



## **Relatório de Análise**

Segurança Informática e nas Organizações

Departamento de Eletrónica, Telecomunicações e Informática

Ano Letivo 2023/2024

Projeto 1 Vulnerabilidades - Equipa 21

André Miragaia, 108412

João Rodrigues, 108214

Guilherme Duarte, 107766

André Cruz, 110554

Guilherme Andrade, 107696

# Índice

Índice .....	2
Introdução ao Website .....	3
CWEs .....	4
CWE-79 / XSS.....	4
CWE-89 / SQL Injection .....	4
CWE-22 / Path Traversal .....	4
CWE-352 / CSRF .....	4
CWE-434 / Unrestricted Upload of File with Dangerous Type .....	5
CWE-521 / Weak Password Requirements.....	5
CWE-530 / Exposure of Backup File to an Unauthorized Control Sphere .....	5
<b>Vulnerabilidade 0 (CWE-79) .....</b>	<b>6</b>
<b>Vulnerabilidade 1 (CWE-79) .....</b>	<b>9</b>
<b>Vulnerabilidade 2 (CWE-89) .....</b>	<b>12</b>
<b>Vulnerabilidade 3 (CWE-22) .....</b>	<b>16</b>
<b>Vulnerabilidade 4 (CWE-352) .....</b>	<b>21</b>
<b>Vulnerabilidade 5 (CWE-434) .....</b>	<b>24</b>
<b>Vulnerabilidade 6 (CWE-521) .....</b>	<b>28</b>
<b>Vulnerabilidade 7 (CWE-530) .....</b>	<b>32</b>
<b>Bug/Vulnerabilidade .....</b>	<b>35</b>

## Introdução ao Website

Este documento apresenta uma análise detalhada das medidas de segurança implementadas e das potenciais vulnerabilidades identificadas numa possível implementação de uma Loja Virtual do DETI.

Dentro da nossa Loja Virtual do DETI é possível:

- Criar novos utilizadores;
- Fazer login;
- Fazer logout;
- Ver o perfil de utilizador;
- Alterar a password;
- Ver catálogo de produtos;
- Pesquisar na barra de pesquisa pelos produtos;
- Adicionar a quantidade do produto desejado e adicionar ao carrinho;
- Fazer checkout dos produtos que estão no carrinho para finalizar a compra;
- Fazer comentários aos produtos e dar um rating;
- Adicionar, remover ou alterar produtos quando se é um Administrador (utilizador admin);

## **CWEs**

### **CWE-79 / XSS**

O XSS é uma vulnerabilidade que permite ao atacante injetar scripts em javascript ou até mesmo instruções em HTML ou XML de forma a tirar proveito dos utilizadores. O Stored XSS é um ataque em que o script é injetado na página fazendo com que qualquer um que entre na mesma fique comprometido pois, ao contrário do anterior, este fica armazenado no servidor sendo o ataque XSS mais perigoso já catalogado. O Reflected XSS é exatamente o mesmo conceito da anterior mas difere no facto do script não ficar armazenado no servidor, o que significa que o utilizador tem de por exemplo clicar numa url que irá enviar o utilizador para a página vulnerável em que muitas das vezes o script pode ser visualizado através da URL.

### **CWE-89 / SQL Injection**

O SQLI é uma vulnerabilidade que nos permite executar instruções SQL dentro de um sistema. O atacante que explora essa vulnerabilidade tem acesso total à base de dados podendo visualizar tudo o que está dentro dela incluindo os utilizadores e passwords armazenados, mas além disso, dependendo das permissões do utilizador associado à página que faz a conexão com a base de dados o atacante também pode modificar a base de dados e em alguns casos também pode fazer upload de arquivos para dentro do sistema o que torna uma das vulnerabilidades mais perigosas já descobertas.

### **CWE-22 / Path Traversal**

Esta vulnerabilidade permite que seja fornecido um caminho para o arquivo que se deseja abrir. No entanto, o programa não verifica corretamente o caminho que foi fornecido. Isso significa que se consegue fazer alterações no caminho que fazem com que a página abra arquivos em lugares onde não deveria e acaba por retornar para o atacante o conteúdo presente nessas páginas. Em termos simples, a página não está a ter cuidado suficiente com os arquivos que são solicitados.

### **CWE-352 / CSRF**

O CWE-352 também conhecido como CSRF, baseia-se na “confiança” que uma aplicação web tem sobre o navegador que o utilizador está a utilizar, ou seja a app confia que todos os

pedidos enviados pelo navegador são autorizados pelo utilizador. No fundo, pode existir uma entidade de fora que pode comunicar com a aplicação web através de requests que não são intencionais por parte do utilizador, como estes vão ser tratados como requests autênticos, esta situação pode resultar em exposição de dados indesejada ou até mesmo uma execução de código não pretendida.

#### **CWE-434 / Unrestricted Upload of File with Dangerous Type**

O CWE-434, como o nome já indica, baseia-se na possibilidade de um agente poder fazer upload de ficheiros com extensões que não deveriam ser aceites facilitando o upload de webshells, backdoors, etc. Por exemplo, se um atacante consegue carregar e executar um script PHP num servidor que não tem as devidas restrições, ele poderia potencialmente causar danos à aplicação WEB.

#### **CWE-521 / Weak Password Requirements**

O CWE-521 refere-se a uma categoria de vulnerabilidades de segurança quando um sistema ou aplicação não impõe uma política de criação de passwords robustas. Por norma, os utilizadores utilizam passwords que são fáceis de memorizar e nem sempre seguem boas práticas no que toca à criação das mesmas. Esta fragilidade resulta na utilização de passwords que são facilmente adivinhadas que podem ser facilmente “quebradas” através de ataques de brute force. Neste caso, a ausência de controlos e limitação no que toca ao número de tentativas de login, prazos de validade de senhas, ou requisitos de variação entre senhas antigas e novos, podem aumentar o risco de comprometimento de contas.

#### **CWE-530 / Exposure of Backup File to an Unauthorized Control Sphere**

A CWE-530 é uma fraqueza de segurança que ocorre quando arquivos de backup, que geralmente contêm dados sensíveis, são expostos a pessoas que não têm autorização para acedê-los. Essa exposição pode acontecer por diversos motivos, tais como controlos de acesso inadequados, falta de criptografia ou armazenamento em locais inseguros.

## Vulnerabilidade 0 (CWE-79)

Esta vulnerabilidade foi identificada na barra de pesquisa dos produtos, onde qualquer utilizador mal-intencionado pode introduzir um script e este é executado na página web. Por este não ficar armazenado no servidor este pode ser catalogado na categoria de Reflected XSS.

### O que é que o atacante ganha?

Num ataque de Cross-Site Scripting (XSS) bem-sucedido, o atacante pode ganhar acesso a informações sensíveis dos utilizadores, como dados de login, cookies de sessão, informações pessoais ou credenciais financeiras.

### Código exemplo não seguro:

```
<?php
include("../config.php");
if($_SERVER["REQUEST_METHOD"] === "POST") {
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);
    if(!($json != NULL && key_exists("keyword", $json))){
        echo json_encode(array("result" => 0, "result_text" => "JSON inválido!"));
        die();
    }else if(empty($json["keyword"])){
        echo json_encode(array("result" => 0, "result_text" => "O campo está vazio!"));
        die();
    }else{
        $keyword = $json["keyword"];
        $search_term = "%" . $keyword . "%";
        $sql = "SELECT * FROM products WHERE name LIKE '$search_term' OR description LIKE '$search_term' OR categories LIKE '$search_term'";
        $stmt = $conn->prepare($sql);
        $stmt->execute();
        $result = $stmt->fetchAll();
        if(count($result) > 0){
            echo json_encode(array("result" => 1, "result_text" => $result, "keyword" => $keyword));
        }else{
            echo json_encode(array("result" => 0, "result_text" => "Produto não encontrado!", "keyword" => $keyword));
        }
    }
}
?>
```

Fig 1- Código não seguro (api/search\_products.php)

Como podem ver no código o arquivo search\_products.php recebe um JSON no seguinte formato: {"keyword": "produto a ser pesquisado"} em que depois o mesmo é reenviado num novo request em formato JSON junto com outras informações para depois serem exibidas na página. O grande problema que faz a vulnerabilidade existir é o facto do JSON recebido não ter qualquer tipo de controlo no que toca a informação que é passada o que faz com que qualquer script enviado seja depois executado no navegador.

## Como testámos a vulnerabilidade:

Para testar a vulnerabilidade decidimos enviar um pequeno script e como podemos visualizar nas figuras subsequentes o script foi executado com sucesso.

O script utilizado no teste foi: ``.

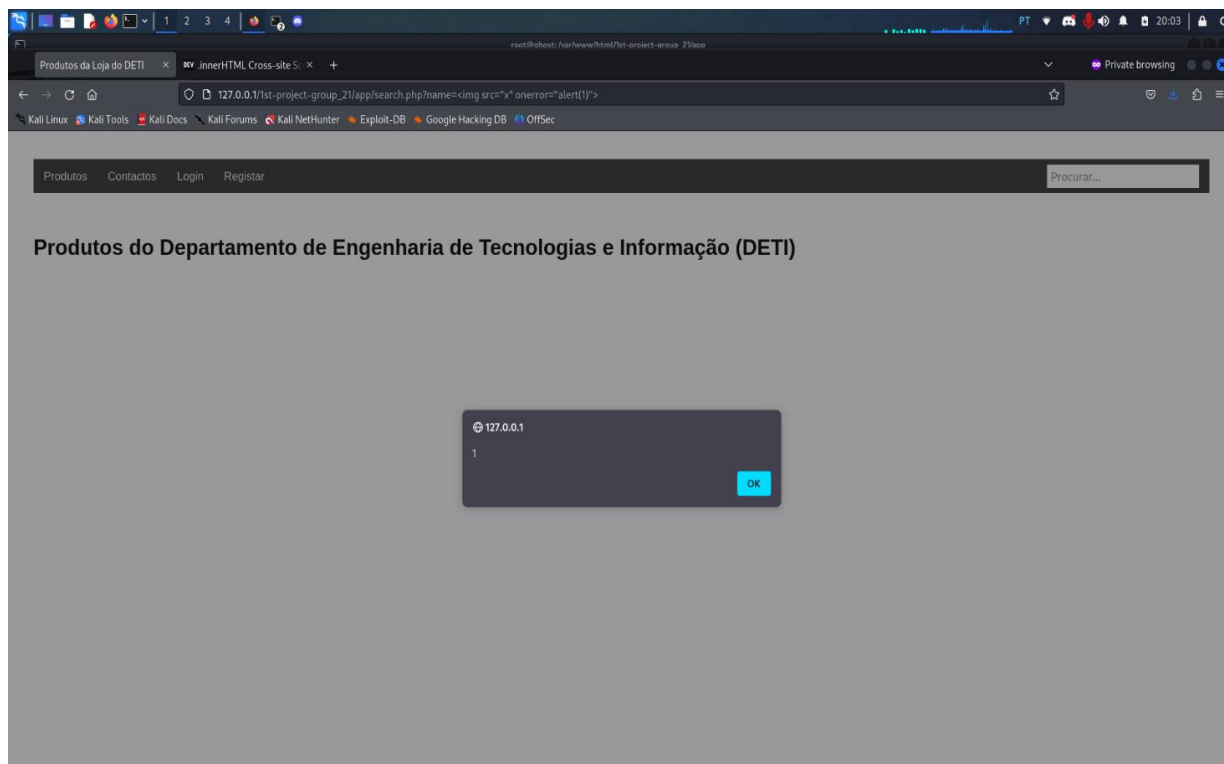


Fig 2- Resultado 1

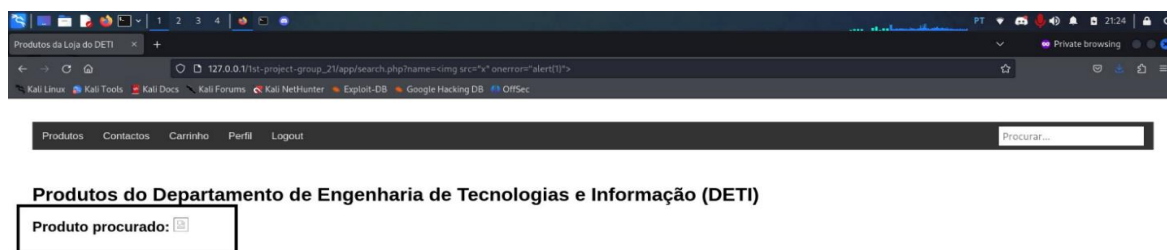


Fig 3- Resultado 2

## Como corrigimos a vulnerabilidade:

Como foi elucidado nos tópicos anteriores a conclusão que tiramos é que a vulnerabilidade existe devido ao facto da informação passada não receber qualquer tipo de filtro e por isso a abordagem utilizada para solucionar o problema foi com o uso de uma função presente no PHP chamada `htmlspecialchars()` que substitui caracteres especiais por entidades em html.

