

Projeto 1 - Vulnerabilidades

Segurança Informática e nas Organizações

Universidade de Aveiro

Afonso Cardoso, Carlos Costa, Diogo Correia, João Simões

Equipa 8

2021/2022



Projeto 1 - Vulnerabilidades

DEPARTAMENTO IV

Universidade de Aveiro

Afonso Cardoso (88964) afonsocardoso@ua.pt,
Carlos Costa (88755) carlospalmacosta@ua.pt,
Diogo Correia (90327) diogo.correia99@ua.pt,
João Simões (88930) jtsimoes@ua.pt

12 de novembro de 2021

Conteúdo

1	Introdução	3
2	Setup	5
3	Vulnerabilidades - CWE	6
3.1	CWE-79: Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)	6
3.1.1	Reflected XSS (ou Non-Persistent)	6
3.1.2	Stored XSS (ou Persistent)	8
3.1.3	DOM-Based XSS (ou Type-0 XSS)	10
3.2	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	11
3.3	CWE-200: Exposure of Sensitive Information to an Unauthorized Actor	13
3.4	CWE-256: Plaintext Storage of a Password & CWE-311 - Missing Encryption of Sensitive Data	15
3.5	CWE-306: Missing Authentication for Critical Function	15
3.6	CWE-425: Direct Request ('Forced Browsing') & CWE-288: Authentication Bypass Using an Alternate Path or Channel	17
3.7	CWE-434: Unrestricted Upload of File with Dangerous Type & CWE-20: Improper Input Validation	17
3.8	CWE-472: External Control of Assumed-Immutable Web Parameter	20
3.9	CWE-521: Weak Password Requirements	20
3.10	CWE-532: Insertion of Sensitive Information into Log File	21
3.11	CWE-549: Missing Password Field Masking	21
3.12	CWE-552: Files or Directories Accessible to External Parties	22
3.13	CWE-799: Improper Control of Interaction Frequency & CWE-307: Improper Restriction of Excessive Authentication Attempts	23
3.14	CWE-862: Missing Authorization & CWE-522: Insufficiently Protected Credentials	24
4	Soluções às Vulnerabilidades	26
4.1	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	26
4.1.1	Reflected XSS (ou Non-Persistent)	26
4.1.2	Stored XSS (ou Persistent)	27
4.1.3	DOM-Based XSS (ou Type-0 XSS)	27
4.2	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	27
4.3	CWE-200: Exposure of Sensitive Information to an Unauthorized Actor	29
4.4	CWE-256: Plaintext Storage of a Password & CWE-311 - Missing Encryption of Sensitive Data	30
4.5	CWE-306: Missing Authentication for Critical Function	30

4.6	CWE-425: Direct Request ('Forced Browsing') & CWE-288: Authentication Bypass Using an Alternate Path or Channel	31
4.7	CWE-434: Unrestricted Upload of File with Dangerous Type & CWE-20: Improper Input Validation	31
4.8	CWE-472: External Control of Assumed-Immutable Web Parameter	33
4.9	CWE-521: Weak Password Requirements	34
4.10	CWE-532: Insertion of Sensitive Information into Log File	34
4.11	CWE-549: Missing Password Field Masking	35
4.12	CWE-552: Files or Directories Accessible to External Parties	35
4.13	CWE-799: Improper Control of Interaction Frequency & CWE-307: Improper Restriction of Excessive Authentication Attempts	36
4.14	CWE-862: Missing Authorization & CWE-522: Insufficiently Protected Credentials .	38
4.15	Mecanismos de segurança adicionais	39
4.15.1	Extensão .php dos URLs	39
4.15.2	robots.txt	40
4.15.3	Página de erro 403 e 404	40
5	Conclusão	41

Capítulo 1

Introdução

Uma aplicação *web* permite, muitas vezes, a interação com o utilizador, seja através de *inputs* de informação, como através da própria navegação pelo diretório onde esta se encontra. Tendo isto em conta, é preciso aplicar uma certa camada de proteção contra possíveis ações maliciosas, ou acidentais, por parte do utilizador.

O sistema Common Weakness Enumeration (CWE) categoriza uma grande quantidade de vulnerabilidades que uma aplicação pode estar sujeita, e neste trabalho exploramos algumas dessas vulnerabilidades numa aplicação de administração de um *blog* de notícias, e como as resolver, tendo como objetivo final a apresentação de duas versões desse mesmo *back-office*. As duas versões representam uma aplicação insegura, sem qualquer mecanismo de segurança, e uma aplicação segura onde são abordadas um conjunto específico de CWEs que a nossa equipa considerou convenientes.

Este relatório analisa as duas versões da aplicação e explica, para cada CWE escolhida, o porquê do *site* ser vulnerável, com exemplos de ataque (no caso da aplicação insegura), e como é que a vulnerabilidade é resolvida, com exemplos do ataque sem sucesso (no caso da aplicação segura).

Vão ser abordadas as seguintes CWEs:

- **CWE-79:** Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- **CWE-89:** Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- **CWE-200:** Exposure of Sensitive Information to an Unauthorized Actor
- **CWE-256:** Plaintext Storage of a Password & **CWE-311:** Missing Encryption of Sensitive Data
- **CWE-306:** Missing Authentication for Critical Function
- **CWE-425:** Direct Request ('Forced Browsing') & **CWE-288:** Authentication Bypass Using an Alternate Path or Channel
- **CWE-434:** Unrestricted Upload of File with Dangerous Type & **CWE-20:** Improper Input Validation
- **CWE-472:** External Control of Assumed-Immutable Web Parameter

- **CWE-521:** Weak Password Requirements
- **CWE-532:** Insertion of Sensitive Information into Log File
- **CWE-549:** Missing Password Field Masking
- **CWE-552:** Files or Directories Accessible to External Parties
- **CWE-799:** Improper Control of Interaction Frequency & **CWE-307:** Improper Restriction of Excessive Authentication Attempts
- **CWE-862:** Missing Authorization & **CWE-552:** Insufficiently Protected Credentials

Capítulo 2

Setup

Primeiro, é preciso garantir que o **Docker** está instalado na máquina. [\[How to here\]](#)

De seguida, correr os seguintes comandos no CLI:

```
1 $ sudo chmod +x run.sh
2 $ ./run.sh
```

ou

```
1 $ sudo chmod -R a+rx ${PWD}/app
2 $ sudo chmod -R a+rx ${PWD}/app_sec
3
4 $ sudo docker build -t webapp .
5 $ sudo docker run -dti --name app -p 80:80 webapp
```

O servidor *web* estará, então, a correr em localhost:80.

Capítulo 3

Vulnerabilidades - CWE

Para demonstrar as vulnerabilidades a que um site pode estar sujeito, foi criada uma aplicação *web* insegura em **app/**.

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

Este ataque acontece quando o *software* não neutraliza de forma correta *input* por parte do utilizador antes de este ser tratado como *output* a ser usado por uma *web page* utilizada por mais utilizadores. Existem 3 tipos de XSS - ('Cross-site Scripting'):

3.1.1 Reflected XSS (ou Non-Persistent)

O servidor lê dados diretamente do HTTP *request* e reflete de volta na HTTP *response*. Visto de forma prática, este ataque faz com que a vítima forneça conteúdo perigoso a uma aplicação *web* vulnerável, conteúdo esse que é depois refletido de volta para o *web browser* e executado. A forma mais comum de realizar este ataque é através de URL's com parâmetros adulterados.

Depois de o atacante conseguir autenticar-se com sucesso na plataforma, este pode efetuar uma pesquisa na barra de pesquisa no topo. Rapidamente repara que a palavra procurada é retornada através de um *echo* e que a *query* é passada no URL através de um método GET pelo parâmetro **?s=**.

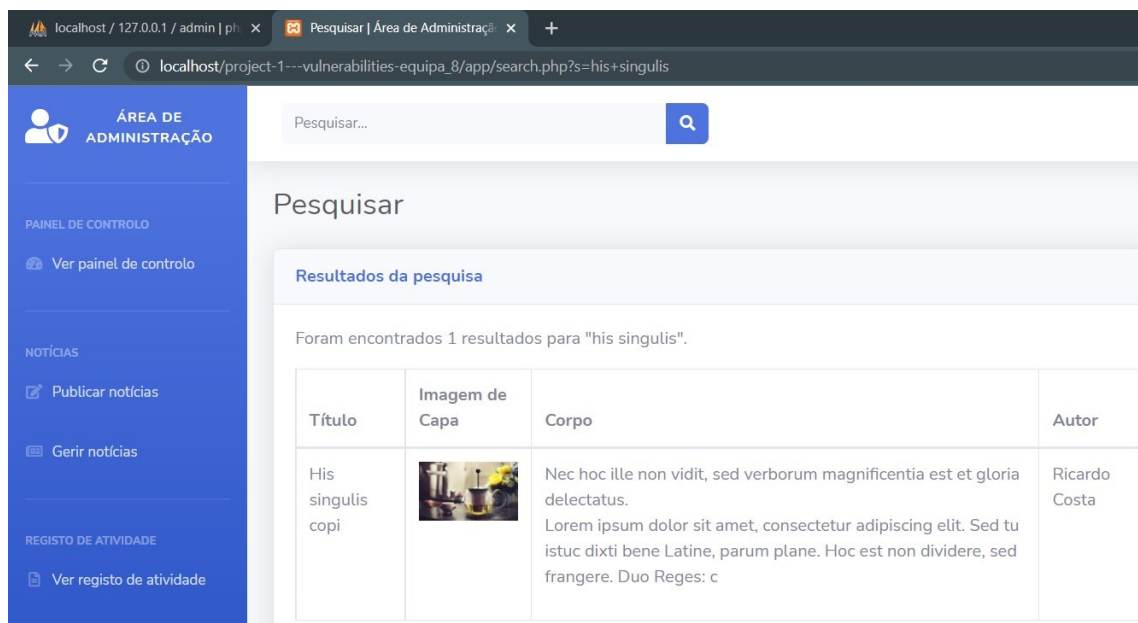


Figura 3.1: Demonstração de uma pesquisa, onde se observa o echo do texto pesquisado e também o parâmetro "s" presente no URL

Ora, se este *echo* da *query* não estiver devidamente protegido, pode levar a sérios problemas de segurança. O atacante facilmente colocaria uma *tag* HTML `<script>` para testar se a *app* permite executar *scripts*.

```

1 (...) search.php?s=<script>alert("Test");</script>
2
3 <!-- o que, convertido para URL/percent encoding, ficará: -->
4
5 (...) search.php?s=<script>alert%28"Test"%29%3B%2Fscript>

```

Felizmente, os programadores desta *app* estavam atentos para esta vulnerabilidade e tiveram o cuidado de implementar alguns mecanismos de segurança no ficheiro `.htaccess`. Em servidores que executam em Apache (como é o caso do nosso), este é um poderoso ficheiro de configuração que permite fazer alterações na configuração do servidor, sem ter que se editar os arquivos de configuração do servidor.

```

1 # Block out any script trying to base64_encode data within the URL
2 RewriteCond %{QUERY_STRING} base64_encode\[^(.*)\([^)]*\)\ [OR]
3 # Block out any script that includes a <script> tag in URL
4 RewriteCond %{QUERY_STRING} (<|%)3C([^\s]*s)+cript.*(>|%)3E [NC,OR]
5 # Block out any script trying to set a PHP GLOBALS variable via URL
6 RewriteCond %{QUERY_STRING} GLOBALS(=|\[|\\%[0-9A-Z]{0,2}) [OR]
7 # Block out any script trying to modify a _REQUEST variable via URL
8 RewriteCond %{QUERY_STRING} _REQUEST(=|\[|\\%[0-9A-Z]{0,2})
9 # Return 403 Forbidden header when showing the content of the root homepage
10 RewriteRule .* index.php [F]

```

Estas regras estão presentes em *frameworks* como WordPress e permitem proteger os *websites* das vulnerabilidades mais comuns. A segunda regra acabou de detetar uma *tag* `<script>` no URL e conseguiu proteger a nossa plataforma desta vulnerabilidade, retornando um erro 403.

