMIT 1")

            cursor.execute("SELECT U.ID, Nome, Num_Utente FROM (Paciente JOIN Utilizador U on U.ID = Paciente.ID) WHERE U.ID=" + str(ID_Pac[0]) + " LIMIT 1")

            patient = cursor.fetchone()
            print(patient)
            cursor.execute("SELECT Data FROM Consulta WHERE ID_Pac=" + str(ID_Pac[0]) + " ORDER BY Data LIMIT 1")

            data = cursor.fetchone()
            print(data)
            if data is not None:
                last_appointment = data[0]
            else:
                last_appointment = "Nenhuma"

            print(last_appointment)
            # Adicionar info ao params_dict
            params_dict["patients"].append({"name": patient[1], "niss": patient[-1], "id": {"_id": ID_Pac[0]}, "last_appointment": last_appointment})
            params_dict["total_patients"] += 1
```

```
<div class="card-body px-0 pb-2">
  <div class="table-responsive">
    <table class="table align-items-center mb-0">
      <thead>
        <tr>
          <th class="text-uppercase text-secondary text-xs font-weight-bolder opacity-7">Patient Name</th>
          <th class="text-uppercase text-secondary text-xs font-weight-bolder opacity-7 ps-2">NISS</th>
          <th class="text-center text-uppercase text-secondary text-xs font-weight-bolder opacity-7">Last appointment</th>
          <th class="text-center text-uppercase text-secondary text-xs font-weight-bolder opacity-7">More details</th>
        </tr>
      </thead>
      <tbody>
        {% for patient in params.patients %}
          <tr>
            <td>
              <div class="d-flex px-2 py-1">
                <div class="d-flex flex-column justify-content-center">
                  <h6 class="mb-0 text-sm">{{ patient.name }}</h6>
                </div>
              </div>
            </td>
            <td>
              <div class="avatar-group mt-2">
                <a href="javascript:;" class="avatar avatar-xs rounded-circle" data-bs-toggle="tooltip" data-bs-placement="bottom" title="Alfredo">
                  <h6 class="mb-0 text-sm">{{ patient.niss }}</h6>
                </a>
              </div>
            </td>
            <td class="align-middle text-center text-sm">
              <span class="text-xs font-weight-bold"> {{ patient.last_appointment }} </span>
            </td>
            <td class="align-middle text-center text-sm">
              <a href="{{ url_for('doctor_dashboard_patient_info', **patient.id) }}"><i class="fa fa-search"></i></a>
            </td>
          </tr>
        {% endfor %}
      </tbody>
    </table>
  </div>
```

Ao carregar inicialmente a página da listagem dos pacientes associados ao médico, os IDs dos pacientes são enviados através de parâmetros para a template que os coloca como argumentos na função “url\_for” que redireciona para a página de informação do paciente com o ID passado na função.

```
def doctor_dashboard_patient_info(_id):
    if session.get('user_id') is None:
        flash("You must login to access this page!")
        return redirect(url_for('login'))

    _id = int(_id)

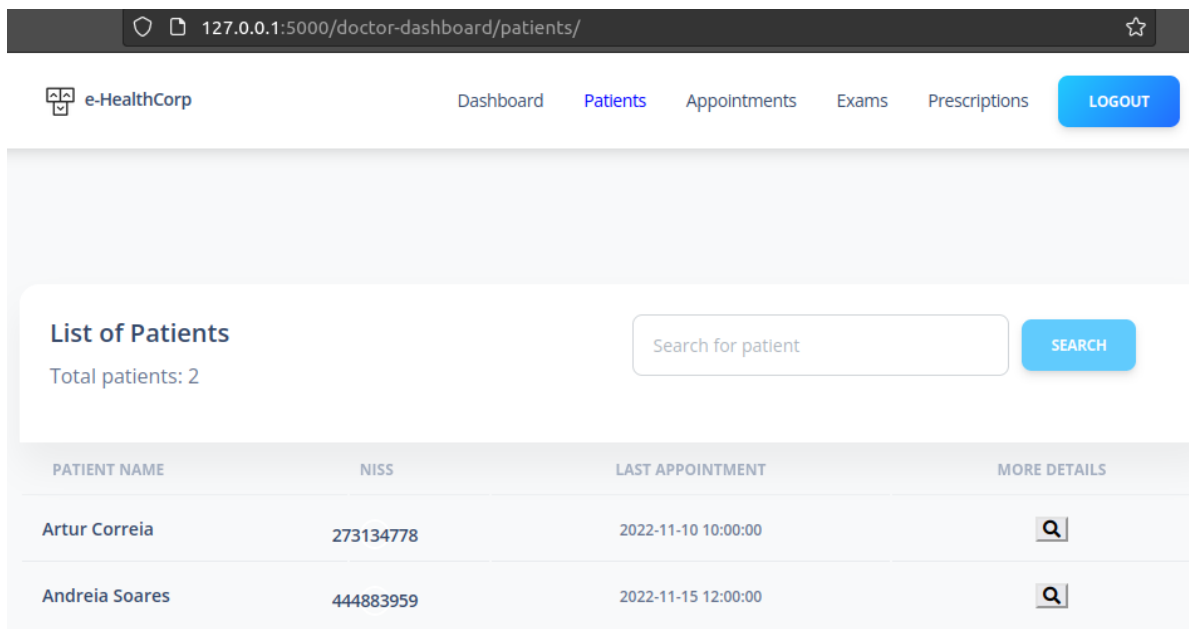
    # Get personal info about the patient
    cursor = db.cursor()
    cursor.execute("SELECT Num Utente, Nome, Email, Tel, Idade, Morada, NIF FROM Paciente JOIN Utilizador U "
                  "on Paciente.ID = U.ID WHERE Paciente.ID =" + str(_id))

    patient = cursor.fetchone()



