



INSTITUTO POLITECNICO CASTELO BRANCO

ESCOLA SUPERIOR DE TECNOLOGIA

Curso de Mestrado

DESENVOLVIMENTO DE SOFTWARE E SISTEMA INTERACTIVO

Disciplina: Segurança de software

Vulnerabilidades das páginas web

Aluno:

Leandro Tito Manjate

Professor:

Osvaldo Santos

Castelo Branco, ao 25 de janeiro 2023

Conteúdo

Introdução	4
1. Proteção contra divulgação de informação não planeada.	5
2. Proteção contra ataque de sessão <i>hijacking</i>	6
3. Protecção contra SQL injection	8
4. Proteção contra ataque de Cross site scripting	11
4.1. Falha sem uso da base de dados.....	11
4.2. Falha com uso de base de dados.....	13
5. Proteção de ficheiro de upload	15
6. Proteção das palavras-chaves	16
6.1. Falha de encriptação da palavra-chave.....	16
6.2. Falha no Login	18
7. Proteção de base de dados	19
7.1. Criação de utilizadores de base de dados	19
7.2. Criação de procedure store.....	22
7.3. Atribuição de privilégio mínimo para cada utilizador.	24
Conclusão	25
Bibliografia	26

Índice de Tabelas

Tabela 1 Credencias de utilizadores da base de dados.....	19
---	----

Índice de Figuras

Figura 1 Script que indica que a visualização de erros está ativo	5
Figura 2 Resultado quando a visualização de erros está ativo	5
Figura 3 Desativar a visualização de erros	6
Figura 4 Resultado apos desligar a visualização de erros	6
Figura 5 Login com Leandro e a verificação do cookie antes da correção.....	7
Figura 6 Login com Nildo e a verificação do cookie antes da correção	7
Figura 7 Script corrido que cria um cookie.	7
Figura 8 Login depois da correcção.....	8
Figura 9 Login depois da correcção.....	8
Figura 10 Script desprotegido contra SQL Injection.....	8
Figura 11 Testado o site com dados correcto	9
Figura 12 Testar a página web com SQL injection	9
Figura 13 Utilização de PDO em vez de MySQLi	9
Figura 14 Query preparado para entrar na base de dados.....	9
Figura 15 Transferir os dados de \$id para :id.....	10
Figura 16 Script utilizado para visualizar o resultado no ecrã.....	10
Figura 17 Erro mostrado quando um script sql é injectado na entrada	11
Figura 18 Script vulnerável a Cross-Site-Script.....	11

Figura 19 Teste com paramentos de entrada correcta.....	11
Figura 20 Resultado com parâmetros correto	12
Figura 21 Inserido um script malicioso da entrada	12
Figura 22 Resultado do Script malicioso	12
Figura 23 script protegido de Cross-Site-Script.....	13
Figura 24 Resultado quando o Cross-Site-Script encontra-se protegido.....	13
Figura 25 Script vulneravel.....	13
Figura 26 Teste da página com parâmetros corretos	14
Figura 27 Script utilizado para evitar a inserção de script malicioso na base de dados	14
Figura 28 Utilizado para proteger os dados que serão visualizado no ecrã	14
Figura 29 Resultado da pagina web quando corrigida.....	14
Figura 30 Inserção de file que não são imagem.....	15
Figura 31 Resultado quando inserido um file que nao seja imagem	15
Figura 32 Script que mostra as variáveis que são aceite e a extensão do ficheiro.....	15
Figura 33 Condição para verificar se o file é uma imagem	16
Figura 34 A condição que transferido ficheiro para o target file	16
Figura 35 Resultado quando o codigo encontra corretamente codificado	16
Figura 36 variavel que inserem a palavra-chave em texto	17
Figura 37 Teste de registar na base de dados.....	17
Figura 38 Visualização dos dados inserido na base de dados.....	17
Figura 39 Palavra-chave encriptada com hash	17
Figura 40 Teste quando a palavra chave foi encriptada	18
Figura 41 Visualização dos dados na base de dados.....	18
Figura 42 Condição incorreta onde a comparação é exata com a informação na base de dados	18
Figura 43 Comapração de hash da palavra chave inserida com o hash da palavra chave inserido na base de dados.....	19