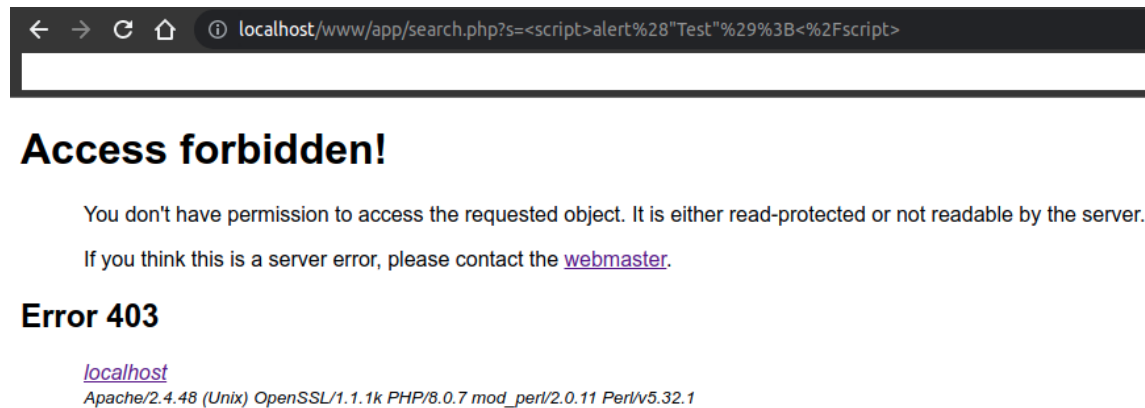


Figura 3.2: Acesso negado a uma tentativa de correr um script através de parâmetros no URL

No entanto, usar a *tag* `<script>` não é a única forma de executar JavaScript. Também é possível utilizar JavaScript no próprio elemento HTML fazendo uso dos atributos `"onmouseover"` ou `"onclick"`. Assim, fazendo uso de *tags* inofensivas como `` ou `<p>`, é possível na mesma realizar um ataque deste tipo.

Por exemplo, fazendo uso de uma outra vulnerabilidade que vamos analisar mais à frente (Seção 3.7), o atacante poderá conseguir enviar um ficheiro malicioso para o servidor. Para redirecionar o utilizador para essa página, o atacante pode colocar uma imagem, também já existente no servidor, invisível e preenchendo o ecrã inteiro (fazendo uso de estilos CSS *inline*) e colocar um *link* (*tag* `<a>`) com um evento `"onmouseover"` associado, fazendo com que seja quase impossível para o utilizador não ser redirecionado para a página que terá código malicioso.



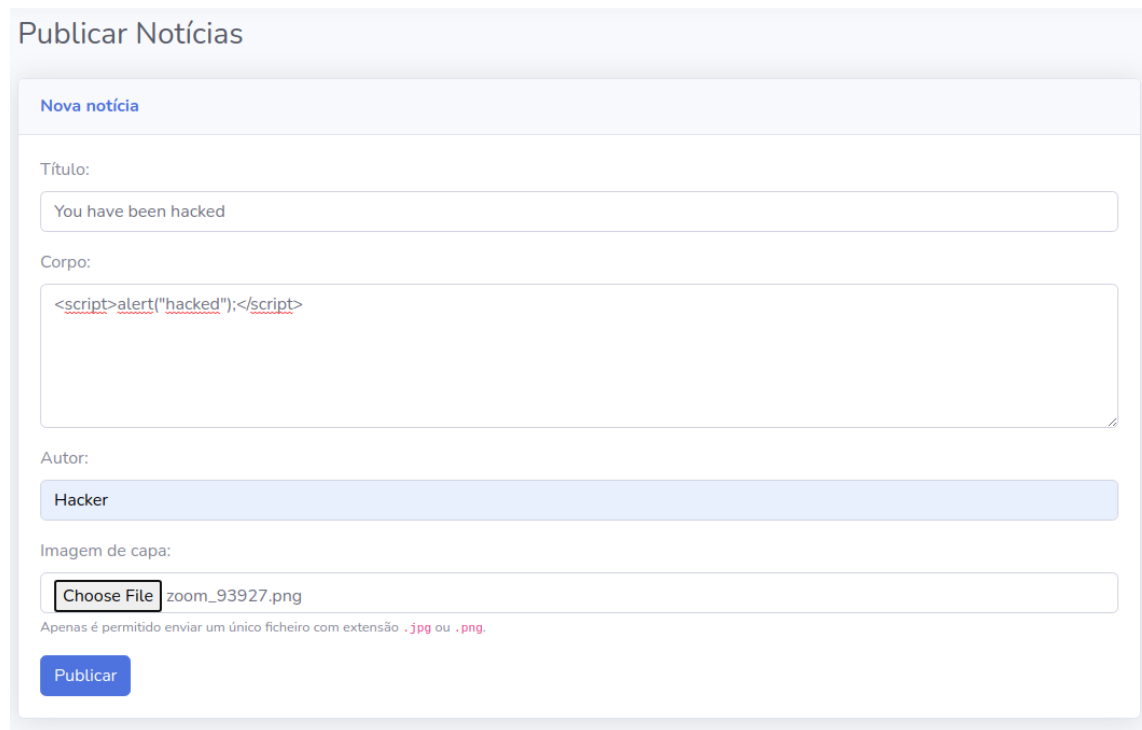
Por vezes, mesmo que estejamos atentos para estas vulnerabilidades e nos preocupemos em implementar mecanismos de segurança contra elas, poderemos não estar a fazê-lo da maneira correta. Todo o cuidado é pouco.

3.1.2 Stored XSS (ou Persistent)

A aplicação armazena conteúdo perigoso na sua base de dados ou outro local de armazenamento de dados. Mais tarde, esse conteúdo malicioso é lido de volta para a aplicação e incluído em conteúdo dinâmico.

Um atacante pode tirar proveito da opção de publicar notícias da nossa Web App Insegura para originar um ataque deste tipo.

Ao criar uma nova notícia, o atacante pode meter qualquer código HTML, até mesmo *scripts*, como o que é demonstrado a seguir:



The screenshot shows a web form titled "Publicar Notícias" with a sub-header "Nova notícia". It contains the following fields and content:

- Título:** A text input field containing "You have been hacked".
- Corpo:** A text area containing the JavaScript payload: `<script>alert("hacked");</script>`.
- Autor:** A text input field containing "Hacker".
- Imagem de capa:** A file upload section with a "Choose File" button and the filename "zoom_93927.png". Below this, a note states: "Apenas é permitido enviar um único ficheiro com extensão .jpg ou .png."
- Publicar:** A blue button to submit the form.

Figura 3.3: Publicação de notícia com o intuito de enganar o utilizador

Se não houver qualquer tipo de cuidado com a identificação e neutralização de elementos HTML, quando a página `news.php` carregar o conteúdo dessa notícia da base de dados e este for processado pelo DOM, o *script* vai correr.

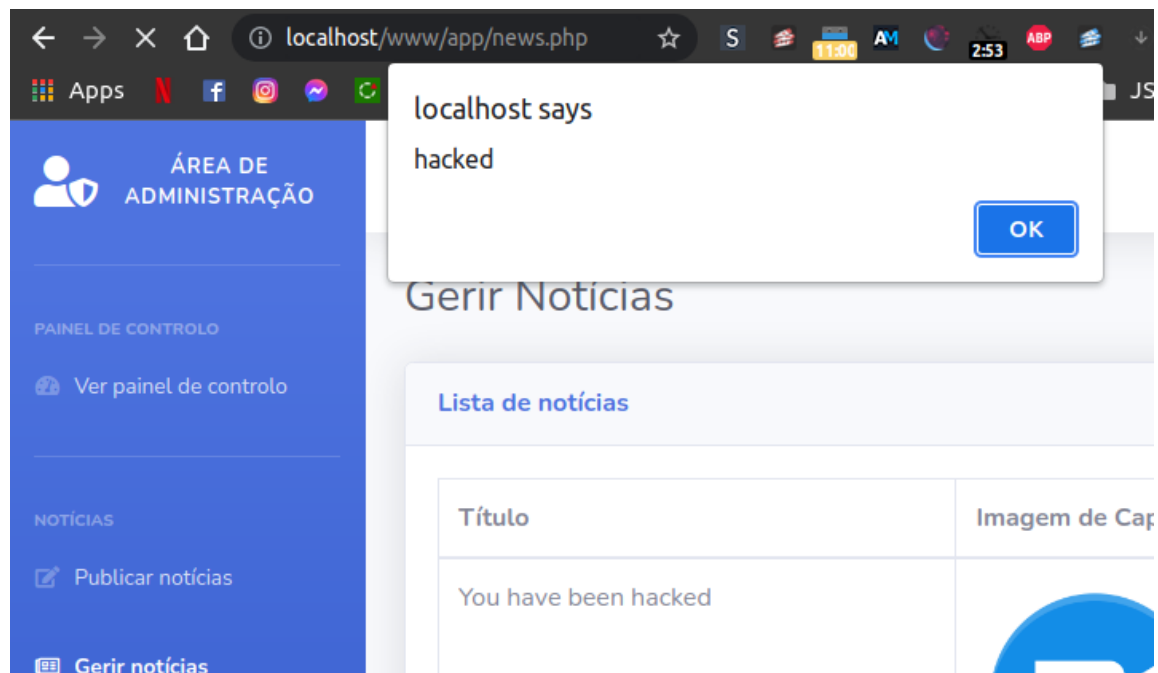


Figura 3.4: Notícia publicada com elementos HTML enganosos

3.1.3 DOM-Based XSS (ou Type-0 XSS)

Consiste em um *script*, à partida confiável, que é enviado para o cliente, que pode realizar operações básicas sobre a página *web*. Contrariamente ao Reflected XSS, este tipo de ataque nunca chega a fazer chamadas ao servidor, sendo que é tudo feito no próprio DOM (JavaScript trabalha o DOM do lado do cliente apenas - *Client-side language*).

Olhando para a listagem de notícias, em `news.php`, pode-se tirar vantagem do facto de existir uma opção de *sorting* que por sua vez pode ser definida através do URL (Esta vulnerabilidade acabou por não ser implementada na *web app*, mas será na mesma explicada uma forma de a explorar).

Um comportamento normal seria aceder a um *link* deste género:

```
localhost/www/app/news.php?sort=ascending
```

Isto ordenaria as notícias de forma crescente.

