

1º projeto

Avaliação e exploração de vulnerabilidade



universidade
de aveiro

[Alfredo Matos](#)
[Vitor Cunha](#)
[Paulo Bartolomeu](#)

Guilherme Lopes (103896)
Gabriel Teixeira (107876)
Pedro Rodrigues (102431)

Índice

Introdução.....	2
Tecnologias usadas.....	3
(1) MYSQL.....	3
(2) PHP.....	3
(3) Apache.....	3
Vulnerabilidades.....	4
CWE-89: Improper Neutralization of Special Elements used in an SQL Command.....	4
CWE-79: Improper Neutralization of Input During Web Page Generation.....	8
CWE-352: Cross-Site Request Forgery (CSRF).....	9
CWE-22: Improper Limitation of a Pathname to a Restricted Directory.....	10
CWE-20: Improper Input Validation.....	11
CWE-311: Missing Encryption of Sensitive Data.....	12
CWE-434: Unrestricted Upload of File with Dangerous Type.....	14
CWE-839: Numeric Range Comparison Without Minimum Check.....	15
CWE-863: Incorrect Authorization.....	16
Conclusão.....	17
Bibliografia.....	18

Introdução

Este documento tem como objetivo apresentar as vulnerabilidades identificadas em um website desenvolvido pelo nosso grupo, juntamente com suas respectivas correções. O projeto consistiu na criação de dois ambientes: um site inseguro e outro seguro, ambos com o tema de uma loja denominada DetiShop.

Neste relatório, abordaremos as vulnerabilidades encontradas no site inseguro e descreveremos as soluções adotadas para tornar o site seguro. Essa análise visa não apenas identificar os problemas de segurança, mas também destacar as melhores práticas para proteger uma aplicação web

Tecnologias usadas

(1) MYSQL

- ☐ O sistema de gerenciamento de banco de dados MySQL é uma parte fundamental do nosso projeto. Ele atua como o repositório de dados para a loja online DetiShop. O MySQL é um sistema de banco de dados relacional que organiza dados em tabelas relacionadas. Ele nos permite armazenar informações sobre produtos, pedidos, clientes e muito mais. Utilizamos a linguagem SQL para inserir, recuperar e modificar dados de maneira eficaz.

(2) PHP

- ☐ A linguagem de programação PHP desempenha um papel crucial na criação da lógica dinâmica do nosso site DetiShop. Ele é executado no servidor e é responsável por processar solicitações dos utilizador, gerar páginas da web dinâmicas e interagir com o banco de dados MySQL. Com o PHP, podemos criar páginas personalizadas, gerar conteúdo com base nas preferências dos clientes e garantir uma experiência interativa e dinâmica para os visitantes do site.

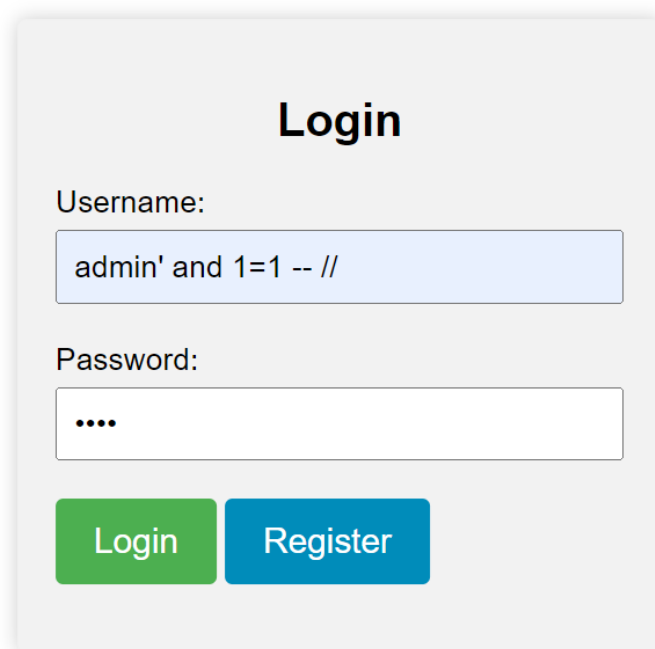
(3) Apache

- ☐ O servidor web Apache é o servidor HTTP escolhido para hospedar o nosso site. Ele atua como intermediário entre os navegadores dos utilizador e o nosso aplicativo web. O como é altamente configurável e suporta recursos essenciais de segurança, autenticação e manipulação de solicitações HTTP. Ele é responsável por servir o conteúdo gerado pelo PHP aos visitantes do site DetiShop de forma eficiente e segura.

