

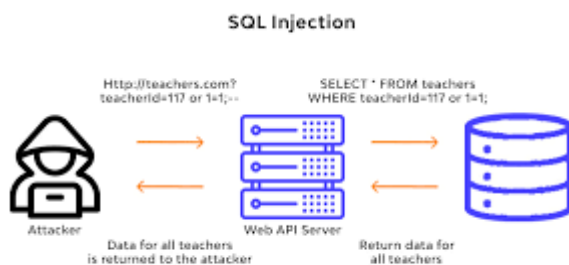
SQL INJECTION

I) DEFINITION

La SQL injection est une technique d'attaque utilisée par les hackers pour exploiter les vulnérabilités des applications Web qui utilisent des requêtes SQL. Elle consiste à injecter du code SQL malveillant dans une requête SQL afin de contourner les mécanismes de sécurité et d'obtenir des informations sensibles ou d'altérer les données stockées dans la base de données

L'attaque se produit lorsque une application ne valide pas ou ne nettoie pas correctement l'entrée utilisateur avant de l'inclure dans une requête SQL. Par exemple, si un formulaire de connexion comprend un champ de nom d'utilisateur et de mot de passe, un attaquant peut injecter du code SQL malveillant dans le champ de nom d'utilisateur, qui peut ensuite être exécuté par l'application.

Pour prévenir l'injection SQL, les développeurs devraient utiliser des requêtes paramétrées ou des instructions préparées, qui séparent le code SQL de l'entrée utilisateur et valident et nettoient l'entrée avant de l'utiliser dans une requête. Il est également important de limiter l'accès à la base de données et de minimiser les privilèges accordés aux utilisateurs de l'application. Des audits de sécurité réguliers et des tests de vulnérabilité peuvent aider à identifier et à remédier aux vulnérabilités d'injection SQL dans une application



LES TYPES SQL INJECTION

Il existe différents types de SQL injection :

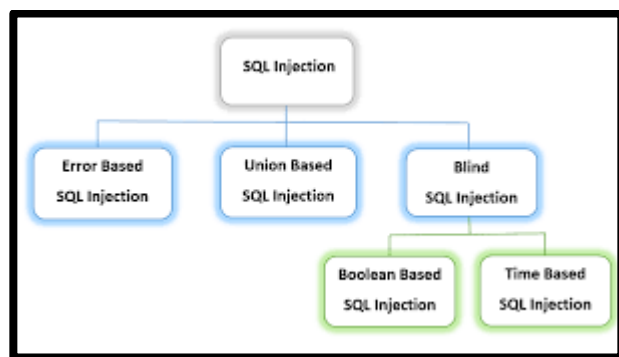
Injection basée sur les valeurs :

Il s'agit d'injecter du code SQL malveillant dans les valeurs saisies par l'utilisateur dans les champs de formulaire. Par exemple, si l'application Web demande un nom d'utilisateur et un mot de passe, un hacker peut saisir une valeur telle que " ' OR 1=1;- - " , ce qui permettrait d'obtenir toutes les informations de la table utilisateur.

Injection basée sur les paramètres :

Ce type d'injection consiste à injecter du code SQL malveillant dans les paramètres d'une requête SQL. Par exemple, si une requête SQL est construite de la manière suivante : "SELECT * FROM utilisateurs WHERE id = ?" , un hacker peut modifier la valeur du paramètre pour injecter du code SQL malveillant

Injection basée sur les erreurs : Il s'agit d'exploiter les messages d'erreur renvoyés par la base de données pour obtenir des informations sur la structure de la base de données. Par exemple, si une erreur est renvoyée pour une requête SQL mal formée, un hacker peut utiliser cette information pour élaborer une attaque SQL injection plus sophistiquée.



PRACTICE

Request to https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net:443 [34.246.129.62]

Forward Drop Intercept is on Action Open Browser Comment this item

Pretty Raw Hex

```
1 POST /login HTTP/1.1
2 Host: 0a01004e03657cb1809d94ec006a00b2.web-security-academy.net
3 Cookie: session=Fny0pbWDX4BWD5MU7BuyoZwAU4Qgrkse
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0)
  Gecko/20100101 Firefox/102.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,i
  mage/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 64
10 Origin:
  https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net
11 Referer:
  https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net/
  login
12 Upgrade-Insecure-Requests: 1
13 Sec-Fetch-Dest: document
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?1
17 Te: trailers
18 Connection: close
19
20 csrf=hKdql8y3ZR3iclfX0IkSm3ld7HBv21S8&username=ghh&password=hhhh
```

Inspector

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 3

Request Cookies 1

Request Headers 17

Web Security Academy SQL injection vuln

Back to lab description >>

Login

Username

ghh

Password

....