Internamente, o JavaScript pega no valor de `'sort'` e faz um *echo* no DOM para criar uma `<option>` no `<select>` respetivo da ordenação.

```
1 <select>
2   <script>
3     const sortingMethods = ["ascending", "descending"];
4
5     // get value from url
6     const default = decodeURIComponent(document.location.href.substring(doc_
7     ↪ ument.location.href.indexOf("sort=")+5));
8     document.write("<OPTION value=0>" + default + "</OPTION>");
9
10    // put all other options
11    sortingMethods.filter(method => method !== default)
12      .forEach((method, i) => document.write("<OPTION
13      ↪ value="+i+">" + method + "</OPTION>"));
14  </script>
15 </select>
```

Tendo isto em conta, um atacante pode fornecer o seguinte URL a um utilizador:

```
localhost/www/app/news.php?sort=<script>alert("Hacked!")</script>
```

Mediante isto, `document.location` vai fazer *parse* do URL e vai executar o *script* malicioso:

```
1 alert("Hacked!")
```

É preciso ter em atenção que o *script* malicioso não é executado aquando da resposta do servidor, mas sim quando o JavaScript manipula o DOM com a variável `document.location`, em *runtime*.

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

Este ataque acontece quando o *software* executa um comando SQL construído, seja na totalidade ou parcialmente, usando um *input* influenciado externamente e não neutraliza de forma correta elementos que possam modificar o comando SQL inicialmente pretendido.

Isto possibilita um utilizador de correr comandos SQL que podem comprometer o sistema e/ou a privacidade dos dados armazenados.

Uma maneira de um atacante tirar partido desta vulnerabilidade é através do sistema de autenticação.

A maneira de validar a autenticação é através de uma única *query* SQL, em que as variáveis \$username e \$pwd contém os valores dos *inputs* apresentados ao utilizador.

```
SELECT * FROM users WHERE username='".$username."' AND pwd='".$pwd."'
```

Uma maneira de adulterar esta *query* é através de um OR, fazendo com que, mesmo que não haja nenhum *username* igual ao fornecido pelo utilizador, a seleção não vai falhar porque vai ainda tentar a *query* apresentada depois desse mesmo OR. Para evitar que haja ainda a verificação da *password*, que é feita logo após a *query* fornecida pelo utilizador e que muito provavelmente ia cancelar a seleção de qualquer linha da tabela *users*, é usada uma *tag* de comentário -- // para inutilizar o resto da *query* que venha após isso.

Sendo assim, sabendo que 1=1 é sempre verdade, a seguinte injeção de SQL vai selecionar todos os utilizadores.

```
' OR 1=1 -- //
```

Mesmo que não haja nenhum utilizador com o *username* vazio (' '), o 1=1 dá a "autorização" de seleção e o -- // descarta a necessidade de a *password* pertencer ao utilizador selecionado. Como houve a seleção de pelo menos um utilizador (neste caso específico são todos selecionados), é feita a autenticação com sucesso.

The screenshot shows a login interface with the heading "Bem-vindo de volta!". It features a text input field containing the SQL injection payload "' OR 1=1 -- //". Below the input field is a checkbox labeled "Manter sessão iniciada". A large blue button labeled "Iniciar sessão" is positioned below the checkbox. At the bottom of the form, there are two links: "Recuperar palavra-passe" and "Criar uma conta".

Figura 3.5: Tentativa de SQL Injection em app.



Figura 3.6: Autenticação com sucesso, mesmo sem um utilizador ou password

O atacante, ainda que tenha entrado sem especificar nenhum *username*, entra na conta do primeiro registo da tabela dos *users*. Isto deve-se ao facto de, ao fazer a condição $1=1$, o `SELECT` vai selecionar todas as linhas da tabela (a condição é sempre *true* para cada linha) e na altura de ser processado o valor da linha selecionada é usada a primeira linha que apareça no resultado da *query* à base de dados, ou seja, o primeiro registo da tabela.

Esta mesma vulnerabilidade é possível ser explorada também, ainda na nossa Web App Insegura, na página `change-password.php`, onde apresenta um comportamento diferente.

Enquanto que, no exemplo que acabámos de explorar, a vulnerabilidade permitia o atacante de se autenticar sem quaisquer dados de utilizador, na troca de *password* é possível despoletar uma mesma ação a todos os utilizadores da tabela, ao mesmo tempo. Isto é, com a *query* certa, todas as linhas da tabela *users* ficam com a mesma *password*.

				username	email	pwd	id
<input type="checkbox"/>	Edit	Copy	Delete	admin	admin	admin	4
<input type="checkbox"/>	Edit	Copy	Delete	digas99	diogo@diogo.pt	1234	5
<input type="checkbox"/>	Edit	Copy	Delete	roberto	roberto@ua.pt	asd1234	6

Figura 3.7: Garantia de que todos os utilizadores têm passwords diferentes

Na Figura 3.7, verificamos que, inicialmente, todos os utilizadores apresentam *passwords* diferentes.

Ao acedermos à página de troca de *password*, com qualquer utilizador, podemos usar a seguinte injeção de SQL:

Pesquisar...

Q

roberto

Alterar palavra-passe

Alteração da palavra-passe

Nova palavra-passe:

hacked' -- //

Confirmar nova palavra-passe:

hacked' -- //

Alterar

Figura 3.8: Injeção de SQL no processo de alteração da password

Aqui, *'hacked'* pode ser substituído por qualquer *password* desejada.

Isto funciona porque, tendo em consideração a *query* SQL usada para atualizar a base de dados com a nova *password*:

```
UPDATE users SET pwd = '$pwd.' WHERE
↪ username='$_SESSION["userUsername"]';
```

veificamos que a pelica (') depois de *'hacked'* fecha o valor atribuído a *'pwd'*, e depois o comentário -- // inutiliza todo o código a partir de WHERE. Se não é especificado **onde** a *password* deve ser mudada, então vai acabar por ser mudada em todas as linhas da tabela.

				username	email	pwd	Id
<input type="checkbox"/>		Edit	Copy Delete	admin	admin	hacked	4
<input type="checkbox"/>		Edit	Copy Delete	digas99	diogo@diogo.pt	hacked	5
<input type="checkbox"/>		Edit	Copy Delete	roberto	roberto@ua.pt	hacked	6

Figura 3.9: Todos os utilizadores apresentam a mesma password, escolhida pelo utilizador, depois do ataque

3.3 CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

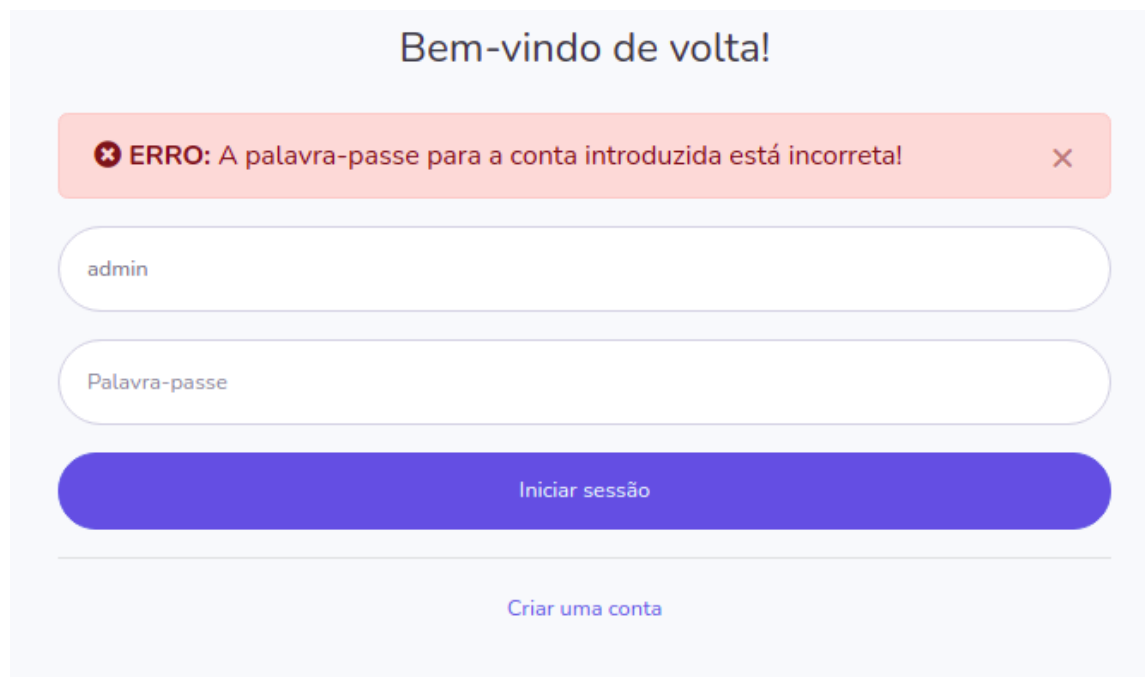
Este ataque ocorre quando a aplicação expõe informação sensível, como mensagens pessoais, dados financeiros, localização geográfica, etc, a um atuator que não possui autorização para ter acesso à mesma.

Na nossa Web App Insegura, quando o utilizador faz *login*, se o *username* ou a *password* inseridos não corresponderem aos dados, é enviado uma mensagem de erro a discriminar o que correu mal. Isto fornece informações do conteúdo da base de dados (se o *username* existe), ou até mesmo de uma conta em específico (caso o *username* exista, se a *password* está ou não errada). Esta discriminação do que está errado é favorável a ataques de *brute-force*.



The image shows a login interface with the heading "Bem-vindo de volta!". Below the heading is a red error message box that says "ERRO: O nome de utilizador introduzido não existe!". There are two input fields: "Nome de utilizador" and "Palavra-passe". Below the input fields is a blue button labeled "Iniciar sessão". At the bottom, there is a link that says "Criar uma conta".

Figura 3.10: Mensagem que informa que o username não existe na base de dados



The image shows a login interface with the heading "Bem-vindo de volta!". Below the heading is a red error message box that says "ERRO: A palavra-passe para a conta introduzida está incorreta!". The "Nome de utilizador" input field contains the text "admin". The "Palavra-passe" input field is empty. Below the input fields is a blue button labeled "Iniciar sessão". At the bottom, there is a link that says "Criar uma conta".

Figura 3.11: Mensagem que informa que a password inserida está incorreta para o username correspondente

Isto acontece devido à maneira de como é implementado a verificação de dados no *login*:


```

1 <?php
2 if (!username exists...) {
3     // username doesn't match
4     header("Location: login.php?error=usernameinvalid");
5     exit();
6 }

```

```

1 <?php
2 if (!password matches...) {
3     // password doesn't match
4     header("Location: login.php?error=pwdinvalid&username=".$username);
5     exit();
6 }

```

3.4 CWE-256: Plaintext Storage of a Password & CWE-311 - Missing Encryption of Sensitive Data

Esta vulnerabilidade ocorre quando uma *password* é armazenada sem qualquer tipo de encriptação.

No caso de acontecer algum ataque à base de dados e a informação dos utilizadores ficar exposta, o atacante fica a conhecer todos os dados de autenticação

Por outro lado, se a *password* tivesse sido guardada encriptada, o atacante continuaria às escuras pois iria apenas ter acesso a um conjunto aleatório de caracteres que não serviriam de autenticação.

No `signup.php` da *app* insegura é corrida a seguinte *query*:

```

INSERT INTO users (username, email, pwd) VALUES ('".$username."', '".$email."',
↳ '".$pwd."')

```

A *password* inserida na tabela da base de dados é o valor que vem diretamente do POST, e não passa por qualquer método de encriptação.




<div>↔ T ↔</div>				username	email	pwd	Id
<input type="checkbox"/>	 Edit	 Copy	 Delete	admin	admin	admin	1

Figura 3.12: Password armazenada sem encriptação, numa linha da tabela "users" na base de dados

3.5 CWE-306: Missing Authentication for Critical Function

Ocorre quando o *software* não realiza qualquer verificação de autenticação para uma funcionalidade que necessita de um utilizador autenticado previamente.

