

# Vulnerabilidades relatório

## Segurança informática e nas Organizações

Aluno:

Bruno Lemos 98221

João Viegas 98372

João Amaral 98373

Pedro Rocha 98256

Professor:

André Zuquete

João Barraca

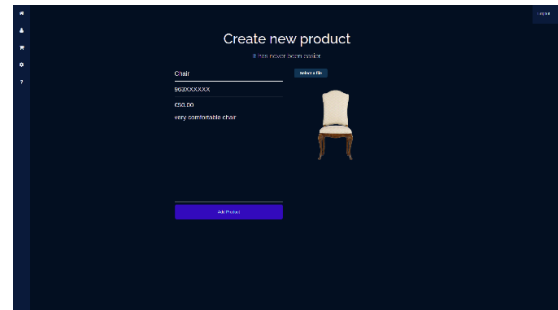
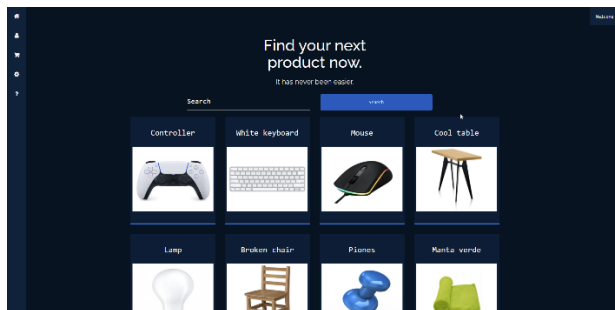
Curso:

Licenciatura Engenharia de Computadores e  
Informática

# Tema do Projeto

O nosso projeto consiste no desenvolvimento de uma web app dedicada à compra e venda de produtos.

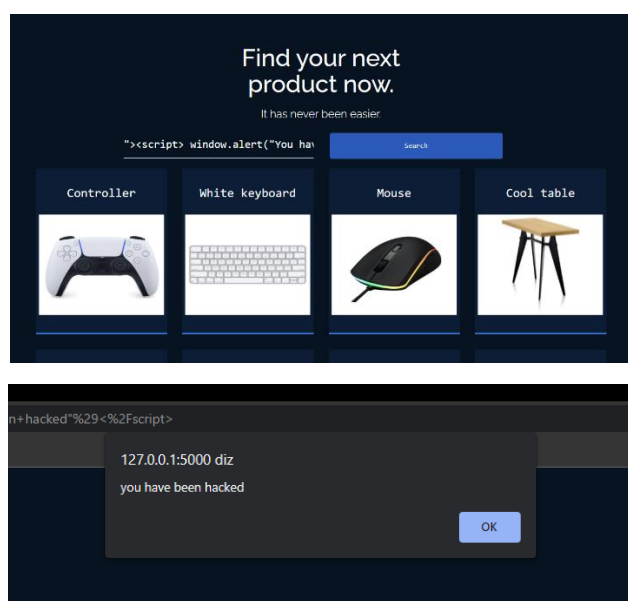
É possível pesquisar produtos no site sem ter feito autenticação, porém é necessário criar uma conta ou iniciar sessão para adicionar produtos ao “mercado”. Os utilizadores desta web app têm acesso aos seus perfis e a todos os produtos criados por si, com a possibilidade de os eliminar. É também possível gerir as informações de um perfil, alterando a password ou mesmo apagando a conta.



# Vulnerabilidades na aplicação

## Cross-site-Scripting CWE-79

Nesta vulnerabilidade, a nossa web app não “neutraliza” o possível input dos utilizadores antes de ser usado como fonte de pesquisa para produtos. Por outras palavras, quando um utilizador pesquisa por uma determinada palavra ou frase, essa mesma frase é apresentada na página de procura. O problema, é que a frase de pesquisa é integrada no código HTML da página de pesquisa apresentada, o que pode levar a inputs mais sofisticados que introduzam código executável quando se partilha esta página:



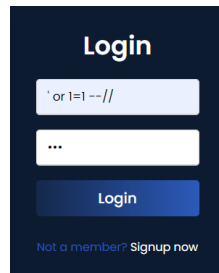
Para resolver esta vulnerabilidade, basta trocar os elementos específicos que alteram o código html (como por exemplo os símbolos '>', '<' e '"') pelos seus respetivo codigos html:

```
query = request.args.get('query').lower()
#PROTECAO CONTRA XSS
#-----
query = query.replace(">", "&#62;")
query = query.replace("<", "&#60;")
query = query.replace('"', "&#34;")
#-----
```

Com esta proteção, a página dos resultados de procura deixa de executar código inserido e apresenta apenas a pesquisa feita pelo utilizador.