Log in

Volume 125%

Request to https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net:443 [34.246.129.62]

Forward Drop Intercept is on Action Open Browser Comment this item

Pretty Raw Hex

```
1 POST /login HTTP/1.1
2 Host: 0a01004e03657cb1809d94ec006a00b2.web-security-academy.net
3 Cookie: session=Fny0pbWDX4BWD5MU7BuyoZwAU4Qgrkse
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0)
  Gecko/20100101 Firefox/102.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/
  image/webp,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 64
10 Origin:
  https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net
11 Referer:
  https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net/
  login
12 Upgrade-Insecure-Requests: 1
13 Sec-Fetch-Dest: document
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?1
17 Te: trailers
18 Connection: close
19
20 csrf=hKdql8y3ZR3iclfX0IkSm3ld7HBv21S8&username=ghh&password=hhhh
```

Web Security Academy SQL injection vuln

Back to lab description >>

Login

Username

ghh

Password

....

Log in

The screenshot displays a web security lab environment. On the left, the Burp Suite 'HTTP History' tab shows a POST request to `/login` with the following details:

- Host: `0a01004e03657cb1809d94ec006a00b2.web-security-academy.net`
- Cookie: `session=xQ4zoDond0SEcsB4X7p0NjpmN3tpNQx2`
- User-Agent: `Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0`
- Accept: `text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8`
- Accept-Language: `en-US,en;q=0.5`
- Accept-Encoding: `gzip, deflate`
- Content-Type: `application/x-www-form-urlencoded`
- Content-Length: `99`
- Origin: `https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net`
- Referer: `https://0a01004e03657cb1809d94ec006a00b2.web-security-academy.net/login`
- Upgrade-Insecure-Requests: `1`
- Sec-Fetch-Dest: `document`
- Sec-Fetch-Mode: `navigate`
- Sec-Fetch-Site: `same-origin`
- Sec-Fetch-User: `?1`
- Te: `trailers`
- CSRF: `jbvVNF521eRhu00cSxrk724Ai94FoE&username=administrator%27--&password=qqqqqqqqqqqqqqqqqqqqqqqqqqqqqq`

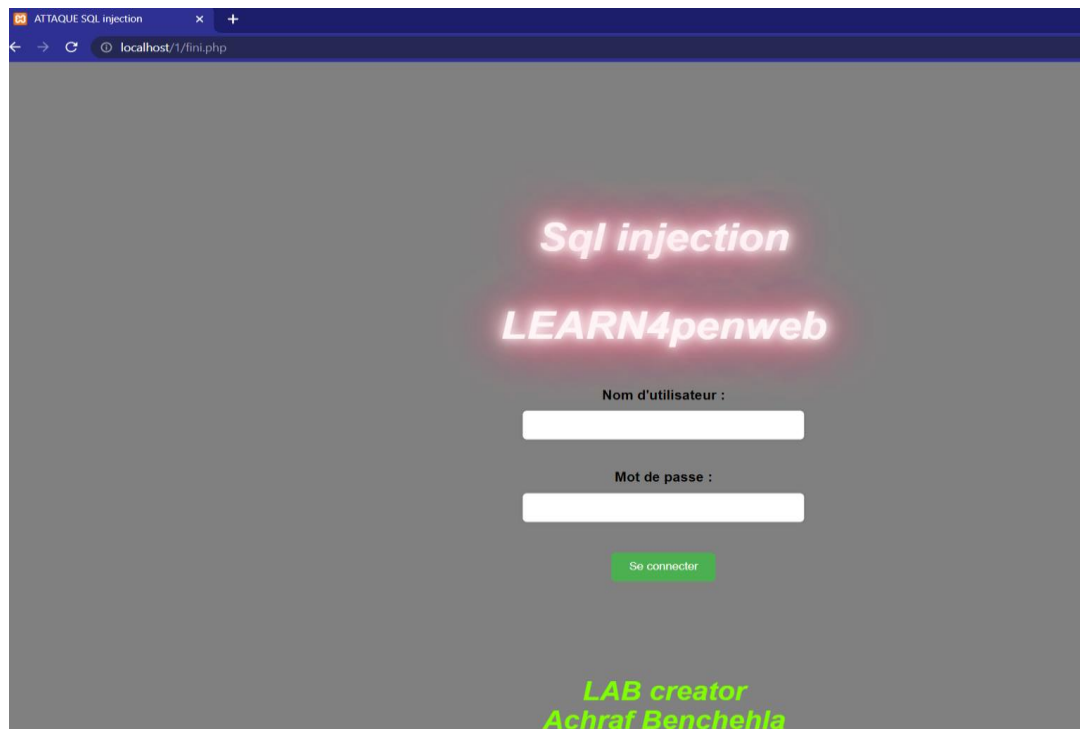
In the center, a panel lists request attributes with counts: Request Attributes (2), Request Query Parameters (0), Request Body Parameters (3), Request Cookies (1), and Request Headers (19).

On the right, the 'Academy' website is shown. It features a 'Login' form with fields for 'Username' (containing `administrator' --`) and 'Password' (masked with dots). A 'Log in' button is at the bottom. An orange banner above the form reads 'Congratulations, you solved the lab!'. A 'Back to lab description' link is in the top right corner.

avec l'aide de la description sachant que le nom du utilisateur est administrator ou ajout ' –

Explication :

En utilisant Burp Suite, nous avons pu visualiser le contenu de la requête pour comprendre comment les données sont transférées du client vers le serveur. Cette méthode permet également de détecter les caractères spéciaux qui peuvent être bloqués par l'utilisateur. Par exemple, l'instruction "--" permet de commenter le reste de la requête et peut être utilisée pour contourner certaines restrictions imposées par l'utilisateur.