Exemplo:

`<script>alert('xss')</script>` é convertido para `&lt;script&gt;alert(&#039;xss&#039;)&lt;/script&gt;`

```
$keyword = htmlspecialchars($json["keyword"]);
$search_term = "%" . $keyword . "%";
$sql = "SELECT * FROM products WHERE name LIKE '$search_term' OR description LIKE '$search_term' OR categories LIKE '$search_term'";
$stmt = $conn->prepare($sql);
$stmt->execute();
$result = $stmt->fetchAll();
if(count($result) > 0){
    echo json_encode(array("result" => 1, "result_text" => $result, "keyword" => $keyword));
}else{
    echo json_encode(array("result" => 0, "result_text" => "Produto não encontrado!", "keyword" => $keyword));
}
```

Fig 4- Código seguro (api/search\_products.php)

Produtos Carrinho Perfil Logout

## Produtos do Departamento de Engenharia de Tecnologias e Informação (DETI)

Produto procurado: ``

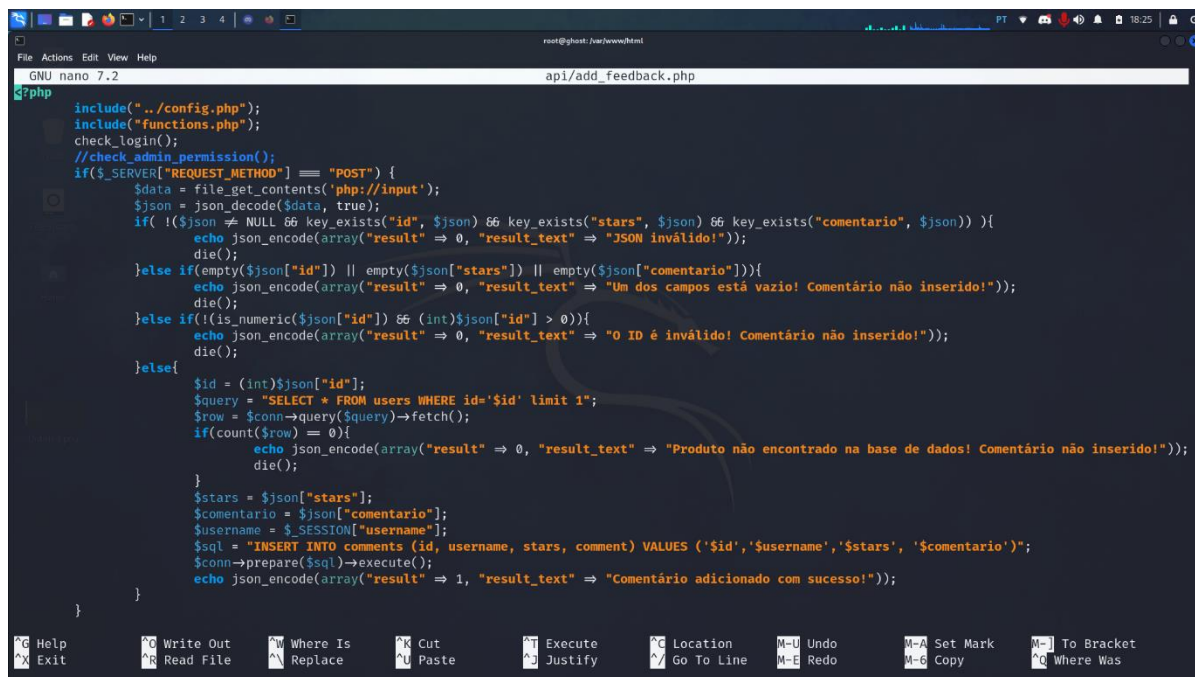
Fig 5- Execução do ataque (mal sucedido)



## Vulnerabilidade 1 (CWE-79)

Esta vulnerabilidade foi identificada na caixa de comentários e está também associada à CWE-79, mas ao contrário da anterior, esta pode ser catalogada como Stored XSS visto que o script introduzido pelo atacante fica armazenado no servidor.

### Código exemplo não seguro:



```
api/add_feedback.php

include("../config.php");
include("Functions.php");
check_login();
//check_admin_permission();
if($_SERVER["REQUEST_METHOD"] === "POST") {
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);
    if (!($json != NULL && key_exists("id", $json) && key_exists("stars", $json) && key_exists("comentario", $json))) {
        echo json_encode(array("result" => 0, "result_text" => "JSON inválido!"));
        die();
    } else if (empty($json["id"]) || empty($json["stars"]) || empty($json["comentario"])) {
        echo json_encode(array("result" => 0, "result_text" => "Um dos campos está vazio! Comentário não inserido!"));
        die();
    } else if (!is_numeric($json["id"]) && (int)$json["id"] > 0) {
        echo json_encode(array("result" => 0, "result_text" => "0 ID é inválido! Comentário não inserido!"));
        die();
    } else {
        $id = (int)$json["id"];
        $query = "SELECT * FROM users WHERE id=$id limit 1";
        $row = $conn->query($query)->fetch();
        if(count($row) == 0){
            echo json_encode(array("result" => 0, "result_text" => "Produto não encontrado na base de dados! Comentário não inserido!"));
            die();
        }
        $stars = $json["stars"];
        $comentario = $json["comentario"];
        $username = $_SESSION["username"];
        $sql = "INSERT INTO comments (id, username, stars, comment) VALUES ('$id', '$username', '$stars', '$comentario')";
        $conn->prepare($sql)->execute();
        echo json_encode(array("result" => 1, "result_text" => "Comentário adicionado com sucesso!"));
    }
}
```

Fig 6- Código não seguro(api/add\_feedback.php)

Ao analisarmos o código podemos concluir que o arquivo add\_feedback.php recebe um JSON no formato `{"id": "1", "stars": "5", "comment": "comentario random"}` que depois é armazenado na base de dados para depois ser exibido na página de comentários de um determinado produto. O grande problema é que a chave **"comment"** não filtra a informação presente no seu valor associado, o que significa que se introduzirmos algum script ele será armazenado na base de dados e depois será inserido na página quando algum utilizador solicitar ao servidor para visualizar os comentários de um determinado produto.

### Como testámos a vulnerabilidade:

No campo de texto onde é possível adicionar o feedback, quando o atacante introduz um script pode-se verificar que este é executado com sucesso e é também guardado no servidor.

O script utilizado no teste foi: ``

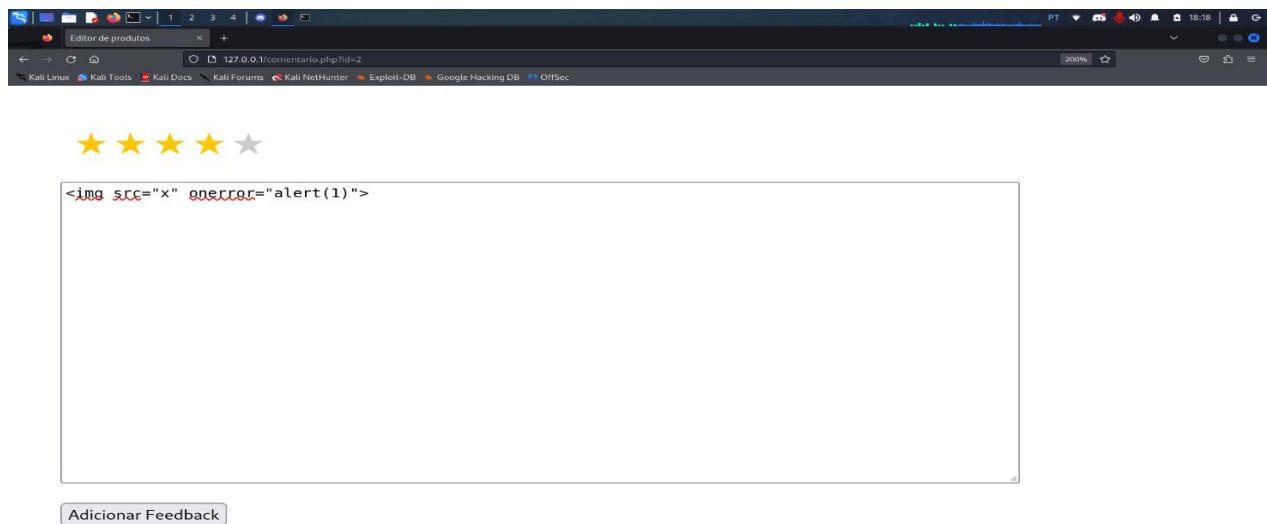


Fig 7- Injeção do script

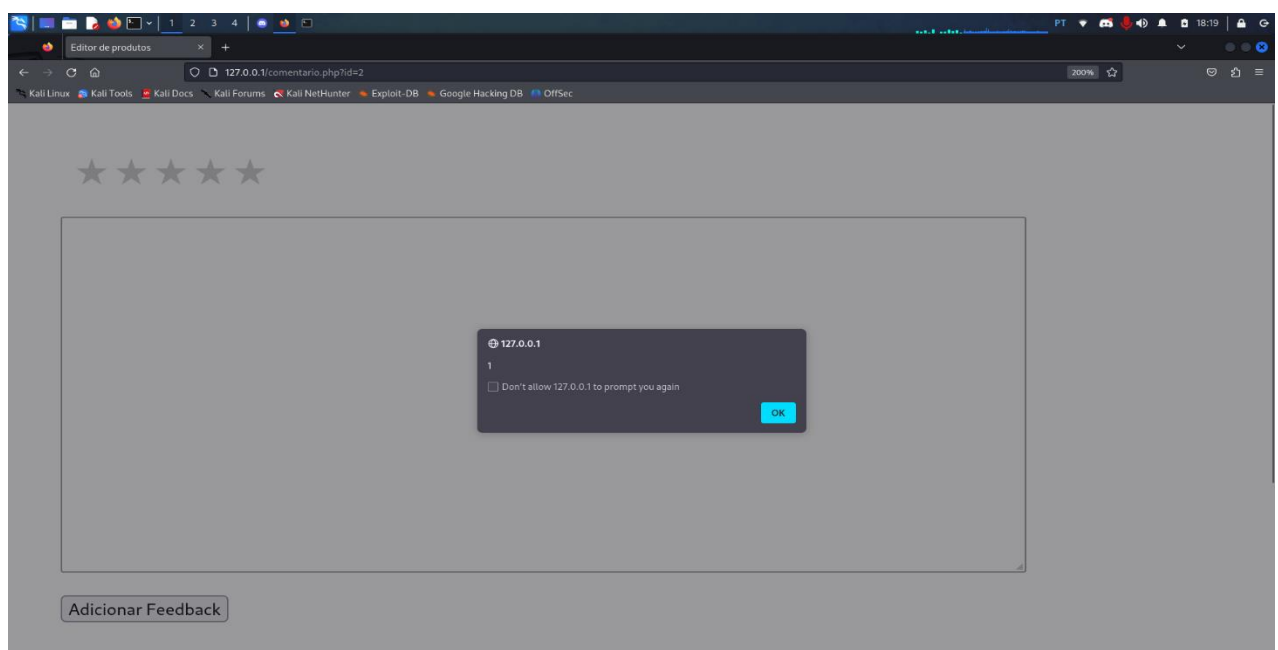


Fig 8- Evidência de que o script é executado

### Como corrigimos a vulnerabilidade:

A correção da vulnerabilidade é a mesma que a da vulnerabilidade anterior, sendo a introdução de um filtro usando a função em PHP **htmlspecialchars()** de modo a transformar todos os caracteres em entidades de HTML.

Exemplo:

**<script>alert('xss')</script>** é convertido para **&lt;script&gt;alert(&#039;xss&#039;)&lt;/script&gt;**

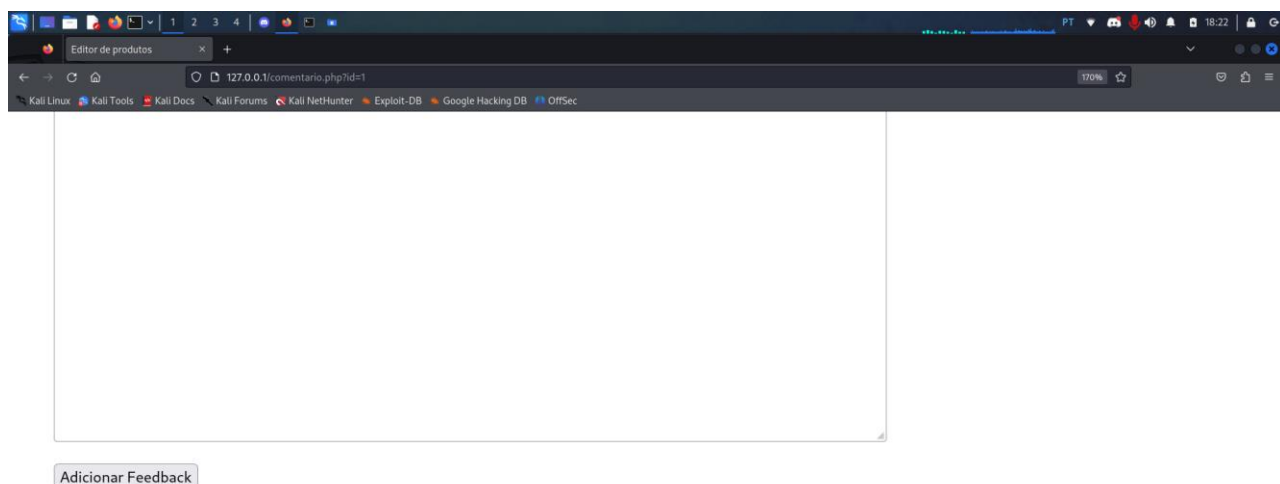
```

GNU nano 7.2 api/add_feedback.php *
<?php
include("../config.php");
include("functions.php");
check_login();
//check_admin_permission();
if($_SERVER["REQUEST_METHOD"] === "POST") {
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);
    if( !($json != NULL && key_exists("id", $json) && key_exists("stars", $json) && key_exists("comentario", $json)) ){
        echo json_encode(array("result" => 0, "result_text" => "JSON inválido!"));
        die();
    }else if(empty($json["id"]) || empty($json["stars"]) || empty($json["comentario"])){
        echo json_encode(array("result" => 0, "result_text" => "Um dos campos está vazio! Comentário não inserido!"));
        die();
    }else if(!is_numeric($json["id"]) && (int)$json["id"] > 0){
        echo json_encode(array("result" => 0, "result_text" => "O ID é inválido! Comentário não inserido!"));
        die();
    }else{
        $id = (int)$json["id"];
        $query = "SELECT * FROM users WHERE id='$id' limit 1";
        $row = $conn->query($query)->fetch();
        if(count($row) == 0){
            echo json_encode(array("result" => 0, "result_text" => "Produto não encontrado na base de dados! Comentário não inserido!"));
            die();
        }
        $stars = $json["stars"];
        $comentario = htmlspecialchars($json["comentario"]);
        $username = $_SESSION["username"];
        $sql = "INSERT INTO comments (id, username, stars, comment) VALUES ('$id', '$username', '$stars', '$comentario')";
        $conn->prepare($sql)->execute();
        echo json_encode(array("result" => 1, "result_text" => "Comentário adicionado com sucesso!"));
    }
}

```

Fig 9- Código seguro(api/add\_feedback.php)

Vulnerabilidade resolvida:



Utilizador: antonio



Comentario: 

Fig 10- É adicionado o comentário introduzido (script) sem ser executado

## Vulnerabilidade 2 (CWE-89)

Esta vulnerabilidade foi identificada no sistema de login.

