

# **Segurança Informática e nas Operações**

## ***Vulnerabilidades***

Projeto nº1

2021/2022

Artur Romão, nº 98470, P2

Mariana Rosa, nº 98390, P2

Nuno Fahla, nº 97631, P2

Paulo Pereira, nº 98430, P2

# Índice

<b>Índice</b>	<b>2</b>
<b>Introdução</b>	<b>3</b>
<b>Vulnerabilidades</b>	<b>4</b>
CWE-20: Improper Input Validation	4
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	5
CWE-87: Improper Neutralization of Alternate XSS Syntax	6
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	7
CWE-256: Plaintext Storage of a Password	8
CEW-862/863: Missing/Incorrect Authorization	9
CWE-287: Improper Authentication	10
CWE-521: Weak Password Requirements	11
<b>Conclusão</b>	<b>11</b>
<b>Fontes</b>	<b>12</b>

## Introdução

Neste primeiro projeto da disciplina de SIO, cujo objetivo foca-se em encontrar vulnerabilidades (e combatê-las) num sistema informático, o nosso grupo decidiu desenvolver um site de reservas de Alojamento Local: a *Lokals*.

A *Lokals* visa fornecer uma plataforma online e fidedigna de procura e reserva de alojamentos. Existem dois tipos de utilizadores registados/que se podem registar no site: os proprietários, ou *hosts*, que disponibilizam as habitações, e os clientes, que procuram e alugam os imóveis. A *Lokals* funciona como outras plataformas comuns como o Booking e o Airbnb. Existe uma série de alojamentos publicados pelos *hosts*, disponíveis para os restantes utilizadores, que podem reservá-los através do nosso *website*.

Consequentemente, a existência de um *website* com esta funcionalidade, suporta muitos dados importantes associados aos utilizadores, tais como *emails*, *passwords* e até mesmo informações de cartões bancários. Assim, neste projeto, iremos criar duas versões do *website*. Uma com várias vulnerabilidades e outra, segura, em que tentamos combater essas mesmas vulnerabilidades. Neste relatório estará presente a descrição, a aplicação e a prevenção das mesmas.

# Vulnerabilidades

## CWE-20: Improper Input Validation

No mundo da programação e desenvolvimento de aplicações temos de partir do princípio que o utilizador tem muito pouco conhecimento do que fazer/escrever. Esta vulnerabilidade assume que o software mal tenha recebido a informação introduzida pelo utilizador, armazena-a logo na base de dados. Com isto, o atacante pode ser capaz de manipular a informação que entra no sistema, podendo existir várias consequências negativas. Para que isto não aconteça, devem existir validações para assegurar a segurança do sistema. Um exemplo na nossa plataforma é o de formulários de registo na plataforma.

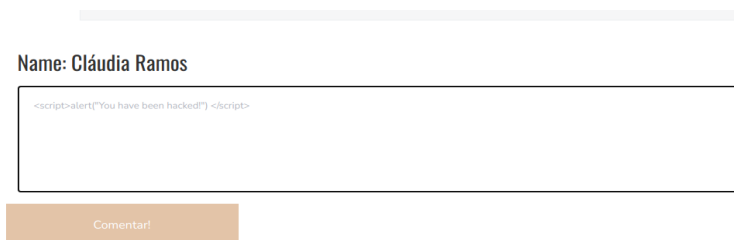
```
$regex = "/^[_a-z0-9-]+(\\.[_a-z0-9-]+)*@[a-z0-9-]+(\\.[a-z0-9-]+)*(\\.[a-z]{2,})$/i"; // regex for email confirmation

if (isset($_POST['confirmar'])) {
    if ((empty($nome) or empty($pass) or empty($mail) or empty($loca) or empty($cell))) {
        $error = "Preencha todos os campos para se registar!";
        echo "<script type='text/javascript'>alert('$error');</script>";
    }
    elseif (!preg_match("/^[a-zA-Z ]*$/", $nome)) {
        $error = "Nome inválido!";
        echo "<script type='text/javascript'>alert('$error');</script>";
    }
    elseif (!preg_match($regex, $mail)) {
        $error = "Insira um e-mail válido!";
        echo "<script type='text/javascript'>alert('$error');</script>";
    }
    elseif (!preg_match("/^\\d+$/", $cell)) {
        $error = "Número de telemóvel inválido!";
        echo "<script type='text/javascript'>alert('$error');</script>";
    }
}
```

Fig. 1 - Exemplo de código para validação dos campos preenchidos pelo utilizador

Neste pedaço de código, avaliamos se os campos estão vazios e/ou mal preenchidos pelo utilizador, isto é, utilizamos as funções “empty” e “preg\_match”, recorrendo a diferentes regex para verificar qualquer anormalidade no conteúdo do input. Na versão insegura, não testamos nenhuma destas validações, pelo que o utilizador pode violar as regras de cada campo (ex. introduzir letras ou caracteres especiais num campo para número de telemóvel, etc).