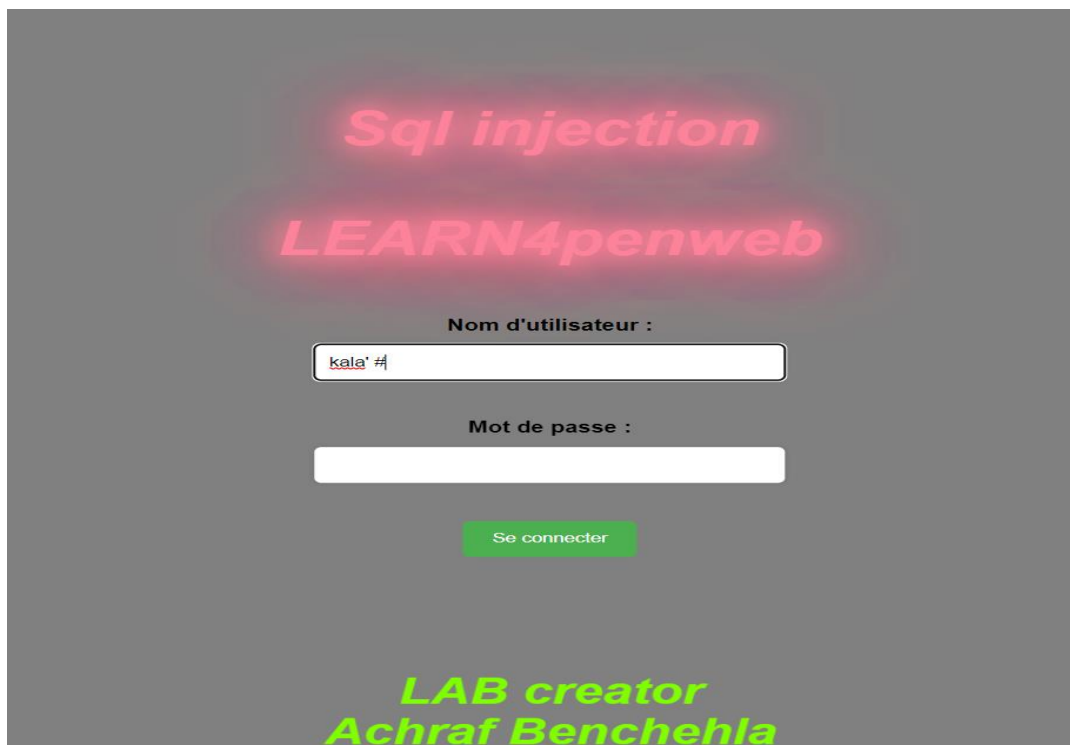
On va se connecter avec un nom d'utilisateur et un mot de passe qui n'existent pas dans la base de données, et on obtient le message d'erreur suivant.



Si on se connecte avec le vrai nom d'utilisateur et le mot de passe valide, on passe à la page suivante.

Login Successful

Maintenant on va entrer « kala' # » dans la case du username et on laisse la case du mot de passe vide



The image shows a login interface with a grey background. At the top, the text "Sql injection" is written in a red, glowing, italicized font, followed by "LEARN4penweb" in the same style. Below this, the label "Nom d'utilisateur :" is positioned above a white input field containing the text "kala' #". Underneath the username field, the label "Mot de passe :" is positioned above an empty white input field. A green button with the text "Se connecter" is located below the password field. At the bottom of the image, the text "LAB creator" and "Achraf Benchehla" is displayed in a green, italicized font.

Sql injection
LEARN4penweb

Nom d'utilisateur :

Mot de passe :

Se connecter

LAB creator
Achraf Benchehla

Login Successful

On a réussi à contourner la page de connexion en entrant « kala ' # » dans le champ du nom d'utilisateur. Ce caractère indique que tout ce qui suit le symbole # est un commentaire, ce qui permet d'avoir une condition toujours vraie

Il est important de prendre des mesures pour prévenir les attaques d'injection SQL (SQLi), une technique d'attaque qui peut être utilisée pour contourner les mesures de sécurité d'une application Web et accéder à des informations confidentielles. Les développeurs doivent utiliser des techniques de validation et de nettoyage de l'entrée utilisateur, telles que les requêtes paramétrées ou les instructions préparées, pour empêcher l'exécution de code SQL malveillant. Il est également recommandé de limiter l'accès à la base de données et de minimiser les privilèges accordés aux utilisateurs de l'application.