Introdução

Hoje em dia várias páginas web são atacadas por causas dos programadores não estarem a utilizar os *Best practice* ao longo do código assim tornando os seus códigos vulneráveis e fáceis de serem hackeados por um utilizador mal intencionado. Portanto, no presente trabalho, serão abordadas as possíveis vulnerabilidades nas páginas web e a forma de protegê-las contra ataques de forma a deixá-las o mais seguras possíveis. Portanto, primeiro será abordado o tema de má configuração onde o utilizador mostrar os erros ao utilizador. Depois será mostrado um cenário onde os cookies mantêm-se o mesmo, até quando um novo utilizador entra no mesmo ecrã. Depois será apresentado um cenário que procura o ID do utilizador que o mesmo pode ser atacado por *SQL injection*. Em seguida será mostrado um cenário onde o utilizador escreve qualquer coisa e nesse cenário o mesmo pode ser atacado por um Cross-Site-Script. Temos um cenário onde será mostrado como proteger os dados que serão inseridos na base de dados e como proteger os dados mostrados no ecrã do utilizador. Depois será verificada a vulnerabilidade de upload de ficheiro onde queremos que somente o upload aceite imagem e outras extensões sejam rejeitadas. Em seguida tem um cenário de Regista que guarda a palavra-passe como um texto normal e objetivo e de encriptar a palavra na chave e guardar encriptado na base de dados. Depois temos o Login que inicialmente faz a verificação direta do texto inserido pelo utilizador com o texto escrito na base de dados e depois a verificação é corrigida. Em seguida serão criadas variáveis de ambiente a fim de esconder as credenciais das bases de dados. Depois será criada a *procedure store* com o objetivo de esconder os *scripts* de *sql* no nosso projeto e finalmente será feito o privilégio mínimo que consiste em cada utilizador somente ter um único privilégio específico.

1. Proteção contra divulgação de informação não planeada.

Normalmente ao produzir o código o programador tem tendência de ativar o *display_errors*, *display_startup_errors*, *error_reporting* com o objetivo de encontrar os erros facilmente quando ao longo do desenvolvimento do projeto. Ao desenvolver o projeto é uma boa técnica, porém quando desejamos concluir o projeto para meter no servidor afim do utilizador poder usar não é uma boa pratica, porque assim os utilizadores vão saber os locais onde o projeto contém o erro e se o utilizador for uma pessoa tecnicamente qualificada e com ma intenções pode utilizar essa informação para atacar a pagina. Portanto no trabalho foi criado um exemplo que esta no ficheiro *displayError.php* que contém código errado propositalmente com objetivo de vermos o que acontece quando ligamos a visualização de erro no código.

Portanto a fim de verificamos os erros no código foi colocado no início do ficheiro *php* os seguinte trecho de código.

```
// simulates turning display errors on
ini_set('display_errors', 1);
ini_set('display_startup_errors', 1);
error_reporting(E_ALL);
```

Figura 1 Script que indica que a visualização de erros está ativo

Tendo esse trecho quando o ficheiro tem erro sempre será visualizado na página do utilizador conforme mostrado na figura seguinte.



Figura 2 Resultado quando a visualização de erros está ativo

Portanto como podemos verificar a nossa página disponibiliza ao utilizador a informação de erros e com esses dados um utilizador mal-intencionado pode fazer muitas coisas. Portanto uma melhor forma para proteger a página é de desativar essa funcionalidade quando o projeto estiver concluído a fim de ser utilizado para utilizar.

Pode se fazer isso de duas formas, a primeira é ir a o ficheiro de *php.ini* afim de desativar essas funcionalidade. A segunda forma é de desativar no código onde em vez de colocar

1 que significa ativo colocamos 0 que significa que não está ativo conforme podemos ver na seguinte imagem.

```

0
1 // simulates turning display errors off
2 ini_set('display_errors', 0);
3 ini_set('display_startup_errors', 0);
4
5

```

Figura 3 Desativar a visualização de erros

Ao correr o programa novamente apesar da página continuar com erro o utilizador não vai poder ver os erros.



Figura 4 Resultado após desligar a visualização de erros

2. Proteção contra ataque de sessão *hijacking*

Normalmente quando faz-se login para uma conta para os utilizadores é gerado um cookie que podemos dizer que guarda as informações feitas do utilizador. Portanto com isso um *hacker* se de alguma forma conseguir pegar esse cookie ele pode aceder a conta do utilizador e os dados cruciais do utilizador. Como podemos verificar no exemplo seguinte vamos fazer login com utilizadores diferentes e será verificado que no mesmo browser é sempre gerado o mesmo *cookie*. O ficheiro utilizado para verificar o cenário é *hijacking.php*.

Portanto primeiro entramos primeiro será feito login com Leandro e podemos verificar que o cookie desse login é *a0vieqko04kdn86jolqblv2ljk*



Figura 5 Login com Leandro e a verificação do cookie antes da correção

Agora podemos testar com um outro utilizador e podemos verificar que o cookie continua sendo o mesmo.



Figura 6 Login com Nildo e a verificação do cookie antes da correção

Dessa forma o projeto fica vulnerável para ataques, portanto será necessário proteger-lho. A fim de fazer isso será necessário acrescentar o trecho `session_regenerate_id()` após o `session_start()` no código `hijacking.php`. Com esse trecho irá criar um cookie sempre que ser feito um login assim dificilmente um atacante poderá aceder ao login de um outro utilizador através de cookies.

Podemos ver na imagem seguinte como o código corrigido ficava.

```
<?php
// preventing session hijacking and for
// use session_regenerate_id() to maint

session_start();

session_regenerate_id();
```

Figura 7 Script corrido que cria um cookie.