### O que é que o atacante ganha?

Um atacante que realiza uma injeção de SQL bem-sucedida pode ganhar acesso não autorizado e realizar diversas ações prejudiciais no banco de dados ou no sistema onde a injeção ocorreu. Alguns dos possíveis ganhos para um atacante incluem:

1. **Acesso a Dados Sensíveis:** O atacante pode ler dados sensíveis do banco de dados, como informações pessoais, detalhes financeiros e outros dados confidenciais.
2. **Bypass de Autenticação:** Se o atacante modificar a consulta SQL para burlar a lógica de autenticação, pode conseguir autenticar-se como qualquer usuário, incluindo administradores.
3. **Exclusão de Dados:** O atacante pode eliminar tabelas, registros ou até mesmo bases de dados inteiras, causando danos à integridade e disponibilidade dos dados.
4. **Manipulação de Dados:** Podem ser inseridos, atualizados ou modificados dados no banco de dados, permitindo, por exemplo, a criação de contas de usuário falsas ou a alteração de permissões.
5. **Controlo do Sistema:** Em casos graves, se o banco de dados estiver mal configurado e o atacante tiver permissões elevadas, ele poderá executar comandos do sistema operacional a partir do SQL, o que pode levar ao controlo total do servidor onde o banco de dados está hospedado.
6. **Elevação de Privilégios:** O atacante pode utilizar a SQLI para alterar o esquema do banco de dados ou a lógica das aplicações, concedendo privilégios de administrador a si mesmo, dentro da aplicação

#### Código exemplo não seguro:

```
}else{
    $username = $json['user_name'];
    $password = $json['password'];
    $query = "SELECT * FROM users WHERE username='$username' limit 1";
    $row = $conn->query($query)->fetch();
    if($username == $row["username"] && hash('sha256', $password) === $row["password"]){
        $token = $conn->query("INSERT INTO tokens (username, password, token) VALUES ('$username', '$password', '$token')");
    }
}
```

Fig 11- Código não seguro(api/login.php)

Como podemos ver no código a conclusão que tiramos é que as variáveis **\$username** e **\$password** recebem sucessivamente **\$json["user\_name"]** e **\$json["password"]** e essas informações são extraídas do JSON que é recebido pelo arquivo login.php, presente na pasta api. Como podemos visualizar, não existe qualquer tratamento na informação que é passada nessas variáveis o que permite a uma pessoa mal-intencionada executar instruções SQL. Na variável **\$password** a situação não é crítica visto que a mesma é criptografada para depois ser comparada com a password presente na base de dados, mas no caso do **\$username** o mesmo não acontece e por isso é de extrema importância o tratamento da informação passada por essa variável.

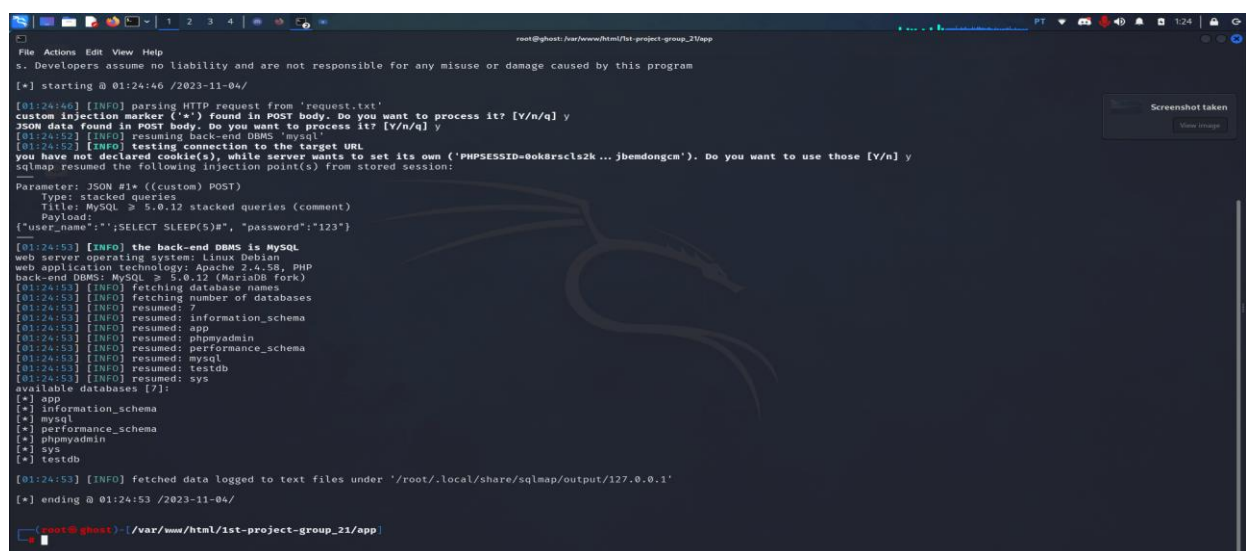
## Como testamos a vulnerabilidade:

Para testar a vulnerabilidade, escolhemos usar o SQLMap, para confirmar que o atacante consegue ter acesso à base de dados através de injeção SQL nos campos de login.

Considerando que o campo da password é cifrado, decidimos proceder da seguinte maneira:

- Criamos um arquivo de texto com o request que pretendíamos fazer, neste colocamos o respetivo host, juntamente com o método que pretendemos (neste caso o POST)
- Passamos no final o JSON que queremos enviar, na chave do “user\_name” colocamos um “\*” de modo a comunicar ao SQLMap que é nessa chave que queremos que os parâmetros SQL sejam testados

Como podemos ver na figura seguinte, a implementação do login está vulnerável a SQL Injection, visto que o SQLMap encontrou todas as bases de dados referentes à aplicação.



```
root@ghost: /var/www/html/1st-project-group_21/app
[*] starting @ 01:24:46 /2023-11-04/

[01:24:46] [INFO] parsing HTTP request from 'request.txt'
custom injection marker ('*') found in POST body. Do you want to process it? [Y/n/q] y
JSON data found in POST body. Do you want to process it? [Y/n/q] y
[01:24:52] [INFO] resuming back-end DBMS 'mysql'
[01:24:52] [INFO] testing connection to the target URL
you have not declared cookie(s), while server wants to set its own ('PHPSESSID=ok8RscIs2k...jbemdongcm'). Do you want to use those [Y/n] y
sqlmap resumed the following injection point(s) from stored session:

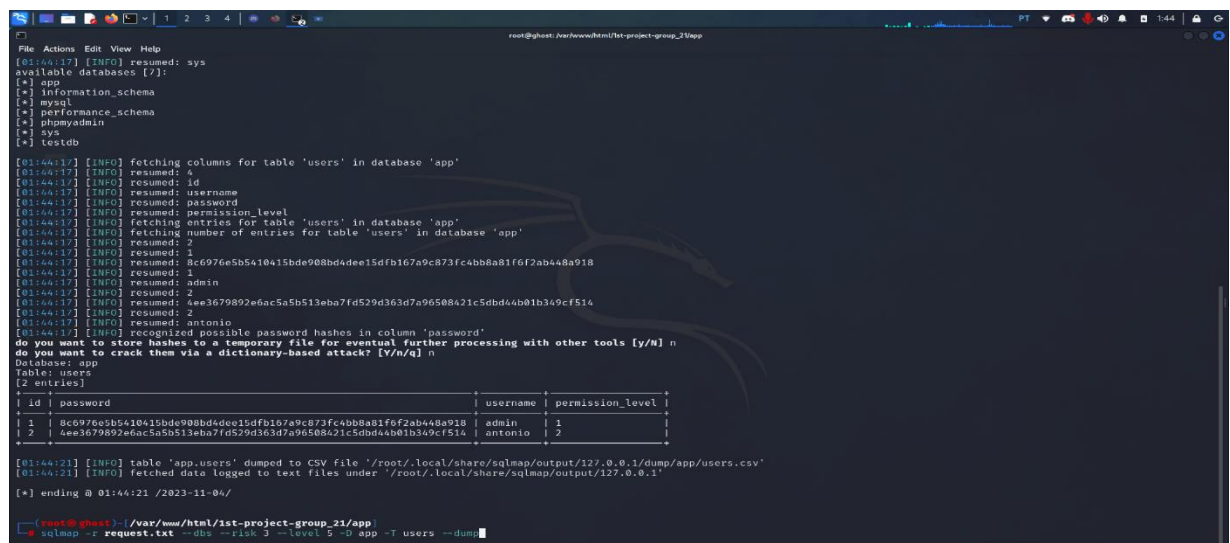
Parameter: JSON #1* ((custom) POST)
Type: stacked queries
Title: MySQL > 5.0.12 stacked queries (comment)
Payload:
["user_name":"*;SELECT SLEEP(5)#", "password":"123"]

[01:24:53] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.58, PHP
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[01:24:53] [INFO] fetching database names
[01:24:53] [INFO] resumed: number of databases
[01:24:53] [INFO] resumed: 7
[01:24:53] [INFO] resumed: information_schema
[01:24:53] [INFO] resumed: app
[01:24:53] [INFO] resumed: phpmyadmin
[01:24:53] [INFO] resumed: performance_schema
[01:24:53] [INFO] resumed: mysql
[01:24:53] [INFO] resumed: testdb
[01:24:53] [INFO] resumed: sys
available databases [7]:
[*] app
[*] information_schema
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] sys
[*] testdb

[01:24:53] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/127.0.0.1'
[*] ending @ 01:24:53 /2023-11-04/

root@ghost: /var/www/html/1st-project-group_21/app
```

Fig 12 – SQLMap encontra as bases de dados



```
root@ghost: /var/www/html/1st-project-group_21/app
[01:44:17] [INFO] resumed: sys
available databases [7]:
[*] app
[*] information_schema
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] sys
[*] testdb

[01:44:17] [INFO] fetching columns for table 'users' in database 'app'
[01:44:17] [INFO] resumed: 4
[01:44:17] [INFO] resumed: id
[01:44:17] [INFO] resumed: username
[01:44:17] [INFO] resumed: password
[01:44:17] [INFO] resumed: permission_level
[01:44:17] [INFO] fetching entries for table 'users' in database 'app'
[01:44:17] [INFO] fetching number of entries for table 'users' in database 'app'
[01:44:17] [INFO] resumed: 2
[01:44:17] [INFO] resumed: 1
[01:44:17] [INFO] resumed: Rc6976e5b5410415bde908bd4dee15dfb167a9c873fc4b8a81f6f2aba448a918
[01:44:17] [INFO] resumed: 1
[01:44:17] [INFO] resumed: admin
[01:44:17] [INFO] resumed: 2
[01:44:17] [INFO] resumed: 4ee3679892e6ac5a5b513eba7fd529d363d7a96508421c5dbd44b01b349cf514
[01:44:17] [INFO] resumed: 2
[01:44:17] [INFO] resumed: antonio
[01:44:17] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with other tools [Y/N] n
do you want to crack them via a dictionary-based attack? [Y/n/q] n
Database: app
Tables: users
[2 entries]
+----+-----+-----+-----+
| id | password | username | permission_level |
+----+-----+-----+-----+
| 1 | Rc6976e5b5410415bde908bd4dee15dfb167a9c873fc4b8a81f6f2aba448a918 | admin | 1 |
| 2 | 4ee3679892e6ac5a5b513eba7fd529d363d7a96508421c5dbd44b01b349cf514 | antonio | 2 |
+----+-----+-----+-----+

[01:44:21] [INFO] table 'app.users' dumped to CSV file '/root/.local/share/sqlmap/output/127.0.0.1/dump/app/users.csv'
[01:44:21] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/127.0.0.1'
[*] ending @ 01:44:21 /2023-11-04/

root@ghost: /var/www/html/1st-project-group_21/app
sqlmap -r request.txt -u http://127.0.0.1:8080/ --level 5 -D app -T users --dump
```

Fig 13 – SQLMap mostra o conteúdo de uma das bases de dados



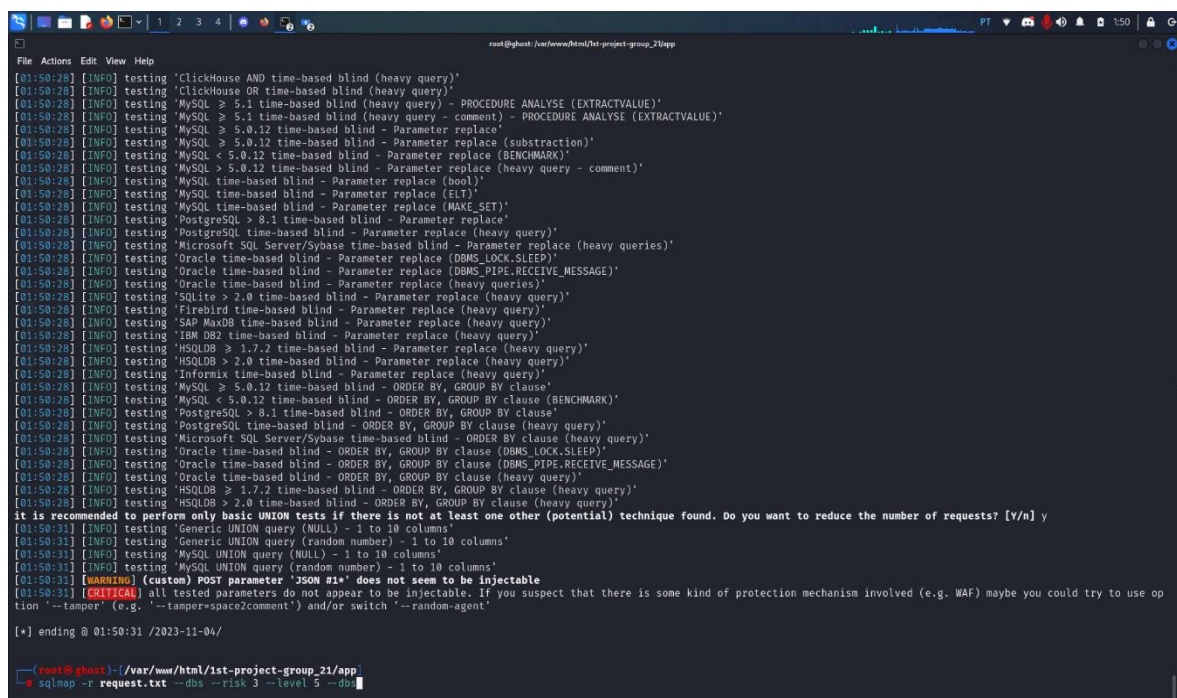
## Como corrigimos a vulnerabilidade:

O código a seguir anula a vulnerabilidade visto que faz com que independentemente do input do utilizador, o mesmo vá ser tratado sempre como uma string (**POO::PARAM\_STR**) e não como uma parte de um possível comando SQL, o que significa que os inputs não vão alterar a estrutura ou a execução do comando SQL.

```
}else{
    $username = $json['user_name'];
    $password = $json['password'];
    // new code
    $query = "SELECT * FROM users WHERE username=:username limit 1";
    $stmt = $conn->prepare($query);
    $stmt->bindParam(':username', $username, PDO::PARAM_STR);
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    if($username == $row["username"] && hash('sha256', $password) === $row["password"]){
```

Fig 14 – Código seguro(api/login.php)

## Vulnerabilidade resolvida:



```
root@ghost: /var/www/html/1st-project-group_21/app
File Actions Edit View Help
[01:50:28] [INFO] testing 'ClickHouse AND time-based blind (heavy query)'
[01:50:28] [INFO] testing 'ClickHouse OR time-based blind (heavy query)'
[01:50:28] [INFO] testing 'MySQL > 5.1 time-based blind (heavy query) - PROCEDURE ANALYSE (EXTRACTVALUE)'
[01:50:28] [INFO] testing 'MySQL > 5.1 time-based blind (heavy query - comment) - PROCEDURE ANALYSE (EXTRACTVALUE)'
[01:50:28] [INFO] testing 'MySQL > 5.0.12 time-based blind - Parameter replace (subtraction)'
[01:50:28] [INFO] testing 'MySQL > 5.0.12 time-based blind - Parameter replace (BENCHMARK)'
[01:50:28] [INFO] testing 'MySQL > 5.0.12 time-based blind - Parameter replace (heavy query - comment)'
[01:50:28] [INFO] testing 'MySQL time-based blind - Parameter replace (bool)'
[01:50:28] [INFO] testing 'MySQL time-based blind - Parameter replace (BIT)'
[01:50:28] [INFO] testing 'MySQL time-based blind - Parameter replace (MAKE_SET)'
[01:50:28] [INFO] testing 'PostgreSQL > 8.1 time-based blind - Parameter replace'
[01:50:28] [INFO] testing 'PostgreSQL time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind - Parameter replace (heavy queries)'
[01:50:28] [INFO] testing 'Oracle time-based blind - Parameter replace (DBMS_LOCK.SLEEP)'
[01:50:28] [INFO] testing 'Oracle time-based blind - Parameter replace (DBMS_PIPE.RECEIVE_MESSAGE)'
[01:50:28] [INFO] testing 'Oracle time-based blind - Parameter replace (heavy queries)'
[01:50:28] [INFO] testing 'SQLite > 2.0 time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'Firebird time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'SAP MaxDB time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'IBM DB2 time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'HSQLDB > 1.7.2 time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'HSQLDB > 2.0 time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'Informix time-based blind - Parameter replace (heavy query)'
[01:50:28] [INFO] testing 'MySQL > 5.0.12 time-based blind - ORDER BY, GROUP BY clause'
[01:50:28] [INFO] testing 'MySQL < 5.0.12 time-based blind - ORDER BY, GROUP BY clause (BENCHMARK)'
[01:50:28] [INFO] testing 'PostgreSQL > 8.1 time-based blind - ORDER BY, GROUP BY clause'
[01:50:28] [INFO] testing 'PostgreSQL time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[01:50:28] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind - ORDER BY clause (heavy query)'
[01:50:28] [INFO] testing 'Oracle time-based blind - ORDER BY, GROUP BY clause (DBMS_LOCK.SLEEP)'
[01:50:28] [INFO] testing 'Oracle time-based blind - ORDER BY, GROUP BY clause (DBMS_PIPE.RECEIVE_MESSAGE)'
[01:50:28] [INFO] testing 'Oracle time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[01:50:28] [INFO] testing 'HSQLDB > 1.7.2 time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[01:50:28] [INFO] testing 'HSQLDB > 2.0 time-based blind - ORDER BY, GROUP BY clause (heavy query)'
[01:50:31] [INFO] it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] y
[01:50:31] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[01:50:31] [INFO] testing 'MySQL UNION query (random number) - 1 to 10 columns'
[01:50:31] [INFO] testing 'MySQL UNION query (random number) - 1 to 10 columns'
[01:50:31] [WARNING] (custom) POST parameter 'JSON #1' does not seem to be injectable
[01:50:31] [CRITICAL] all tested parameters do not appear to be injectable. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use op
tion '--tamper' (e.g. '--tamper=spacecomment') and/or switch '--random-agent'

[*] ending @ 01:50:31 /2023-11-04/

root@ghost: /var/www/html/1st-project-group_21/app
sqlmap -r request.txt --dbs --risk 3 --level 5 --dbs
```

Fig 15– SQLMap não consegue achar um exploit

### Vulnerabilidade 3 (CWE-22)

A vulnerabilidade de *Path Traversal*, ocorre quando um servidor não filtra adequadamente as entradas do utilizador em que essas entradas depois são usadas para aceder aos arquivos e diretórios armazenados no servidor. Isso permite que um atacante manipule essas entradas para navegar através do sistema de arquivos do servidor, potencialmente acedendo a arquivos ou diretórios que não era suposto o mesmo ter acesso.

#### O que é que o atacante ganha?

Um atacante que explora com sucesso uma vulnerabilidade de *Path Traversal* pode ganhar a capacidade de aceder, visualizar ou executar arquivos no servidor que deveriam estar inacessíveis. Aqui estão alguns dos possíveis ganhos para um atacante:

1. **Acesso a Dados Sensíveis:** O atacante pode ler arquivos sensíveis, como arquivos de configuração, que podem conter senhas, chaves criptográficas ou informações da base de dados.
2. **Bypass de Controlos de Acesso:** Ao navegar para diretórios restritos, o atacante pode contornar os controlos de acesso que protegem arquivos críticos.
3. **Execução de Código Remoto:** Se o atacante puder carregar e executar arquivos arbitrários, isso pode levar à execução de código malicioso no servidor.
4. **Alteração de Dados:** Modificação ou exclusão de arquivos críticos que podem afetar a disponibilidade ou integridade do sistema.
5. **Comprometimento do Servidor:** Dependendo das permissões do sistema de arquivos e do ambiente do servidor, o atacante pode ganhar privilégios e controlo total do servidor.



### Código exemplo não seguro:

```
<html lang="pt-PT">
<head>
  <link rel="stylesheet" href="css/default.css">
  <meta charset="ISO-8859-1">
  <title>Produtos da Loja do DETI</title>
</head>
<body>
  <?php
    include ("config.php");
    include ("api/functions.php");
    include ("topbar.php");
  ?>
  <h1>Produtos do Departamento de Engenharia de Tecnologias e Informação (DETI)</h1>
  <?php
    $file = $_GET['file'];
    if(isset($file)){
      include("$file");
    }else{
      header('Location: index.php');
    }
  ?>
</body>
</html>
```

Fig 16 – Código inseguro(api/contactos.php)

Ao analisarmos o código podemos concluir que o arquivo contactos.php recebe como argumento `$_GET['file']` que ao ser armazenado na variável `$file` a mesma é passada numa condição if em que caso `$_GET['file']` exista, o arquivo presente no argumento será exibido na página. Caso o argumento não exista a página irá redirecionar para a página principal. O grande problema neste código é que como não existem restrições nos arquivos que podem ser acedidos acontece que o argumento `$_GET['file']` aceita qualquer arquivo de qualquer diretório e devido a esse facto uma pessoa mal-intencionada pode ler arquivos comprometedores ou até mesmo fazer um brute force para poder saber que arquivos existem num determinado diretório.

## Como testámos a vulnerabilidade:

Para testar esta vulnerabilidade utilizámos a página de contactos. Esta exibe o conteúdo do ficheiro `contactos.txt`. Ao substituir no url da página o argumento do ficheiro a ser mostrado, obtivemos o resultado da Fig.18.

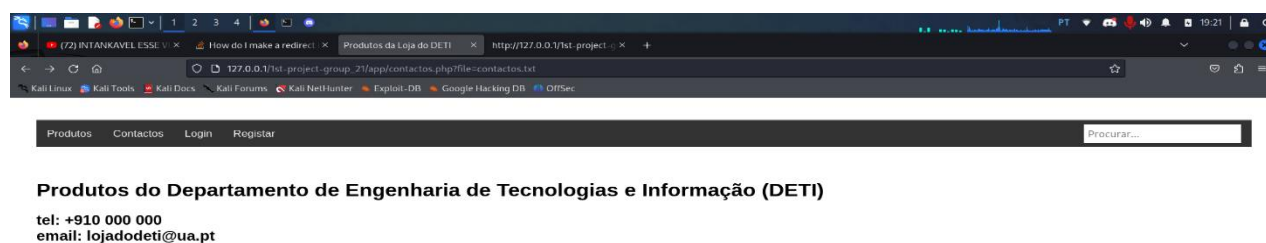


Fig 17 – Página Contactos.php

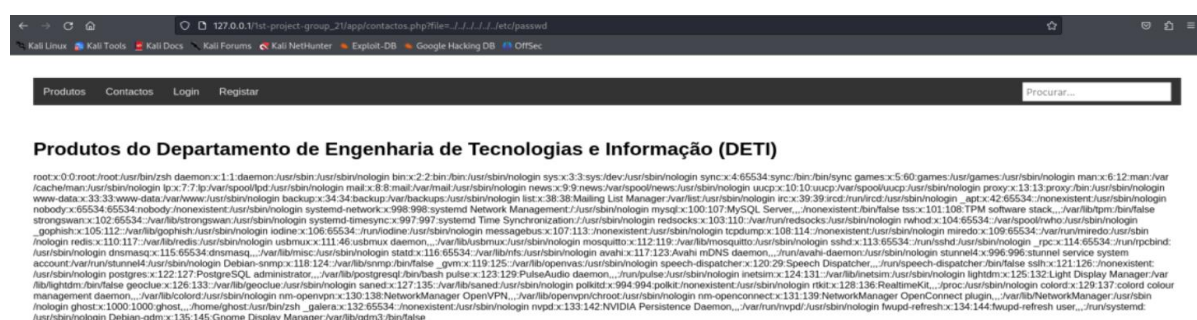


Fig 18 – Conteúdo do `/etc/passwd` a ser mostrado na página principal

Como podemos verificar, ao colocar no url `./etc/passwd` o conteúdo que é mostrado pela página passa a ser o conteúdo do ficheiro “passwd”, pelo que a informação crítica fica exposta ao utilizador.

Neste próximo exemplo, o ficheiro “config.php” foi acedido diretamente na página principal.



### Como corrigimos a vulnerabilidade:

Para corrigir a vulnerabilidade, adicionámos um array com os ficheiros acessíveis, sendo agora impossível navegar diretamente para um ficheiro que não faça parte deste array. Voltámos a repetir o processo anterior, mas agora somos redirecionados para a página inicial.

Exemplo do trecho inserido no fim do url: “**page=php://filter/convert.base64-encode/resource=**”

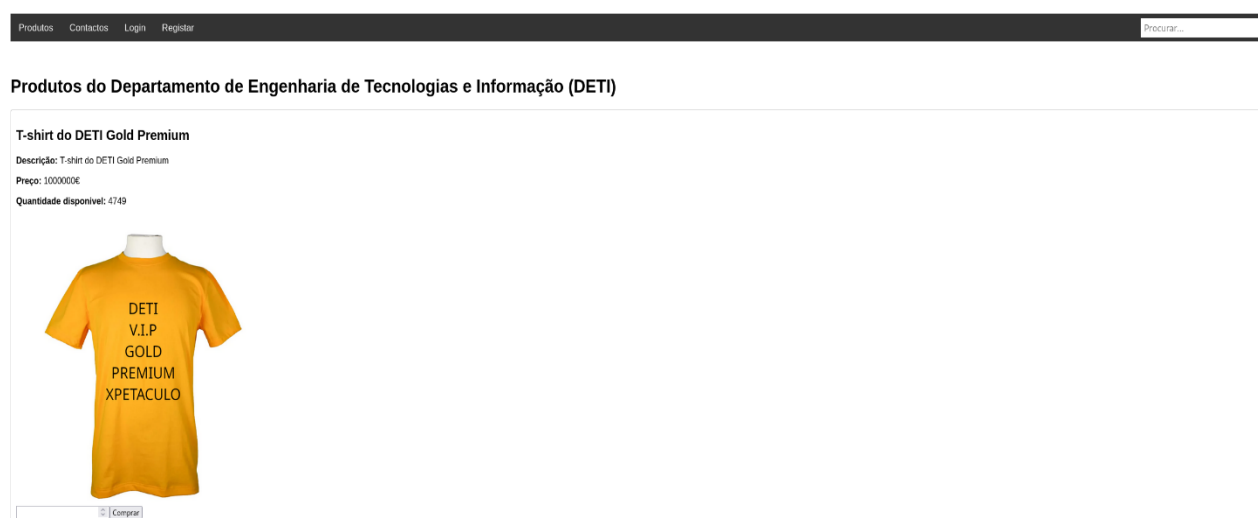


Fig 21 – Resultado após correção do código

### Código exemplo seguro:

```
<?php
$file = $_GET['file'];
$arquivos_disponíveis = array("contactos.txt");
if(isset($file) && in_array($file, $arquivos_disponíveis)){
    include("$file");
}else{
    header('Location: index.php');
}
?>
```

Fig 22 – Código seguro para resolver a vulnerabilidade(api/contactos.php)

## Vulnerabilidade 4 (CWE-352)

A vulnerabilidade de *Cross-Site Request Forgery (CSRF)* ocorre quando um aplicativo web permite que ações indesejadas sejam executadas em nome de um usuário autenticado sem que o usuário decida ou consinta explicitamente com a ação. Um atacante pode enganar um usuário autenticado para enviar uma solicitação a um aplicativo web que ele está atualmente autenticado, utilizando o estado de autenticação do usuário para realizar uma ação mal-intencionada.

