# 1º Projeto

## Segurança Informática e nas Organizações

Universidade de Aveiro

Guilherme Martins, Inês Moreira,
Lara Rodrigues, Ricardo Machado

# 1º Projeto

## Segurança Informática e nas Organizações

**DETI**

Universidade de Aveiro

Guilherme Martins, Inês Moreira,
Lara Rodrigues, Ricardo Machado
(102587) gcmartins@ua.pt, (100084) inesfm@ua.pt,
(93427) laravieirarodrigues@ua.pt, (102737) ricardo.machado@ua.pt

Novembro 2022

# Índice

# Capítulo 2

# Introdução

Este relatório surge no âmbito do primeiro projeto da unidade curricular de Segurança Informática e nas Organizações. O projeto consiste na criação de uma página web simples de uma clínica de saúde, na qual teríamos de criar um front-end que mostre serviços básicos e o back-end desenvolvido com qualquer funcionalidade relevante para essa finalidade. No entanto, este aplicativo também deve sofrer de um conjunto específico de vulnerabilidades, que não são óbvios para o usuário casual, mas podem ser usados para comprometer o aplicativo ou o sistema. Deve ainda ser entregue uma versão da aplicação totalmente segura e com todas as vulnerabilidades corrigidas.

O enunciado do projeto, assim como, o código desenvolvido encontram-se no seguinte repositório do github : https://github.com/detiuaveiro/assignment-1—vulnerable-ehealth-application-grupo_29.

Os comandos para correr o projeto encontram-se no ficheiro README dentro do repositório.

# Capítulo 3

# Vulnerabilidades

## 3.1 CWE-79 (Score - 5,73)

Os ataques Cross-Site Scripting (XSS) são um tipo de injeção, na qual scripts maliciosos são injetados em sites benignos e confiáveis. Os ataques XSS ocorrem quando um invasor usa um aplicativo da Web para enviar um código malicioso, geralmente na forma de um script do lado do navegador, para um usuário final diferente. As falhas que permitem que esses ataques sejam bem-sucedidos são bastante difundidas e ocorrem em qualquer lugar em que um aplicativo da Web use a entrada de um usuário na saída que ele gera sem validá-lo ou codificá-lo.

## 3.2 CWE-89 (Score - 8.66)

A injeção de SQL é um tipo de vulnerabilidade na qual o invasor adiciona código SQL (Structured Query Language) a uma caixa de entrada da página Web para obter acesso a recursos ou fazer alterações nos dados. Um SQL Query é uma solicitação de alguma ação a ser executada diretamente num banco de dados.

Normalmente, num formulário da página Web para autenticação do utilizador (login), quando um utilizador insere seu nome e password nas caixas de texto fornecidas para tal, esses valores são inseridos num SELECT query. Se os valores inseridos forem encontrados conforme o esperado, o utilizador terá acesso permitido.

No entanto, a maioria dos formulários da Web não possui mecanismos para bloquear entradas que não sejam nomes e senhas. A menos que tais precauções sejam tomadas, um invasor pode usar as caixas de entrada para enviar sua própria solicitação ao banco de dados, o que pode permitir que ele interaja com ele de maneiras ilícitas.

## 3.3 CWE-256 (Score - 4)

Os problemas de gestão de password ocorrem quando uma é armazenada em texto sem formatação. Armazenar uma password de texto simples em um arquivo de configuração permite que qualquer pessoa que possa ler o arquivo acesse o recurso protegido por password. Em alguns contextos, até mesmo o armazenamento de uma password de texto simples na memória é considerado um risco de segurança se a senha não for limpa imediatamente após ser usada.

## 3.4 CWE-306 (Score - 8)

Essa vulnerabilidade leva à incapacidade do software de realizar a identificação da funcionalidade. Evitando uma autenticação, os invasores podem obter acesso a quaisquer dados confidenciais, cumprir funções administrativas e até mesmo executar um código arbitrário. O utilizador pode identificar a vulnerabilidade apenas usando análise manual, por exemplo, teste de penetração, modelagem de ameaças e modificação de sessão ativa.

## 3.5 CWE-425 (Score - 5)

A página Web não é capaz de conduzir a conformidade adequada de autorização limitada de URLs, scripts ou arquivos e aplica a autorização apenas em determinados pontos no caminho que permite aos invasores obter privilégios, ler e modificar dados do aplicativo e executar casos ou comandos.

## 3.6 CWE-521 (Score - 5)

Não há exigência de que os usuários tenham senhas fortes, o que torna mais fácil para os invasores comprometerem as contas dos usuários. Os mecanismos de autenticação geralmente dependem de um segredo memorizado (também conhecido como password) para fornecer uma afirmação de identidade para um utilizador do sistema. Portanto, é importante que essa senha seja de complexidade suficiente e impraticável para um adversário adivinhar.

## 3.7 CWE-549 (Score - 5)

O software falha em tapar senhas durante o login, aumentando o potencial de invasores observarem e capturarem senhas.