Podemos verificar o seguinte exemplo onde o cookie possui o id `l9924072p0m4vv3iecg16ddlv0`



Figura 8 Login depois da correcção

E agora se tentar fazer o login com Nildo podemos verificar que fica com um cookie diferente com Id `4i4isudjbm8h24vqf628f4aacq`



Figura 9 Login depois da correcção

3. Protecção contra SQL injection

Hoje em dia o *injection* é uma das formas mais usadas pelos atacantes a fim de conseguir inserir dados indesejada na base de dados ou mesmo apagar a base de dados e outros e isso acontece porque os *query* utilizado para aceder a base de dados não tem o seu devido tratamento. Portanto no ficheiro *index.php* temos um trecho que código que procura os estudante com base de id e mostra o nome do utilizador e o seu respetivo email.

A fim de termos o resultado mencionado em cima é utilizado o seguinte trecho

```
if(isset($_GET['user_id'])&&!empty($_GET['user_id']))
{
    $user_id=$_GET['user_id'];

    $result=mysqli_query($conn,"Select * from users where user_id =".$user_id);

    while($row =mysqli_fetch_assoc($result))
    {
        echo $row['user_name']. "-". $row['user_email'];
    }
}
```

Figura 10 Script desprotegido contra SQL Injection

Portanto se iremos a página *web* e procuramos pelo utilizador 1 iremos ter a seguinte resposta



Figura 11 Testado o site com dados correcto

Portanto parece que o projeto esta funcionar normalmente mais se o atacante colocar na box 1 OR 1=1 o resultado será mostrar todos utilizados contido na base de dados conforme podemos verificar na imagem seguinte



Figura 12 Testar a página web com SQL injection

Portanto isso significa que o site esta vulnerável a *SQL Injection* portanto o que deve ser feito. Deve-se fazer o tratamento de *query* no código. Portanto no ficheiro inicialmente precisamos de utilizar o *PDO* para conectarmos a base de dados ao nosso projeto. O *PDO* é uma camada de abstração para suas consultas de banco de dados e é uma alternativa incrível para *MySQLi*.

```
$con = new PDO('mysql:dbname=php_project;host=localhost', $dbuser, $dbpass);
```

Figura 13 Utilização de PDO em vez de MySQLi

Depois disso temos que preparar o nosso *query* utilizado prepare assim dessa forma se o utilizador *injectar* qualquer *query* não ira funcionar, *:id* é uma forma de variável que vai guardar a informação enviada pelo utilizador

```
$stmt = $con->prepare("SELECT 'user_name', 'user_email' from 'users' WHERE 'user_id' = :id");
```

Figura 14 Query preparado para entrar na base de dados.

Depois temos o binParam que vai passar os dados de utilizador da variável \$id para :id e no fim execute executa a query.

```
// bind parameters for markers
$stmt->bindParam(':id', $id);

// execute query
$stmt->execute();
```

Figura 15 Transferir os dados de \$id para :id

Depois disso utiliza o *bincolumn* que vai enviar o resultado obtido da base de dados para variável *\$user_name* e *\$user_email* e por fim temos uma estrutura de repetição que vai permitir nos visualizar a resposta na página web.

```
$stmt->bindColumn(1,$user_name);  
$stmt->bindColumn(2,$user_email);  
  
print "NAME \t\t\t\t\t Email <br>";  
  
while ($stmt->fetch(PDO::FETCH_BOUND)) {  
    print $user_name . "- \t" . $user_email . "\n";  
}
```

```
$stmt->bindColumn(1,$user_name);  
$stmt->bindColumn(2,$user_email);  
  
print "NAME \t\t\t\t\t\t\t\t\t\t Email <br>";  
  
while ($stmt->fetch(PDO::FETCH_BOUND)) {  
    print $user_name . "\t" . $user_email . "\n";  
}
```

Figura 16 Script utilizado para visualizar o resultado no ecrã.

Portanto depois de todas alterações se testarmos a página web novamente a injeção não vai funcionar conforme podemos verificar na imagem seguinte



Figura 17 Erro mostrado quando um script sql é injectado na entrada

Como podemos ver o código agora está protegido porque sempre se for colocada um código de SQL no código iremos ter erro sempre quando estivermos a executar a query.

4. Proteção contra ataque de Cross site scripting

Também um ataque extremamente utilizado que consiste em colocar um script malicioso que pode fazer que o próximo utilizado sinta os efeitos do mesmo. Portanto para esse tipo de protecção sera analisada duas falhas.

4.1. Falha sem uso da base de dados

Portanto no ficheiro *search.php* foi escrito o seguinte trecho.

```
search.php
1  <?php
2
3  if(isset($_GET['search']))
4  {
5      echo $_GET['search'];
6  }
7
8
9
10 ?>
11
```

Figura 18 Script vulnerável a Cross-Site-Script

Portanto com o código ao irmos a página web podemos escrever qualquer conforme a imagem.

Escreva qualquer coisa

Eu quero passar

Figura 19 Teste com paramentos de entrada correcta

Pos isso teremos o resultado mostrado a abaixo.