II)VULNARABILITE & Code

Le code suivant décrit un script en PHP pour faire une connexion avec la base de donne en se connectant avec les information de login

```
<?php
// Connexion à la base de données
$host = "localhost";
$user = "root";
$password = "";
$dbname = "achraf";
$conn = mysqli_connect($host, $user, $password, $dbname);
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// cette partie sert a collecter les formations entre par les utilisateurs //
if($_POST){
    $unam = $_POST['username'];
    $pass = $_POST['password'];

    // Verification des informations d'identification dans la base de données
    $query = "SELECT * FROM users WHERE username='$unam' AND password='$pass'";
```

Précédemment, dans le lab du Lear4PenWeb, ce qui s'est réellement passé est :

```
// ce qui passe si utilisateur entre kala ' #
$qyyuery = "SELECT * FROM users WHERE username='ach' or 1=1# ' AND password='$pass'";
$qyyuery = "SELECT * FROM users WHERE username='kala' # ' AND password='$pass'";
/*
```

Comme vous remarquez le reste de la commande SQL se transforme en un commentaire donc les autres conditions devient inutile donc même si on laisse la case du mot de passe vide la condition reste toujours vrai

III) SOLUTION

```
// Connexion à la base de données
$host = "localhost";
$user = "root";
$password = "";
$dbname = "achraf";
$conn = mysqli_connect($host, $user, $password, $dbname);
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// cette partie sert à collecter les informations envoyées par les utilisateurs //
if($_POST){
    $uname = mysqli_real_escape_string($conn, $_POST['username']);
    $pass = mysqli_real_escape_string($conn, $_POST['password']);

    // Vérification des informations d'identification dans la base de données
    $query = "SELECT * FROM users WHERE username='$uname' AND password='$pass'";
}
```

l'utilisation de `mysqli_real_escape_string()` pour échapper les caractères spéciaux dans les entrées de l'utilisateur et empêcher les injections SQL.

Si l'utilisateur entre des caractères spéciaux tels que `'`, `--`, `<`, `>`, `#`, la fonction `mysqli_real_escape_string` les échappera automatiquement pour qu'ils puissent être utilisés en toute sécurité dans la requête SQL. Par exemple, si l'utilisateur entre `'; DROP TABLE users;--` comme nom d'utilisateur, la fonction `mysqli_real_escape_string` l'échappera en `\'; DROP TABLE users\;\\-` pour éviter l'injection de code SQL malveillant.

il existe d'autres précautions que vous pouvez prendre pour éviter les attaques par injection SQL

```
$stmt = mysqli_prepare($conn, "SELECT * FROM users WHERE username=? AND password=?");
```

```
mysqli_stmt_bind_param($stmt, "ss", $username, $password);  
mysqli_stmt_execute($stmt);
```

```
// astuce //
```

La requête préparée est construite en utilisant des marqueurs de paramètres (?) à la place des valeurs des paramètres dans la requête SQL. Dans cet exemple, la requête SQL est

```
"SELECT * FROM users WHERE username=? AND password=?".
```

Les marqueurs de paramètres permettent de séparer la requête SQL de ses valeurs de paramètres. Cela permet à MySQL de traiter la requête et les valeurs des paramètres séparément, ce qui rend la requête plus sûre contre les attaques par injection SQL.

Ensuite, la fonction `mysqli_stmt_bind_param()` permet de lier les valeurs des paramètres aux marqueurs de paramètres dans la requête préparée. Dans cet exemple,

les valeurs des paramètres `$username` et `$password` sont liées aux deux marqueurs de paramètres dans la requête SQL à l'aide de la chaîne de formatage "ss".

Enfin, la requête préparée est exécutée en utilisant la fonction `mysqli_stmt_execute()`

. Cette fonction exécute la requête préparée avec les valeurs de paramètres liées aux marqueurs de paramètres.

Mettez à jour régulièrement votre application et votre infrastructure pour corriger les vulnérabilités connues et les failles de sécurité. Les mises à jour logicielles peuvent également inclure des correctifs pour les vulnérabilités de sécurité nouvellement découvertes.

Utilisez des bibliothèques ou des Framework de développement web sécurisés qui sont conçus pour prévenir les attaques par injection SQL. Ces outils peuvent être utiles car ils ont été testés et validés par la communauté de sécurité et ont été conçus pour prévenir les vulnérabilités les plus courantes.

XSS CROSS-SITE SCRIPTING

1) DEFINITION

Les attaques de type Cross-Site Scripting (XSS) sont un type d'injection, dans lequel des scripts malveillants sont injectés dans des sites Web autrement bénins et fiables. Les attaques XSS se produisent lorsqu'un attaquant utilise une application Web pour envoyer un code malveillant, généralement sous la forme d'un script côté navigateur, à un autre utilisateur final. Les failles qui permettent à ces attaques de réussir sont assez répandues et se produisent partout où une application Web utilise l'entrée d'un utilisateur dans la sortie qu'elle génère sans la valider ou l'encoder.

Un attaquant peut utiliser XSS pour envoyer un script malveillant à un utilisateur sans méfiance. Le navigateur de l'utilisateur final n'a aucun moyen de savoir que le script ne doit pas être approuvé et exécutera le script. Parce qu'il pense que le script provient d'une source fiable, le script malveillant peut accéder à tous les cookies, jetons de session ou autres informations sensibles conservés par le navigateur et utilisés avec ce site. Ces scripts peuvent même réécrire le contenu de la page HTML

La plupart des attaques XSS exploitent en effet des vulnérabilités dans du code JavaScript. Il est important de noter que toutes les attaques XSS ne se produisent pas nécessairement en utilisant JavaScript. Les attaques XSS peuvent également exploiter des vulnérabilités dans d'autres langages de programmation Web, tels que PHP ou Python.