Isto verifica-se na página `search.php` da nossa Web App Insegura. Ainda que, quando o utilizador não está autenticado, não exista propriamente uma maneira, na interface, de aceder à página de *search* (não existe uma barra de navegação no topo da página onde estaria o *input* relativo à pesquisa), se o utilizador aceder à página `search.php` através do URL, esta página não vai verificar se existe uma sessão de autenticação válida, e o utilizador não é redirecionado de volta para a página de *login*.



Figura 3.13: Interface de login na App Insegura, sem qualquer acesso à pesquisa

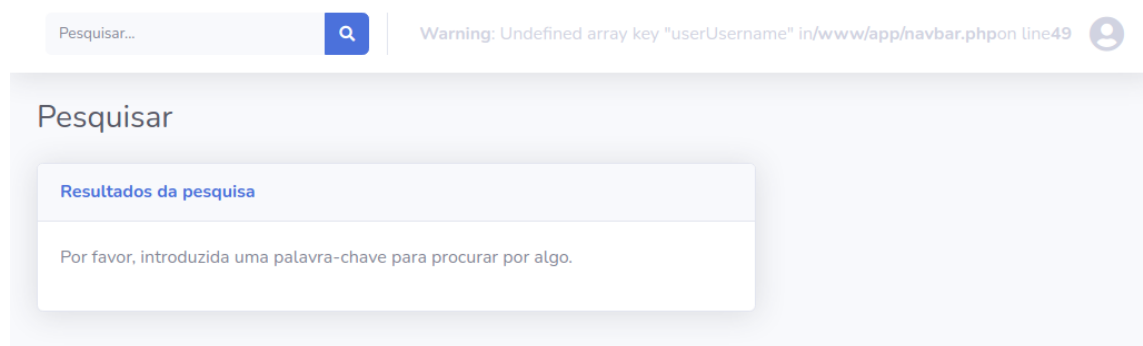


Figura 3.14: Acesso à página de pesquisa sem autenticação válida

Como se observa na Figura 3.14, o utilizador não está autenticado, como mostra o *warning* no canto superior direito da página, onde o valor `userUsername` da sessão de autenticação não foi definido (aquando de uma autenticação válida, `$_SESSION['userUsername']` recebe o valor do *username* do utilizador autenticado). No entanto, o utilizador tem acesso à página e à funcionalidade de pesquisa sem qualquer restrição.

3.6 CWE-425: Direct Request ('Forced Browsing') & CWE-288: Authentication Bypass Using an Alternate Path or Channel

Esta CWE ocorre quando uma aplicação web não impõe de forma adequada a autorização apropriada em todos os URLs, *scripts* ou ficheiros. Esta falha de segurança pode ser descoberta por utilizadores mal intencionados através dos chamados ataques por *direct request*.

No caso da versão insegura da nossa app, conseguimos observar este problema em duas instâncias:

- Ao forçar o acesso à página de pesquisa `search.php`;
- Ao forçar o acesso ao ficheiro de registo de atividade `log.php` (dentro da pasta `app/php/`).

Quanto ao primeiro ponto, este já foi explicado na secção anterior (Secção 3.5). Os programadores fizeram a falsa suposição de que esta página/funcionalidade de pesquisa só poderia ser alcançada por meio de um determinado caminho de navegação e, portanto, apenas aplicaram verificações de autenticação e de autorização num determinado *path* apenas.

No entanto, isto não é verdade. Através de "*forced browsing*", um atacante poderá facilmente tentar um *request* e aceder com sucesso à página em questão, sem precisar passar pelo processo de *login*. Isto porque `/search` é um dos URLs *standart* e mais comuns utilizados entre aplicações web, por isso a probabilidade de o *request* retornar um erro 404 é baixa.

Quanto ao segundo ponto, a situação é muito semelhante. A página de consultar o registo de atividade (`logs.php`) tem código que lê e apresenta o conteúdo do ficheiro `log.php` (presente na pasta `php/`) ao utilizador, ficheiro este que contém o registo de todas as ações realizadas dentro da área de administração. Embora a página `logs.php` esteja devidamente protegida e necessite que o utilizador esteja autenticado e autorizado para visualizar a página, já o ficheiro `log.php` não faz nenhuma dessas verificações.

Fica então extremamente fácil para um atacante descobrir que este ficheiro *log* existe (por exemplo, através da vulnerabilidade do acesso desprotegido à vista de diretórios na pasta `php/` - figura 3.22 -, processo que vamos analisar mais à frente na secção 3.12) e aceder diretamente a ele pelo próprio URL, sem qualquer processo de verificação pelo meio.

3.7 CWE-434: Unrestricted Upload of File with Dangerous Type & CWE-20: Improper Input Validation

Ocorre quando o *software* permite ao atacante fazer *upload* ou transferência de ficheiros de tipos perigosos que podem ser automaticamente processados pelo ambiente do software.

No caso da versão insegura da nossa app, quando o utilizador escreve uma notícia, existe a possibilidade de fazer o *upload* de ficheiros, sendo o pretendido ficheiros de imagens como forma de complementar a notícia a ser criada. Embora a aplicação diga ao utilizador que "Apenas é permitido enviar um único ficheiro com extensão .jpg ou .png", o tipo de ficheiro enviado não é verificado no *back-end*, podendo assim ser possível fazer *upload* de ficheiros maliciosos.

Ainda sim, os programadores desta app tiveram o cuidado de implementar o atributo `accept=".jpg,.jpeg,.png"` no `<input>` de *upload* do ficheiro. Este atributo faz com que, na janela de escolha do ficheiro, só seja mostrados ficheiros com o tipo de imagem permitido. No entanto, esta proteção pode ser facilmente ultrapassada com apenas dois cliques e sem qualquer conhecimento avançado de informática.

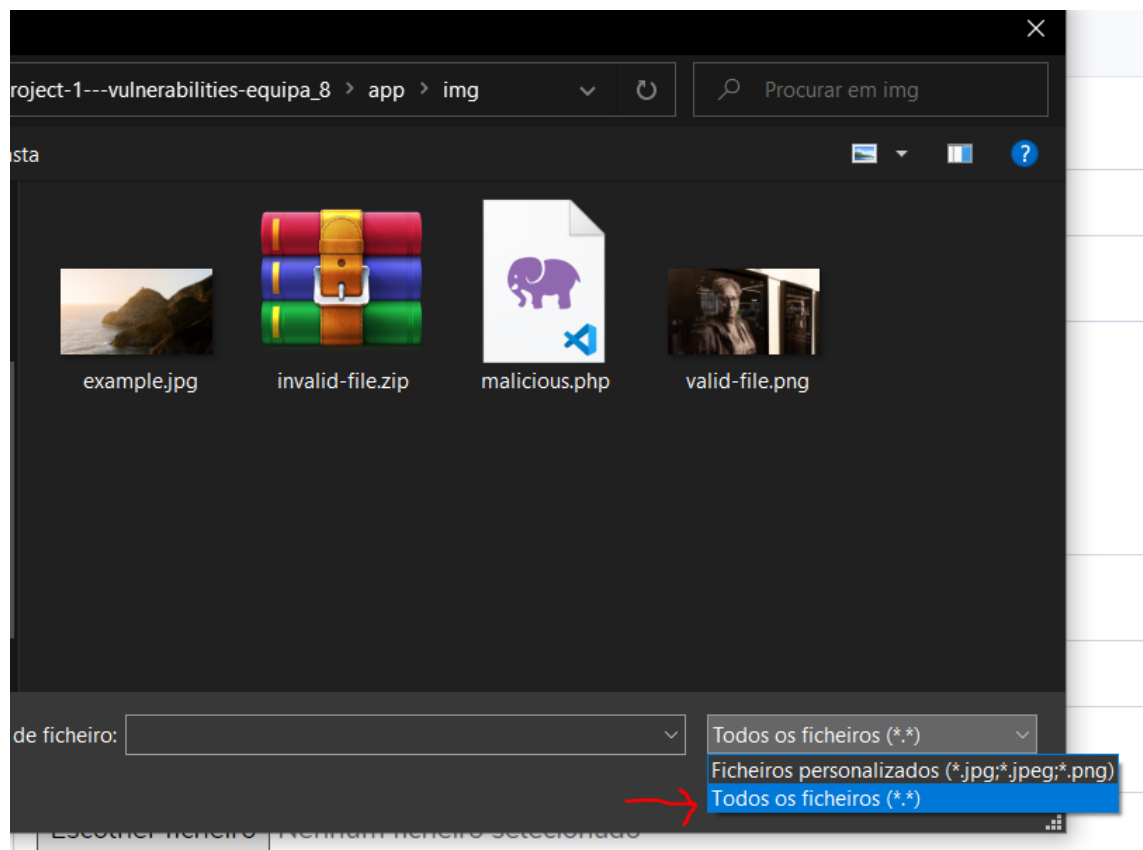


Figura 3.15: Forma de contornar o atributo "accept" no Windows (Windows 10)

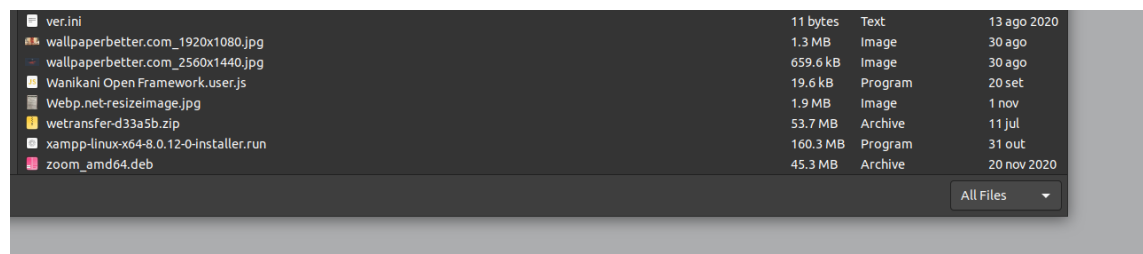


Figura 3.16: Forma de contornar o atributo "accept" no Linux (Ubuntu), em que a opção default são os ficheiros personalizados, mas que dá para trocar para todos os ficheiros

Uma forma ainda mais fácil de ultrapassar esta restrição via *client-side*, seria fazer *drag-and-drop* do ficheiro pretendido para cima do `<input type="file">`. Isto funciona, mesmo que a extensão do ficheiro não seja uma das declaradas no atributo `accept`.

Publicar Notícias

Nova notícia

Título:

Corpo:

You have been attacked

Autor:

Imagem de capa:

Apenas é permitido enviar um único ficheiro com extensão .jpg ou .png.

Figura 3.17: Possibilidade de fazer upload de um ficheiro para a aplicação

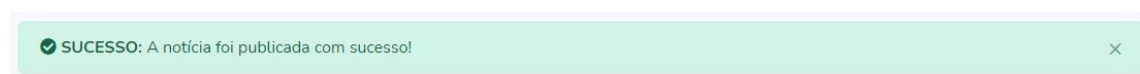


Figura 3.18: Sucesso na publicação da notícia publicada contendo conteúdo malicioso

Gerir Notícias

Lista de notícias

Título	Imagem de Capa	Corpo	Autor	Ações
Incoming Attack		You have been attacked	Your	

DevTools - localhost/project-1---vulnerabilities-equipa_8/app/news.php?submit=success

Elements

```

</thead>
<tbody>
  <tr> => $0
    <td>Incoming Attack</td>
    <td>
      
    </td>
    <td>You have been attacked</td>
    <td>Your</td>
  </tr></tbody>

```

Styles

```

element.style {
}
*,
:after, :before {
  box-sizing: border-box;
}
tr {
  user agent stylesheet

```

Figura 3.19: Conteúdo malicioso presente na base de dados da aplicação

3.8 CWE-472: External Control of Assumed-Immutable Web Parameter

Esta vulnerabilidade acontece quando uma aplicação web não verifica suficientemente os vários *inputs*, que supostamente seriam imutáveis mas que na verdade podem ser controlados externamente via *client-side*.