Vulnerabilidades

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

Esta vulnerabilidade de injeção SQL ocorre quando um aplicativo da web permite que um invasor insira comandos SQL maliciosos em campos de entrada, como neste caso no formulário de login, permitindo que o invasor acesse informações, faça login como utilizador ou administrador, ou prejudique o sistema. Isso ocorre devido à falta de validação adequada das entradas do utilizador. Para mitigar essa vulnerabilidade, os desenvolvedores devem validar e sanitizar as entradas e utilizar consultas parametrizadas.



The image shows a login form with the title "Login" in bold. Below the title, there are two input fields. The first field is labeled "Username:" and contains the text "admin' and 1=1 -- //". The second field is labeled "Password:" and contains three dots "....". Below the input fields, there are two buttons: a green button labeled "Login" and a blue button labeled "Register".

exemplo de ataque em (SQL)

Código Inseguro:

```
$username = $_POST['username'];
$password = $_POST['password'];
//unsafe
$sql = "SELECT * FROM user WHERE nome = '$username' AND pass = '$password'";
$result = mysqli_query($con, $sql);
```

(Img 1)

A principal razão pela qual este código é inseguro é a concatenação direta de dados não tratados do utilizador em uma consulta SQL. Isso pode permitir que um atacante execute consultas maliciosas no banco de dados.

Código seguro:

```
$row = mysqli_fetch_array($result);
if(mysqli_num_rows($result) == 1 && password_verify($password, $row['pass'])){ ...
}else{
    echo "Login failed";
}
```

(Img 2)

```
$stmt = mysqli_prepare($con, "SELECT * FROM user WHERE nome = ?");
mysqli_stmt_bind_param($stmt, "s", $username);
mysqli_stmt_execute($stmt);
$result = mysqli_stmt_get_result($stmt);
```

(Img 3)

Na imagem 2 (Img 2) as variáveis de sessão são utilizadas para autenticação, o que é uma prática segura, ao mesmo tempo que se verifica se a consulta SQL retornou exatamente uma linha de resultado, garantindo a existência do utilizador no banco de dados.

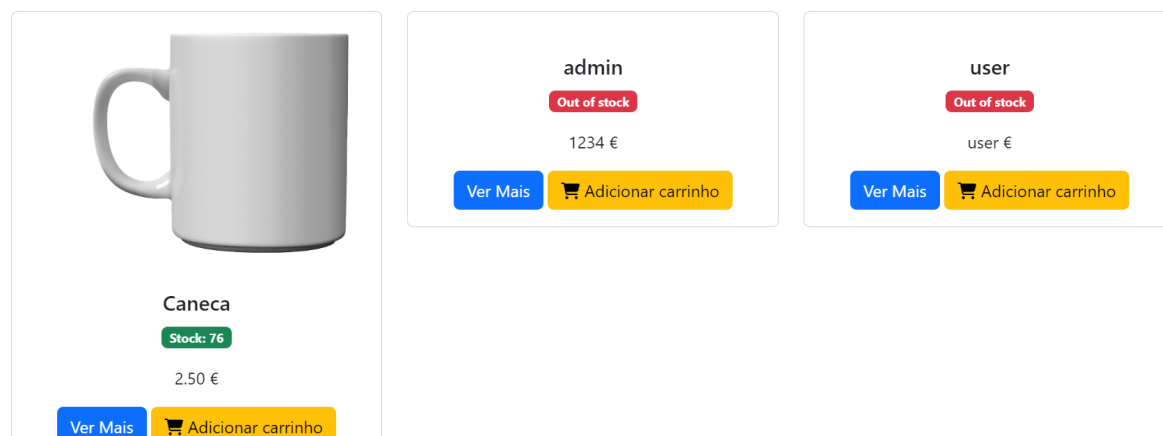
É usado a função **password_verify()** serve para comparar a senha fornecida com a senha armazenada no banco de dados, protegendo as senhas com cifragem segura. Isso garante que apenas utilizadores válidos com senhas corretas possam acessar o sistema.

Na imagem 3 (Img 3) o código utiliza declarações preparadas e associação segura de parâmetros para garantir que os valores do utilizador sejam tratados como dados e não como parte da consulta SQL. Isso previne com eficácia ataques de injeção SQL, uma vez que os valores do utilizador não têm a oportunidade de alterar a estrutura da consulta.