Type des XSS

Reflected :

Les attaques XSS réfléchies sont celles où le script malveillant est inclus dans un lien ou une requête HTTP envoyée à un site Web, qui reflète le script malveillant dans la réponse. Un exemple de cela serait une requête de recherche malveillante qui inclut du code JavaScript injecté dans l'URL de la recherche, qui est ensuite renvoyé dans les résultats de la recherche

STORED XSS:

LES ATTAQUES XSS STOCKÉES SONT CELLES OÙ LE SCRIPT MALVEILLANT EST STOCKÉ SUR LE SERVEUR WEB, GÉNÉRALEMENT DANS UNE BASE DE DONNÉES, ET EXÉCUTÉ CHAQUE FOIS QU'UN UTILISATEUR CONSULTE LA PAGE AFFECTÉE. UN EXEMPLE DE CELA SERAIT UN COMMENTAIRE MALVEILLANT QUI CONTIENT DU CODE JAVASCRIPT INJECTÉ DANS UNE SECTION DE COMMENTAIRES D'UN SITE WEB

DOM XSS :

C'est une attaque de sécurité Web qui exploite les vulnérabilités des sites Web pour injecter des scripts malveillants dans les pages Web visualisées par les utilisateurs. Cette attaque permet aux attaquants d'injecter des scripts

malveillants dans les pages Web, qui peuvent ensuite être exécutés dans le navigateur Web de l'utilisateur, généralement sans que l'utilisateur en ait connaissance. Les attaquants peuvent utiliser le DOM XSS pour voler des informations sensibles telles que des cookies ou des données d'identification, et pour exécuter des actions malveillantes telles que la redirection de l'utilisateur vers des sites Web frauduleux. Les sites Web peuvent se protéger contre le DOM XSS en utilisant des méthodes de validation et de nettoyage des entrées utilisateur, ainsi qu'en limitant l'utilisation de JavaScript dans les pages Web

Pratique



y | Reflected XSS into HTML context with nothing encoded LAB Solved

[Back to lab description >>](#)

ou solved the lab! [Share your skills!](#) [Continue learning >>](#)

[Home](#)

0 search results for "

[Search](#)

[< Back to Blog](#)

Dans ce cas particulier, le code JavaScript affiche une boîte de dialogue "achraf" en utilisant la fonction alert(). Ainsi, lorsqu'un utilisateur soumet cette recherche contenant le code, le navigateur affichera immédiatement la boîte de dialogue avec le message "achraf". Explication Donc, c'est juste un test, mais un attaquant

peut utiliser les différentes fonctions disponibles dans JavaScript pour injecter du code malveillant

Dans le cas d'une injection de script, l'attaquant peut par exemple insérer du code JavaScript dans une page web, qui sera ensuite exécuté par le navigateur web de l'utilisateur sans que ce dernier ne s'en aperçoive. Le code JavaScript peut alors effectuer des actions malveillantes, comme récupérer des informations confidentielles de l'utilisateur, modifier le contenu de la page web, ou encore rediriger l'utilisateur vers un site malveillant.

Si l'attaquant injecte un code de type "`<script>alert('XSS!');</script>`" dans une page web, cela va provoquer l'affichage d'une boîte de dialogue contenant le message "XSS!" lorsque la page sera chargée par un utilisateur. Cette technique est couramment utilisée par les attaquants pour tester si un site web est vulnérable aux attaques XSS..

Web Security Academy

DOM XSS in document.write sink using source location.search

Back to lab description >>

Home

0 search results for 'achraf'

Search the blog... Search

< Back to Blog

Source Map URL: labsBlog.css.map

Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.w...
Source Map URL: academyLabHeader.css.map [Learn More](#)

Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.w...
Source Map URL: labsBlog.css.map [Learn More](#)

```
>> window.location.search  
← "?search=achraf"  
>> |
```

Ici, on va entrer des chaînes de caractères et des lettres, et on va inspecter le code pour comprendre comment nos entrées sont stockées. On va tester avec la valeur "achraf". Puis, dans la capture d'écran suivante, on va entrer des caractères spéciaux et on remarque qu'ils sont soit remplacés par d'autres caractères ou bien supprimés.

L'obfuscation peut être utilisée comme mesure de sécurité pour rendre le code source plus difficile à comprendre pour les attaquants potentiels. Cependant, cela ne garantit pas une sécurité absolue car les techniques d'obfuscation peuvent être contournées par des attaquants déterminés.