### O que é que o atacante ganha?

Um atacante que explora com sucesso uma vulnerabilidade CSRF pode realizar ações no site como se fosse o usuário. Dependendo dos privilégios do usuário, isso pode levar a várias consequências, como:

1. **Transações Não Autorizadas:** Executar operações, como transferências de dinheiro em um banco online, sem o conhecimento ou consentimento do usuário.
2. **Alteração de Configurações do Usuário:** Mudar as configurações da conta do usuário, incluindo senhas, endereços de e-mail e informações de contato.
3. **Comprometimento de Dados:** Postar conteúdo indesejado ou aceder e potencialmente excluir dados privados.
4. **Ações Maliciosas:** Inscrever o usuário em serviços, aceitar termos de contrato ou realizar compromissos em nome do usuário.
5. **Aumento de Privilégios:** Se o ataque for combinado com outras vulnerabilidades, pode levar à elevação de privilégios dentro do site.

## Código exemplo não seguro:

```
<?php
include("../config.php");
include("../functions.php");
if($_SERVER['REQUEST_METHOD'] === "POST"){
    $sql="SELECT * FROM users WHERE id= ?";
    $statement= $conn->prepare($sql);
    $statement->bind_param('i',$_SESSION["id"]);
    $statement->execute();
    $result = $statement->get_result();
    $row= $result->fetch_assoc();

    if (! empty($row)) {
        $hashedPassword = $row["password"];
        $password = PASSWORD_HASH($_POST["newPassword"], PASSWORD_DEFAULT);
        if (password_verify($_POST["currentPassword"], $hashedPassword)) {
            $sql = "UPDATE users set password=? WHERE id=?";
            $statement = $conn->prepare($sql);
            $statement->bind_param('si', $password, $_SESSION["id"]);
            $statement->execute();
            $message = "Password Changed";
        } else {
            $message = "Current Password is not correct";
        }
    }
}
?>
```

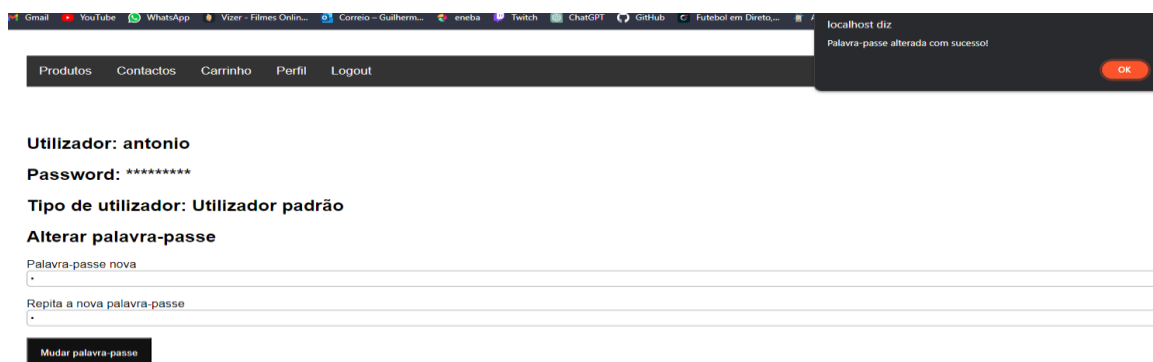
Fig 23- Código não seguro da alteração da password(api/change\_password.php)

A vulnerabilidade neste código deve-se ao facto de não verificarmos a password atual para a troca da nova password.

Depois da análise do nosso código reparámos que seria a CWE-352(CSRF), pois qualquer utilizador mal-intencionado possuindo o token da sessão no seu browser poderia alterar a palavra-passe sem saber a atual.

## Como testamos a vulnerabilidade:

Na imagem seguinte conseguimos perceber que é possível inserir a palavra-passe nova apenas estando logado, não sendo necessário saber a password atual.

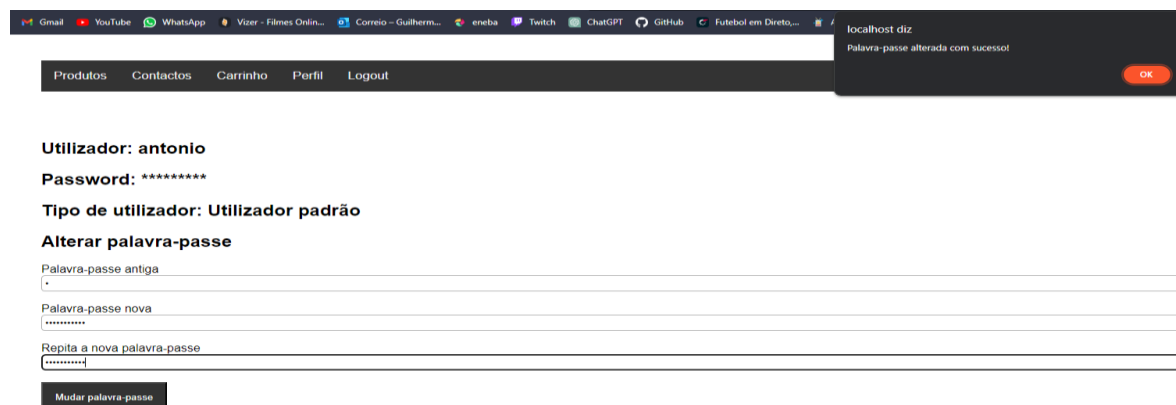


The screenshot shows a web browser window with a navigation bar at the top containing links: Produtos, Contactos, Carrinho, Perfil, and Logout. A dark notification box in the top right corner displays the message "Palavra-passe alterada com sucesso!" (Password changed successfully) with an "OK" button. Below the navigation bar, the user is logged in as "Utilizador: antonio". The "Password:" field is masked with asterisks. Under the heading "Tipo de utilizador: Utilizador padrão", there is a section titled "Alterar palavra-passe" (Change password). This section contains two input fields: "Palavra-passe nova" (New password) and "Repita a nova palavra-passe" (Repeat the new password), both with asterisk masking. A "Mudar palavra-passe" (Change password) button is located at the bottom of this section.

Fig 24- Alteração da password no código não seguro

## Como resolvemos a vulnerabilidade:

Para resolver esta vulnerabilidade pede-se que seja inserida a password que está associada ao utilizador antes deste colocar a nova password, não sendo possível alterar a password de outra forma.



The screenshot shows a web browser window with a dark theme. At the top, there's a navigation bar with links: Produtos, Contactos, Carrinho, Perfil, and Logout. Below this, the user is logged in as 'antonio'. The 'Password' field is masked with asterisks. The 'Tipo de utilizador' is 'Utilizador padrão'. The section is titled 'Alterar palavra-passe'. It contains three input fields: 'Palavra-passe antiga' (containing a single asterisk), 'Palavra-passe nova' (containing asterisks), and 'Repita a nova palavra-passe' (containing asterisks). A 'Mudar palavra-passe' button is at the bottom. A notification box in the top right corner says 'localhost diz: Palavra-passe alterada com sucesso!' with an 'OK' button.

Fig 25- Alteração da password no código seguro

## Código exemplo seguro:

```
<?php
include("../config.php");
include("../functions.php");
check_login();
if($_SERVER['REQUEST_METHOD']== "POST"){
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);
    // vai receber var password e new_password
    $id = $_SESSION["id"];
    $sql="SELECT * FROM users WHERE id= :id ";
    $statement= $conn->prepare($sql);
    $statement->bindParam(':id', $id, PDO::PARAM_INT);
    $statement->execute();
    $row = $statement->fetch(PDO::FETCH_ASSOC);

    $hashedPassword = $row["password"];
    $password = hash("sha256",$json["password"]);
    $new_password = hash("sha256",$json["new_password"]);
    $new_password1 = hash("sha256",$json["new_password1"]);

    if ($password == $hashedPassword) {
        if($new_password == $new_password1){
            $sql = "UPDATE users set password= :new_password WHERE id = :id ";
            $updateStmt = $conn->prepare($sql);
            $updateStmt->bindParam(':id', $id, PDO::PARAM_INT);
            $updateStmt->bindParam(':new_password', $new_password, PDO::PARAM_STR);
            $updateStmt->execute();
            echo json_encode(array("result" => 1, "result_text" => "Palavra-passe alterada com sucesso!"));
        } else {
            echo json_encode(array("result" => 0, "result_text" => "Palavras-passe novas não são iguais"));
        }
    } else {
        echo json_encode(array("result" => 0, "result_text" => "Palavra-passe não coincide com a atual"));
    }
}
```

Fig 26- Código seguro da alteração da password(api/change\_paasword.php)

## Vulnerabilidade 5 (CWE-434)

A vulnerabilidade de *Unrestricted Upload of File with Dangerous Type* ocorre quando uma API permite que usuários façam upload de arquivos executáveis ou de outros tipos que possam ser automaticamente processados no lado do servidor ou no lado do cliente de maneira perigosa. Se os controlos adequados não forem implementados, um atacante pode carregar um arquivo malicioso que pode ser usado para executar código, realizar ataques de script entre sites (XSS), ou causar negação de serviço, entre outros ataques.

### O que é que o atacante ganha?

Um atacante que explora uma vulnerabilidade de *Unrestricted Upload of File with Dangerous Type*:

1. **Execução de Código:** Carregar e executar scripts ou programas maliciosos no servidor, o que pode levar ao comprometimento completo do servidor.
2. **Ataques Cross-Site Scripting (XSS):** Inserir scripts maliciosos em arquivos que serão entregues a outros usuários, executando código no navegador do usuário final.
3. **Aumento de Privilégios:** Utilizar arquivos carregados para explorar outras vulnerabilidades no sistema, potencialmente elevando os privilégios de acesso do atacante.
4. **Denegation of Service (DoS):** Enviar arquivos que consomem recursos do servidor excessivamente, levando a uma negação de serviço.
5. **Phishing ou Engano de Outros Usuários:** Usar o sistema confiável para hospedar phishing ou outros tipos de páginas da web maliciosas.
6. **Espalhar Malware:** Utilizar a plataforma como um ponto de distribuição para malware, afetando outros usuários e sistemas.



### Código exemplo não seguro:

```
<?php
include("functions.php");
include("../config.php");

$path = '../uploads/';
$img = basename($_FILES['image']['name']);
$tmp = $_FILES['image']['tmp_name'];
$final_image = rand(1000, 1000000) . "_" . $img;
$path = $path . strtolower($final_image);
$image = "uploads/" . strtolower($final_image);
if(move_uploaded_file($_FILES['image']['tmp_name'], $path)){
    echo json_encode(array("result" => 1, "result_text" => $image));
}else{
    echo json_encode(array("result" => 0, "result_text" => "Imagem não foi adicionada com sucesso!"));
}
```

Fig 27: Código não seguro(api/upload\_image.php)

A vulnerabilidade associada a este código deve-se ao facto de não se verificar o tipo do ficheiro que é utilizado no upload. No caso da nossa loja, o único local onde é possível fazer upload de ficheiros é quando se adiciona uma imagem a um produto novo, sendo feito pela função “upload\_image.php”.

### Como testámos a vulnerabilidade:

Para testarmos esta vulnerabilidade, decidimos dar upload de um ficheiro que não fosse uma imagem (.png, .gif, .jpg...). Neste caso enviamos o “b374k-2.8.source.php” para averiguar se o mesmo iria ser aceite. Este ficheiro é uma Web Shell.

