universidade de aveiro

Projeto eHealth Corp

Relatório

Report

Segurança Informática e nas Organizações

Prof.João Paulo Barraca

Departamento de Eletrónica,Telecomunicações e Informática

Ano Letivo 2022-2023

Equipa 38

Bruno Lins- 101077

Filipe Barbosa-103064

Miguel Gomes-103826

Pedro Durval-103173

universidade de aveiro

Índice

Descrição do projeto

O nosso projeto implementa um website de saúde. Este tem funções como registar novos utilizadores, realizar login, marcar consulta e escrever e ler críticas acerca da empresa, dos doutores, etc.

Na página inicial o utilizador poderá visualizar os contactos da empresa, os serviços prestados e ainda uma descrição acerca da ***eHealth Corp***. Poderá ainda fazer login.

Assim que o utilizador faça login com as suas credenciais, entrará numa nova página com várias novas funções, como visualizar as suas consultas marcadas, marcar novas consultas, visualizar a lista de todos os doutores, alterar a sua palavra-passe e aceder ao resultado do exame através de um download do ficheiro.

Para que o utilizador marque uma consulta será necessário, primeiro e obviamente, a realização do login no site (ou registo, caso não tenha conta). Depois poderá proceder à marcação inserindo o seu nome, o tipo de consulta, a hora e a data.

CWES

Neste ponto iremos abordar as definições dos tipos de vulnerabilidades que são apresentadas no nosso site inseguro e devidamente corrigidas/resolvidas na nossa plataforma segura.

CWE-20

CWE-20 trata-se de, quando o servidor/plataforma recebe um ***input*** mas este é recebido ou validado de forma inapropriada, isto é, não corresponde àquilo que era previsto do próprio.

Score: 7,1.

CWE-79

CWE-79, ou ***Cross-Site Scripting***, é uma das vulnerabilidades mais usadas pelos atacantes.

Trata-se de um ataque de injeção de código JavaScript malicioso embutido num site onde será executado pelo utilizador sem que este se aperceba.

Score: 8,4 .

CWE-89

CWE-89, ou como é chamado, ***SQLInjection*** é um tipo de ataque onde ocorre a execução de instruções SQL mal-intencionadas. Normalmente, este tipo de ataques tem como foco o controlo da base de dados, onde, caso seja bem-sucedido, é possível modificar, adicionar, remover registos na base de dados.

Estes ataques são bastante prejudicais à plataforma, pois põe em risco toda a informação dos utilizadores.

Score: 9,8.

CWE-256

CWE-256 consiste em guardar as palavra-passes dos utilizadores em texto simples. Assim, caso o atacante consiga aceder à base de dados consegue ter facilmente acesso às palavra-passes.

Score: 7,9.

CWE-620

CWE-620, refere-se, na fase de alteração de password, a não verificação da palavra-passe atual. Assim, é possível o atacante alterar a palavra-passe sem ter de ter conhecimento da palavra-passe atual.

Score: 5,2

CWE-756

CWE-756 refere-se à falta de uma ***error page*** , isto é, uma página para onde o utilizador é redirecionado assim que ocorre um erro no servidor. O grande problema acontece quando: com a falta desta página, quem implementa a página de erro é a biblioteca usada pelo backend, e, com isto, é mostrada informação sensível(como ***query***, nome das tabelas, etc).

Score: 8,6

Score Total - 47

Resolução das Vulnerabilidades

Neste ponto iremos detalhar a resolução das vulnerabilidades que são descritas acima.

CWE-20

Para a resolução desta vulnerabilidade basta colocar o input devidamente correto ao pretendido. No caso do nosso site inseguro, temos o input relativo à hora da consulta implementado em forma de texto, e no site seguro temos como tipo de input ***time*** .

Para esta realização decidimos implementar um ***if else*** na nossa página html com uma variável ***safe*** onde no servidor seguro vai ser **True** e no servidor inseguro vai ser **False.**

Trecho baseado no nosso HTML:

```
<div class = "form-group">
      {% if safe%}
      <input class = "timepicker" type = "time"/>
      {% else %}
      <input type = "text"/>
      {% endif %}
</div>
```

CWE-79

Com a utilização do ***Flask***, o ***Cross-Site Scripting*** revela-se já protegido. No entanto, a caso de exemplo de implementação decidimos tornar o site inseguro de forma a conseguirmos fazer uma implementação de XSS.

```
{% autoescape false %}
    Content
{% endautoescape %}
```

O ***autoescape*** é um sistema implementado no Flask que defende o site contra ataques CWE-79. Portanto, a nível de teste, desativamos este sistema com um pedaço de código como o de cima para que o nosso site inseguro "permita" XSS.

CWE-89

Em vista a utilização do ***Flask*** na nossa plataforma, tudo o que precisamos de fazer para prevenir ***SQLInjection*** foi alterar a chamada das variáveis quando fazemos ***querys***.