0 search results for '<<<<>>>///'

[Search](#)[< Back to Blog](#)

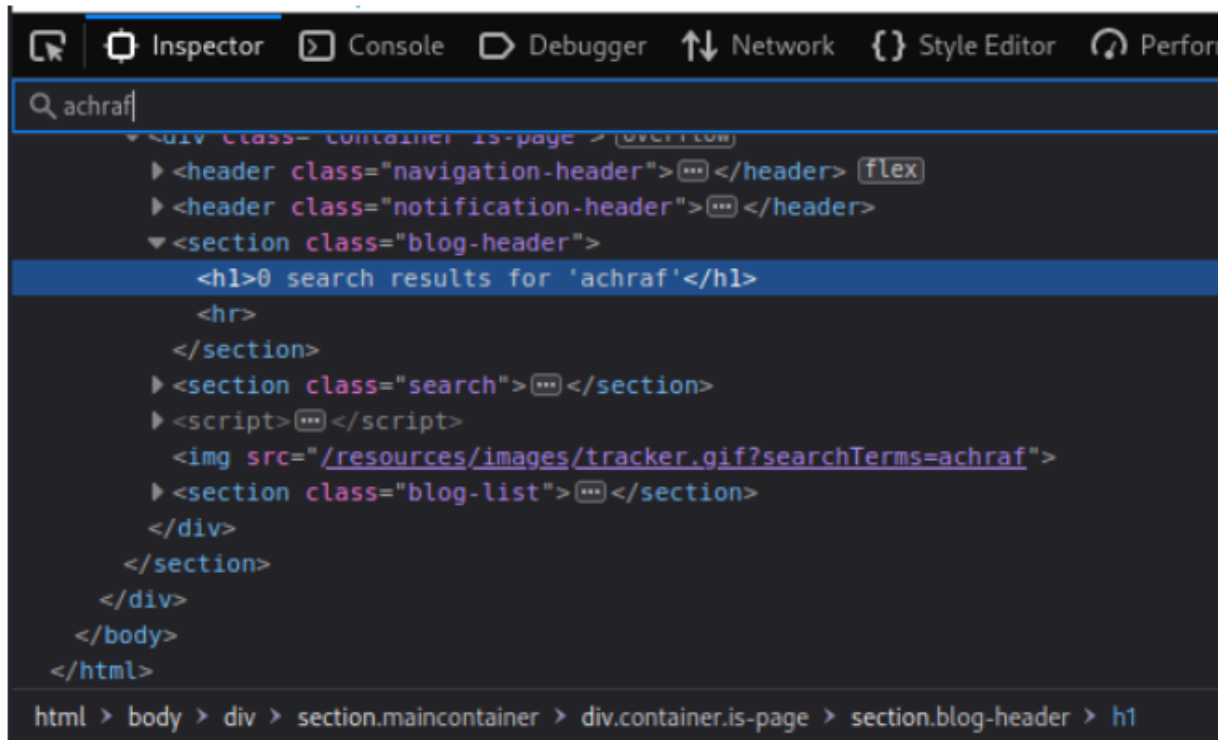
```
Inspector Console Debugger Network Style Editor Performance Memory Storage Accessibility Application
Filter Output Errors Warnings Logs Info Debug CSS XHR
Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.web-security-academy.net/resources/css/labsBlog.css
Source Map URL: labsBlog.css.map [Learn More]
Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.web-security-academy.net/resources/labheader/css/academyLabHeader.css
Source Map URL: academyLabHeader.css.map [Learn More]
Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.web-security-academy.net/resources/css/labsBlog.css
Source Map URL: labsBlog.css.map [Learn More]
Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.web-security-academy.net/resources/labheader/css/academyLabHeader.css
Source Map URL: academyLabHeader.css.map [Learn More]
>> window.location.search
← "?search=%3C%3C%3C%3C%3E%3E%3E%2F%2F%2F"
>> |
```

```
⚠ Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.web
Source Map URL: labsBlog.css.map [Learn More]

⚠ Source map error: Error: request failed with status 404
Resource URL: https://0ac800ae03631586824b473a00ac006d.web
Source Map URL: academyLabHeader.css.map [Learn More]

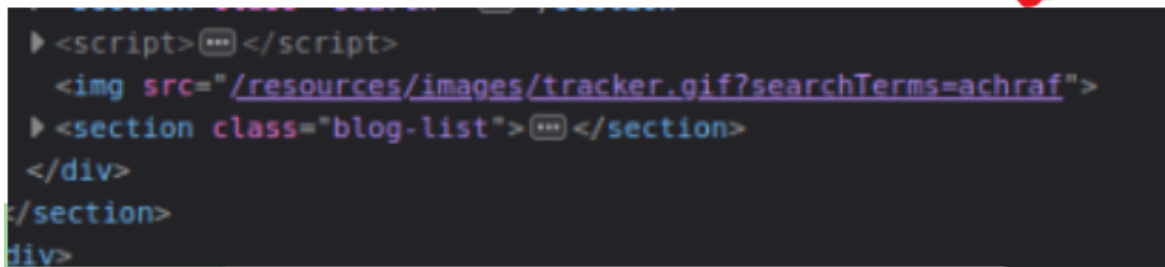
>> window.location.search
← "?search=%3C%3C%3C%3C%3E%3E%3E%2F%2F%2F"

>> |
```

```
Q achraf|
<div class="container is-page">
  <header class="navigation-header">
  <header class="notification-header">
  <section class="blog-header">
    <h1>0 search results for 'achraf'</h1>
    <hr>
  </section>
  <section class="search">
  <script>
  
  <section class="blog-list">
</div>
</section>
</div>
</body>
</html>

html > body > div > section.maincontainer > div.container.is-page > section.blog-header > h1
```



```
<script>

<section class="blog-list">
</div>
</section>
div>
```

On inspecte l'élément pour comprendre le mécanisme
Donc on compris ce qu'on doit faire c'est tout simplement
il faut s'échapper de la balise ``

Et ensuite, on va voir le contenu de notre requête lors du transfert pour nous assurer que cette dernière n'a pas été modifié.