Welcome to my Detishop

Here you can find the products for the best prices

Categorias



(Img 4)

A vulnerabilidade "Search Pass and Username" ocorre por meio de injeção SQL ocorre quando um invasor manipula campos de entrada em um sistema, como formulários de login, inserindo comandos SQL maliciosos. Isso pode permitir que o invasor acesse senhas e nomes de utilizador armazenados no banco de dados, obtendo acesso não autorizado a contas. Para mitigar essa vulnerabilidade, é fundamental validar entradas, usar consultas parametrizadas e seguir boas práticas de segurança de aplicativos.

Código não seguro:

```
$sql = "SELECT * FROM products";
if (isset($_GET['category'])) {
    $cat = $_GET['category'];
    $sql = "SELECT * FROM products WHERE category = '$cat'";
}
```

(Img 5)

No exemplo de código a seguir simplesmente insere o valor da variável “\$cat” diretamente na consulta SQL, onde pode permitir que o atacante injete comandos indesejados tornando inseguro e pode resultar em comprometimento da base de dados, ao mesmo tempo que executa uma consulta Dinâmica em SQL diretamente com base em dados fornecidos pelo utilizador. Em vez disso, você deve usar consultas preparadas ou funções de escape para evitar a injeção de SQL.

Código seguro:

```
$stmt = mysqli_prepare($con, "SELECT * FROM products");  
if (isset($_GET['category'])) {  
    $cat = $_GET['category'];  
    $stmt = mysqli_prepare($con, "SELECT * FROM products WHERE category = ?");  
    mysqli_stmt_bind_param($stmt, "s", $cat);  
}
```

(Img 6)

Foi desenvolvido com consultas preparadas, isto é uma técnica de execução de consultas SQL , com segunda em PHP.Com isso conseguimos separar os dados do SQL real,permitindo que a base de dados lide com os dados de forma segura representado por “\$stmt”.

Na última linha de código “*mysqli_stmt_bind_param*” o “s” indica que “ \$cat ” é uma string, onde conseguimos evitar injeções de SQL.

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

O Cross-Site Scripting (XSS) é uma vulnerabilidade comum em aplicações web que permite a inserção de código JavaScript malicioso. Isso pode levar a roubo de informações e ações não autorizadas. Para prevenir o XSS, é essencial validar entradas, escanear dados de saída e aplicar medidas de segurança.

Como no exemplo a seguir:

Adicionar uma Avaliação

Comentário:

Avaliação (1-5):

(Img 7)

Seu efeito gerado:



(Img 8)

Código do Site não seguro:

```
<div class="card my-2 p-3">
  <p class="mb-0"><strong>Review:</strong> <?= $review['comment'] ?></p>
  <p class="mb-0"><strong>Rating:</strong> <?= $review['rating'] ?></p>
  <p class="mb-0"><strong>Author:</strong> <?= $review['nome'] ?></p>
</div>
```

(Img 9)

Código do site seguro:

```
<div class="card my-2 p-3">
  <p class="mb-0"><strong>Review:</strong> <?= htmlentities($review['comment']) ?></p>
  <p class="mb-0"><strong>Rating:</strong> <?= htmlentities($review['rating']) ?></p>
  <p class="mb-0"><strong>Author:</strong> <?= htmlentities($review['nome']) ?></p>
</div>
```

(Img 10)

Prevenção contra Cross-Site Scripting (XSS):

Ao exibir dados na página, o código utiliza a função **htmlspecialchars()** para escapar os dados, tornando-os seguros para exibição no navegador. XSS é um tipo de ataque em que scripts maliciosos são injetados em páginas da web e executados no navegador dos utilizador. htmlspecialchars() previne esse tipo de ataque, garantindo que qualquer código JavaScript ou HTML injetado seja tratado como texto simples e não seja interpretado pelo navegador.