    params_dict = {"name": patient[1], "niss": patient[0], "email": patient[2], "tel": patient[3], "age": patient[4], "address": patient[5],
                  "nif": patient[6], "illnesses": []}
```

Sendo o pedido processado normalmente.

→ App segura:



The screenshot displays the 'List of Patients' page in the e-HealthCorp application. The browser address bar shows the URL '127.0.0.1:5000/doctor-dashboard/patients/'. The navigation bar includes 'Dashboard', 'Patients', 'Appointments', 'Exams', 'Prescriptions', and a 'LOGOUT' button. The 'List of Patients' section shows 'Total patients: 2' and a search bar. Below is a table with patient information.

PATIENT NAME	NISS	LAST APPOINTMENT	MORE DETAILS
Artur Correia	273134778	2022-11-10 10:00:00	
Andreia Soares	444883959	2022-11-15 12:00:00	

127.0.0.1:5000/doctor-dashboard/patient-info

e-HealthCorp Dashboard Patients Appointments Exams Prescriptions LOGOUT

← Go Back

**Artur Correia**

NISS 273134778	NIF 262190185	Age 20
-------------------	------------------	-----------

**Contactos**

E-Mail art.afo@ua.pt	Telefone 930577403	Endereço Quinta do Bosque, Lote 110, 2ºEsq
-------------------------	-----------------------	---

**Diagnósticos**

- Common Cold

← → ↻ 127.0.0.1:5000/doctor-dashboard/patients/3 ☆

Como podemos observar o URL após pressionar o botão de “More Details” não apresenta a informação sensível do paciente que está a ser visualizado.

## Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

Ao adotar a mesma abordagem da aplicação insegura, ou seja, adicionar ao URL o ID de um paciente ao qual pretendemos aceder à sua ficha de paciente é nos apresentado o erro 404 “Not Found”.

Em termos de código, foi alterado na template a linha de redireccionamento para a página de detalhes do paciente, sendo que agora é feito não com o método `url_for` e a passagem de argumentos no mesmo, mas através de um formulário com o método POST.



```

{% for patient in params.patients %}
<tr>
  <td>
    <div class="d-flex px-2 py-1">
      <div class="d-flex flex-column justify-content-center">
        <h6 class="mb-0 text-sm">{{ patient.name }}</h6>
      </div>
    </div>
  </td>
  <td>
    <div class="avatar-group mt-2">
      <a href="javascript:;" class="avatar avatar-xs rounded-circle" data-bs-toggle="tooltip" data-bs-placement="bottom" title="Alfredo">
        <h6 class="mb-0 text-sm">{{ patient.niss }}</h6>
      </a>
    </div>
  </td>
  <td class="align-middle text-center text-sm">
    <span class="text-xs font-weight-bold"> {{ patient.last_appointment }} </span>
  </td>
  <td class="align-middle text-center text-sm">
    <form action="{{url_for('doctor_dashboard_patient_info')}}" method="POST">
      <button type="submit" class="fa fa-search" name="patient_id" value="{{ patient.id_id }}" id="{{ patient.id_id }}"></button>
    </form>
  </td>
</tr>
{% endfor %}

```

```

def doctor_dashboard_patient_info():
    if session.get('user_id') is None:
        flash("You must login to access this page!")
        return redirect(url_for('login'))

    if request.method == "POST":
        _id = request.form["patient_id"]

        # Get personal info about the patient
        cursor = db.cursor()
        cursor.execute("SELECT Num_Utente, Nome, Email, Tel, Idade, Morada, NIF FROM Paciente JOIN Utilizador U "
                       "on Paciente.ID = U.ID WHERE Paciente.ID = %s", (_id, ))

        patient = cursor.fetchone()

        params_dict = {"name": patient[1], "niss": patient[0], "email": patient[2], "tel": patient[3], "age": patient[4], "address": patient[5],
                       "nif": patient[6], "illnesses": []}

```

## 7. CWE-307: Improper Restriction of Excessive Authentication Attempts

### CVSS Estimado: 7.0

Esta CWE refere-se a situações em que o produto não aplica medidas suficientes para a prevenção de múltiplas tentativas de autenticação falhadas dentro de um pequeno intervalo de tempo.

Como consequência da ausência das medidas de segurança anteriormente referidas, as aplicações desenvolvidas são bastante mais suscetíveis a ataques do tipo *brute force* para acesso indevido a contas pertencentes a outros utilizadores.

### 7.1. Exploração da Vulnerabilidade

Na nossa aplicação, esta vulnerabilidade é encontrada durante a realização do login para uma conta de utilizador.

The screenshot shows a login interface with the title "Login" in blue. Below the title, there are two input fields: "Email" containing "art.ifo@ua.pt" and "Password" containing four dots. Below the password field, a red error message states "Email or password incorrect". Underneath the error message is a link "Forgot password?". A blue "Login" button is positioned below the link. A horizontal line separates the login section from the registration section, which contains the word "or" and a blue "Create New Account" button.

Na versão insegura da aplicação, quando um utilizador coloca a palavra-passe incorreta num e-mail relacionado com uma conta existente, apenas é apresentado um aviso de que uma das credenciais de autenticação está errada, não sendo imposto qualquer limite no número de tentativas falhadas de login com credenciais incorretas.

### Login

---

Email

Password

Password is incorrect! You have 4 attempts remaining.

[Forgot password?](#)

[Login](#)

---

or

[Create New Account](#)

### Login

---

Email

Password

You will not be able to attempt a login into this profile for the next 5 minutes.

[Forgot password?](#)

[Login](#)

---

or

[Create New Account](#)

Por sua vez, na versão segura da aplicação, o utilizador tem um limite de 5 tentativas de autenticação erradas, ao fim dos quais qualquer tentativa de login é bloqueada durante um total de 5 minutos. Este bloqueio é feito de acordo com o IP do utilizador, independentemente de este ter errado na autenticação do e-mail e/ou palavra-passe.

## 7.2. Resolução da Vulnerabilidade:

→ App insegura:

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

    params_dict = {"email": "", "password": "", "id": ""}

    if request.method == 'GET' and len(request.args) > 0:
        params_dict["email"] = request.args['email']
        params_dict["password"] = request.args['password']

        # Buscar email e pass à base de dados
        cursor = db.cursor(buffered=True)
        print(params_dict["email"])
        print(params_dict["password"])
        cursor.execute("SELECT ID, Email, Password FROM Utilizador WHERE Email = " + \
            + str(params_dict["email"]) + \
            " AND Password=" + \
            + str(params_dict["password"]) + ";")
        user_data = cursor.fetchone()
        print(user_data)
        if user_data is None:
            flash("Email or password incorrect")
            cursor.close()
            return redirect(url_for('login'))
```

Na aplicação insegura, o login é feito com recurso ao método GET, sendo executada uma *query* na base de dados que verifica se o e-mail e a palavra-passe passadas no formulário existe, ou não, na tabela Utilizador Conforme anteriormente descrito, esta *query* não é parametrizada, permitindo a ocorrência de SQL Injection.