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



Name: Cláudia Ramos

`<script>alert('You have been hacked!')</script>`

Comentar!

Fig. 2 - Exemplo de XSS numa textbox

Sendo uma das vulnerabilidades mais importantes, o *Cross-site Scripting* (XSS) é um tipo de *injection* que ocorre quando são introduzidos scripts maliciosos nos *websites*. A aplicação obtém as informações fornecidas pelo utilizador e envia de volta ao navegador sem realizar qualquer validação do conteúdo. O XSS permite aos atacantes executarem scripts no navegador da vítima, podem ser scripts tal como este exemplo (Fig. 1 e 2) de um simples alerta na zona de comentários (`<script>alert("You have been hacked!")</script>`), mas também pode ser algo mais robusto para tentar roubar informações do utilizador, como cookies, expor vírus, etc.

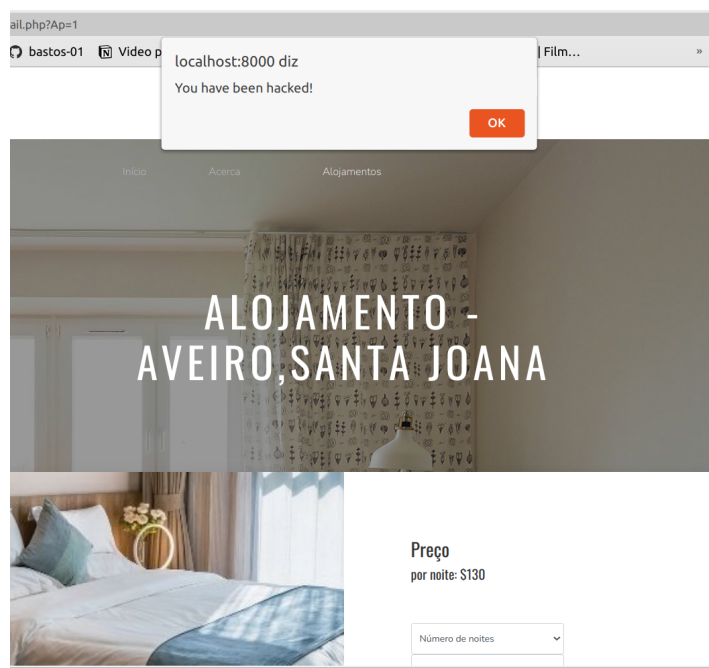


Fig. 3 - Execução de um script com o XSS

Como maneira de prevenir, é necessário verificar que todos os parâmetros da aplicação são validados e/ou recodificados antes de serem incluídos em páginas HTML. Com PHP foi simples de verificar, em todos os inputs onde o utilizador poderia escrever incluímos a seguinte linha de código:

```
#no xss
$comuserName = htmlspecialchars($comuserName_nosql);
$comEmail = htmlspecialchars($comEmail_nosql);
$comCom = htmlspecialchars($comCom_nosql);
```

Fig. 4 - Função “htmlspecialchars”

A função “htmlspecialchars” transforma todos os caracteres especiais nas suas entidades HTML, ou seja, as aspas (") passam a &quot; e o símbolo & passa a &amp;, isto previne Cross-site scripting, pois os scripts iniciam-se com a tag <script> e os caracteres ">" e "<" são transformados em &gt; e &lt;, respetivamente.

## CWE-87: Improper Neutralization of Alternate XSS

### Syntax

Esta vulnerabilidade é filha da anterior. Implica que o software neutralize alternativas sintáticas para os scripts, como por exemplo:

New Pylance

```
<Script>alert("this is a vulnerability")</Script>
```

Name: New Pylance

Your text

Fig. 5 - Aplicação da CWE-87

Fazendo esta linha de comando, o nosso *website* reage como se fosse um simples comentário, tratando da vulnerabilidade e impedindo o atacante de executar o script na página.

Uma forma possível de prevenir scripts, é não permitir que o campo aceite “script” como parte do seu valor, isto levanta problemas, pois variações na capitalização dão a volta a esta medida de segurança. A função `htmlspecialchars` resolve este problema também.

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

Tal como na vulnerabilidade anterior, uma aplicação que contenha esta CWE, está facilmente exposto a que um atacante consiga executar comandos para manipular qualquer dado disponível na aplicação. O atacante tanto pode entrar numa conta de um utilizador sem ter a palavra-passe, como ter acesso a todas as palavras-passes com um simples comando.