No caso da nossa *app* insegura, o exemplo mais crítico desta CWE dá-se na página `news.php`, nomeadamente na funcionalidade de eliminar notícias, utilizando o botão vermelho com o ícone do caixote do lixo. Embora o método escolhido pelos programadores para esta opção de eliminar notícias tenha sido o POST (que sempre é mais seguro que o GET), isto por si só não é suficiente para garantir a segurança desta funcionalidade.

Isto porque o ID da notícia que se pretende apagar, está a ser enviado pelo campo `value` do próprio botão, que é do tipo `submit`. O atacante poderá aperceber-se disto e mudar este valor, editando o HTML através das ferramentas de programador do *browser*.

```
1 <!-- antes de o atacante editar o HTML da página -->
2 <button title="Eliminar notícia 1" type="submit" class="btn btn-danger
3   ↪ btn-block" name="delete-submit" value="1">
4   <i class="fas fa-trash"></i>
5 </button>
6
7 <!-- depois de o atacante editar o HTML da página -->
8 <button title="Eliminar notícia 1" type="submit" class="btn btn-danger
9   ↪ btn-block" name="delete-submit" value="100; DROP DATABASE admin; -- //">
10  <i class="fas fa-trash"></i>
11 </button>
```

Mais uma vez, isto poderá ser uma porta aberta para os atacantes injetarem comandos SQL, vulnerabilidade já abordada na Seção 3.2.

```
1 DELETE FROM news WHERE id=.$_POST['delete-submit'].; ALTER TABLE news
2   ↪ AUTO_INCREMENT = 1
3
4 -----
5
6 DELETE FROM news WHERE id=100; DROP DATABASE admin; -- //; ALTER TABLE news
7   ↪ AUTO_INCREMENT = 1
8
9 -- SQL injection cujo código vai resultar na eliminação permanente de toda a
10 ↪ base de dados, tornado a aplicação web totalmente inutilizável (é necessário
11 ↪ colocar um número, caso contrário a primeira query vai falhar e depois a
12 ↪ segunda não chega a ser executada)
```

3.9 CWE-521: Weak Password Requirements

Esta vulnerabilidade dá-se quando o software não exige que os utilizadores definam palavras-passe fortes, o que torna mais fácil para os atacantes invadirem as contas dos utilizadores.

A versão insegura da nossa *app* sofre deste problema. Ao criar uma conta ou ao alterar a palavra-passe, o utilizador tem a árdua tarefa de escolher uma palavra-passe para a sua conta. Mas a plataforma não tem qualquer medidas para garantir que essa palavra-passe escolhida é segura.

Por exemplo, nada impede que um utilizador escolha uma palavra-passe com um caracter apenas. Isto não só é problemático para o utilizador, porque poderá ficar com os seus dados pessoais comprometidos, como também para a própria *app*, pois o atacante ganha acesso a mais páginas e isso traduz-se num maior leque de possíveis falhas para explorar.

3.10 CWE-532: Insertion of Sensitive Information into Log File

A informação escrita nos ficheiros de registo de atividade (*log files*) pode: ser de natureza sensível e fornecer um valioso fio condutor ao atacante; expor informação sensível de utilizadores.

No caso da nossa Web App Insegura, foi implementada a escrita de um registo de atividade (cujo *back-end* foi omitido) com demasiadas informações sobre cada ação do utilizador. O que era suposto ser um simples ficheiro com um *tracking* de ações realizadas pelos vários utilizadores para controlo de tarefas, acaba por ser uma fonte de informações para possíveis indivíduos mal intencionados, tais como IP, *browser*, dispositivo usado, sistema operativo, ID de sessão, etc..

Um utilizador desta plataforma (que tem acesso ao registo de atividade pela barra lateral) nunca necessitará, por exemplo, de saber o IP ou o ID de sessão de outro utilizador. Mesmo que seja necessário que alguém tenha acesso a estas informações (para controlo de inícios de sessão suspeitos ou para auditorias de segurança, deverá ser feito pelos *logs* internos do servidor e não deverá ser acessível para utilizadores que não "*super-admins*" ou *webmasters*, com acesso privilegiado.

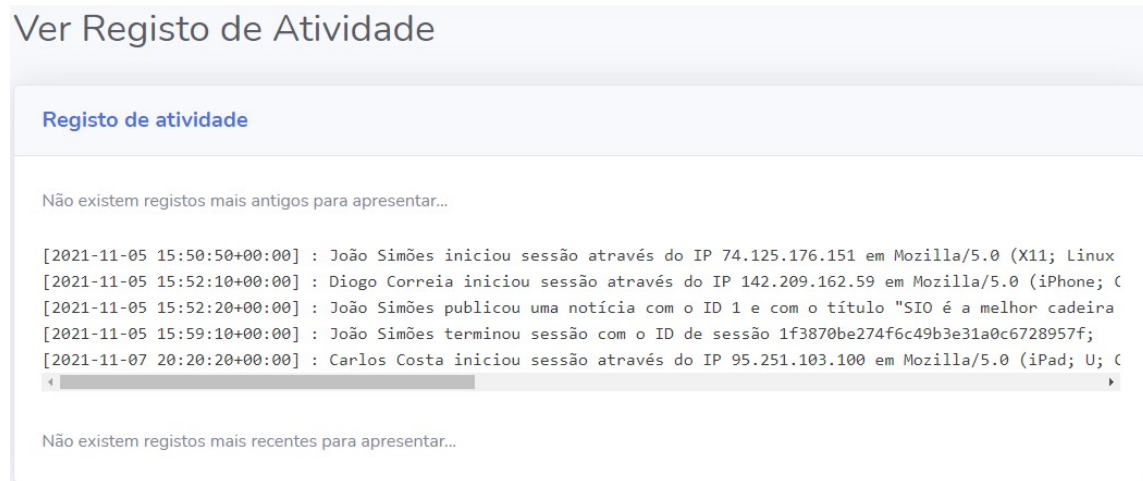


Figura 3.20: Informação sensível disponibilizada ao utilizador comum

3.11 CWE-549: Missing Password Field Masking

Esta vulnerabilidade acontece quando o software não encobre os campos onde os utilizadores introduzem dados sensíveis (tais como palavras-passe, CVV de cartões de crédito, códigos de segurança, etc.), facilitando o trabalho dos atacantes que poderão observar e conseguir descobrir parte ou até mesmo a totalidade desses dados.

Na versão insegura da nossa *app*, esta vulnerabilidade pode ser observada nos campos de palavra-passe. O *browser* deteta a ausência da especificação de um atributo `type` por parte dos programadores e assume o `<input>` como sendo do tipo `type="text"` (o valor predefinido).

Bem-vindo de volta!

Mark Zuckerberg

EnterTheMeta

Iniciar sessão

[Criar uma conta](#)

Figura 3.21: Password com ausência de *masking*

3.12 CWE-552: Files or Directories Accessible to External Parties

Ocorre quando o software cria ficheiros ou diretórios que podem ser acedidos por atuadores não autorizados.

Exemplos deste tipo de vulnerabilidades é o acesso do utilizador a diretórios com ficheiros que manipulam o comportamento da aplicação, como por exemplo estilos (CSS) e *scripts* (JavaScript).

No caso da nossa Web App Insegura, o utilizador consegue aceder a certos diretórios que deveriam ser restritos, como por exemplo um diretório chamado `php/` dentro de cada App onde seriam armazenadas funções PHP que não apresentam qualquer tipo de interface *web* e que o utilizador não tem a necessidade de saber da sua existência nem deveriam ser acessíveis.

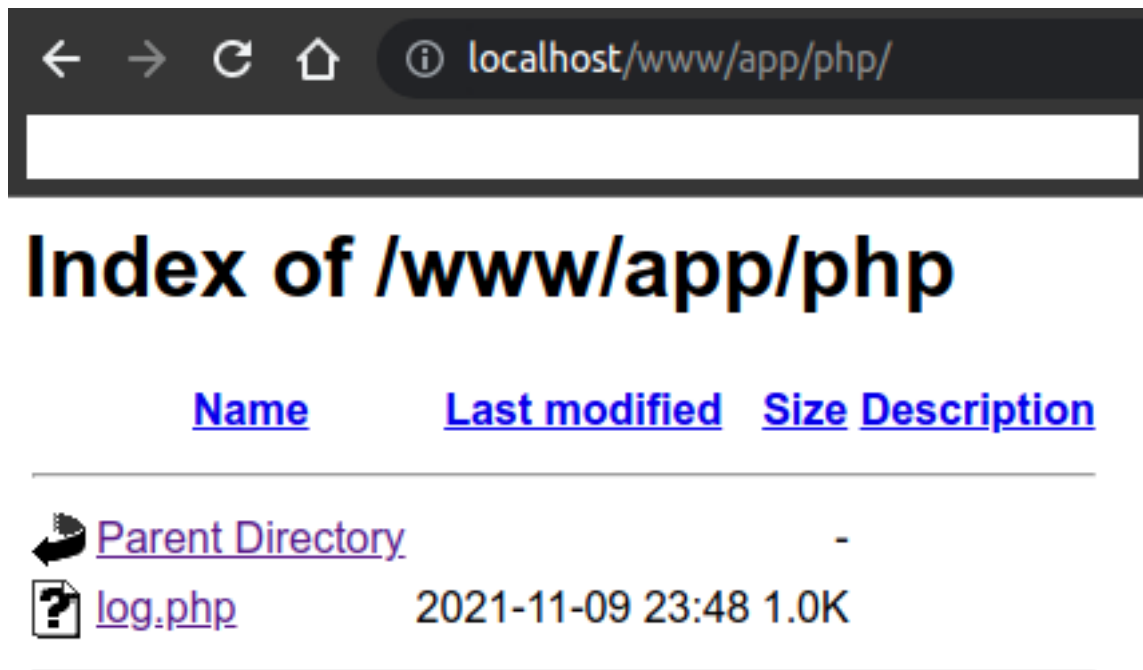


Figura 3.22: Acesso ao diretório php/ que deveria estar interdito

3.13 CWE-799: Improper Control of Interaction Frequency & CWE-307: Improper Restriction of Excessive Authentication Attempts

Ocorre quando o *software* não define um número limite ou frequência de interações que pode ter com um atuador, tal como o número de *requests* recebidos.

No caso da nossa Web App Insegura, um dado utilizador poderá tentar o processo de autenticação o número de vezes que quiser, pois não tem um limite de tentativas erradas.

The image shows a login interface with a light blue background. At the top, it says 'Bem-vindo de volta!'. Below this is a red error message box that says 'ERRO: O nome de utilizador introduzido não existe!'. There are two input fields: 'Nome de utilizador' and 'Palavra-passe'. A blue button labeled 'Iniciar sessão' is below the fields. At the bottom, there is a link 'Criar uma conta'.

Figura 3.23: Mensagem informa que o login não obteve sucesso mas não possui qualquer restrição

Este tipo de vulnerabilidade permite, principalmente, ataques de *brute-force*.