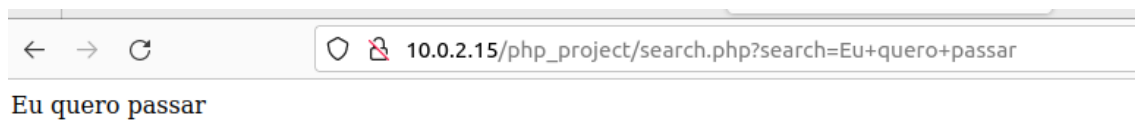


Figura 20 Resultado com parâmetros correto

Portanto até agora parece que o nosso site está a funcionar corretamente. Portanto agora vamos tentar inserir um `<script>alert("A sua pagina foi hackeado")</script>` na caixa e verificar o que vai acontecer.

Escreva qualquer coisa



Figura 21 Inserido um script malicioso da entrada

Depois de inserimos o *script* temos o seguinte resultado



Figura 22 Resultado do Script malicioso

Portanto podemos concluir que estamos vulnerável *Cross site scripting*. Mas apesar de se encontrarmos vulnerável a esse tipo de ataque a forma de prevenir se contra essa ataque é extremamente simples. Somente precisamos de utilizar a função `htmlentities()` que converte todos os caracteres para html assim tornando qualquer dado inserido como um texto normal. Portanto o trecho de código protegido ficaria da seguinte forma.

```

<?php
if(isset($_GET['search']))
{
    echo htmlentities($_GET['search']);
    // Convert speatial character to html
    //echo htmlspecialchars($_GET['search']);
}
?>

```

Figura 23 script protegido de Cross-Site-Script

Portanto com o código protegido teríamos o seguinte resultado.

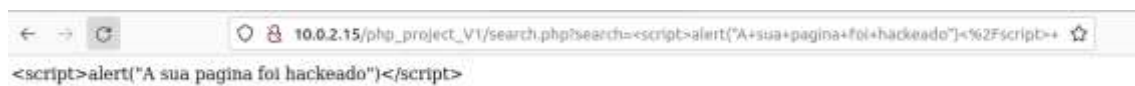


Figura 24 Resultado quando o Cross-Site-Script encontra-se protegido

Assim o código fica seguro porque o script foi tratado com um texto normal.

4.2. Falha com uso de base de dados

A seguir também temos um outro cenário que consiste em atualização de email através de ID onde o utilizador inserir o seu id e depois atualiza o seu respetivo email. Portanto o trecho para atualização do código *atualizar.php* é o seguinte

```

$mysqli = new mysqli("localhost","root","aluno123","php_project");

if(isset($_POST['id']) && isset($_POST['email']))
{
    $id = $_POST['id'];
    $email=$_POST['email'];

    $stmt = $mysqli->prepare('UPDATE users SET user_email=? WHERE user_id=? ');

    $stmt->bind_param("si",$email,$id);

    $result = $stmt->execute();

    echo $email;
}

```

Figura 25 Script vulneravel

Portanto no cenário que desejamos atualizar um email na base de dados funcionaria da seguinte forma.

Manjate

UPDATE EMAIL

ID:

EMAIL:

Figura 26 Teste da página com parâmetros corretos

Portanto parece que funciona normalmente, porém se colocar um script no email o script será escrito na base de dados e o resultado o mal será executado. Portanto devemos corrigir o de forma que isso não aconteça.

Portanto ao atribuir o valor na variável de email teremos que utilizar o seguinte trecho

```
$email=$mysqli->real_escape_string( $_POST['email'] );
```

Figura 27 Script utilizado para evitar a inserção de script malicioso na base de dados

Esse trecho é utilizado normalmente sempre que desejamos inserir algo dado na base de dados assim para o valor ser tratado com uma string de script.

Depois para mostrar no ecrã utiliza-se o *htmlentities* sempre que desejamos mostrar alguma variável que contem um script armazenado.

```
echo htmlentities($email);
```

Figura 28 Utilizado para proteger os dados que serão visualizado no ecrã

Portanto se tentar-nos inserir uma script na caixa de email não vai funcionar conforme vamos mostrar a seguir

```
<script>alert(\"A sua pagina foi hackeado\")</script>
```

UPDATE EMAIL

ID:

EMAIL:

Figura 29 Resultado da pagina web quando corrigida

Assim podemos verificar que a página se encontra protegida.

5. Proteção de ficheiro de upload

A proteção de upload é muito importante porque existe utilizador maliciosa que podem enviar um *malware* executável que pode trazer danos ao sistema ou ao servidor. Portanto para esse caso foi criado um cenário que o utilizador deve somente inserir imagem. Mas se tentarmos inserir um ficheiro html o programa vai funcionar.

File: enviar_mensagem.html

Figura 30 Inserção de file que não são imagem

A seguir podemos ver o resultado

O ficheiro foi carregado de uma forma bem sucedida
File: No file selected.

Figura 31 Resultado quando inserido um file que nao seja imagem

E isso não deveria acontecer, portanto o seguinte deve ser feito.