# Capítulo 4

# Solução para as Vulnerabilidades

## 4.1 CWE-79

Uma maneira de colmatar este tipo de ataque, que apresenta muitas semelhanças com as soluções apresentadas aos os vários tipos de ataques XSS que estamos a abordar neste projeto, é neutralizando as tags HTML. O que está implementado é a troca dos símbolos que representam uma HTML tag, '<', '>', pelos respetivos char codes ou entity names. Desta forma, quando o DOM processar a notícia publicada, já não vai criar HTML tags porque vai encontrar '&lt', '&gt' no sítio desses carateres especiais. Esta conversão de caracters pode ser feita recorrendo à função de PHP htmlspecialchars(), usada também na solução ao ataque anterior.

```php
$fullname = $_POST['fullname'];
$email = $_POST['email'];
$date = $_POST['date'];
$departement = $_POST['departement'];
$number = $_POST['number'];
$message = $_POST['message'];
```

Figura 4.1: Vulnerabilidade CWE-79 não solucionada

```php
$fullname = htmlspecialchars($_POST['fullname'], ENT_QUOTES);
$email = htmlspecialchars($_POST['email'], ENT_QUOTES);
$date = htmlspecialchars($_POST['date'], ENT_QUOTES);
$departement = htmlspecialchars($_POST['departement'], ENT_QUOTES);
$number = htmlspecialchars($_POST['number'], ENT_QUOTES);
$message = htmlspecialchars($_POST['message'], ENT_QUOTES);
```

Figura 4.2: Solução à vulnerabilidade CWE-79

## 4.2 CWE-89

De forma a prevenir que o utilizador possa executar SQL queries a partir dos inputs que lhe são apresentados na altura de iniciar sessão, é usada uma técnica de prepared statements aquando das chamadas à base de dados. Desta forma, se o utilizador tentar iniciar sessão com uma injeção de SQL, esse input é interpretado como o nome do utilizador e não como código SQL, o que causa um erro.

```php
$sql = "SELECT * FROM users_sec WHERE username=?;";
$stmt = mysqli_stmt_init($conn);
if (!mysqli_stmt_prepare($stmt, $sql)) {
    header("Location: login.php?submit=error");
    exit();
}
else {
    mysqli_stmt_bind_param($stmt, 's', $username);
    mysqli_stmt_execute($stmt);
```

Figura 4.3: Solução à vulnerabilidade CWE-89

## 4.3 CWE-521

A solução para esta vulnerabilidade é implementar uma série de verificações de segurança para as quais a palavra-passe que o utilizador escolhe tem de passar. No caso da versão segura da nossa app, optámos por implementar 3 requisitos obrigatórios para tornar as palavras-passe dos utilizadores o mais seguras possível. São elas:

- A palavra-passe tem de ter pelo menos 8 caracteres;
- A palavra-passe tem de ter pelo menos uma letra maiúscula;
- A palavra-passe tem de ter pelo menos um símbolo/caracter especial.

```php
}else if(strlen($pwd) < 8 || !preg_match('/[A-Z]/', $pwd) ||
        !preg_match('/[\'^£$%&*()}!{@#~?><>,|=_+¬-]/', $pwd)){
    header("Location: signup.php?submit=pwdnotvalid&username=".$username."&email=".$email);
    exit();
}
```

Figura 4.4: Solução à vulnerabilidade CWE-521

## 4.4 CWE-306 e CWE-425

A solução para estas CWE's é bastante evidente e auto-explicativa. É necessário portanto implementar mecanismos que verifiquem se o utilizador está autenticado ou não para executar essa determinada ação crítica, mesmo que a página que permita essa funcionalidade já esteja aparentemente protegida e segura. Para isso, implementamos a seguinte linha de código imediatamente no início das páginas que sofrem desta vulnerabilidade:

```php
<?php
    require '../php/check-session.php';
?>
```

Figura 4.5: Solução às vulnerabilidades CWE-306 e CWE-425

```php
<?php

session_start();

// if not yet in session then go to login
if(!isset($_SESSION["userId"])){
    header("Location: login.php");
    exit();
}
```

Figura 4.6: Código presente em check-session.php

## 4.5 CWE-549

Tudo que é necessário para prevenir esta vulnerabilidade são apenas 15 caracteres: type="password". Isto vai camuflar as credenciais introduzidas pelo utilizador e vai substituir cada caracter por um ponto/círculo preto.

## 4.6 CWE-256

Para proteger as passwords dos utilizadores, é implementada a função password_hash() para a encriptação da mesma.

```
mysqli_stmt_bind_param($stmt, 'sss', $username, $email, password_hash($pwd, PASSWORD_DEFAULT));
mysqli_stmt_execute($stmt);
```

Figura 4.7: Solução à vulnerabilidade CWE-256

# Capítulo 5

# Soma da pontuação CVSS de todas as vulnerabilidades

| Vulnerabilidades | Pontuação |
|---|---|
| CWE-79 | 5,73 |
| CWE-89 | 8,66 |
| CWE-256 | 4 |
| CWE-306 | 8 |
| CWE-425 | 5 |
| CWE-521 | 5 |
| CWE-549 | 5 |
| **Soma total** | **41,39** |