## SQL Injection CWE-89

A vulnerabilidade SQL injection baseia-se maioritariamente em executar código SQL no lado do servidor, com o intuito de revelar informação indesejada a potenciais atacantes, assim como a possibilidade de alterar todos os valores de uma base de dados e até apagar esta mesma. Um dos campos vulneráveis para introdução deste código maligno é o campo 'username' da página login:



The image shows a login form with a dark blue background. At the top, the word "Login" is written in white. Below it, there are two input fields. The first field contains the text "'or 1=1 --//". The second field contains three dots "...". Below the input fields is a blue button with the text "Login" in white. At the bottom, there is a link that says "Not a member? Signup now" in white.

Usando esta string como valor de entrada do username e uma palavra-passe qualquer, é possível iniciar sessão na primeira conta presente na base de dados, tendo acesso total aos dados e produtos deste utilizador. Tendo esta fraqueza, também é possível introduzir “ 'or 1=1; DROP TABLE users --// ” o que resultaria em eliminar toda a informação da base de dados em questão.

Para contornar este problema, a maneira qual escolhemos proteger a web app destes possíveis ataques foi garantir que tanto o campo do username como a password não permitam certos caracteres especiais:

```
username = request.form['username']
##protecao contra SQL injection
#-----
if not re.match(r'[A-Za-z0-9_]+', username):
    return redirect(url_for('login', msg="Invalid username!"))
#-----
```

Deste modo, tanto na criação de uma password como no campo username, torna-se impossível utilizar caracteres especiais principais na técnica de SQL injection.

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

Esta vulnerabilidade consiste na passagem de informação a mais ao utilizador no momento do login. Ou seja, ao tentar uma autenticação é informado qual das credenciais está errada (password ou username):

**Login**

Wrong password!

Username

Password

Login

Not a member? [Signup now](#)

Isto torna a app muito mais vulnerável a ataques de força bruta e reduz circunstancialmente o número de tentativas a serem realizadas para conseguir entrar na conta de alguém.

Facilmente se corrige esta vulnerabilidade, indicando apenas que as credenciais estão erradas na mensagem que é passada.

**Login**

Wrong credentials!

Username

Password

Login

Not a member? [Signup now](#)

## Insufficiently Protected Credentials CWE-522

Esta vulnerabilidade pode ser utilizada caso a web app permita ao utilizador alterar credenciais de autenticação suscetíveis a intercessão ou recuperação não autorizada.

Neste caso em especifico seria possível alterar a password de um dado utilizador sem saber a sua password atual.

```
# CWE 522 -> User changing pass might not be the user loggedin
# if user != session['username']:
#     return redirect(url_for('changepassword', error="The username does not match the logged in username!"))

cursor = con.cursor()
query = f"SELECT * FROM users WHERE username = '{user}' and password = '{md5(oldPass.encode()).hexdigest()}'"
cursor.execute(query)
account = cursor.fetchone()

# CWE 522 -> User changing pass might not be the user loggedin
# if not account:
#     return redirect(url_for('changepassword', error="The password does not match the logged in username!"))

if not re.match(r'[A-Za-z0-9_]+', newPass):
    return redirect(url_for('changepassword', error="Invalid special characters!"))

if newPass != confirm_new_pass:
    return redirect(url_for('changepassword', error="The new passwords aren't matching!"))

query = f'UPDATE users SET password="{md5(newPass.encode()).hexdigest()}" WHERE username="{user}"'
```

No código acima apresentado, no presente estado em que se encontra, não era verificada a autenticidade do utilizador em que estava a tentar mudar a password. Assim, introduzindo um nome de um outro utilizador uma password à sua escolha, era possível alterar as credenciais desse utilizador para sempre, ganhando controlo total dessa conta.

Para resolver este problema, temos que verificar que o utilizador que está a mudar a password é o mesmo utilizador que tem a sessão iniciada, e que todos os campos preenchidos são de acordo com a informação presente na base de dados. Descomentando o código selecionado no script apresentado acima, tornamos esta vulnerabilidade segura.

## Conclusão

Bruno Lemos

---

25%

João Amaral

---

25%

Pedro Rocha

---

25%

João Viegas

---

25%

O trabalho foi igualmente distribuído entre os membros do grupo fazendo passar um pouco por todos as várias tarefas do mesmo, como: desenvolvimento da interface gráfica (frontend), o desenvolvimento do servidor/app (backend), decisão das vulnerabilidades implementadas, escrita do relatório e testes da aplicação.