Portanto para corrigir esse defeito deve-se criar uma variável composta *\$allowed* que armazenara as extensão permitida para o upload e o *\$filename* com objetivo de adquirir o nome para depois adquirir a extensão.

```
//Se o utilizador carregar ficheiros com outra extensão que essas, então não vai aceitar
$allowed = array('gif','png','jpg','jpeg');
$filename= $_FILES['picture']['name'];

//get file extention
$ext = pathinfo($filename,PATHINFO_EXTENSION);
```

Figura 32 Script que mostra as variáveis que são aceite e a extensão do ficheiro

Em seguida devemos criar umas condições que verifica se a extensão adquirida do *\$filename* é diferente das extensões permitida no programa por caso seja significa que não é uma imagem.


```
if(!in_array($ext,$allowed))
{
    $uploadOk=0;
    echo 'Não é uma imagem';
}
```

Figura 33 Condição para verificar se o file é uma imagem

Depois verifica-se o ficheiro é maior do que 1Mbyte e caso não seja no final o ficheiro é enviado para o \$target_file.

```
if(move_uploaded_file($_FILES["picture"]["tmp_name"],$target_file))
{
    $uploadOk=1;
    echo 'O ficheiro foi carregado de uma forma bem sucedida';
}
```

Figura 34 A condição que transferido ficheiro para o target file

Portanto se tentarmos inserir um ficheiro que não seja imagem o programa não vai aceitar e vai aparecer como resultado o seguinte.

Não é uma imagem
File: No file selected.

Figura 35 Resultado quando o código encontra corretamente codificado

Assim podemos verificar que o ficheiro se encontra seguro.

6. Proteção das palavras-chaves

Aqui será abordado duas falhas onde na primeira, a palavra chave não é encriptada e na segunda falha a verificação da palavra chave é feita diretamente com o texto inserido na base de dados e vez de comparar o *hash* do valor inserido.

6.1. Falha de encriptação da palavra-chave

Normalmente programadora inexperiente tem tendência de nas contas para registar armazenar as palavras-chaves em texto assim se alguém consegue aceder a base de dados consegue imediatamente descobrir a palavra-chave de cada utilizador, portanto isso é um problema que deve ser resolvido. Portanto temos o ficheiro *registar.php* que tem as seguintes variáveis


```
$name = $_POST['name'];
$email = $_POST['email'];
$password = $_POST['password'];
```

Figura 36 variável que inserem a palavra-chave em texto

Portanto de forms na página web registrar ira acontecer o seguinte

Register

Username:

Email:

Password:

Figura 37 Teste de registrar na base de dados

A conta será criada de uma forma bem-sucedida. Mas se verificamos na base de dados os utilizador foi registado mas pode se ver a palavra chave da pessoa na base de dados.

24	mateus	mateus@gmail.com	\$2y\$10\$tg4v4eKsvdacUgNae4YmV
25	elias	elias@gmail.com	Procme2022

Figura 38 Visualização dos dados inserido na base de dados

Portanto registrar dessa forma não é segura, portanto devemos modificar o código. Para resolver esse assunto o que pode se fazer é converter a *string* e fazer um *hash* com o objetivo de que na base de dados fique armazenado dados e *hash* que não serão fáceis de ser decodificado.

```
$name = $mysqli->real_escape_string($_POST['name']);
$email = $mysqli->real_escape_string($_POST['email']);
$password = password_hash($_POST['password'], PASSWORD_DEFAULT );
```

Figura 39 Palavra-chave encriptada com hash

Portanto as variáveis mencionado em cima deve estar dessa forma assim o código fica encriptado e o nome e email ficam protegido contra cross site script.

Agora podemos criar mais um utilizador conforme a imagem seguinte.

Register

Username:

Email:

Password:

Figura 40 Teste quando a palavra chave foi encriptada

Depois de registrar podemos verificar na base de dados como a palavra-chave ficou na imagem seguinte.

25	elias	elias@gmail.com	Procme2022
26	yara	yara@gmail.com	\$2y\$10\$B7dt0c30hdc0vgCY.LbEPuVwpk6M0nk0p1WJP3eNXSFUUYm0T.NRW

Figura 41 Visualização dos dados na base de dados.

Assim a palavra-chave fica, mas protegida.

6.2. Falha no Login

Depois temos o login onde no início a verificação da password era feita direta com a informação exata que se encontra na base de dados com o seguinte trecho de código.

```

if (!empty($row)) { // checks if the user actually exists(true/false returned)
    //Login sem a encriptação da palavra chave
    if($password==$user_password){
        echo 'logged in successfully'; // password_verify success!
        echo "<br/>";
        echo htmlentities($user_email);
        echo "<br/>";
        echo htmlentities($user_name);
    } else {
        echo 'failed';
    }
}

```

Figura 42 Condição incorreta onde a comparação é exata com a informação na base de dados

Mas esse código somente funciona se a palavra-chave inserida na base de dados não foi encriptada. Portanto quando a palavra chave inserida na base de dados é encriptada devemos utilizar `password_verify($password, $user_password)` que vai fazer a comparação da palavra chave inserida com o valor da palavra chave adquirida pelo *hash*. O trecho de código para verificação correta é a seguinte.

```

if (!empty($row)) { // checks if the user actually exists(true/false returned)
    //Login com a encriptação da palavra chave
    if (password_verify($password, $user_password)) {
        echo 'logged in successfully'; // password_verify success!
        echo "<br/>";
        echo htmlentities($user_email);
        echo "<br/>";
        echo htmlentities($user_name);
    } else {
        echo 'failed';
    }
}

```

Figura 43 Comapração de hash da palavra chave inserida com o hash da palavra chave inserido na base de dados.