Caso não seja encontrada nenhuma linha na tabela Utilizador correspondente à *query* efetuada, é apresentada uma mensagem em flash a informar o utilizador de que as credenciais estão erradas, sendo feito um redirecionamento de novo para a página de login, onde esta mensagem é apresentada.

→ App segura:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    params_dict = {"email": "", "password": "", "id": ""}
    if request.method == 'POST':
        params_dict["email"] = request.form['email']
        params_dict["password"] = request.form['password']

        user_IP = request.remote_addr
        current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

        # Verificar se o IP está bloqueado
        cursor = db.cursor()

        cursor.execute("SELECT * FROM Login_Attempts WHERE IP=%s", (user_IP, ))
        data = cursor.fetchone()

        if data is not None:
            # Caso o IP já tenha anteriormente errado nas credenciais de login
            attempts = data[1]
            last_attempt = data[2]

            duration = datetime.now() - last_attempt
            duration_in_min = divmod(duration.total_seconds(), 60)[0]

            if attempts >= 5 and duration_in_min < 5:
                flash('You will not be able to attempt a login into this profile for the next 5 minutes.')

                cursor.execute('UPDATE Login_Attempts SET Num_Tentativas = %s, Ult_tentativa = %s WHERE IP=%s'
                               , (attempts + 1, current_time, user_IP))
                db.commit()
                cursor.close()
                return redirect(url_for('login'))
```

Por sua vez, na aplicação segura, foi adicionada uma nova tabela à base de dados, denominada Login\_Attempts, para permitir o tratamento desta vulnerabilidade. Assim, nesta tabela serão guardadas as listas de IPs que falharam a autenticação de login, bem como o número de tentativas falhadas e a timestamp da última tentativa de acesso ao site.

Deste modo, a lógica do tratamento do login começa por verificar se o IP que está a tentar aceder ao site se encontra ou não nesta tabela, já bloqueado, isto é, com o número de tentativas de acesso superior a 5, e com uma timestamp de último acesso correspondente a um intervalo de tempo inferior a 5 minutos, comparado com o timestamp da tentativa de acesso atual.

```

cursor.execute("SELECT Utilizador.ID, Email, Password FROM Utilizador WHERE Email = %s", (params_dict["email"],))
user_data = cursor.fetchone()

if user_data is None or not check_password_hash(user_data[2], params_dict["password"]):
    # Caso o E-Mail ou a Pass estejam errados
    if attempts == 0:
        flash(f'Password is incorrect! You have 4 attempts remaining.')

        cursor.execute('INSERT INTO Login_Attempts (IP, Num_Tentativas, Ult_tentativa) VALUES (%s, 1, %s)'
                        , (user_IP, current_time))

        db.commit()
        cursor.close()
        return redirect(url_for('login'))
    elif attempts < 3:
        rem = 5 - (attempts + 1)
        flash(f'Password is incorrect! You have {rem} attempts remaining.')

        cursor.execute('UPDATE Login_Attempts SET Num_Tentativas = %s, Ult_tentativa = %s WHERE IP=%s'
                        , (attempts+1, current_time, user_IP))

        db.commit()
        cursor.close()
        return redirect(url_for('login'))
    elif attempts == 3:
        flash('Password is incorrect! This is your last login attempt before you are blocked for 5 minutes!')

        cursor.execute('UPDATE Login_Attempts SET Num_Tentativas = %s, Ult_tentativa = %s WHERE IP=%s'
                        , (attempts + 1, current_time, user_IP))

        db.commit()
        cursor.close()
        return redirect(url_for('login'))
    else:
        cursor.execute('UPDATE Login_Attempts SET Num_Tentativas = %s, Ult_tentativa = %s WHERE IP=%s'
                        , (attempts + 1, current_time, user_IP))

        db.commit()

        flash('You will not be able to attempt a login for any account for the next 5 minutes.')
        return redirect(url_for('login'))

```

Caso o IP não se encontrasse bloqueado, é então feita uma *query* similar às anteriormente descritas para verificar se o e-mail se encontra na base de dados, e se a palavra-passe passada corresponde, ou não, à palavra-passe a si associada. Caso se verifique que este não é o caso, é então inserida/atualizada a informação correspondente ao IP na tabela Login\_Attempts previamente mencionada, sendo que o aviso apresentado ao utilizador varia de acordo com o número de tentativas que este ainda tem disponíveis para realizar o login.

```

        return redirect(url_for('login'))
    else:
        ID = user_data[0]
        email = user_data[1]

        session['user_id'] = ID
        session['email'] = email

        cursor.execute("SELECT * FROM Login_Attempts WHERE IP=%s", (user_IP,))
        data = cursor.fetchone()

        if data is not None:
            # Caso o IP já tenha anteriormente errado nas credenciais de login
            cursor.execute('DELETE FROM Login_Attempts WHERE IP=%s'
                            , (user_IP, ))

            db.commit()

            # Verificar se é médico ou paciente e redirecionar para a página correta
            cursor.execute("SELECT ID FROM Medico WHERE ID = %s", (ID,))
            medico_data = cursor.fetchone()

            if medico_data is None:
                # É paciente
                cursor.close()
                return redirect(url_for('logged'))
            else:
                # É médico
                cursor.close()
                return redirect(url_for('doctor_dashboard'))