CWE-352: Cross-Site Request Forgery (CSRF)

É quando nós conseguimos adicionar uma imagem de outra fonte de informação. Este ataque é possível porque os navegadores enviam automaticamente cookies de autenticação ao site alvo. Como por exemplo aqui em baixo:

Adicionar uma Avaliação

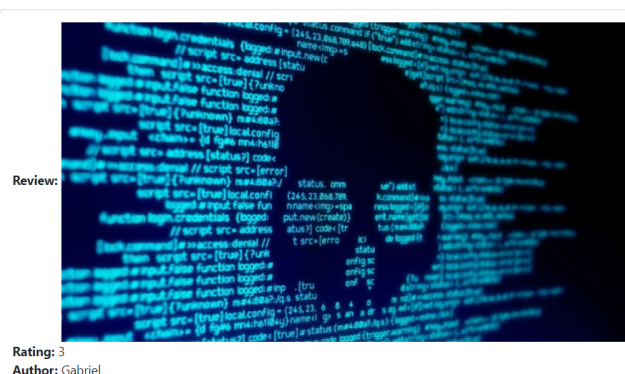
Comentário:

Avaliação (1-5):

5

Enviar Avaliação

E com isso concluímos que pode ser um pouco perigoso porque nós conseguimos adicionar qualquer coisa que queríamos ao nosso site:



CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')


Esta vulnerabilidade tem uma aplicação para acessar diretórios e arquivos que normalmente não seriam acessíveis. Esse tipo de ataque geralmente ocorre em aplicações web que não validam corretamente as entradas do utilizador usadas para construir caminhos de arquivo ou URLs. Quando um atacante consegue explorar uma falha de segurança e acessar diretórios sensíveis ou arquivos críticos, ele pode comprometer a segurança do sistema, acessar informações confidenciais ou até mesmo executar código malicioso.

Exemplo da vulnerabilidade inserida nos comentários de um produto:

```
<a href="images"></a>
or
<a href="images">Olá foste hackeado</a>
or
<a href="images"></a>
<a href="images"><img class=""></a>
```

(Img 11)

Exemplo como se apresenta no site:

Review:  or [Olá foste hackeado](#) or ☐






Rating: 5

Author: user

(Img 12)

Exemplo do que consegue obter:

Index of /detishop/images

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 hoodie.png	2023-10-10 14:55	265K	
 laptop_bag.png	2023-10-10 14:55	448K	
 mug.png	2023-10-10 14:55	101K	
 tshirt.png	2023-10-10 14:55	19K	
 usb_pen.png	2023-10-25 15:13	596K	

(Img 13)

Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4 Server at localhost Port 80

Como protegemos contra esse ataque:

Usamos a função **htmlspecialchars()** é uma função em PHP que converte caracteres especiais em entidades HTML. Isso evita problemas de segurança, como ataques de Cross-Site Scripting (XSS), ao escapar de caracteres que podem ser interpretados como código HTML ou JavaScript quando renderizados em uma página da web. Por exemplo, converte o caractere < em < e o caractere > em >, evitando que o utilizador obtenha informações confidenciais.

CWE-20: Improper Input Validation

Login

Username:

Password:

Adicionar uma Avaliação

Comentário:

Avaliação (1-5):

Insert Product

Product Name:

Product Price:

Product Description:

Product Category:

Product Stock:

O código valida se os campos do formulário de revisão são preenchidos (required no HTML) antes de aceitar o envio. Isso garante que os dados necessários sejam fornecidos antes de serem processados, ajudando a manter a integridade dos dados no banco de dados.

CWE-311: Missing Encryption of Sensitive Data

Esta vulnerabilidade existe quando informação sensível, neste caso as passwords dos utilizadores não são armazenadas de forma segura. O que as torna acessíveis a potenciais ataques e a acessos não autorizados.

A mitigação desta vulnerabilidade passa por armazenar a informação de forma cifrada, adotar políticas de hash seguros e a obrigatoriedade de passwords fortes.

Inicialmente, asseguramos que a senha tenha pelo menos 8 caracteres, garantindo um requisito mínimo de comprimento. Em seguida, utilizamos a função **password_hash()**, uma função nativa do PHP projetada para simplificar a criação de hashes seguros. A principal vantagem dessa função é sua capacidade de escolher automaticamente um algoritmo de hash seguro e, ao mesmo tempo, criar um salt, um valor aleatório, para cada hash gerado. Tornando os hashes únicos, mesmo quando senhas idênticas são processadas, aumentando significativamente a segurança e a integridade dos dados sensíveis.