7. Proteção de base de dados

A base de dados e informação relativamente a mesma devem- ser privado e as pessoas não devem ter acesso a mesma. Portanto no trabalho a primeira tarefa feita seria criar utilizadores de base de dados a fim de atribuir privilégio mínimos a cada utilizador de bases de dados. Depois iremos criar funções a fim de esconder os scripts de *sql* e depois teremos que dar o privilegio mínimo a cada utilizador da base de dados.

7.1. Criação de utilizadores de base de dados

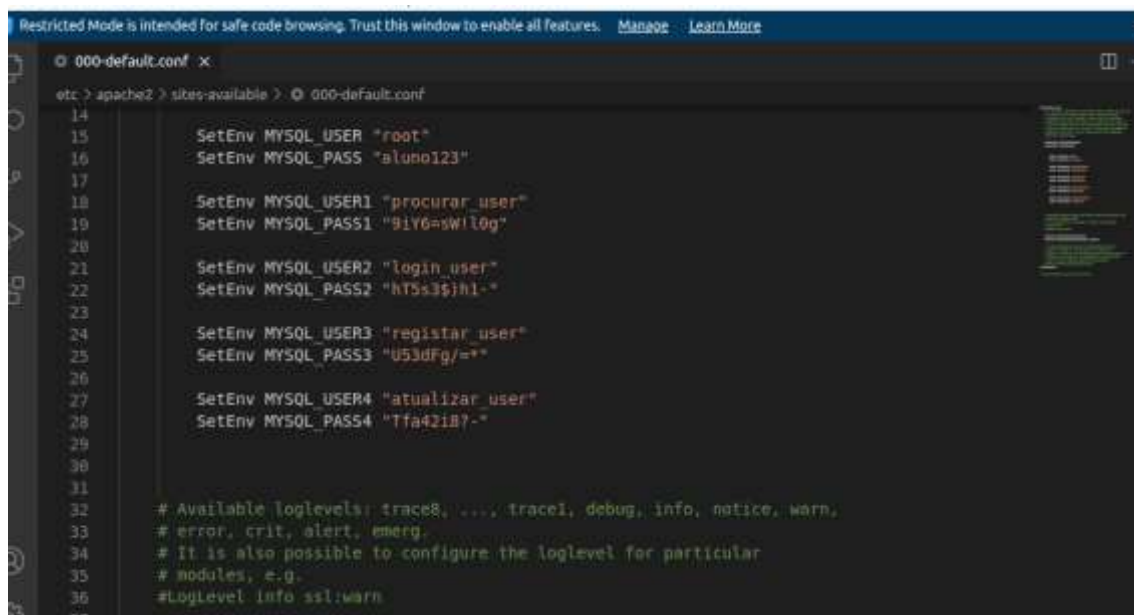
Portanto no início para conexão todo ficheiro estava conectado ao *root* mas na pagina web segura foi utilizado os seguinte utilizador

Variavel de ambiente	utilizador de base de dados	o que o utilizador da base de dados faz
MYSQL_USER1	procurar_user	procurar o utilizador
MYSQL_PASS1	9iY6=sW!l0g	
MYSQL_USER2	login_user	login check
MYSQL_PASS2	hT5s3\$h1-	
MYSQL_USER3	registar_user	registar utilizador
MYSQL_PASS3	U53dFg/=*	
MYSQL_USER4	atualizar_user	atualizar o email do utilizador
MYSQL_PASS4	Tfa42i8?-	

Tabela 1 Credencias de utilizadores da base de dados

Esses utilizadores são criados no *Mysql WorkBench* onde todos privilégios do utilizador criado são retirado e depois cria-se as variáveis e atribui-se a utilizadores de base de dados.