        return render_template('login.html')

```

Caso o utilizador realize a autenticação de forma bem-sucedida, antes de ser feito o redireccionamento para a sua página de paciente/médico, é verificado se o seu IP estava na tabela Login\_Attempts, sendo apagado o registo caso se encontrasse nesta tabela. Uma vez que no início do processo se verificava se o IP já estava nesta tabela ou não, e se já estava bloqueado, é impossível a execução da função chegar a este passo para um utilizar bloqueado, mesmo que na nova tentativa ele coloque credenciais de login corretas.

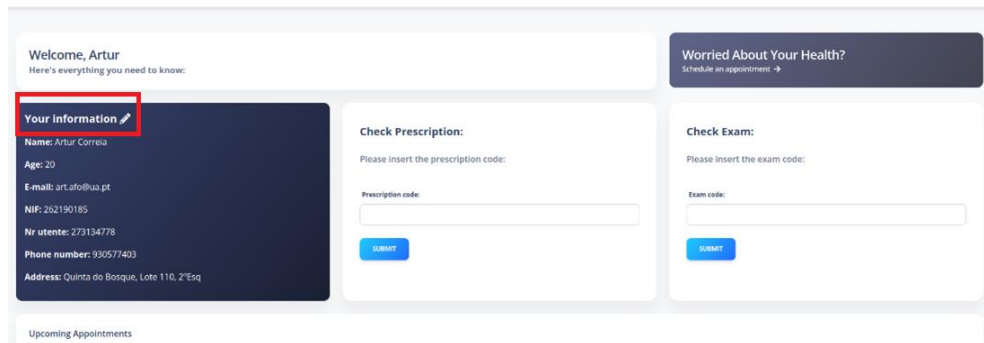
## 8. CWE-522: Insufficiently Protected Credentials

### CVSS Estimado: 6.5

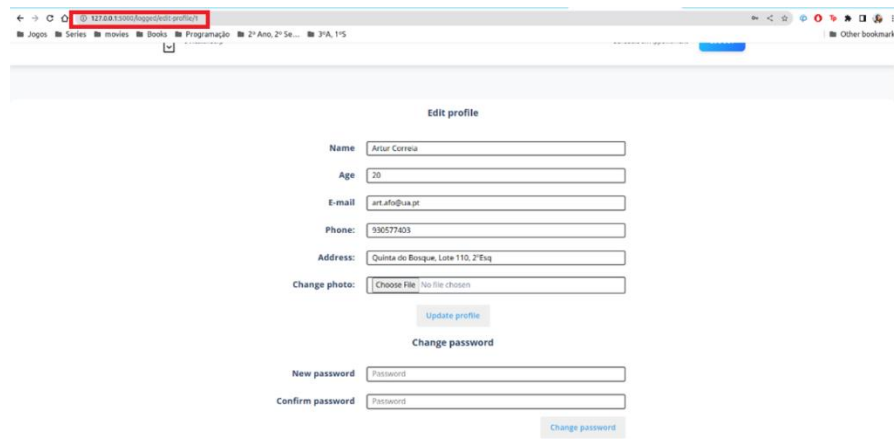
Esta CWE é verificada quando o produto desenvolvido transmite ou armazena credenciais de autenticação, mas de um modo inseguro e suscetível a interações por parte de agentes não autorizados.

A ocorrência desta vulnerabilidade numa aplicação que requer o login para acesso às suas funcionalidades pode, então, permitir que um utilizador seja capaz de obter as credenciais de acesso de outras contas, conseguindo alterar ou editar estas. Assim, ele pode ganhar acesso a esta conta, ao mesmo tempo que impede que o dono legítimo se consiga autenticar e entrar no seu perfil.

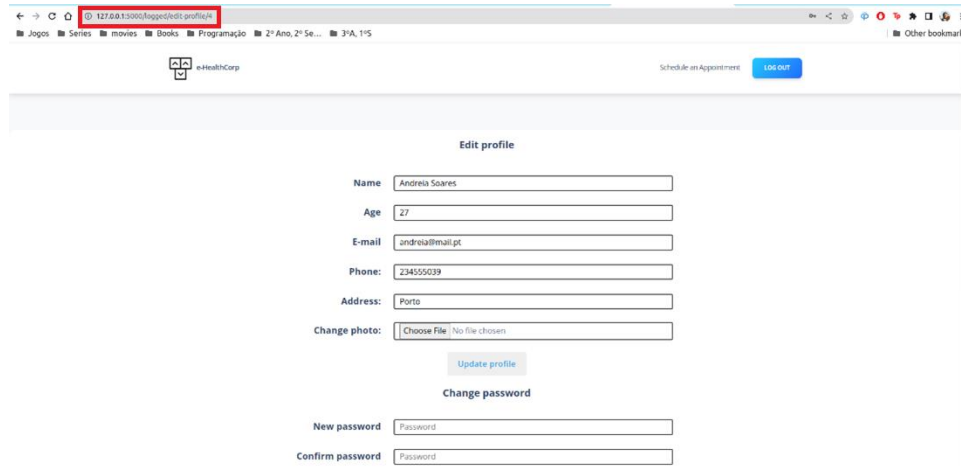
### 8.1. Exploração da Vulnerabilidade



Na versão insegura da aplicação desenvolvida, esta vulnerabilidade encontra-se exposta na página de edição de perfil de um Paciente, acessível através da sua dashboard.



O acesso à página de edição do perfil é feito com um pedido GET, sendo apresentada no URL o ID do utilizador cujo perfil está a ser editado.



Deste modo, é possível aceder às informações do perfil de outro utilizador, bastando para isso alterar o ID passado no caminho URL. Uma vez que esta página contém um formulário para alteração da palavra-passe, que não está protegido pela necessidade de autenticação com a palavra-passe atual aquando da submissão do formulário, é então possível alterar as credenciais de acesso de outras contas que não a correspondente ao nosso login, sendo que estamos, portanto, na presença de uma falha grave de segurança.

## 8.2. Resolução da Vulnerabilidade

### → Versão Insegura

```
@app.route('/logged/edit-profile/<_id>', methods=['GET', 'POST'])
def edit_profile(_id):
    ctx = {
        'user_id': {'_id': session.get('user_id')},
        'patient_info': [],
    }

    if request.method == "GET":
        cursor = db.cursor()
        cursor.execute("SELECT * FROM Utilizador WHERE ID = " + str(_id))
        ctx['patient_info'] = cursor.fetchone()
        print(ctx['patient_info'])
        cursor.close()

    return render_template('edit-patient-profile.html', ctx=ctx)
```

Conforme anteriormente mencionado, na versão insegura da aplicação o acesso à página de edição do perfil é feita apenas com recurso a um método GET, tratado na função `edit_profile()`. Esta função recebe como parâmetro do GET o ID da conta a ser editada, sendo este portanto apresentado na URL da página. É de notar que o acesso à página é feito através da página principal da conta do paciente, conforme anteriormente descrito, sendo aí enviado no GET o ID do perfil da conta autenticado; no entanto, alterando o ID presente no URL da página, é possível aceder à página de edição de outros perfis.



Por sua vez, a função vai utilizar este ID recebido para realizar uma *query* na base de dados, onde serão retornados todos os campos relacionados com as informações do perfil, a serem apresentadas como “placeholder” nos campos do formulário de edição de perfil presente na página.