Repeater Window Help

Target: <https://0ac800ae03631586824b473a00ac006d.web-security-academy.net>

Hex

586824b473a00ac006d.web-security-academy

on=kMN46No6gllA9XGtUKWmgxuj2VG3al6q
ozilla/5.0 (X11; Linux x86_64; rv:102.0)
01 Firefox/102.0

location/xhtml+xml,application/xml;q=0.9
image/webp,*/*;q=0.8
ge: en-US,en;q=0.5
ng: gzip, deflate

0ae03631586824b473a00ac006d.web-security
?search=%3C%3C%3C%3C%3E%3E%3E%2F%2F%2F
ure-Requests: 1
t: document
e: navigate
e: same-origin
r: ?1

lose

Response

Pretty Raw Hex Render

```
</p>
<div>
  </div>
</section>
</header>
<header class="notification-header">
</header>
<section class=blog-header>
<div>
  0 search results for 'gggg'
</div>
</hr>
</section>
<section class=search>
<form action=/ method=GET>
  <input type=text placeholder='Search
blog...' name=search>
  <button type=submit class=button>
    Search
  </button>
</form>
</section>
<script>
function trackSearch(query) {
  document.write(
    ' <script>alert("achraf");</script>

Search the blog...

Search

Inspector

achraf

```
<!DOCTYPE html>
<html>
 <head>
 <script src="/resources/labheader.js"></script>
 <div id="academyLabHeader"></div>
 <div these="blog"></div>
 </body>
</html>
```

**Bienvenue dans notre site QSTINFO**

Nom :  
user

Email :  
benchehlaachraf@gmail.com

Votre question :  
votre question |

Poser la question

**LEAR4PENWEB**

**VULNERABLE XSS**

Cette page indique 1

**Bie** **INFO**

Nom :  
hacker

Email :  
benchehlaachraf@gmail.com

Votre question :  
"><img src=x onerror=javascript:alert(('1'))>

Poser la question

**LEAR4PENWEB**

**VULNERABLE XSS**

Dans ce cas, un utilisateur normal entre ses informations et pose sa question. Mais si un utilisateur malveillant entre un payload en JavaScript, dans ce cas, l'attaquant peut récupérer des informations très critiques sur l'utilisateur, par exemple les cookies et les sessions, etc.

Dans cette exemple on a simplement tester par un code qui affiche un message d'alerte mais on peut injecter un code malveillant qui peut faire des tache plus dangereuse a l'utilisateur

Ce payload XSS est conçu pour afficher une boîte de dialogue contenant le message « 1 » lorsqu'il est exécuté dans le navigateur de la victime. Il utilise une balise d'image <img> avec une source « x » qui n'existe pas, et utilise l'attribut onerror pour exécuter du code JavaScript. Dans ce cas, le code JavaScript est simplement une alerte qui affiche le message « 1 ».

L'objectif de ce payload peut être de tester si le site web est vulnérable à l'injection de code malveillant ou de causer des problèmes à l'utilisateur en affichant des alertes non désirées.

## II) VULNARABILITE & Code

```
<script>
// partie de code javascript LEARN4penweb
// récupérer le formulaire
const form = document.querySelector("form");

// ajouter un événement "submit" sur le formulaire
form.addEventListener("submit", (event) => {
 // empêcher la soumission du formulaire

 // récupérer la valeur du champ de texte de question
 const questionInput = document.querySelector("#question");
 const QST = questionInput.value;

 // Afficher la valeur de QST sur la page sans l'échapper
 const div = document.createElement("div");
 div.innerHTML = QST;
 document.body.appendChild(div);
});
</script>
<script>
function afficher() {
 var valeur = document.getElementById("question").value;
 document.getElementById("resultat").innerHTML = question;
}
</script>
```

Ce code est vulnérable aux attaques XSS car il n'échappe pas les caractères spéciaux dans la variable QST avant de les afficher sur la page. Cela signifie qu'un utilisateur malveillant peut saisir du code JavaScript dans le champ de texte de question et l'envoyer avec le formulaire, ce qui peut ensuite être exécuté sur la page.

Par exemple, si un utilisateur entre la chaîne de caractères suivante dans le champ de question:

Alors, lorsque le formulaire est soumis, cette chaîne de caractères est affichée directement sur la page sans être échappée, ce qui peut entraîner l'exécution du code malveillant `alert("Je suis un attaquant!");`.

### III) SOLUTION

```
// xss securise //
<script>
 // partie de code javascript LEARN4penweb
 // récupérer le formulaire
 const form = document.querySelector("form");

 // ajouter un événement "submit" sur le formulaire
 form.addEventListener("submit", (event) => {
 // empêcher la soumission du formulaire
 event.preventDefault();