3.14 CWE-862: Missing Authorization & CWE-522: Insufficiently Protected Credentials

Uma funcionalidade que precisa de autenticação e de uma autorização é a alteração da *password*. É importante esta camada extra de proteção pois trata-se de um aspeto crítico dentro das interações do utilizador com a *app*.

No caso da nossa Web App Insegura, o utilizador estar autenticado é suficiente para que ele possa alterar a *password* e não existe um campo que peça a introdução da *password* antiga. Este campo extra daria a possibilidade de uma revalidação da autenticação e garantir que é mesmo o utilizador quem quer realizar a alteração, e não um atacante a fazer passar-se por ele, através de *session hijacking* por exemplo.

Pesquisar...

admin

Alterar palavra-passe

Alteração da palavra-passe

Nova palavra-passe:

Confirmar nova palavra-passe:

Alterar

Figura 3.24: Ainda que autenticado, não está implementada uma forma de verificar se o utilizador está autorizado a alterar a password da conta em questão (solicitando a password antiga, por exemplo)

Capítulo 4

Soluções às Vulnerabilidades

Para solucionar as vulnerabilidades apresentadas anteriormente, foi criada uma aplicação *web* segura em `app_sec/`.

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

4.1.1 Reflected XSS (ou Non-Persistent)

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.

Quando, na Web App Segura, numa *search*, passamos o valor abordado na Secção 3.1.1 para o parâmetro "s" do URL, temos um desfecho diferente quando a página é processada:

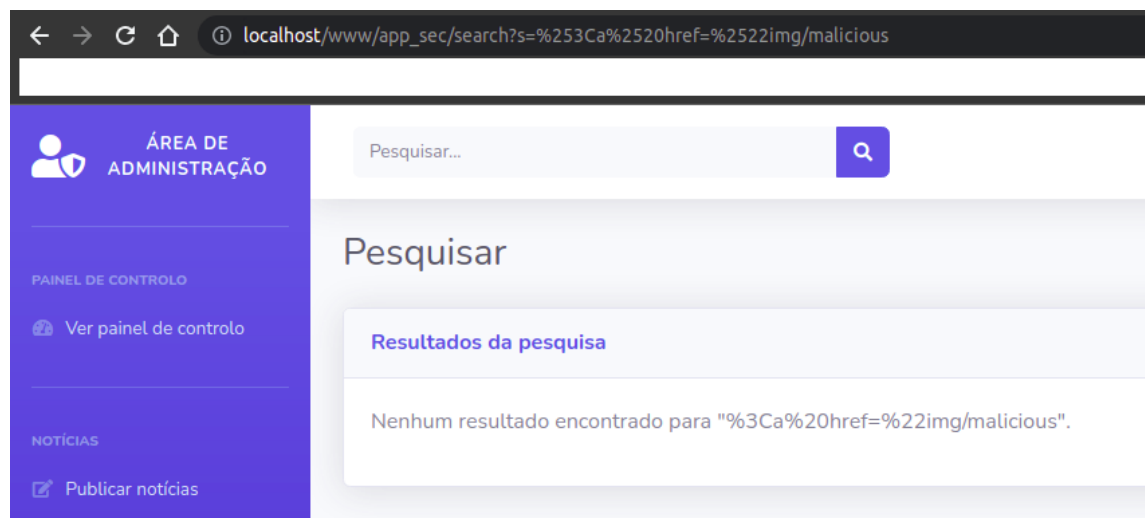


Figura 4.1: O resultado da pesquisa foi parsed para não originar elementos HTML

Desta vez, nenhuma imagem aparece no resultado da pesquisa, pois o *echo* do parâmetro do URL foi *parsed* ao ponto de ser impossível ao DOM de criar qualquer elemento HTML a partir desse valor.

O parsing é feito com uma *built-in function* do PHP:

```
1 <?php
2 $search = htmlspecialchars($_GET["s"], ENT_QUOTES);
```

4.1.2 Stored XSS (ou Persistent)

Formas de resolver este tipo de ataques passam por detetar HTML *tags* nos inputs do utilizador e atuar sobre elas.

Pode-se anular todo o conteúdo envolvido pelas *tags*, não havendo assim nada a ser processado.

Por outro lado, e é o que está implementado na nossa Web App Segura, pode-se trocar os 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 '<', '>' no sítio desses caracteres especiais.

Esta conversão de caracteres pode ser feita recorrendo à função de PHP `htmlspecialchars()`, usada também na solução ao ataque anterior.

```
1 <?php
2 $title = htmlspecialchars($_POST['title'], ENT_QUOTES);
3 $body = htmlspecialchars($_POST['body'], ENT_QUOTES);
4 $author = htmlspecialchars($_POST['author'], ENT_QUOTES);
```

4.1.3 DOM-Based XSS (ou Type-0 XSS)

Uma maneira de atacar esta vulnerabilidade é garantir que as funções que fazem uso de valores provenientes de parâmetros do URL neutralizam as *tags* de HTML.

Outra forma é bloquear o uso de *script* HTML *tags* no URL, através do `.htaccess`, da seguinte forma:

```
1 # Block out any script that includes a <script> tag in URL
2 RewriteCond %{QUERY_STRING} (<|%3C)([~s]*s)+cript.*(>|%3E) [NC,OR]
```

A primeira solução é *client-side* enquanto que a segunda é *server-side*.

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

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.

```
1 <?php
2 $sql = "SELECT * FROM users WHERE username=?";
3 // initializing sql statement
4 $stmt = mysqli_stmt_init($conn);
5 // binding placeholder with variable $username
6 mysqli_stmt_bind_param($stmt, 's', $username);
7 // executing statement
8 mysqli_stmt_execute($stmt);
```

Neste caso, o '?' na linha 1 é um *placeholder* para o valor armazenado na variável \$username. Desta forma, o texto que o utilizador preencher no formulário será sempre uma *string* (parâmetro 's' na linha 3), anulando a possibilidade de se correr uma SQL *query* indesejada.

Desta forma, se o utilizador tentar iniciar sessão com uma injeção de SQL, como por exemplo:

```
' OR 1=1 -- //
```

esse *input* é interpretado como o nome do utilizador e não como código SQL, o que causa um erro de "Invalid username or password!".

The screenshot shows a login interface with the heading "Bem-vindo de volta!". It features a light blue rounded rectangular input field containing the SQL injection payload "' OR 1=1 -- //". Below this is a standard password input field with a single character 'a' and a toggle icon. There is an unchecked checkbox labeled "Manter sessão iniciada". A large blue button labeled "Iniciar sessão" is positioned below the inputs. At the bottom, there are two links: "Recuperar palavra-passe" and "Criar uma conta".

Figura 4.2: Tentativa de SQL Injection em app_sec

Bem-vindo de volta!

✖ ERRO: As credenciais introduzidas são inválidas! Por favor tente novamente. ✖

Nome de utilizador

Palavra-passe

Iniciar sessão

[Criar uma conta](#)

Detailed description: This is a login form titled 'Bem-vindo de volta!'. It features a red error message box at the top stating 'ERRO: As credenciais introduzidas são inválidas! Por favor tente novamente.' with a close button. Below the error message are two input fields: 'Nome de utilizador' and 'Palavra-passe'. The password field has a toggle icon (an eye) to its right. A large blue button labeled 'Iniciar sessão' is positioned below the input fields. At the bottom, there is a link 'Criar uma conta'.

Figura 4.3: SQL Query comporta-se como string e não como código SQL

4.3 CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Se a mensagem de erro for a mesma para quando o utilizador erra no *username* ou na *password*, então um possível atacante que estivesse a tentar aceder à conta não percebe qual dos dois campos de autenticação estão errados.

Bem-vindo de volta!

✖ ERRO: As credenciais introduzidas são inválidas! Por favor tente novamente. ✖

Nome de utilizador

Palavra-passe

Iniciar sessão

[Criar uma conta](#)

Detailed description: This is a login form identical in layout to Figure 4.3, titled 'Bem-vindo de volta!'. It has a red error message box at the top with the text 'ERRO: As credenciais introduzidas são inválidas! Por favor tente novamente.' and a close button. Below the error message are two input fields: 'Nome de utilizador' and 'Palavra-passe'. The password field has a toggle icon (an eye) to its right. A large blue button labeled 'Iniciar sessão' is positioned below the input fields. At the bottom, there is a link 'Criar uma conta'.

Figura 4.4: Mensagem de erro mais geral, não especifica o qual dos campos está errado

4.4 CWE-256: Plaintext Storage of a Password & CWE-311 - Missing Encryption of Sensitive Data

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

```
1 <?php
2 // use binding to prevent executing queries from the user
3 mysqli_stmt_bind_param($stmt, 'sss', $username, $email, password_hash($pwd,
4     ↪ PASSWORD_DEFAULT));
5 mysqli_stmt_execute($stmt);
6
7 header("Location: login.php?username=".$username);
8 exit();
```

Deste modo, se houver um ataque à base de dados da Web App, o atacante irá continuar sem ter acesso aos dados das *passwords* do utilizador.

	username	email	pwd
<input type="checkbox"/>  Editar  Copiar  Apagar	admin	admin	\$2y\$10\$ngUhef8W7.zWZKmrIhVCHO/QpEqi8nXuYWc2rerVILY...
<input type="checkbox"/>  Editar  Copiar  Apagar	carlo	carlospalmacosta@gmail.com	\$2y\$10\$YzVow5y85bS6Z2HaOd6wQ.MgEeP0NB.S99TCUfScSle...

Figura 4.5: Passwords com encriptação, numa linha da tabela users

4.5 CWE-306: Missing Authentication for Critical Function

A solução para esta CWE 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, apenas é necessário implementar a seguinte linha de código imediatamente no início das páginas que sofrem desta vulnerabilidade (no caso particular analisado na secção 3.5: na página `search.php`).

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

Este ficheiro `check-session.php` contém o código necessário para verificar se o utilizador já se encontra autenticado ou não. Uma vez que não permitimos que os utilizadores mantenham a sessão iniciada no *login*, podemos fazer uso da variável superglobal nativa do PHP `$_SESSION`, em vez de `$_COOKIE`. Para além disso é também uma camada de proteção adicional, já que assim o UID (*Unique Identifier*) do utilizador fica guardado no servidor (*server-side*) e não no *browser* do utilizador (*client-side*), prevenindo ataques como o "*Pass-The-Cookie*" e outros relacionados com *session hijacking*.

```
1 <?php
2 session_start();
```



```

4 // if not yet in session then go to login
5 if(!isset($_SESSION["userId"])){
6     header("Location: login.php");
7     exit();
8 }

```

4.6 CWE-425: Direct Request ('Forced Browsing') & CWE-288: Authentication Bypass Using an Alternate Path or Channel

À semelhança da solução para a vulnerabilidade apresentada na secção anterior (secção 4.5), a solução para esta CWE acaba por ser o mesmo processo: verificar se o utilizador está autenticado e autorizado a aceder à página ou ficheiro pretendido, através de um `require()` do ficheiro `check-session.php` comum às duas apps.

Deste modo, caso o utilizador tente aceder "à força" ao URL `/app_sec/php/log.php` ou a `/app_sec/search.php` sem estar autenticado, então é redirecionado imediatamente para a página de *login*.

4.7 CWE-434: Unrestricted Upload of File with Dangerous Type & CWE-20: Improper Input Validation

Nesta altura já está provado que o programador não pode confiar nos utilizadores. Mesmo que todos os avisos possíveis e imaginários sejam feitos no *front-end*, estes podem ser facilmente ignorados sem querer, ou até propositadamente.