```
@app.route('/logged/edit-profile-details/<id>', methods=['POST'])
def edit_profile_details(id):
    ctx = dict()

    ID = id
    name = request.form.get("name")
    age = request.form.get("age")
    email = request.form.get("email")
    tel = request.form.get("tel")
    morada = request.form.get("morada")
    photo = request.files["photo"]

    if photo:
        photo.save(os.path.join(app.config['UPLOAD_FOLDER'], photo.filename))

    cursor = db.cursor()
    cursor.execute("UPDATE Utilizador SET Nome = " + name + ", Idade = " + age + ", Email = " + email + ", Tel = " + tel + ", Morada = " + morada + ", Image_path = "
    db.commit()

    ctx = {
        'user_id': {"_id": id},
        'patient_info': [],
    }
    cursor.execute("SELECT * FROM Utilizador WHERE ID = " + str(id))
    ctx["patient_info"] = cursor.fetchone()

    cursor.close()

    return render_template('edit-patient-profile.html', ctx=ctx)
```

A alteração de informação utilizando este formulário é tratado na função `edit_profile_details()`, que recebe os parâmetros submetidos pela submissão com recurso ao método POST do formulário, e que vai realizar uma *query* SQL para atualização da informação do perfil na base de dados, com recurso ao ID passado como parâmetro.

```
@app.route('/logged/edit-profile-password/<id>', methods=['GET'])
def edit_profile_password(id):
    new_password = str(request.args.get("psw"))

    cursor = db.cursor()
    cursor.execute("UPDATE Utilizador SET Password=" + new_password + " WHERE ID=" + str(id) + ";")
    db.commit()

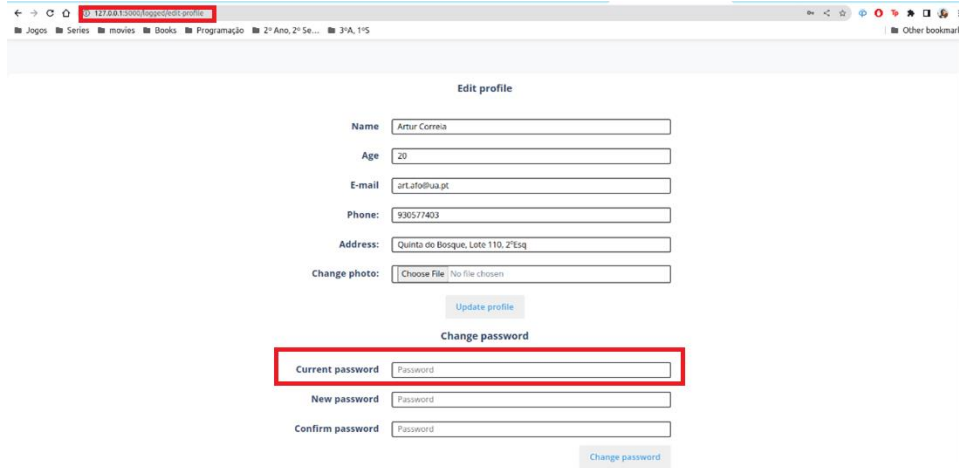
    cursor.execute("SELECT * FROM Utilizador WHERE ID = " + str(id))

    ctx = {
        'user_id': {"_id": id},
        'patient_info': [],
    }
    ctx["patient_info"] = cursor.fetchone()

    return render_template('edit-patient-profile.html', ctx=ctx)
```

Por fim, o tratamento da submissão do formulário referente à alteração da password funciona de forma semelhante à descrita anteriormente, estando alojado na função `edit_profile_password()`. Esta função vai também realizar um *query* á base de dados para atualização do campo password da entrada corresponde ao ID passado, na tabela Utilizador.

## → Versão Segura



A versão segura da aplicação foi desenvolvida de forma a que o ID do perfil a editar não seja exposto no caminho URL, impossibilitando portanto o acesso à edição de outros perfis que não o do utilizador autenticado através de uma simples mudança do link.

Além disso, de forma a redobrar a segurança na alteração das credenciais de autenticação, é requerido que o utilizador coloque a sua palavra-passe atual antes da submissão do formulário correspondente.

```
@app.route('/logged/edit-profile', methods=['GET', 'POST'])
def edit_profile():
    ctx = {
        'user_id': {'_id': session.get('user_id')},
        'pacient_info': [],
    }

    if request.method == "GET":
        cursor = db.cursor()
        cursor.execute("SELECT * FROM Utilizador WHERE ID = " + str(session.get('user_id')))
        ctx['pacient_info'] = cursor.fetchone()
        print(ctx['pacient_info'])
        cursor.close()
    return render_template('edit-patient-profile.html', ctx=ctx)
```

Para tornar isto possível, a função `edit_profile()` deixou de receber como parâmetro do método GET o ID do perfil a editar, sendo este ID determinado antes pelo acesso à cookie `"user_id"` da sessão atual, evitando a exposição do ID no URL da página. Desta forma, apenas é possível apresentar a página de edição de perfil de uma conta que tenha sido autenticada com sucesso através do login implementado, e que esteja, portanto, guardada na cookie `Session`.

A mesma lógica foi também aplicada na função `edit_profile_details()`, que trata do POST da submissão do formulário de alteração de informações do perfil, e na função

edit\_profile\_password(), que aborda o POST da submissão do formulário de alteração da palavra-passe.