Por exemplo, na nossa versão não segura ao executar o comando `' UNION SELECT userId,password,email,name_,cellphone,country FROM users --//'`, obtém-se a palavra-passe da conta deste utilizador (TL7RY9fmECgOKrl).



Fig. 6 - Exemplo de SQL Injection

Tal como na vulnerabilidade anterior, aqui também conseguimos prevenir apenas com uma linha de código.

```
#No sql injection
$comuserName_nosql =mysqli_real_escape_string($con,$_POST["Nome"]);
$comEmail_nosql =mysqli_real_escape_string($con,$_POST["Email"]);
$comCom_nosql =mysqli_real_escape_string($con,$_POST["Com"]);
```

Fig. 7 - Função `mysqli_real_escape_string`, para combater o SQL Injection

Esta função “escapa” todos os caracteres especiais tornando assim impossível correr comandos sql como “--//” no nosso exemplo.

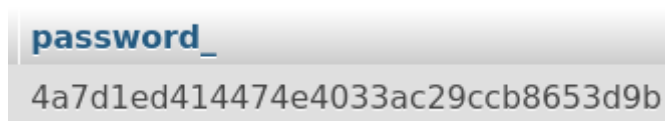
## CWE-256: Plaintext Storage of a Password

Há certas informações dos utilizadores que se têm de proteger ao máximo, tal como as passwords. Desta feita, não podemos armazená-las na base de dados como fazemos com o nome ou o email (como *plaintext*), uma pessoa com más intenções pode facilmente lá chegar.

Para prevenir essa situação, *password hashing* é uma boa solução, onde se encripta a palavra-passe. Na nossa aplicação segura, decidimos utilizar o algoritmo MD5, que gera uma string de 32 caracteres, através de uma função:

```
$pass_nosql =mysqli_real_escape_string($con,md5($_POST["password"]));
$pass = htmlspecialchars($pass_nosql);
```

Fig. 8 - Código responsável pela encriptação das palavras passe



password\_  
4a7d1ed414474e4033ac29ccb8653d9b

Fig. 9 - Exemplo de palavra passe encriptada na base de dados



## CEW-862/863: Missing/Incorrect Authorization

Foram implementadas várias verificações de controlo de acessos. Não é possível, a qualquer utilizador, executar ações exclusivas aos utilizadores que estão registados. O acesso a certos botões é restrito, bem como a certas páginas do *website*. Havendo apenas dois tipos de utilizadores, registados ou não, missing authorization acaba por ser, invalid authorization.

De maneira a resolver este problema, foi incluído, no início do segmento de PHP o seguinte código:

```
if(!isset($_SESSION['loggedin'])){
    header("Location:missing_auth.php");
}
```

Fig. 10 - Segmento de PHP que verifica se existe uma sessão de login ou não

Sendo “*loggedin*” uma variável booleana que indica se a sessão tem um utilizador registado. A página *missing\_auth.php* é reservada para estas situações.

Na página *index.php*, no fundo, existe um botão de candidatura, o qual redireciona para a página *candidatura\_alojamento.php*, no caso do utilizador estar registado e, para a página *candidatura.php*, no caso de não estar. Esta é uma das situações em que existem diferenças dependendo da identidade do utilizador. Este é o código associado para o caso descrito acima:

```
if (isset($_SESSION['loggedin']) && $_SESSION['loggedin'] == true) {
    echo "
    <p style=\"padding-top: 40px; padding-bottom: 30px;\">Deseja expor o seu alojamento no site? Adicione-o agora!</p>
    <p><a href=\"candidatura_alojamento.php\" class=\"btn btn-primary px-4 py-3\">Adicionar Alojamento</a></p>";
} else {
    echo "
    <p style=\"padding-top: 40px; padding-bottom: 30px;\">Deseja expor o seu alojamento no site? Candidate-se agora!</p>
    <p><a href=\"candidatura.php\" class=\"btn btn-primary px-4 py-3\">Junte-se a nós</a></p>";
}
```

Fig. 11 - Alteração do código HTML conforme o valor da variável `$_SESSION['loggedin']`

Deseja expor o seu alojamento no site? Candidate-se agora!

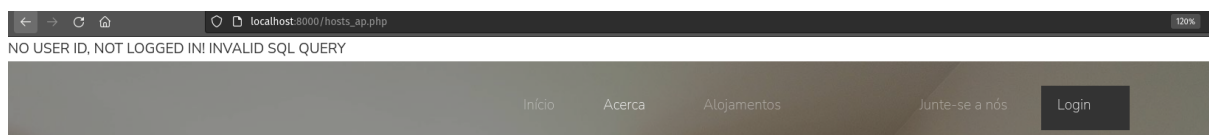
Deseja expor o seu alojamento no site? Adicione-o agora!