Depois temos que criar as variáveis de ambiente no ficheiro *000-default.conf*. Portanto ao abrir esse ficheiro devemos criar as variáveis e atribuir os utilizadores de base de dados.



```

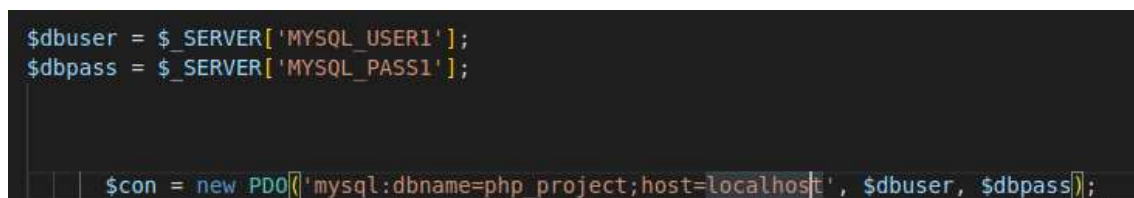
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More

000-default.conf x
etc > apache2 > sites-available > 000-default.conf
14
15     SetEnv MYSQL_USER "root"
16     SetEnv MYSQL_PASS "aluno123"
17
18     SetEnv MYSQL_USER1 "procurar_user"
19     SetEnv MYSQL_PASS1 "9iY6=sWl0g"
20
21     SetEnv MYSQL_USER2 "login_user"
22     SetEnv MYSQL_PASS2 "hT5s3$jhl-"
23
24     SetEnv MYSQL_USER3 "registar_user"
25     SetEnv MYSQL_PASS3 "U53dFg/=*"
26
27     SetEnv MYSQL_USER4 "atualizar_user"
28     SetEnv MYSQL_PASS4 "Tfa42iB?-*"
29
30
31
32     # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
33     # error, crit, alert, emerg.
34     # It is also possible to configure the loglevel for particular
35     # modules, e.g.
36     #LogLevel info ssl:warn
37

```

Figura 44 Criação de variável de ambiente no ficheiro *000-default.conf*

Na ligação da base de dados no ficheiro *index.php* utiliza-se o seguinte script.



```

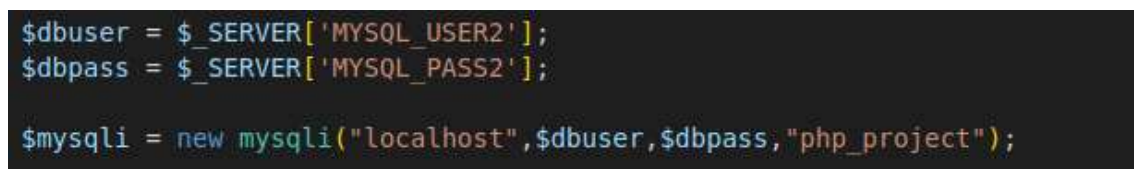
$dbuser = $_SERVER['MYSQL_USER1'];
$dbpass = $_SERVER['MYSQL_PASS1'];

$con = new PDO('mysql:dbname=php_project;host=localhost', $dbuser, $dbpass);

```

Figura 45 Script do *index.php* com variável de ambiente

Na ligação da base de dados no ficheiro *login.php* utiliza-se o seguinte script.



```

$dbuser = $_SERVER['MYSQL_USER2'];
$dbpass = $_SERVER['MYSQL_PASS2'];

$mysqli = new mysqli("localhost", $dbuser, $dbpass, "php_project");

```

Figura 46 Script do *login.php* com variável de ambiente

Na ligação de base de dados do ficheiro *registar.php* utiliza-se o seguinte script.

```

$dbuser = $_SERVER['MYSQL_USER3'];
$dbpass = $_SERVER['MYSQL_PASS3'];
|

$mysqli = new mysqli("localhost",$dbuser,$dbpass,"php_project");

```

Figura 47 Script de registrar.php com variável de ambiente

Na ligação de base de dados do ficheiro *atualizarEmail.php* utiliza o seguinte script.

```

$dbuser = $_SERVER['MYSQL_USER4'];
$dbpass = $_SERVER['MYSQL_PASS4'];

// $mysqli = new mysqli("localhost","root","aluno123","php_project");
$mysqli = new mysqli("localhost",$dbuser,$dbpass,"php_project");

```

Figura 48 Script de atualizarEmail.php com variável de ambiente

Assim tornado o código mais seguro porque a pessoa que tiver o script não saber as credencias da base de dados.

7.2. Criação de procedure store

Alem de tirar as credencias dos nossos ficheiros também podemos tirar os scripts de base de dados e somente colocar métodos de forma a manter mais seguro o nosso sistema.

Para ficheiro *index.php* foi criado o procedimento *procura_utilizador* da seguinte forma

```
CREATE DEFINER='root'@'localhost' PROCEDURE `procurar_utilizador` (
  IN param_id int(11)
)
BEGIN
  SELECT user_name, user_email from users WHERE user_id = param_id;
END
```

Figura 49 Procedimento para procurar o utilizador

Apos a criação do procedimento deve se chamar o método no script CALL *procurar_utilizador(:id)* como podemos ver a seguir.

```
$stmt = $con->prepare("CALL procurar_utilizador(:id);");
```

Figura 50 Usando o CALL para chamar o método procurar_utilizador(:id)

No ficheiro *login.php* também foi criado um método *login_utilizador(?)* com o seguinte script no Workbench.

```
CREATE DEFINER='root'@'localhost' PROCEDURE `login_utilizador` (
  IN param_user_email varchar(100)
)
BEGIN
  SELECT user_name, user_email, user_password FROM users WHERE user_email = param_user_email;
END
```

Figura 51 Procedimento store para login_utilizador

Depois no ficheiro é necessário chamara o método com as suas entradas.

```
$stmt = $mysqli->prepare("CALL login_utilizador( ? );");
```

Figura 52 O chamamento do método login_utilizador(?)