```
@app.route('/logged/edit-profile-password/', methods=['POST'])
def edit_profile_password():
    id = session.get('user_id')

    current_password = str(request.form.get("oldpsw"))
    new_password = str(request.form.get("psw"))
    confirm_password = str(request.form.get("pswc"))

    cursor = db.cursor()
    cursor.execute("SELECT Password FROM Utilizador WHERE ID = %s", (id,))
    db_password = cursor.fetchone()

    if not check_password_hash(db_password[0], current_password):
        flash("Incorrect Current Password")
    elif new_password != confirm_password:
        flash("Passwords do not match")
    else:
        cursor.execute("UPDATE Utilizador SET Password = %s WHERE ID = %s", (generate_password_hash(new_password), id,))
        db.commit()
        cursor.close()

    return redirect(url_for('edit_profile'))
```

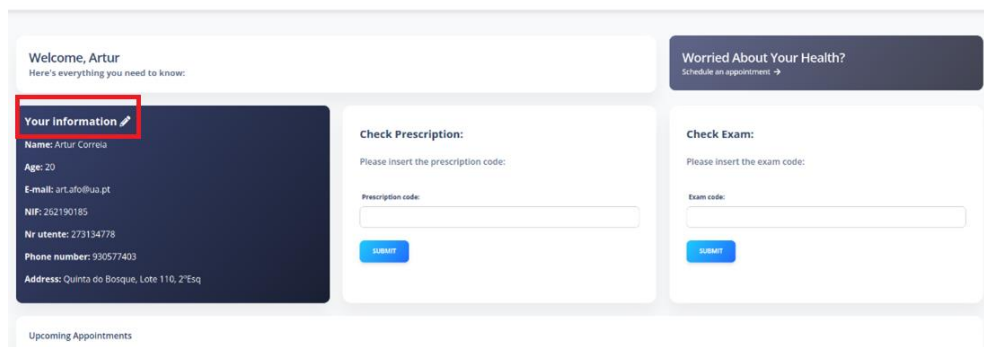
## 9. CWE-434: Unrestricted Upload of File with Dangerous Type

### CVSS Estimado: 7.7

Esta CWE ocorre quando o software permite ao atacante fazer o upload ou a transferência de ficheiros de tipos perigosos, que serão processados dentro do ambiente do produto.

A exploração deste tipo de vulnerabilidades pode permitir, entre outras coisas, o upload de código perigoso para o sistema que, se executado, poderá ser usado pelo atacante para obter controlo total do sistema, realizar ataques no lado do cliente, aceder à base de dados, entre outros, sendo esta vulnerabilidade, portanto, um risco significativo para uma aplicação.

### 9.1. Exploração da Vulnerabilidade

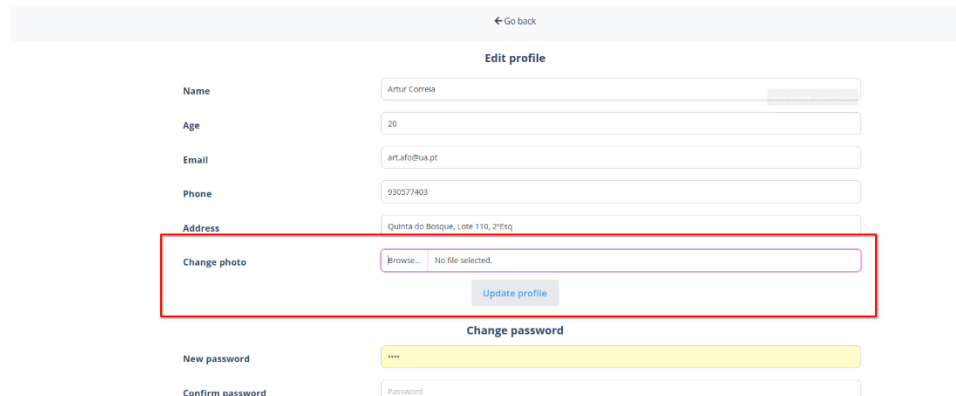


The screenshot shows a user profile page for 'Artur'. The 'Your information' section is highlighted with a red box. It contains the following details:

- Name: Artur Correia
- Age: 20
- E-mail: art.ato@ua.pt
- NIF: 262190185
- Nr utente: 273134778
- Phone number: 930577403
- Address: Quinta do Bosque, Lote 110, 2ª Esq

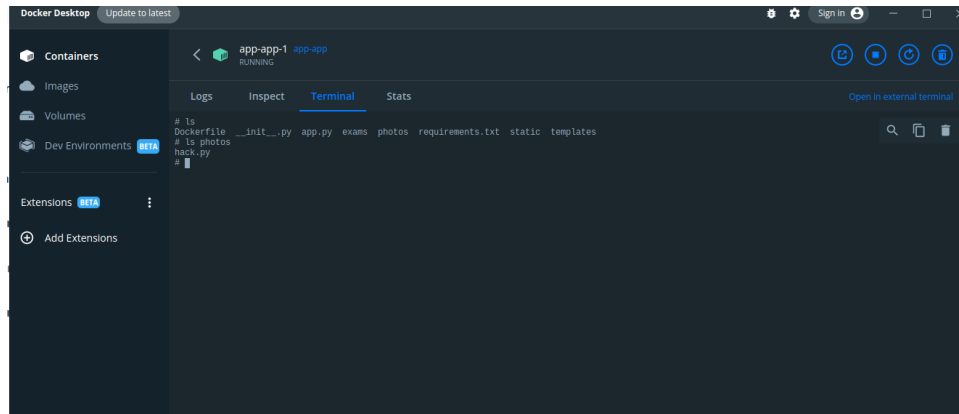
Other sections visible include 'Check Prescription:', 'Check Exam:', and 'Upcoming Appointments'.

Esta vulnerabilidade encontra-se na página de edição de perfil do utilizador, mais concretamente quando se faz o upload de uma imagem de perfil.



The screenshot shows the 'Edit profile' form. The 'Change photo' field is highlighted with a red box. It contains a 'Browse...' button and the text 'No file selected.' Below the field is an 'Update profile' button. Other fields include 'Name', 'Age', 'Email', 'Phone', 'Address', 'New password', and 'Confirm password'.

Como não se faz validação do tipo de ficheiro que é enviado, é possível enviar para o sistema um ficheiro maligno, como por exemplo o hack.py criado para demonstração.



Deste modo o script maligno fica guardado dentro da pasta das fotos dos utilizadores.

## 9.2. Resolução da Vulnerabilidade

→ Versão Insegura

```
@app.route('/logged/edit-profile-details/<id>', methods=['POST'])
def edit_profile_details(id):
    ctx = dict()