No nosso servidor inseguro chamamos as variáveis da seguinte forma:

```
cur.execute(f'SELECT * FROM login WHERE username = "{user}" AND password = "{key}";')
```

A chamada das variáveis é feita de forma "manual", isto é, o que se escrever no input do ***user*** ou da ***key*** é escrito de forma direta na query.

No servidor seguro fazemos a chamada da variável da seguinte forma:

```
cur.execute('SELECT * FROM logim WHERE username = ? AND password = ?;', (user,key))
```

Assim com a alteração para: ***?,*** o próprio ***Flask*** previne possíveis ***SQLInjection***.

Nota : Caso não se utilize o framework Flask o necessário a fazer seria chamar a variável seguido de:

tilte.replace("\'","'")

Assim desta forma, não existe qualquer tipo de possibilidade de ***SQLInjection*** na plataforma.

CWE-256

Para a resolução deste problema, tivemos de encriptar as palavras-passe. Portanto, quando as palavras-passes são inseridas na base de dados já estão encriptadas de forma a prevenir e a defender os dados dos utilizadores.

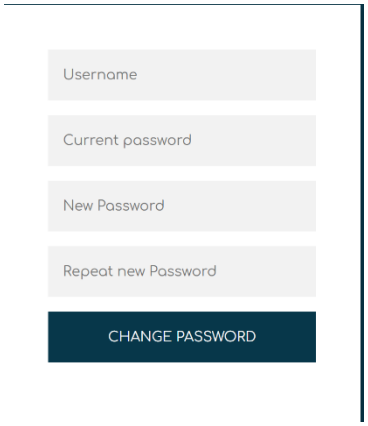
Assim que é feito o registo fazemos a encriptação da palavra-passe da seguinte forma:

```
# import the library
from werkzeug.security import check_password_hash
hashpass = generate_password_hash(password)
```

Assim quando o ***hashpass*** é enviado para a base de dados já é feito de forma encriptada.

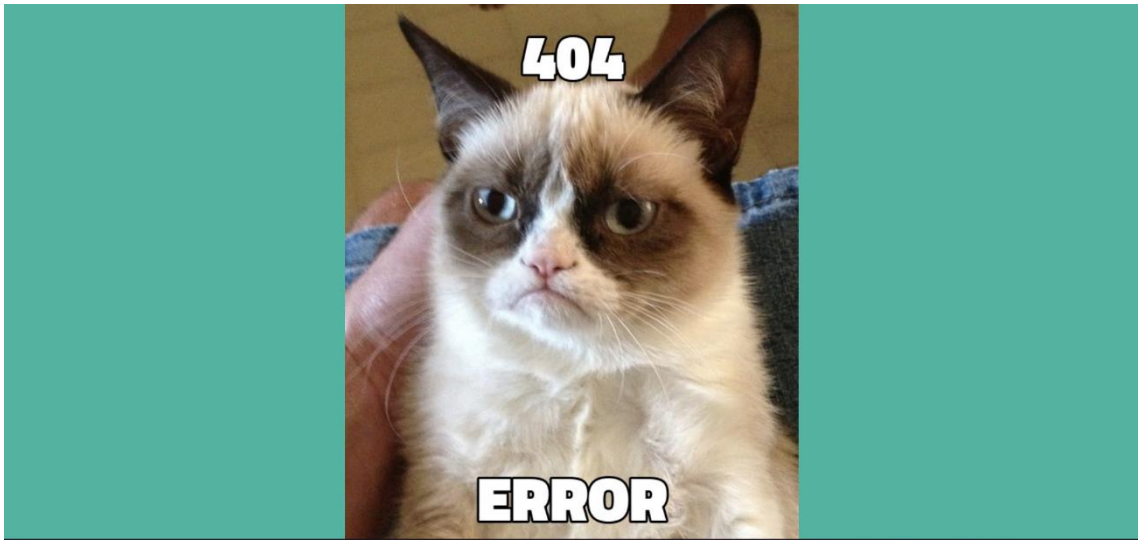
CWE-620

A resolução deste tipo de ataque é bastante fácil. Para resolver o problema basta apenas, assim que o utilizador quiser alterar a sua palavra-passe, requerer a sua password atual, assim o utilizador é obrigado a utilizar a palavra-passe correspondente ao utilizador.



CWE-756

Para resolver este tipo de problema, basta criar uma página de erro, que, no nosso caso, é a seguinte:



Depois de criar esta página criamos funções para que, quando houvesse um erro no servidor 500,404 chamasse esta página de forma a não mostrar conteúdo sensível da nossa base de dados.

A função que usamos para chamar a página de erro foram, a título de exemplo, a seguinte:

```
     @app.errorhandler(num_erro)
     def internal_error(error):
          return render_template("error.html",top = num_erro,
bottom = escape("Error ")),num_erro
```

Bibliografia

Templates HTML/CSS:

https://codepen.io/colorlib/pen/rxddKy?editors=0011

https://github.com/sonorangirl/web-development-guide

https://codepen.io/baahubali92/pen/vvvraZ

Tutorial Flask:

https://www.javatpoint.com/