No ficheiro *registar.php* foi criado o método *registar_estudante(?,?,?)* que deve ser criado no *MysqlWorbench*


```

CREATE DEFINER='root'@'localhost' PROCEDURE `registrar_estudante` (
  IN param_user_name varchar(100) ,
  IN param_user_email varchar(100) ,
  IN param_user_password varchar(100)
)
BEGIN
  INSERT INTO users (user_name,user_email,user_password) VALUES (param_user_name,param_user_email,param_user_password);
END

```

Figura 53 Procedimento store registrar_estudante

Depois deve se chamar o método no *script*.

```
$stmt = $mysqli->prepare('CALL registrar_estudante(?,?,?)');
```

Figura 54 Invocar o registrar_estudante(?,?,?)

Depois criamos o método *atualizarEmail(?,?)* e esse método no *Workbench* é criado da seguinte forma.

```

CREATE DEFINER='root'@'localhost' PROCEDURE `atualizarEmail` (
  IN param_user_email varchar(100) ,
  IN param_id int(11)
)
BEGIN
  UPDATE users SET user_email=param_user_email WHERE user_id=param_id;
END

```

Figura 55 Procedimento store atualizarEmail

Depois da criação dos métodos deve-se chamar o método da seguinte forma.

```
$stmt = $mysqli->prepare('CALL atualizarEmail(?,?)');
```

Figura 56 Invocar o método atualizarEmail(?,?)

Assim depois de criar todos métodos e chamar todos métodos o nosso código fica mais seguro pois de um hacker conseguir ter acesso ao nosso código não vai saber qual foi o *script* de *sql* inserido a fim de obter os resultados.

7.3. Atribuição de privilégio mínimo para cada utilizador.

Apos a criação métodos devemos atribuir esses métodos a cada utilizador afim de dar o minimo de privilegio a cada utilizador de base de dados. Portanto teremos de colocar o seguinte script na consola de sql.

```
GRANT EXECUTE ON PROCEDURE php_project.atualizarEmail TO atualizar_user@localhost;  
  
GRANT EXECUTE ON PROCEDURE php_project.registar_estudante TO registar_user@localhost;  
  
GRANT EXECUTE ON PROCEDURE php_project.login_utilizador TO login_user@localhost;  
  
GRANT EXECUTE ON PROCEDURE php_project.procurar_utilizador TO procurar_user@localhost;
```

Figura 57 Atribuição dos privilegio mínimo a cada utilizador de base de dados

Assim cada utilizador da base de dados somente tem um único ação assim evitado certos tipos de ataque a pagina web.

Conclusão

Com o presente trabalho foi possível identificar que existe várias forma que uma hacker pode atacar a uma pagina web e praticas as boas praticas de programação e utilizado certas técnicas podemos garantir que temos um projeto autentico fiel e seguro. Portanto foi verificado que o método *real_escape_string()* deve ser sempre utilizado no caso que desejamos inserir ao dado na base de dados, o método *htmlentities()* deve ser sempre utilizado no caso de se desejar exibir algo no ecrã. Sempre numa determina sessão devemos sempre colocar o método *session_regenerate_id()* como o objectivo de criar sempre um novo cookie sempre que ocorra um nova sessão. A visualização de erros é bom ao longo do desenvolvimento do projeto mas na produção final para entregar o cliente essa opção deve ser desativada. Quando se desenvolve o código de upload devemos sempre definir a extensão aceite e o seu tamanho assim para evitar que o utilizador inserir executável malicioso ou ficheiro demasiado grande que pode dar cabo da memoria de armazenado. Nos caso de registo e for necessário inserir a palavra chave sempre deve-se guardar na base de dados a palavra chave encriptada assim dessa forma mesmo se um hacker aceder a base de dados dificilmente vai conseguir descobrir a palavra chave. Todos os dados relacionado a base de dados no script do projecto deve ser escondido assim para quem apanhar o ficheiros não obter as credencias e os script de base de dados facilmente.

Bibliografia

- Alawi, M. (s.d.). *Security Best Practices For PHP*. Obtido em 24 de janeiro de 2023, de <https://www.udemy.com/course/security-best-practices-for-php/learn/lecture/28880364?start=435#overview>
- Bierer, D. (s.d.). *Infinite Skills' PHP Security Tutorial*. Obtido em 23 de Janeiro de 2023, de <https://www.prweb.com/releases/2013/10/prweb11246608.htm>
- K. S. (s.d.). *PHP: Creating Secure Websites*. Obtido em 24 de Janeiro de 2023, de <https://www.linkedin.com/learning/php-creating-secure-websites-8399320/the-need-for-security-with-php?autoplay=true>