    ID = id
    name = request.form.get("name")
    age = request.form.get("age")
    email = request.form.get("email")
    tel = request.form.get("tel")
    morada = request.form.get("morada")
    photo = request.files["photo"]

    if photo:
        photo.save(os.path.join(app.config['UPLOAD_FOLDER'], photo.filename))

    cursor = db.cursor()
    cursor.execute("UPDATE Utilizador SET Nome = '" + name + "', Idade = '" + age + "'")
    db.commit()

    ctx = {
        'user_id': {"_id": id},
        'pacient_info': [],
    }
    cursor.execute("SELECT * FROM Utilizador WHERE ID = " + str(id))
    ctx["pacient_info"] = cursor.fetchone()

    cursor.close()

    return render_template('edit-patient-profile.html', ctx=ctx)
```

Na versão insegura apenas se recolhe o ficheiro através do formulário e guarda-se na pasta onde se encontram todas as fotos de utilizador, sem qualquer verificação de tipo de ficheiro.

## → Versão Segura

```
app = Flask(__name__)
app.config['SECRET_KEY'] = 'mysecretkey'
app.config['UPLOAD_EXAMS'] = 'exams'
app.config['UPLOAD_PHOTOS'] = 'photos/'
app.config['SAFE_EXTENSIONS'] = set(['png', 'jpg', 'jpeg', 'gif'])
```

Na versão segura são guardadas as extensões dos ficheiros que se consideram seguras.

```
@app.route('/logged/edit-profile-details/', methods=['POST'])
def edit_profile_details():
    ID = session.get('user_id')
    name = request.form.get("name")
    age = request.form.get("age")
    email = request.form.get("email")
    tel = request.form.get("tel")
    morada = request.form.get("morada")
    photo = request.files["photo"]
    secure_photo = None

    if photo:
        secure_photo = secure_filename(photo.filename)
        if secure_photo.split(".")[-1] not in app.config["SAFE_EXTENSIONS"]:
            flash("Invalid file type")
            return redirect(url_for('edit_profile'))

    photo.save(os.path.join(app.config['UPLOAD_PHOTOS'], secure_photo))
```

E antes de se adicionar o ficheiro na pasta das imagens é verificado se a sua extensão faz parte das extensões aceites.

Invalid file type

[Go back](#)

### Edit profile

Name	<input type="text" value="Artur Correia"/>
Age	<input type="text" value="20"/>
Email	<input type="text" value="art.afa@ua.pt"/>
Phone	<input type="text" value="930577403"/>
Address	<input type="text" value="Quinta do Bosque, Lote 110, 2ªEsq"/>
Change photo	<input type="button" value="Browse..."/> <input type="text" value="No file selected."/>

Se o ficheiro não tiver uma das extensões aceites é recusado.

## 10. CWE-552: Files or Directories Accessible to External Parties

### CVSS Estimado: 6.0

A exploração desta CWE permite tornar ficheiros e/ou diretórios acessíveis a autores não-autorizados. Por exemplo, servidores web podem armazenar um conjunto de ficheiros por baixo do diretório raiz acessível aos seus utilizadores, que poderão ficar expostos caso não seja feito controlo de acesso aos diretórios. O mesmo pode acontecer para aplicações comprimidas em ficheiros ZIP ou tar.

### 10.1. Exploração da Vulnerabilidade

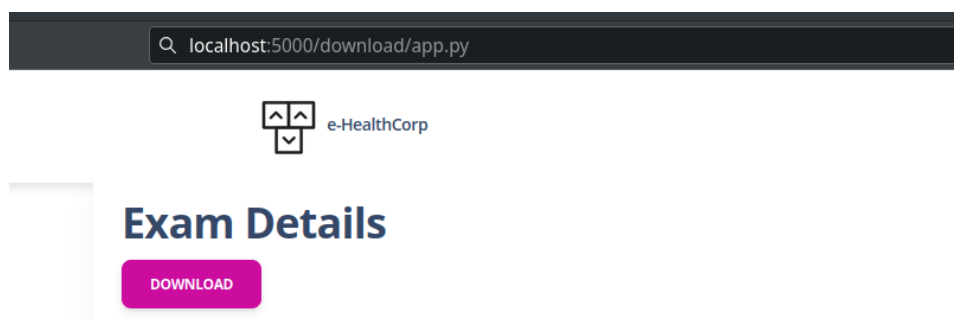
The screenshot shows the eHealthCorp patient portal. At the top, there is a navigation bar with the eHealthCorp logo, a 'Schedule an Appointment' link, and a 'Log Out' button. Below the navigation bar, there is a main content area. On the left, there is a sidebar with a 'vrtur' profile and a list of appointments. The main content area has a 'Check Prescription' section and a 'Check Exam' section. The 'Check Exam' section is highlighted with a red box. It contains a form with a label 'Check Exam:', a text input field for 'Exam code:', and a 'SUBMIT' button. The text input field contains the value 'HGK10'. Above the 'Check Exam' section, there is a dark blue banner with the text 'Worried About Your Health?' and a 'Schedule an appointment' link.

Esta vulnerabilidade encontra-se na página dos exames, no lado do paciente, mais concretamente quando se pretende efetuar o download do exame.

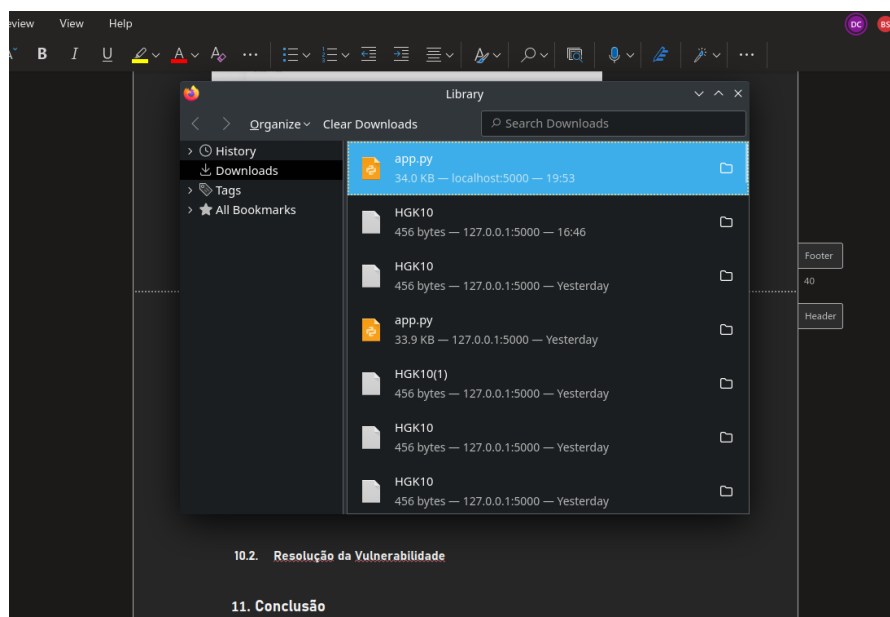
The screenshot shows the 'Exam Details' page. At the top, there is a navigation bar with the eHealthCorp logo. Below the navigation bar, there is a 'Go back' link. The main content area has a title 'Exam Details' and a 'DOWNLOAD' button. Below the 'DOWNLOAD' button, there is a list of details: 'ID: HGK10', 'Patient: Artur Correia', 'Data de Emissão: 2022-11-15', 'Data de Validade: 2022-11-22', and 'Tests'.