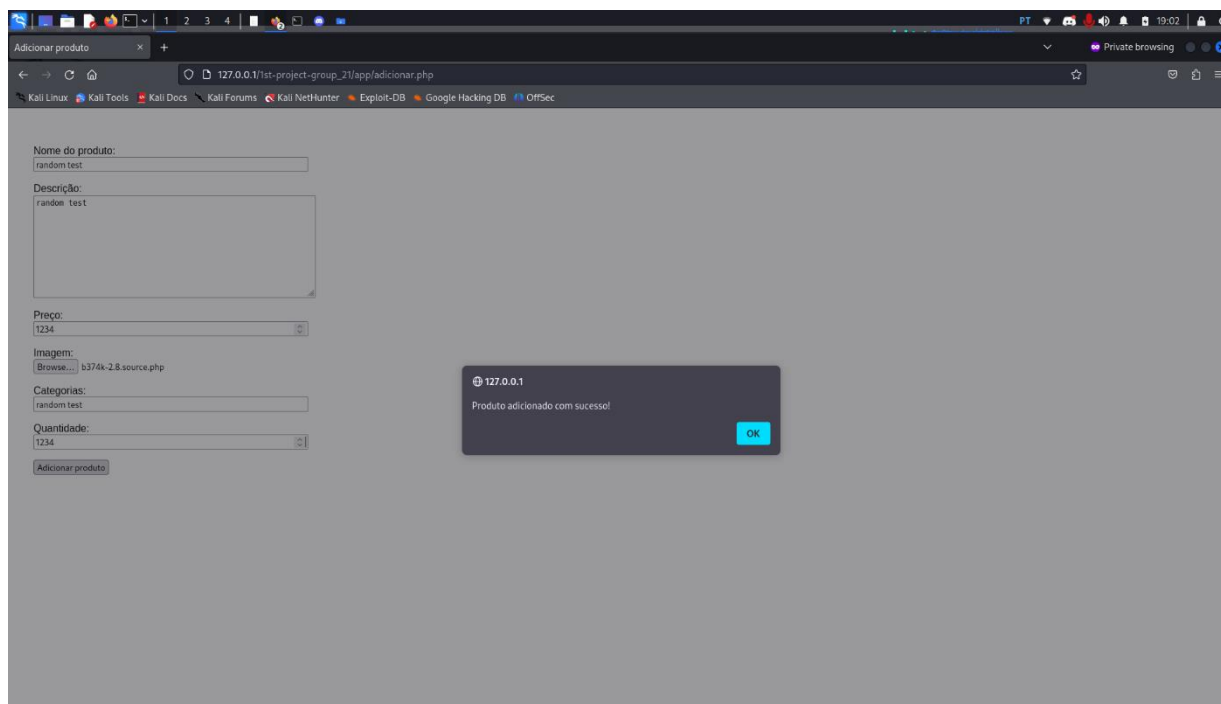


Fig 28: Upload com sucesso do ficheiro de teste

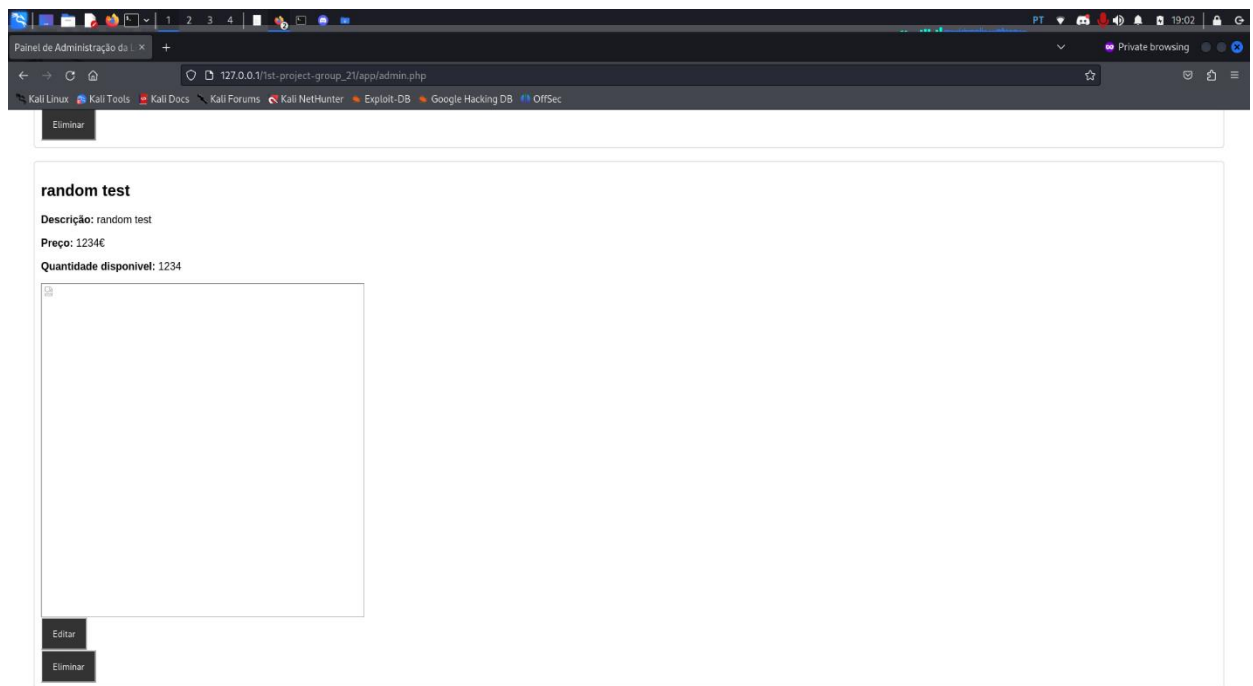


Fig 29: Visualização do ficheiro b374k-2.8.source.php na aba de escolha de imagens

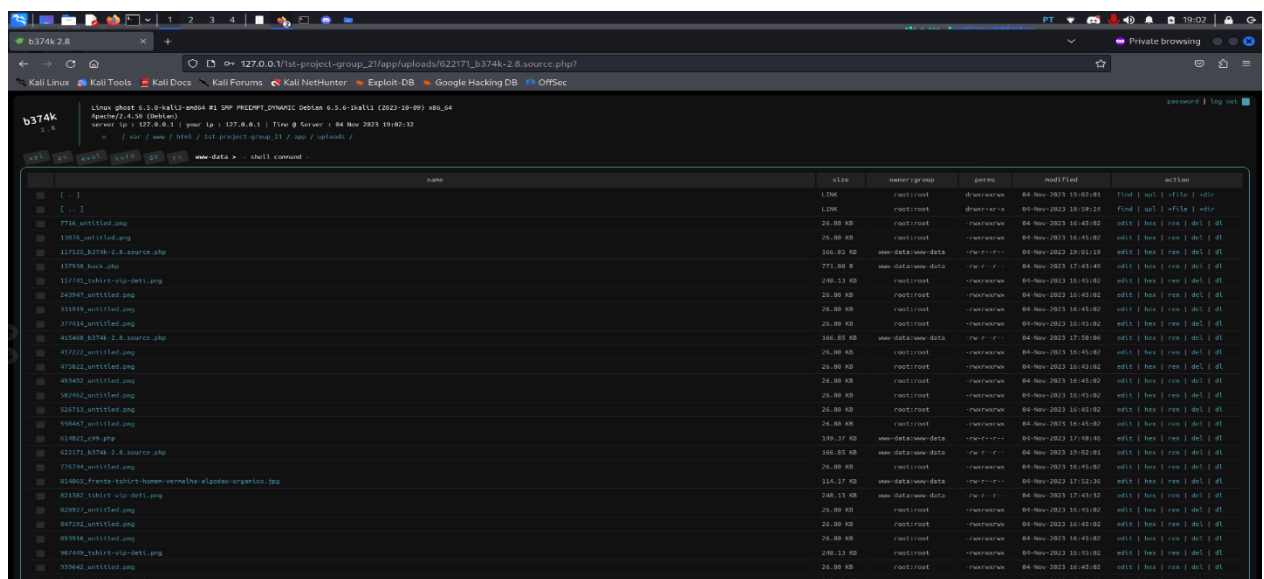


Fig 30: Web shell injection

Após o teste, chegou se à conclusão de que realmente a vulnerabilidade estava presente na respetiva função e apesar de apenas o administrador (na teoria) ter as permissões necessárias para isso, não deixa de existir a vulnerabilidade, visto que qualquer tipo de ficheiros podem ser uploaded, abrindo espaço para uma backdoor.

## Como corrigimos a vulnerabilidade:

Para corrigir a vulnerabilidade foi adicionada uma verificação de modo que não seja possível fazer upload de arquivos que não tenham extensão '.gif', '.png', '.jpg'. Caso o ficheiro que o utilizador tente dar upload tenha outro tipo de extensão, a aplicação não vai aceitar e retorna uma mensagem de insucesso no upload. No exemplo tentámos dar upload a ficheiro .js.

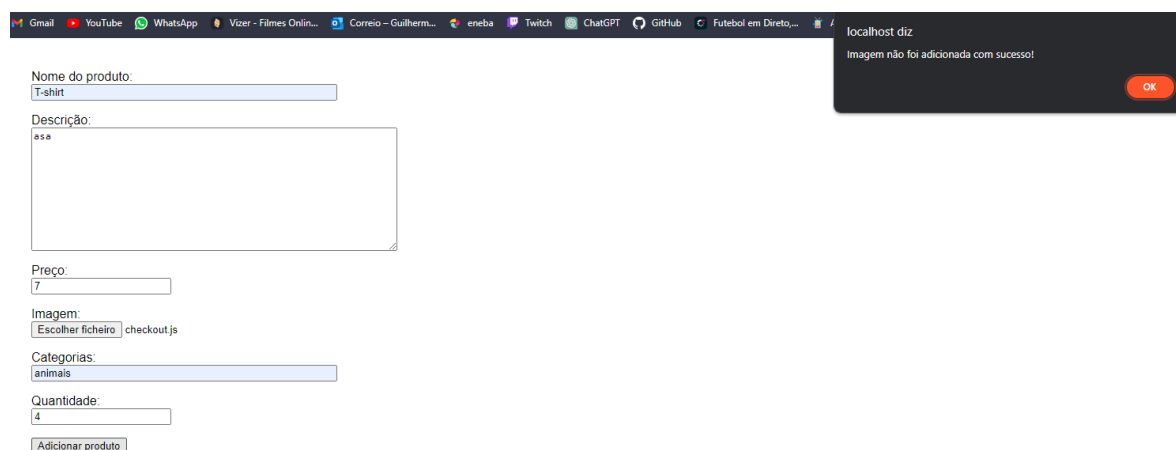


Fig 31: Upload sem sucesso

## Código exemplo seguro:

```
<?php
include("functions.php");
include("../config.php");

$path = '../uploads/';
$img = basename($_FILES['image']['name']);
$tmp = $_FILES['image']['tmp_name'];
$final_image = rand(1000, 1000000) . "_" . $img;
$path = $path . strtolower($final_image);
$image = "uploads/" . strtolower($final_image);
$allowed = array('gif', 'png', 'jpg');
$filename = $_FILES['image']['name'];
$ext = pathinfo($filename, PATHINFO_EXTENSION);
if (!in_array($ext, $allowed)) {
    echo json_encode(array("result" => 0, "result_text" => "Imagem não foi adicionada com sucesso!"));
}
else {
    if(move_uploaded_file($_FILES['image']['tmp_name'], $path)){
        echo json_encode(array("result" => 1, "result_text" => $image));
    }else{
        echo json_encode(array("result" => 0, "result_text" => "Imagem não foi adicionada com sucesso!"));
    }
}
//criar um verificador de extensão de ficheiros
?>
```

Fig 32: Código seguro(api/upload\_image.php)

## Vulnerabilidade 6 (CWE-521)

A vulnerabilidade de *Weak Password Requirements* ocorre quando um sistema ou aplicativo não impõe uma política de senhas fortes. Isso pode permitir que usuários escolham senhas simples ou comuns, que são fáceis de adivinhar ou quebrar usando métodos como brute force attacks, dicionário ou engenharia social.

### O que é que o atacante ganha?

Um atacante que explora a vulnerabilidade de *Weak Password Requirements* pode:

1. Acesso Não Autorizado: Ganhar acesso a contas de usuário, dados sensíveis ou sistemas através da “quebra” de senhas fracas.
2. Elevação de Privilégios: Uma vez dentro do sistema, um atacante pode explorar outras vulnerabilidades para obter privilégios maiores do que os de uma conta de usuário normal.
3. Roubo de Identidade: Usar as credenciais obtidas para se passar por um usuário legítimo e realizar ações mal-intencionadas.
4. Disseminação de Ataques: Utilizar uma conta comprometida como ponto de partida para ataques subsequentes dentro de uma rede ou contra sistemas associados.
5. Sabotagem ou Alteração de Dados: Modificar ou excluir dados para comprometer a integridade dos sistemas e das informações.

## Código exemplo não seguro:

```
<?php
include("../config.php");
if($_SERVER['REQUEST_METHOD'] === "POST"){
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);
    if(!($json != NULL && key_exists("password", $json) && key_exists("password1", $json) && key_exists("user_name", $json) )){
        echo json_encode(array("result" => 0, "result_text" => "JSON inválido!"));
        die();
    }else if(empty($json['user_name'])){
        echo json_encode(array("result" => 0, "result_text" => "Utilizador não definido!"));
        die();
    }else if( empty($json["password"]) || empty($json["password1"]) ){
        echo json_encode(array("result" => 0, "result_text" => "As palavras-passe não estão definidas!"));
        die();
    }else if( $json["password"] != $json["password1"]){
        echo json_encode(array("result" => 0, "result_text" => "As palavras-passe não são iguais!"));
        die();
    }else{
        $user_name = $json['user_name'];
        $password = $json['password'];
        $password1 = $json['password1'];
        $query = "SELECT * FROM users WHERE username = '$user_name'";
        $row = $conn->query($query)->fetch();
        if($row == 0){
            $hash = hash("sha256", $password);
            $query1 = "INSERT INTO users (username, password, permission_level) VALUES ('$user_name', '$hash', 2)";
            $value = $conn->query($query1)->fetch();
            echo json_encode(array("result" => 1, "result_text" => "Utilizador criado com sucesso!"));
        }else{
            echo json_encode(array("result" => 0, "result_text" => "O Utilizador já existe!"));
        }
    }
}
?>
```

Fig 33- Código não seguro aceita qualquer password(api/signup.php)

Esta vulnerabilidade foi identificada no registo de um novo utilizador e na mudança de password. Neste código o utilizador não necessita de qualquer verificação para a password sendo aceite qualquer uma.

## Como testámos a vulnerabilidade:

Para testar esta vulnerabilidade introduzimos uma password muito simples de apenas um caracter e verificámos que esta foi aceite.

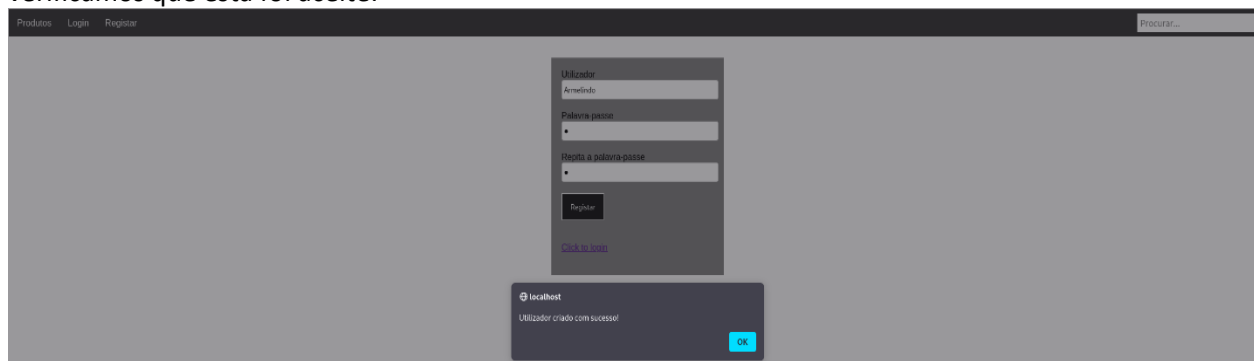


Fig 34- Teste da password fraca no código não seguro

## Como corrigimos a vulnerabilidade:

Para corrigir esta vulnerabilidade é feita uma verificação na password para que o utilizador tenha

que introduzir uma password que cumpra os seguintes requisitos mínimos:

- mais de 8 caracteres;
- pelo menos uma letra maiúscula;
- pelo menos um número e um símbolo;

Testámos primeiro com uma palavra-passe de um caracter, como anteriormente, e de seguida com uma palavra-passe que cumpre os requisitos citados acima (**Secure\_pass123**), verificando que agora só é aceite a segunda tentativa.