Junte-se a nós

Adicionar Alojamento

Fig. 12 e 13 - Botões na página web para diferentes utilizadores. Na figura 12, para utilizadores não registados e, na figura 13, para utilizadores registados.

Na versão não segura, qualquer utilizador que saiba qual link aceder, pode-o fazer e até mesmo preencher o formulário, o que leva a um SQL error devido à falta de userID.



## CWE-287: Improper Authentication

Para aceder a certas páginas do nosso website, a identidade é confirmada de duas maneiras. Verificamos se o utilizador está registado (se o seu *id* consta na tabela de utilizadores) e se efetuou o login corretamente. Na versão insegura, não há este calibre de verificação, na versão segura, caso algo corra mal, o acesso é negado através do seguinte código:

```
if(!isset($_SESSION['loggedin'])){
    header("Location:missing_auth.php");
}else{
    $usrmail = $_SESSION["username_"];
    $query = "SELECT * FROM `users` WHERE `email` LIKE '$usrmail' ";
    $result = mysqli_query($con,$query);
    if(mysqli_num_rows($result)!=1){
        header("Location:missing_auth.php");
    }
}
```

Fig. 14 - Condições para verificar se o utilizador efetuou login

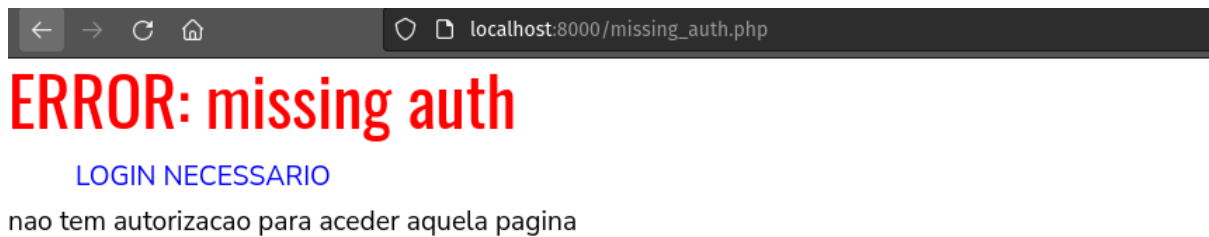


Fig. 15 - Caso o utilizador não esteja identificado/registado encaminha para esta página

## CWE-521: Weak Password Requirements

Para um sistema ser seguro para os utilizadores é necessário que os mesmos também se protejam. Uma das maneiras de o fazer é exigir uma password forte o suficiente, nomeadamente pedir ao utilizador para ter pelo menos uma letra maiúscula e números.

```
$password = $_POST["password"];
// Validate password strength
$uppercase = preg_match('@[A-Z]@', $password);
$lowercase = preg_match('@[a-z]@', $password);
$number    = preg_match('@[0-9]@', $password);
if(!$uppercase || !$lowercase || !$number || strlen($password) < 8) {
    echo 'Password should be at least 8 characters in length and should include at least one upper case letter and one number.';
}else{
    echo 'Strong password.';
    $password_nosql = mysqli_real_escape_string($con,$_POST["password"]);
    $password_ = htmlspecialchars($password_nosql);
}
```

Fig. 16 - Verificações do conteúdo da palavra-passe escolhida

O facto de ser pedido uma palavra-passe mais forte já deve ser suficiente para prevenir ataques que descubram por brute force a palavra de um utilizador.

## Conclusão

Ao longo deste trabalho, a nível teórico, consolidamos os nossos conhecimentos sobre variadas vulnerabilidades, o impacto que têm e como preveni-las. Aprofundamos os nossos conhecimentos em HTML, CSS, JavaScript, PHP e ainda em trabalhar com Dockers. Permitiu aprimorar as nossas habilidades de programação ao aprender novas linguagens de programação, como por exemplo trabalhar com PHP.

Em suma, chegamos a uma conclusão que há muitas vulnerabilidades que podem ser facilmente resolvidas para tornar o site mais seguro para os utilizadores e proprietários.

## Fontes

Para a concretização deste trabalho, para além das nossas bases adquiridas em semestres transatos, também nos baseamos na matéria lecionada durante as aulas teóricas e práticas da Unidade Curricular “Sistema Informática e nas Organizações”.

Ao desenvolver o projeto consultamos o Common Weakness Enumeration (<https://cwe.mitre.org/index.html>) para explorar as diversas vulnerabilidades que poderiam existir e obter informações sobre as mesmas.