```
Search HTML
<div class="row">
  <div class="col-12">
    <h2>Exam Details</h2>
    <form action="/download//exams/HGK10" type="GET">
      <button class="btn btn-primary">Download</button>
    </form>
  </div>
</div>
<div class="row">
  <div class="col-12">
    <h5>
      ID
```

Ao efetuar a inspeção da página com o browser o link utilizado para se efetuar o download é descoberto e para além disso o método utilizado para o formulário é GET em vez de POST, o que leva à dedução de que o caminho do ficheiro é retirado através do URL.



Com isso é possível descarregar qualquer ficheiro pertencente à aplicação, sendo apenas necessário saber o nome do que se pretende, como por isso o código base da aplicação, app.py





## 10.2. Resolução da Vulnerabilidade

### → Versão Insegura

```
@app.route('/download/<path:path>', methods=['GET', 'POST'])
def download(path):
    return send_file(path, as_attachment=True)
```

Na versão insegura é utilizado o método GET para recolher o caminho do ficheiro pretendido, que não é verificado como sendo do utilizador que o pretende e para além disso é utilizada a função `send_file()` do flask, não sendo a mais recomendada para este caso de uso, visto que o único parâmetro que necessita para descarregar um ficheiro é a sua localização na aplicação.

### → Versão Segura

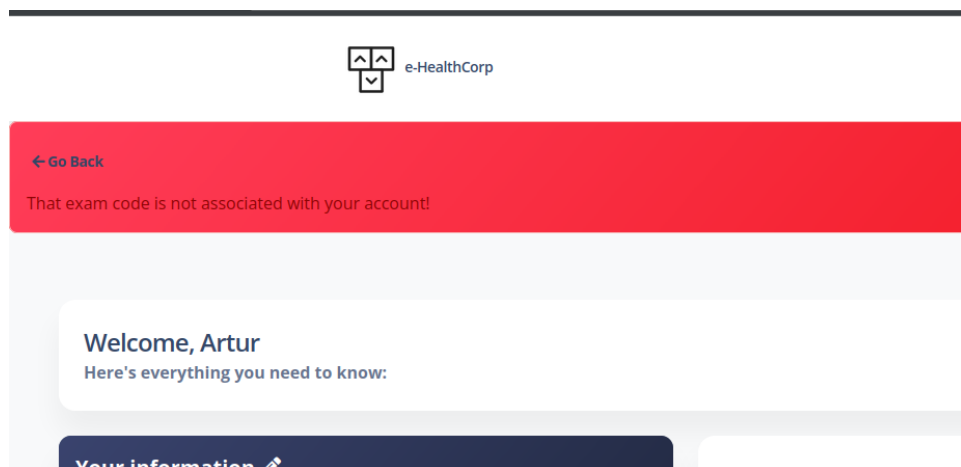
```
@app.route('/download/<path:filename>', methods=['GET', 'POST'])
def download(filename):
    cursor = db.cursor()
    cursor.execute("SELECT * FROM Analise WHERECodigo = %s AND ID_Pac = %s", (filename, session['user_id'], ))
    exam = cursor.fetchall()
    if len(exam) == 0:
        flash("That exam code is not associated with your account!")
        return redirect(url_for("logged"))

    return send_from_directory(directory=app.config['UPLOAD_EXAMS'], path=filename, as_attachment=True)
```

Na versão segura são feitas algumas alterações, primeiramente é verificado se o código (caminho) que o utilizador indica corresponde a uma análise dele na base de dados. Para melhorar foi trocada a função `send_file()` pela `send_from_directory()` que é considerada mais segura devido ao facto de utilizar mais argumentos para localizar o ficheiro, sendo o primeiro muito importante para restringir o acesso dos utilizadores, pois é um local pré configurado chamado `UPLOAD_EXAMS` que remete à pasta dos exames dos pacientes e o segundo é o nome do ficheiro que se pretende descarregar.

```
app = Flask(__name__)
app.config['SECRET_KEY'] = 'mysecretkey'
app.config['UPLOAD_EXAMS'] = 'exams'
app.config['UPLOAD_PHOTOS'] = 'photos/'
app.config['SAFE_EXTENSIONS'] = set(['png', 'jpg', 'jpeg', 'gif'])
```

No caso do utilizador não ter autorização para descarregar o ficheiro é remetido para a página do inicial com um alerta.



## 11. Conclusão

A concretização deste projeto auxiliou em grande parte a nossa aprendizagem de implementação de métodos mais seguros em futuros websites ou outras plataformas que iremos desenvolver, de modo que a sua utilização não constitua um perigo para qualquer utilizador (seja administrador, ou um simples utilizador pouco experienciado com a plataforma).

Dessa maneira, a pesquisa aprofundada sobre as várias vulnerabilidades que podem constituir um website permite a prevenção de potenciais ataques à sua infraestrutura, bem como a manutenção da privacidade de dados que utilizadores nos confiem.

## 12. Referências Bibliográficas

- [CWE - Common Weakness Enumeration](#)
- [Common Vulnerability Scoring System Calculator](#)