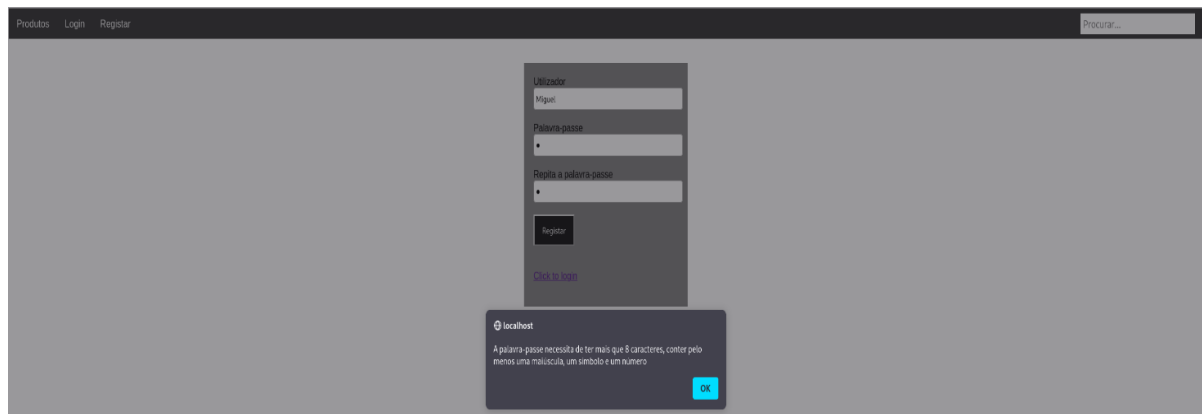


Fig 35- Teste da password fraca("1 caracter") no código seguro

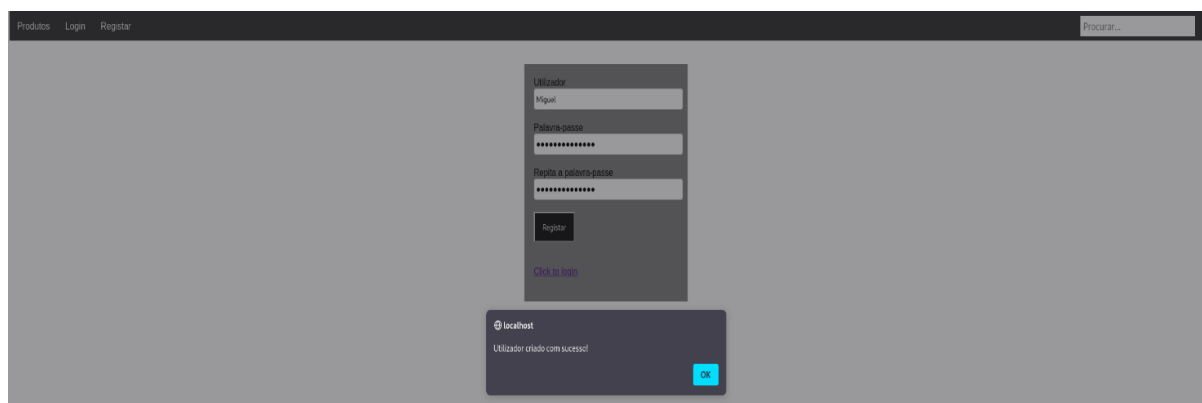


Fig 36- Teste da password forte("Secure\_pass123") no código seguro

## Código exemplo seguro:

```
<?php
include("../config.php");
if($_SERVER['REQUEST_METHOD'] === "POST"){
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);

    $uppercase = preg_match('@[A-Z]@',$json["password"]);
    $number = preg_match('@[0-9]@',$json["password"]);
    $specialChars = preg_match('@^[^w]@',$json["password"]);
    $specialChars1 = preg_match('/[\'\/~\`!@#\$%^&\*\(\)\_\-+=\{\}\[\]\|;:\<\>,\.\?\\\\]/',$json["password"]);

    if(!($json != NULL && key_exists("password", $json) && key_exists("password1", $json) && key_exists("user_name", $json)) ){
        echo json_encode(array("result" => 0, "result_text" => "JSON inválido!"));
        die();
    }else if(empty($json['user_name'])){
        echo json_encode(array("result" => 0, "result_text" => "Utilizador não definido!"));
        die();
    }else if( empty($json["password"]) || empty($json["password1"]) ){
        echo json_encode(array("result" => 0, "result_text" => "As palavras-passe não estão definidas!"));
        die();
    }else if( $json["password"] != $json["password1"]){
        echo json_encode(array("result" => 0, "result_text" => "As palavras-passe não são iguais!"));
        die();
    }else if( !(strlen($json["password"]) >= 8 && $uppercase && $number && $specialChars && $specialChars1)){
        echo json_encode(array("result" => 1, "result_text" => "A palavra-passe necessita de ter mais que 8 caracteres, conter pelo menos uma ma
    }else{

```

Fig 37- Código seguro verifica se a password é realmente forte(api/signup.php)

## Vulnerabilidade 7 (CWE-530)

A vulnerabilidade *Exposure of Backup File to an Unauthorized Control Sphere* acontece quando os arquivos de backup, que frequentemente contêm dados sensíveis, são armazenados ou transferidos de forma insegura. Isso pode levar a que indivíduos não autorizados obtenham acesso a essas cópias de segurança, comprometendo a confidencialidade e a integridade das informações.

### O que é que o atacante ganha?

Um atacante que explora essa vulnerabilidade pode ganhar acesso a várias informações valiosas e realizar ações que podem incluir:

1. **Acesso a Dados Sensíveis:** Visualização ou download de informações confidenciais presentes nos arquivos de backup, como detalhes pessoais, credenciais de login ou dados comerciais secretos.
2. **Restauração de Sistemas:** Potencial para restaurar sistemas usando backups comprometidos, levando a uma possível introdução de malware ou backdoors.
3. **Manipulação de Dados:** Alteração de informações nos backups para causar danos quando os dados forem restaurados ou para inserir dados falsos.
4. **Ransomware ou Extorsão:** Uso de backups comprometidos para ataques de ransomware, exigindo pagamento para a recuperação de dados ou ameaçando divulgar informações.
5. **Bypass de Controles de Segurança:** Acesso a versões mais antigas de sistemas ou aplicativos que podem conter vulnerabilidades conhecidas e não corrigidas, permitindo ataques adicionais.



### Exemplo inseguro:

api	04/11/2023 13:03	Pasta de ficheiros	
css	04/11/2023 13:03	Pasta de ficheiros	
js	04/11/2023 13:03	Pasta de ficheiros	
uploads	04/11/2023 13:03	Pasta de ficheiros	
adicionar.php	04/11/2023 13:03	Arquivo Fonte PHP	2 KB
admin.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
cart.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
config.php	27/10/2023 21:38	Arquivo Fonte PHP	1 KB
database.sql	04/11/2023 13:03	Arquivo Fonte SQL	4 KB
editor.php	04/11/2023 13:03	Arquivo Fonte PHP	2 KB
index.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
login.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
logout.php	04/11/2023 13:01	Arquivo Fonte PHP	1 KB
profile.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
request.txt	04/11/2023 13:03	Documento de te...	1 KB
search.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
signup.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
topbar.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB

Fig 38: Backup dos dados colocado num diretório indevido

Arquivo de backup é armazenado num diretório ou arquivo que fica acessível a utilizadores não autorizados.

### Como corrigimos a vulnerabilidade:

Neste caso tínhamos mais do que uma opção:

1. Eliminar o ficheiro de backup de dados do diretório onde se encontra o index.php
2. Colocar o ficheiro do backup de dados num diretório seguro

Optamos por eliminar o ficheiro do diretório, visto que, o mesmo já se encontra armazenado no diretório do servidor, devidamente seguro.

### Exemplo seguro:

api	04/11/2023 13:03	Pasta de ficheiros	
css	04/11/2023 13:03	Pasta de ficheiros	
js	04/11/2023 13:03	Pasta de ficheiros	
uploads	04/11/2023 13:03	Pasta de ficheiros	
adicionar.php	04/11/2023 13:03	Arquivo Fonte PHP	2 KB
admin.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
cart.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
config.php	27/10/2023 21:38	Arquivo Fonte PHP	1 KB
editor.php	04/11/2023 13:03	Arquivo Fonte PHP	2 KB
index.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
login.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
logout.php	04/11/2023 13:01	Arquivo Fonte PHP	1 KB
profile.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
request.txt	04/11/2023 13:03	Documento de te...	1 KB
search.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
signup.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB
topbar.php	04/11/2023 13:03	Arquivo Fonte PHP	1 KB

Fig 39: Backup dos dados eliminado do diretório indevido

## Bug/Vulnerabilidade

A pedido da disciplina também nos foi sugerido falar sobre um bug e como esse bug poderia causar problemas ao servidor. O bug que encontramos seria o facto das estrelas que são enviadas junto com o comentário ser um valor que tem como base ao valor associado à estrela escolhida que depois enviado junto com o JSON. Devido a isso quando esse valor é convertido para um inteiro para exibir esse comentário é executado um for loop que irá exibir o número de estrelas com base no valor presente na base de dados.

A partir daí podemos modificar o valor de forma a colocar estrelas acima do valor suposto, por exemplo 100.

Como testámos o bug:

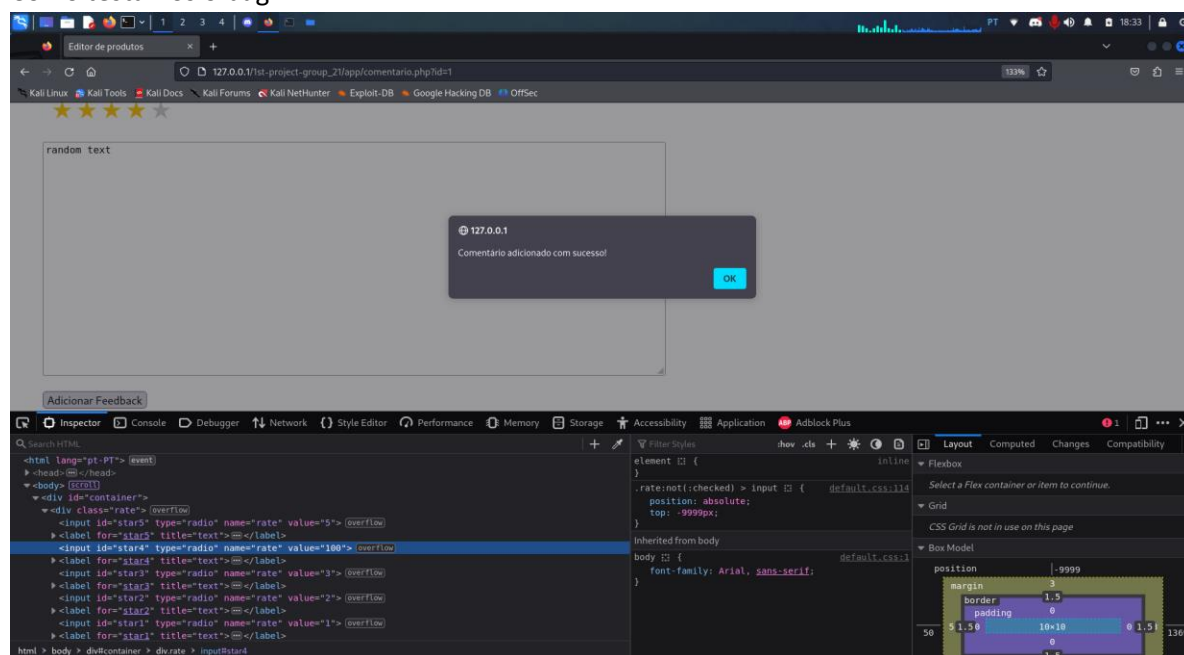


Fig 40: Adição do comentário

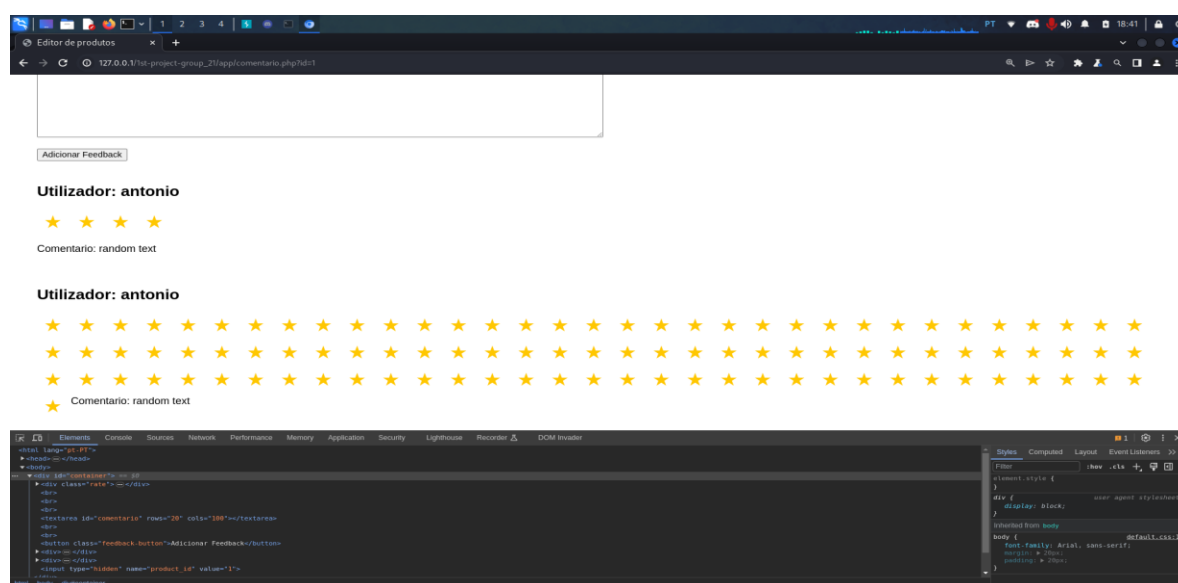


Fig 41: Adição de 100 estrelas através do comentário

## Código exemplo não seguro:

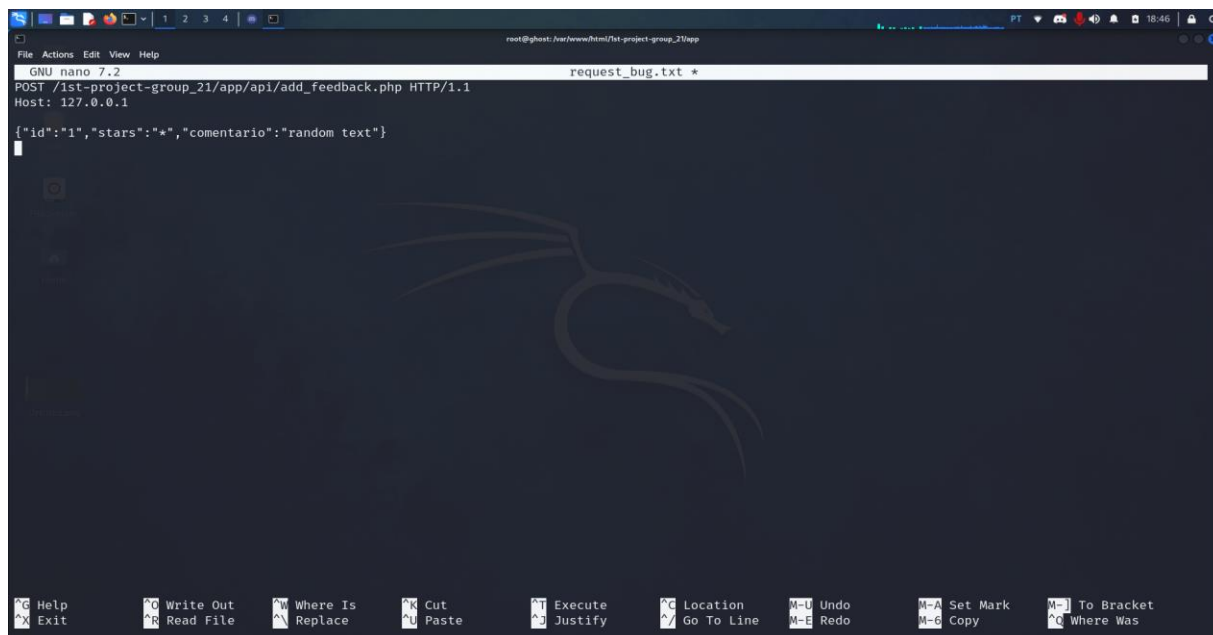
```
<?php
include("../config.php");
include("../functions.php");
if($_SERVER["REQUEST_METHOD"] === "POST") {
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);
    if(!check_login_boolean()){
        echo json_encode(array("result" => 0, "result_text" => "Sessão não iniciada!"));
        die();
    }
    else if($json != NULL && key_exists("id", $json) && key_exists("comentario", $json)) {
        echo json_encode(array("result" => 0, "result_text" => "JSON inválido!"));
        die();
    }
    else if(!key_exists("stars", $json)){
        echo json_encode(array("result" => 0, "result_text" => "Não foi adicionada nenhuma avaliação! Comentário não inserido!"));
    }
    else if(empty($json["id"]) || empty($json["stars"]) || empty($json["comentario"])){
        echo json_encode(array("result" => 0, "result_text" => "Um dos campos está vazio! Comentário não inserido!"));
        die();
    }
    else if(!is_numeric($json["id"]) && (int)$json["id"] > 0){
        echo json_encode(array("result" => 0, "result_text" => "O ID é inválido! Comentário não inserido!"));
        die();
    }
    else{
        $id = (int)$json["id"];
        $query = "SELECT * FROM products WHERE id='$id' limit 1";
        $row = $conn->query($query)->fetch();
        if(count($row) == 0){
            echo json_encode(array("result" => 0, "result_text" => "Produto não encontrado na base de dados! Comentário não inserido!"));
            die();
        }
        $stars = $json["stars"];
        $comentario = $json["comentario"];
        $username = $_SESSION["username"];
        $sql = "INSERT INTO comments (id, username, stars, comment) VALUES ('$id', '$username', '$stars', '$comentario')";
        $conn->prepare($sql)->execute();
        echo json_encode(array("result" => 1, "result_text" => "Comentário adicionado com sucesso!"));
    }
}
}
```

Fig 42: Adição de 100 estrelas através do comentário

Como podemos ver no código acima o mesmo recebe um json no formato {"id": "product\_id", "stars": "3", "comentario": "comentario inserido"} mas, focando na chave "stars" a mesma não recebe qualquer tipo de tratamento perante o que é passado nela, podendo aceitar qualquer valor inteiro e até outros valores que não era suposto receber e se formos mais a fundo podemos deduzir que os valores passados nessa chave se forem instruções SQL podemos ter acesso a base de dados.

## Como testamos a vulnerabilidade:

Para testar a vulnerabilidade nós fizemos o mesmo procedimento como fizemos com o SQL injection em que o que difere do primeiro teste foi o nosso arquivo de request que terá um POST associado a página de comentários e um argumento extra no SQLMAP para associarmos o cookie da nossa sessão aos testes que o script irá fazer.

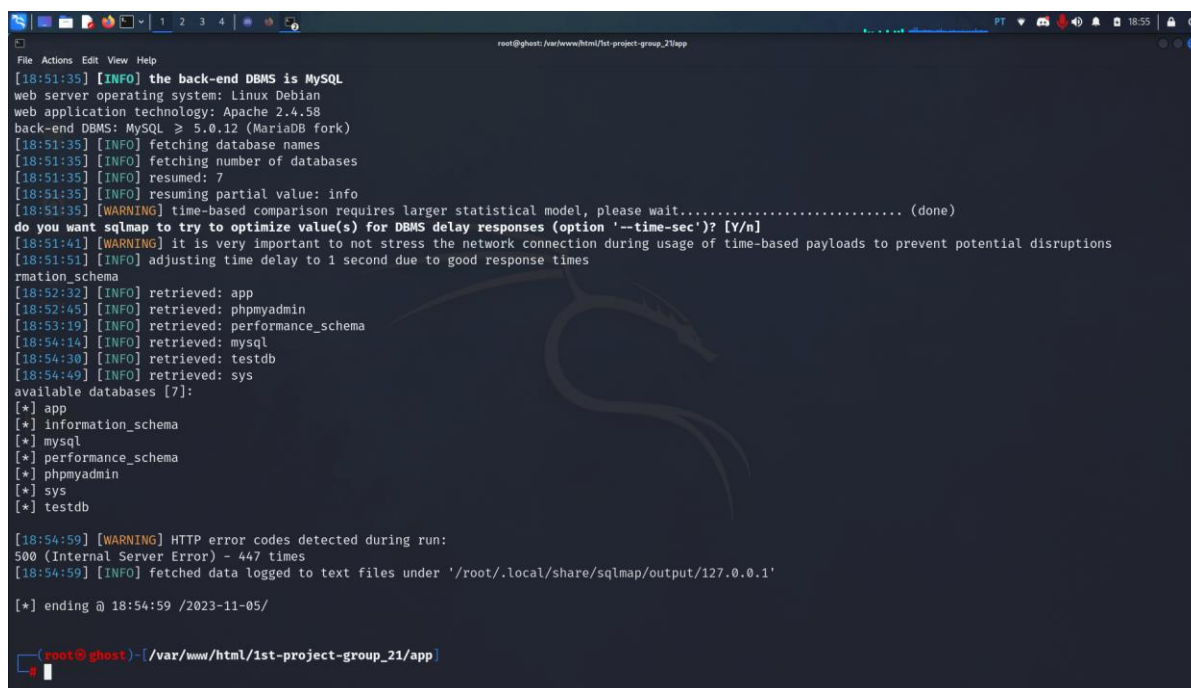


The screenshot shows a terminal window with the nano text editor open. The file being edited is named 'request\_bug.txt'. The content of the file is a JSON object: {"id": "1", "stars": "\*", "comentario": "random text"}. The terminal title bar indicates the user is root@ghost and the current directory is /var/www/html/1st-project-group\_21/app. The nano editor's status bar at the bottom shows various keyboard shortcuts like 'G Help', 'O Write Out', 'W Where Is', etc.

```
GNU nano 7.2 request_bug.txt *
POST /1st-project-group_21/app/api/add_feedback.php HTTP/1.1
Host: 127.0.0.1

{"id": "1", "stars": "*", "comentario": "random text"}
```

Fig 43: Criação de um request\_bug.txt



The screenshot shows a terminal window displaying the output of the SQLMAP tool. The tool has identified the back-end DBMS as MySQL and has fetched the database names: app, phpmyadmin, performance\_schema, mysql, testdb, and sys. It also shows a warning about HTTP error codes detected during the run. The terminal title bar indicates the user is root@ghost and the current directory is /var/www/html/1st-project-group\_21/app.

```
[18:51:35] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Debian
web application technology: Apache 2.4.58
back-end DBMS: MySQL > 5.0.12 (MariaDB fork)
[18:51:35] [INFO] fetching database names
[18:51:35] [INFO] fetching number of databases
[18:51:35] [INFO] resumed: 7
[18:51:35] [INFO] resuming partial value: info
[18:51:35] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)
do you want sqlmap to try to optimize value(s) for DBMS delay responses (option '--time-sec')? [Y/n]
[18:51:41] [WARNING] it is very important to not stress the network connection during usage of time-based payloads to prevent potential disruptions
[18:51:51] [INFO] adjusting time delay to 1 second due to good response times
information_schema
[18:52:32] [INFO] retrieved: app
[18:52:45] [INFO] retrieved: phpmyadmin
[18:53:19] [INFO] retrieved: performance_schema
[18:54:14] [INFO] retrieved: mysql
[18:54:30] [INFO] retrieved: testdb
[18:54:49] [INFO] retrieved: sys
available databases [7]:
[*] app
[*] information_schema
[*] mysql
[*] performance_schema
[*] phpmyadmin
[*] sys
[*] testdb

[18:54:59] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 447 times
[18:54:59] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/127.0.0.1'

[*] ending @ 18:54:59 /2023-11-05/

root@ghost: [/var/www/html/1st-project-group_21/app]
```

Fig 44: Bases de dados encontradas pelo SQLMAP

Como resolvemos o bug/vulnerabilidade:

Para resolvermos o problema a solução adotada foi simplesmente verificar se o valor das estrelas era ou não um valor numérico para depois converter o mesmo para inteiro e verificar se ele se encontra entre os valores [0, 5] e assim conseguimos resolver o problema de forma eficaz e direta.

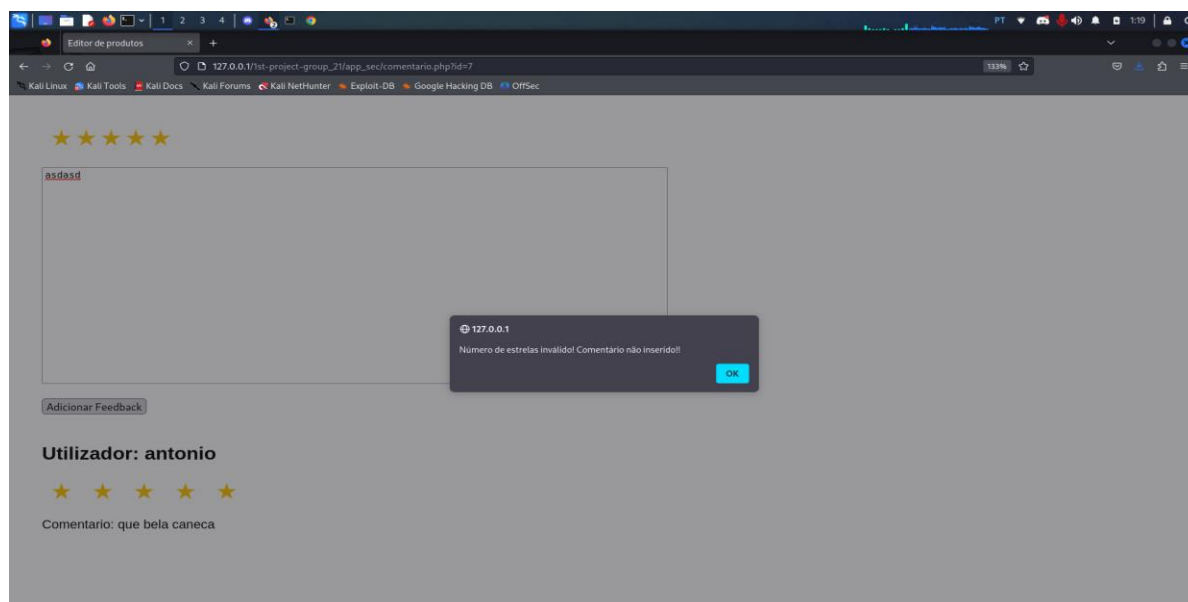


Fig 45: Resultado do que acontece quando enviamos um valor fora do limite estabelecido

**Código exemplo seguro:**

```
<?php
include("../config.php");
include("functions.php");
if($_SERVER["REQUEST_METHOD"] === "POST") {
    $data = file_get_contents('php://input');
    $json = json_decode($data, true);
    if(!check_login_boolean()){
        echo json_encode(array("result" => 0, "result_text" => "Sessão não iniciada!"));
        die();
    }
    else if(!($json != NULL && key_exists("id", $json) && key_exists("comentario", $json))){
        echo json_encode(array("result" => 0, "result_text" => "JSON inválido!"));
        die();
    }
    else if(!key_exists("stars", $json)){
        echo json_encode(array("result" => 0, "result_text" => "Não foi adicionada nenhuma avaliação! Comentário não inserido!"));
    }
    else if(empty($json["id"]) || empty($json["stars"]) || empty($json["comentario"])){
        echo json_encode(array("result" => 0, "result_text" => "Um dos campos está vazio! Comentário não inserido!"));
        die();
    }
    else if(!is_numeric($json["id"]) && (int)$json["id"] > 0){
        echo json_encode(array("result" => 0, "result_text" => "O ID é inválido! Comentário não inserido!"));
        die();
    }
    else if(!is_numeric($json["stars"]) && (int)$json["stars"] >= 0 && (int)$json["stars"] <= 5){
        echo json_encode(array("result" => 0, "result_text" => "Número de estrelas inválido! Comentário não inserido!"));
        die();
    }
    else{
        $id = (int)$json["id"];
        $query = "SELECT * FROM products WHERE id='$id' limit 1";
        $row = $conn->query($query)->fetch();
        if(count($row) == 0){
            echo json_encode(array("result" => 0, "result_text" => "Produto não encontrado na base de dados! Comentário não inserido!"));
            die();
        }
        $stars = $json["stars"];
        $comentario = htmlspecialchars($json["comentario"]);
        $username = $_SESSION["username"];
        $sql = "INSERT INTO comments (id, username, stars, comment) VALUES (:id, :username, :stars, :comentario)";
        $stmt = $conn->prepare($sql);
        $stmt->bindParam(":id", $id, PDO::PARAM_INT);
        $stmt->bindParam(":username", $username, PDO::PARAM_STR);
        $stmt->bindParam(":stars", $stars, PDO::PARAM_STR);
        $stmt->bindParam(":comentario", $comentario, PDO::PARAM_STR);
        $stmt->execute();
        echo json_encode(array("result" => 1, "result_text" => "Comentário adicionado com sucesso!"));
    }
}
```

Fig 46: Código seguro com a implementação de uma condição if que soluciona o problema