No processo de login é utilizada a função **password_verify()**, que compara a pass introduzida com o hash armazenado na database.

id	nome	email	admin	pass
8	admin	admin@detiua.pt	1	admin
9	teste	suport@cx.com	0	1234
13	joao	oo@ff.pt	0	ouououo
15	antonia	antaa@alo.py	0	passw0rd
16	ana maria	anam@ff.pt	0	benfica
17	marcelo	rm@v.t	1	b8

base de dados com a password dos utilizadores desprotegida

id	nome	email	admin	pass
8	admin	admin@detiua.pt	1	\$2y\$10\$nlIRVQhQo/SJs7xIX0ggEO2E28HXps.4eTsnjkBmAR2...
9	teste	suport@cx.com	0	\$2y\$10\$Phe84Rme33K8a6ZgE3bi.FZ0bEkRhqChkzpxlaTyB0...
13	joao	oo@ff.pt	0	\$2y\$10\$8RCyEsqJT2CHhJrghCuGj.ZSsUESScKQ0BSvo4K2RYi...
15	antonia	antaa@alo.py	0	\$2y\$10\$geflBn8sTLBRcVjDAWpRgePh3f39FCOggyJ2HlwMR8....
16	ana maria	anam@ff.pt	0	\$2y\$10\$prJ8ikHIOFRRZuCSlhAD4Owe7DcQAmXFkPWPDraA/2UX...
17	marcelo	rm@v.t	1	\$2y\$10\$01Ax1KnZGbDxTppB1CiXmeZFT3ad82O8FDQSgh4FtJb...

base de dados com a password dos utilizadores cifrada

```

}
if (empty($password)) {
    $errors[] = 'Password is required';
} elseif (strlen($password) < 8) {
    $errors[] = 'Password must be at least 8 characters long';
}
if ($password != $confirm_password) {
    $errors[] = 'Passwords do not match';
}

```

Verificação do tamanho da password por motivos de segurança

```

// Set the parameters and execute the statement
$username = $_POST['username'];
$email = $_POST['email'];
$hashed_password = password_hash($_POST['password'], PASSWORD_DEFAULT);

$stmt->execute();

```

password_hash() cria a hash da password

CWE-434: Unrestricted Upload of File with Dangerous Type

Quando o sistema não faz restrição aos tipos de ficheiros que podem ser inseridos, abre a possibilidade de com a ataque inserir ficheiros perigosos que podem ser executados automaticamente pelo sistema. No nosso sistema a inserção de ficheiro é permitida a utilizadores com permissão de administrador, na criação de um novo produto é pedida uma imagem do mesmo, mas o sistema não verifica se o ficheiro é realmente uma imagem.

A solução desta vulnerabilidade passa por restringir quem pode fazer o upload, limitar o tamanho do ficheiro e definir os tipos de ficheiros permitidos, indicando as extensões permitidas. No nosso caso o upload de ficheiros ocorre apenas na criação ou atualização de um produto, onde é necessário uma imagem do mesmo.

Para verificar se o ficheiro é realmente uma imagem utilizamos o accept que é um elemento html que especifica o tipo de ficheiros que podem ser seleccionados. No entanto, não substitui uma verificação mais robusta por parte do sistema.

```
<div class="me-3">
  <label for="product_img" class="form-label">Product Image:</label>

  <input type="file" class="form-control" id="product_img" name="product_img" required accept="image/*">
</div>
<button type="submit" class="btn btn-primary" name="add_product">Add Product</button>
</div>
```

accept - elemento html

```
$product_stock = $_POST['product_stock'];
$product_img = $_FILES['product_img']['name'];
$img_path = '/detishop/images/' . $product_img;
$target_dir = "images/";
$target_file = $target_dir . basename($_FILES["product_img"]["name"]);
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
$extensions_arr = array("jpg","jpeg","png","gif");
if( in_array($imageFileType,$extensions_arr) ){
    $insert_product = "INSERT INTO products (name, price, descr, stock, img, category)
    mysqli_query($con, $insert_product);
    move_uploaded_file($_FILES['product_img']['tmp_name'],$target_dir.$product_img);
    header("Location: index.php");
}
```

verificação do tipo de ficheiro antes de inserir produto na database

CWE-839: Numeric Range Comparison Without Minimum Check

Esta vulnerabilidade acontece quando o sistema faz a verificação em relação ao valor permitido, mas não verifica se o valor é maior ou igual ao valor mínimo.

No caso da loja cada utilizador está limitado à compra de 10 itens de cada produto mas o facto de a quantidade mínima não ser verificada leva à possibilidade de inserir uma quantidade negativa de produtos, que leva a vários danos ao sistema. Como a diminuição do valor total da compra ou até mesmo de um valor negativo e ao aumento “artificial” do stock dos produtos com quantidades negativas.