Referimos anteriormente (Seção 3.7) que, embora se tenha tido o cuidado de se implementar o atributo `accept` no `<input>` de *upload* do ficheiro (o que faz com que, na janela de seleção, só apareçam ficheiros com o tipo permitido), esta proteção pode ser facilmente ultrapassada com poucos cliques e sem qualquer conhecimento avançado de informática.

Assim, para evitar que esta vulnerabilidade seja realizada, torna-se necessário também uma verificação no *back-end* da app. É feita uma verificação para assegurar que a extensão do ficheiro enviado faz parte da lista de tipos de ficheiros permitidos para *upload* (neste caso imagens JPG/JPEG ou imagens PNG).

```

1 <?php
2 // Check if the extension of the file is allowed
3 if(($fileActualExt != "jpg") && ($fileActualExt != "jpeg") && ($fileActualExt !=
4     "png")){
5     header("Location: publish.php?submit=invalid");
6     exit();
7 }

```

Ou seja, quando o atacante tentar dar *upload* a um ficheiro que não seja `.jpg`, `.jpeg` ou `.png`, não vai ser permitida tal ação, sendo abortada imediatamente antes do *upload* do ficheiro e consequentemente abortada a criação da nova notícia.

Publicar Notícias

Nova notícia

Título:

virus

Corpo:

you got rick rolled

Autor:

rick astley

Imagem de capa:

Choose File exceptions.s

Apenas é permitido enviar um único ficheiro com extensão **.jpg** ou **.png**.

Publicar

Figura 4.6: Tentativa de fazer upload de um ficheiro malicioso

✖ ERRO: O ficheiro enviado tem uma extensão que não é válida! Apenas é permitido enviar imagens, com extensão .jpg ou .png. ✖

Nova notícia

Título:

Escrever o título da notícia

Corpo:

Escrever o corpo da notícia

Autor:

Escrever o autor da notícia

Imagem de capa:

Choose File

No file chosen

Apenas é permitido enviar um único ficheiro com extensão .jpg ou .png.

Publicar

Figura 4.7: Mensagem de erro informando que a extensão do ficheiro enviado é inválida

4.8 CWE-472: External Control of Assumed-Immutable Web Parameter

Esta vulnerabilidade já não é tão trivial de se corrigir. Obviamente que a solução passará por implementar mecanismos de verificação do valor do `<input>` no back-end, mas esta solução vai ser sempre diferente, dependendo do *input* que seja esperado do utilizador.

No caso da nossa app e no caso particular da eliminação de notícias, o valor (atributo `value`) esperado do botão será o ID da notícia, isto é, um número inteiro maior que zero. Devido ao valor no POST ser guardado em `string`, torna-se necessário converter a `string` numérica para `int`, recorrendo à função nativa do PHP `intval()`.

```
1 <?php
2 # make sure that value of ID is a integer (base 10) and clean all other strings
3 $id = intval($_POST['delete-submit'], 10);
```

```

4  if (!is_int($id)){
5      # ID de notícia inválido, aborta a execução da função
6  }
7  # ID de notícia válido, prossegue para a query SQL

```

É aqui que todas as **strings** numéricas são filtradas e ficam apenas com números inteiros (como se se tratasse de um *parse*). Caso seja fornecida uma **string** que não numérica (isto é, apenas tem caracteres e não caracteres e números), então a função coloca na variável `$id` o valor inteiro 0. Portanto, passará na verificação de ser um inteiro `is_int()` mas, chegando à query SQL, a notícia não pode ser encontrada porque não existe nenhuma notícia com o ID zero (o ID é um número inteiro maior que zero) e ação nunca terá qualquer efeito na base de dados.

Caso `$id` seja um número inteiro maior que zero, então aí sim, a notícia com esse ID vai ser eliminada com sucesso da base de dados (caso exista).

4.9 CWE-521: Weak Password Requirements

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.

Esta verificação é feita sempre que uma nova palavra-passe é criada, isto é, tanto na página de criação de conta (`signup.php`) como na página de alteração da palavra-passe (`change-password.php`).

```

1  <?php
2  if(strlen($pwd) < 8 || !preg_match('/[A-Z]/', $pwd) ||
3  → !preg_match('/[\'^!$%&*()]{0,1}[@#~?><>,|=_+~]/', $pwd)){
4      # a palavra-passe não cumpre um dos requisitos obrigatórios
5      # a palavra-passe não foi aceite
6  } else {
7      # a palavra-passe cumpre todos os requisitos obrigatórios
8      # a palavra-passe foi aceite
9  }

```

4.10 CWE-532: Insertion of Sensitive Information into Log File

A correção para esta vulnerabilidade é muito intuitiva. Passa por simplificar o que é escrito no registo e por omitir informação desnecessária e sensível que não interessa ou que não diz respeito a outros utilizadores.

Devido ao *back-end* de escrita do ficheiro de registo ter sido omitido, a solução para esta CWE é apresentada diretamente no novo ficheiro `log.php` na pasta `php/` da *app* segura.

Ver Registo de Atividade

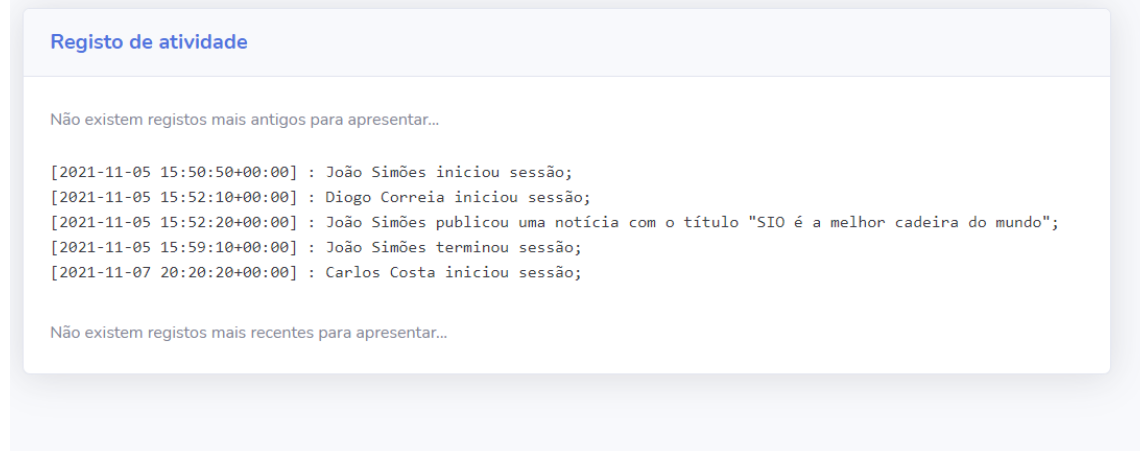


Figura 4.8: Informação sensível omitida e simplificada no log.php

4.11 CWE-549: Missing Password Field Masking

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 carácter por um ponto/círculo preto. Torna-se mais seguro assim o utilizador iniciar sessão em locais públicos, sem receio de oferecer a sua palavra-passe a alguém com más intenções.

Outra alternativa, ainda mais segura, seria não mostrar sequer quantos caracteres o utilizador está a digitar. Esta tática é utilizada por exemplo no terminal quando é necessário introduzir credenciais (Git, `sudo`, ...).

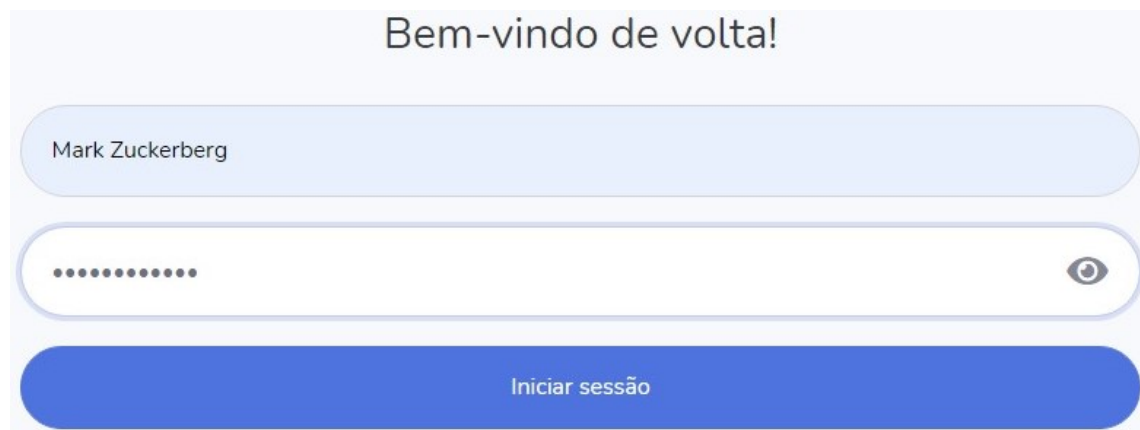


Figura 4.9: Password não visível na versão segura da app

4.12 CWE-552: Files or Directories Accessible to External Parties

Para corrigir esta vulnerabilidade, o processo é bem simples. Fazendo uso do ficheiro `.htaccess` mais uma vez, colocando as seguintes regras é possível ocultar a listagem de ficheiros em qualquer diretório, apresentando um erro 403 *Forbidden* ao utilizador.

```

1 # Hide content of folders returning a 403 Forbidden error
2 IndexIgnore *
3 Options -Indexes

```

Este erro 403 através da regra `ErrorDocument`, também presente no `.htaccess`, redireciona o utilizador para uma página de erro personalizada (`403.php`), em vez de mostrar a página de erro *default* do Apache, como acontece na versão insegura da *app*.



Figura 4.10: Página de erro personalizada - 403 Forbidden

4.13 CWE-799: Improper Control of Interaction Frequency & CWE-307: Improper Restriction of Excessive Authentication Attempts

Para prevenir que um possível atacante, através de um processo de tentativa-erro, descubra informação sensível da nossa aplicação como um *username* existente, procedemos à abordagem seguinte. Para além das colunas tabela *users* já existentes na versão da aplicação insegura, foram adicionadas mais duas, constituindo a tabela *users_sec*: *'login_count'*, que é uma variável inteira que incrementa em 1 de cada vez que um utilizador introduz uma *password* incorreta; *'login_timestamp'*, que é uma variável *timestamp* que regista o momento em que o utilizador está a realizar o *login*.

```

1 <?php
2 // Setup users table secure
3 $users_sec = "CREATE TABLE `admin`.`users_sec` (
4     username VARCHAR(255) NOT NULL ,
5     email VARCHAR(255) NOT NULL ,
6     pwd VARCHAR(255) NOT NULL,
7     login_count INT NOT NULL,
8     login_timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,

```

```

9      id INT NOT NULL PRIMARY KEY AUTO_INCREMENT
10    );

```

Através destas duas é verificado se um dado utilizador falhou em 3 tentivas de *login* seguidas. Se tal acontecer, esse utilizador é colocado no estado *'timeout'* ficando sem acesso à *web app* durante 10 minutos. Esta verificação é feita através do seguinte código:

```

1  <?php
2  $attempts_limit = 3;    // 3 attempts
3  $lockout_time = 600;    // 600 seconds = 10 minutes
4
5  (...)
6
7  // fetch rows
8  if ($row = mysqli_fetch_assoc($result)) {
9
10     $timestamp_failed_login = $row['login_timestamp'];
11     $attempts = $row['login_count'];
12
13     if( ($attempts >= $attempts_limit) &&
14         (time() - strtotime($timestamp_failed_login) < $lockout_time) ){
15         // User is lockout, too many attempts made
16         header("Location: login.php?submit=lockout");
17         exit();
18     }
19 }
20 );