Para resolver esta vulnerabilidade basta adicionar uma verificação para a quantidade mínima de um produto, que neste caso será 1, sendo que o utilizador pode remover o produto do carrinho com um botão.



Carrinho de utilizador com quantidades negativas

```
var newQuantity;
if (action === 'increase' && currentQuantity < 10) {
  // Aumentar a quantidade
  newQuantity = currentQuantity + 1;
  total += price;
  document.getElementById('cart-count').innerText = parseInt(document.getElementById('cart-count').innerText) + 1;
} else if (action === 'decrease' && currentQuantity > 1) {
  // Diminuir a quantidade (não permitir quantidade menor que 1)
  newQuantity = currentQuantity - 1;
  total -= price;
  document.getElementById('cart-count').innerText = parseInt(document.getElementById('cart-count').innerText) - 1;
} else {
  return; // Não faz nada se a ação for inválida
}
```

Verificação da quantidade atual do produto no carrinho

CWE-863: Incorrect Authorization

A autorização incorreta permite que utilizadores sem permissões acessem a zonas restritas que não deviam ter permissão de aceder. O que pode resultar no acesso e alteração não autorizada de dados, comprometendo a segurança do sistema. Esta vulnerabilidade ocorre quando as políticas de controlo de acesso não são aplicadas adequadamente.

Numa loja online apenas utilizadores com privilégios devem poder editar ou adicionar um produto. Na nossa loja, quando um administrador visita um produto, encontra um botão que o redireciona para uma página para editar o produto. Um utilizador normal não tem acesso a esse botão; mas se no url alterar o produto.php por update.php consegue aceder à página de atualizar o produto que estava a visitar. A autenticação apenas verifica se existe um utilizador conectado e não verifica os privilégios que este tem, ficando acessível a alteração de informações sensíveis dos produtos como o preço ou o stock a todos os visitantes da loja, desde que tenham um conta.

Para solucionar esta vulnerabilidade basta corrigir a verificação de autenticação, verificando os privilégios de quem quer aceder a esta zona sensível do sistema.

```
if(!isset($_SESSION['user_id'])){  
    header("location: index.php");  
}
```

verifica apenas se o utilizador está conectado

```
if($_SESSION['admin'] != 1){  
    header("location: index.php");  
}
```

verificação dos privilégios do utilizador

Conclusão

Concluimos que a realização deste projeto destacou a importância crítica da segurança da informação em ambientes online. Ao identificar e corrigir diversas vulnerabilidades comuns em aplicações web, como injeção SQL, Cross-Site Scripting (XSS), ataques de Directory Traversal, validação incorreta de entrada e autorização inadequada, aprendemos valiosas lições sobre as melhores práticas de segurança.

Ao implementar medidas de segurança adequadas é essencial para proteger não apenas os dados dos utilizadores, mas também a integridade e a confiabilidade de todo o sistema. Ao seguir práticas recomendadas, como validar e sanitizar entradas, utilizar consultas parametrizadas, adotar funções de escape e `htmlentities()` para prevenir XSS, impor restrições rigorosas no upload de ficheiros e armazenar senhas com hashing seguro, podemos criar aplicações web mais seguras e confiáveis.

Concluimos que a conscientização sobre as vulnerabilidades comuns e a implementação diligente das melhores práticas de segurança são essenciais para mitigar riscos e proteger tanto os utilizadores quanto as organizações contra potenciais ataques cibernéticos. Este projeto serviu como um valioso exercício de aprendizado, proporcionando insights fundamentais sobre como garantir a segurança em ambientes online, contribuindo para um ecossistema digital mais seguro e confiável para todos.

Bibliografia

<https://cwe.mitre.org/data/definitions/863.html>

<https://cwe.mitre.org/data/definitions/89.html>

<https://cwe.mitre.org/data/definitions/79.html>

<https://cwe.mitre.org/data/definitions/311.html>

<https://cwe.mitre.org/data/definitions/434.html>

<https://cwe.mitre.org/data/definitions/839.html>

<https://www.php.net/manual/en/function.password-verify.php>

<https://www.php.net/manual/en/function.password-hash.php>

<https://w3codegenerator.com/code-snippets/php/how-to-check-file-is-an-image-in-php>

<https://www.php.net/manual/en/function.pathinfo.php>