```

Em seguida segue-se uma representação visual:

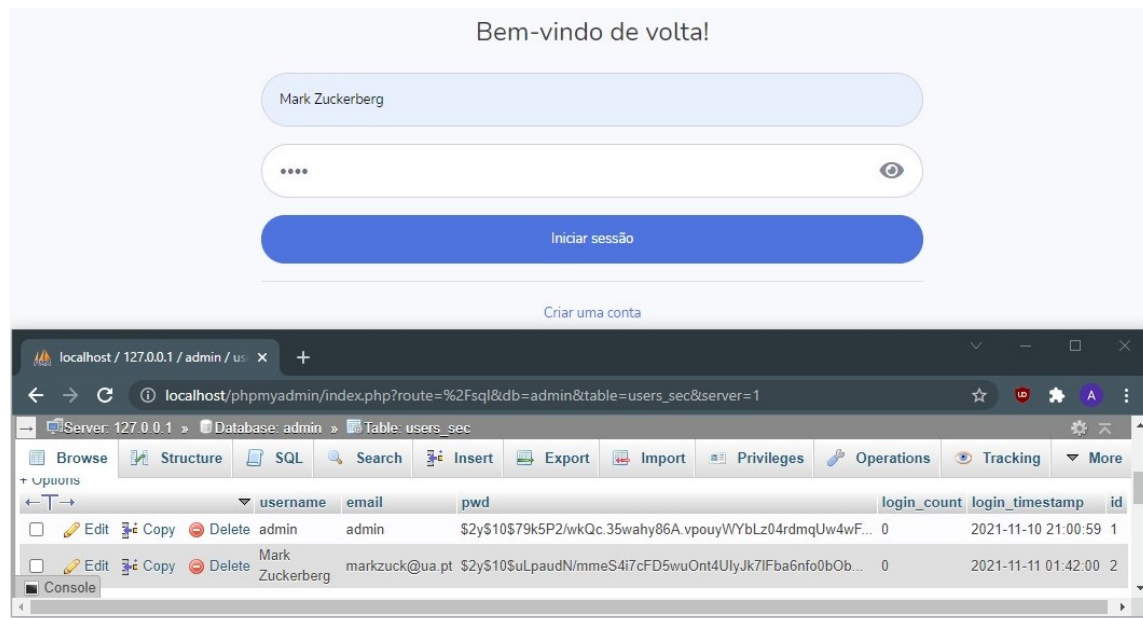


Figura 4.11: Primeira tentativa de login com o username Marc Zuckerberg

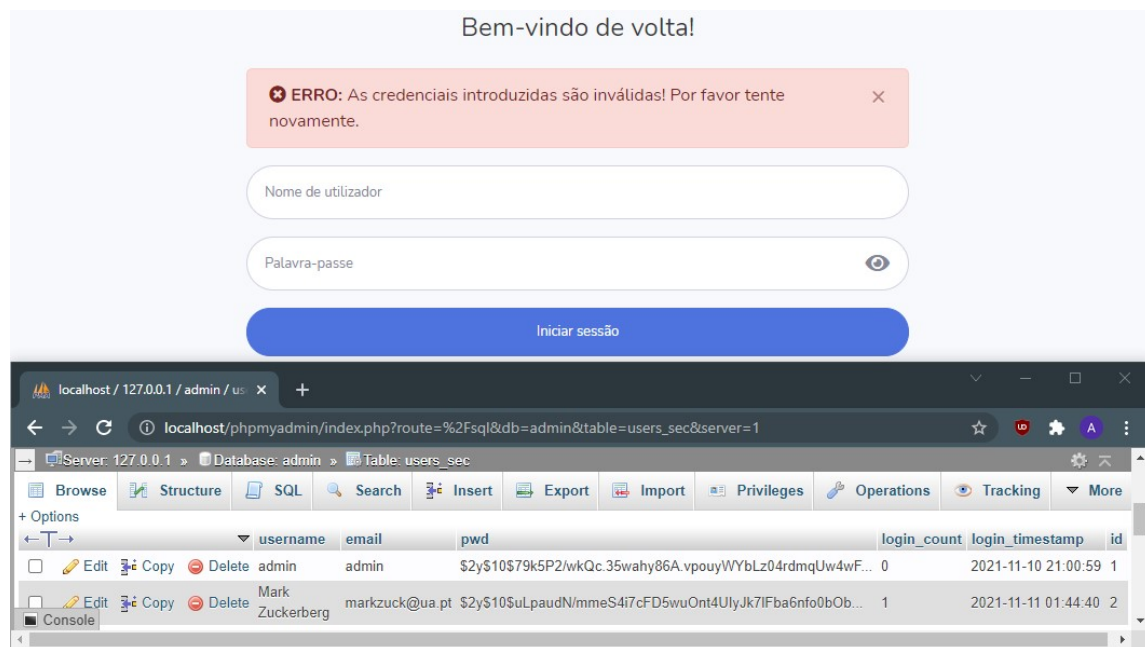


Figura 4.12: Após 1 tentativa de login sem sucesso

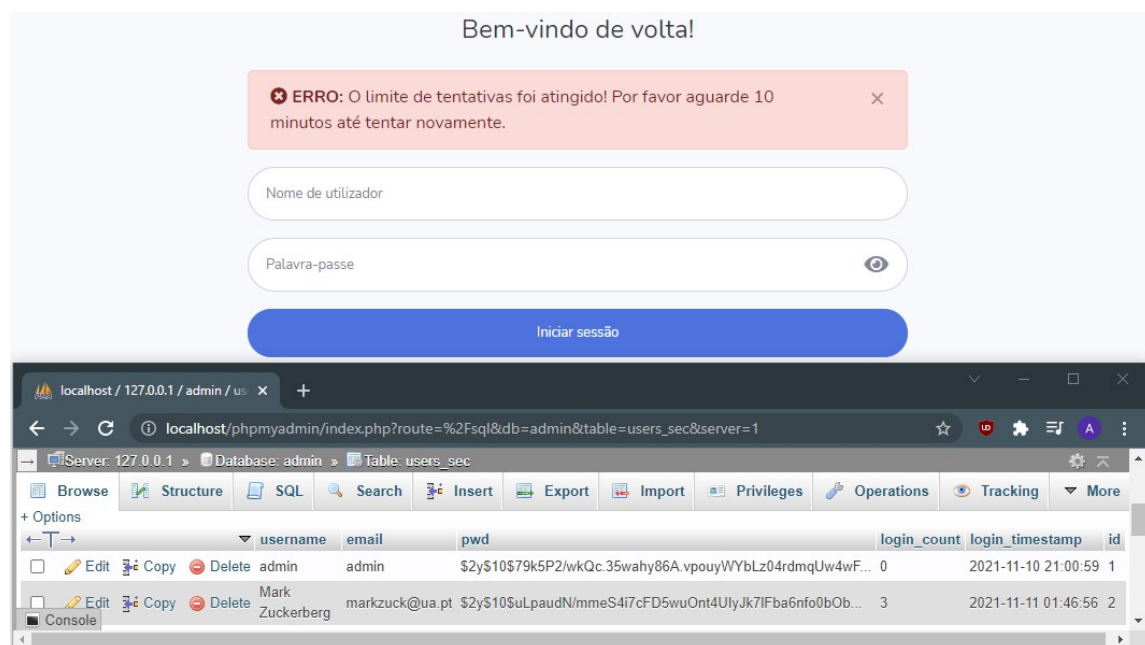


Figura 4.13: Após 3 tentativas de login sem sucesso, o utilizador Marc Zuckerberg fica impedido de fazer login por 10 minutos

4.14 CWE-862: Missing Authorization & CWE-522: Insufficiently Protected Credentials

Para assegurar que o utilizador que está a fazer a alteração da *password* da conta é o dono da mesma, é realizada uma verificação através do pedido da *password* atual da conta antes de ser autorizado tais alterações com a criação do ficheiro `change-password.php`.

Alterar palavra-passe

Alteração da palavra-passe

Palavra-passe antiga:



Nova palavra-passe:



Confirmar nova palavra-passe:



Figura 4.14: Adicionado um novo campo a solicitar a escrita da password antiga antes de se dar autorização para a alteração da mesma

O código usado para validar a *password* antiga é muito semelhante ao código de validação de autenticação usado na página `login.php`.

```
1 <?php
2 // check for old password
3 $sql = "SELECT * FROM users_sec WHERE username=?;";
4 $stmt = mysqli_stmt_init($conn);
5 // ...
6 $row = mysqli_fetch_assoc(mysqli_stmt_get_result($stmt));
7 if (!password_verify($oldPwd, $row['pwd'])) {
8     header("Location: change-password.php?submit=oldpwderror");
9     exit();
10 }
```

4.15 Mecanismos de segurança adicionais

Para além das soluções para as vulnerabilidades descritas, foram ainda implementados outros mecanismos de segurança adicionais na versão segura da *app*.

4.15.1 Extensão .php dos URLs

Por uma questão de estética e também por questões de segurança, optámos por retirar a extensão no final dos *links* das páginas. Não só fica visualmente mais agradável, como também torna menos óbvio que linguagem de programação a *app* utiliza para *back-end*.

4.15.2 robots.txt

Foi também implementado um ficheiro "robots.txt" na versão segura da *app*. Este ficheiro TXT indica para os *bots* de *crawling* do Google, Bing e outros motores de busca quais as páginas da plataforma que possam ou não ser acedidas por estes mecanismos de pesquisa.

Ora, sendo esta uma plataforma de administração de um *blog* de notícias, não faz sentido estas páginas serem indexadas. Então, para evitar que conteúdo (excertos de texto, imagens, ...) destas páginas apareça em resultados dos diversos motores de busca, desativou-se o *crawling* por parte destes robôs para todos os *links* da área de administração.

```
# Aplica a regra a todos os bots (Googlebot, Bingbot, DuckDuckBot, etc.)
User-agent: *
# Impede os bots acima de vasculharem qualquer página
Disallow: /
```

4.15.3 Página de erro 403 e 404

Como já foi referido anteriormente, a versão segura da *app* contém o ficheiro `.htaccess` com regras `ErrorDocument` para redirecionar o utilizador para páginas de erro personalizadas, ao invés de utilizar as páginas de erro *default* do Apache, como acontece na versão insegura da *app*. Estas últimas costumam fornecer alguma informação avançada sobre o servidor e assim sempre é mais uma camada extra de segurança.

No caso da nossa plataforma, as páginas de erro implementadas dizem respeito ao erro HTTP 403 *Forbidden* (figura 4.10) e ao erro HTTP 404 *Not Found* (figura abaixo).



Figura 4.15: Página de erro personalizada - 404 Not Found

Capítulo 5

Conclusão

Durante a realização deste trabalho, não só adquirimos conhecimentos de várias CWEs na implementação da versão insegura da app, mas também de como resolver essas mesmas vulnerabilidades através da realização da `app_sec`.

Acreditamos que ao forçarmos propositadamente as CWEs a acontecerem, conseguimos perceber melhor ainda o seu comportamento e a sua solução. E certamente que estaremos atentos para, no futuro, não cometermos erros semelhantes e protegermos devidamente o nosso código.