 // récupérer la valeur du champ de texte de question
 const questionInput = document.querySelector("#question");
 const QST = questionInput.value;

 // Afficher la valeur de QST sur la page en échappant les caractères spéciaux
 const div = document.createElement("div");
 const text = document.createTextNode(QST);
 div.appendChild(text);
 document.body.appendChild(div);
 // on peut ajouter cette fonction aussi ou même créer des fonctions pour éviter
 // injection du xss

 function sanitizeString(str) {
 const regex = /[<>\"'&]/gi;
 return str.replace(regex, '');
 }

 // utilisation dans le code
 const questionInput = document.querySelector("#question");
 const QST = sanitizeString(questionInput.value);
 const div = document.createElement("div");
 div.textContent = QST;
 document.body.appendChild(div);

 });
</script>
```

## Voici quelques consignes pour éviter XSS

Valider toutes les entrées utilisateur : Il est important de valider toutes les entrées utilisateur et de s'assurer qu'elles ne contiennent pas de code malveillant. Les données entrées par l'utilisateur doivent être filtrées et validées du côté serveur.

Échapper les données d'entrée : Échapper les données entrées par l'utilisateur avant de les afficher dans le code HTML. Les caractères spéciaux comme <, >, & et " doivent être remplacés par leur équivalent en HTML (<, >, & et ").

Utiliser les en-têtes de sécurité HTTP : Utiliser les en-têtes de sécurité HTTP pour protéger votre application contre les attaques XSS. Les en-têtes tels que X-XSS-Protection, Content-Security-Policy et X-Content-Type-Options peuvent aider à réduire le risque d'attaques XSS.

Utiliser une bibliothèque de sécurité : Utiliser une bibliothèque de sécurité telle que OWASP ESAPI ou HTML Purifier pour échapper les données d'entrée et protéger votre application contre les attaques XSS.

Éduquer les utilisateurs : Éduquer les utilisateurs sur les risques de sécurité en ligne et sur la manière de se protéger contre les attaques XSS. Les utilisateurs doivent être conscients des risques liés à l'ouverture de liens ou de fichiers provenant de sources inconnues.



## Utilisation des fonctions en JS pour éviter xss et augmenter la sécurité de vote code

Échapper les caractères spéciaux: Utilisez la fonction `encodeURIComponent()` pour échapper les caractères spéciaux qui pourraient être utilisés dans une attaque XSS.

```
const userInput = "<script>alert('XSS!');</script>";
const escapedInput = encodeURIComponent(userInput);
```

Échapper les caractères HTML: Utilisez la fonction `innerText` pour échapper les caractères HTML dans les éléments HTML.

```
const userInput = "<script>alert('XSS!');</script>";
const outputElement = document.createElement('div');
outputElement.innerText = userInput;
document.body.appendChild(outputElement);
```

Utiliser des entités HTML: Vous pouvez également utiliser des entités HTML pour échapper les caractères spéciaux. Par exemple, utilisez &lt; à la place de < et &gt; à la place de >.

```
const userInput = "<script>alert('XSS!');</script>";
const outputElement = document.createElement('div');
outputElement.innerHTML = userInput.replace(/</g, '<').replace(/>/g, '>');
document.body.appendChild(outputElement);
```

Sanitization de la chaîne de caractères : Utilisez une bibliothèque de nettoyage de chaîne de caractères comme DOMPurify pour nettoyer le contenu HTML et supprimer tout code malveillant potentiel.

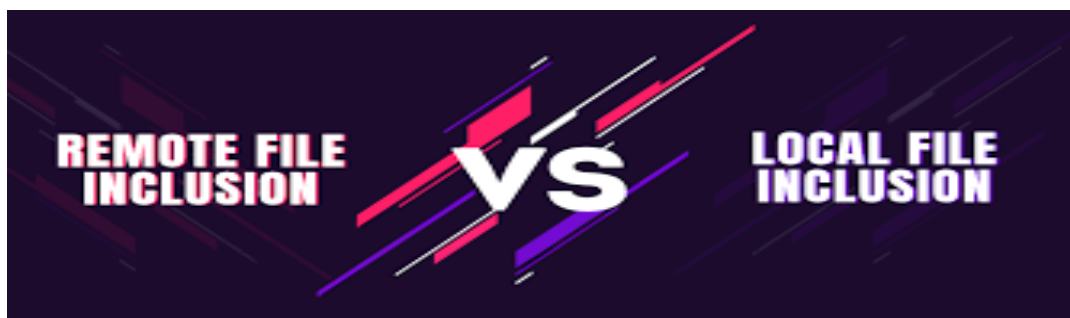
```
const userInput = "<script>alert('XSS!');</script>";
const outputElement = document.createElement('div');
outputElement.innerHTML = DOMPurify.sanitize(userInput);
document.body.appendChild(outputElement);
```



# ***FILE INCLUSION***

## ***1)DEFINITION***

L'inclusion de fichier (file inclusion) est une vulnérabilité courante sur les sites Web qui permet à un attaquant d'inclure et d'exécuter du code arbitraire dans une page Web. Cette vulnérabilité peut être exploitée de différentes manières, mais les deux types les plus courants sont:



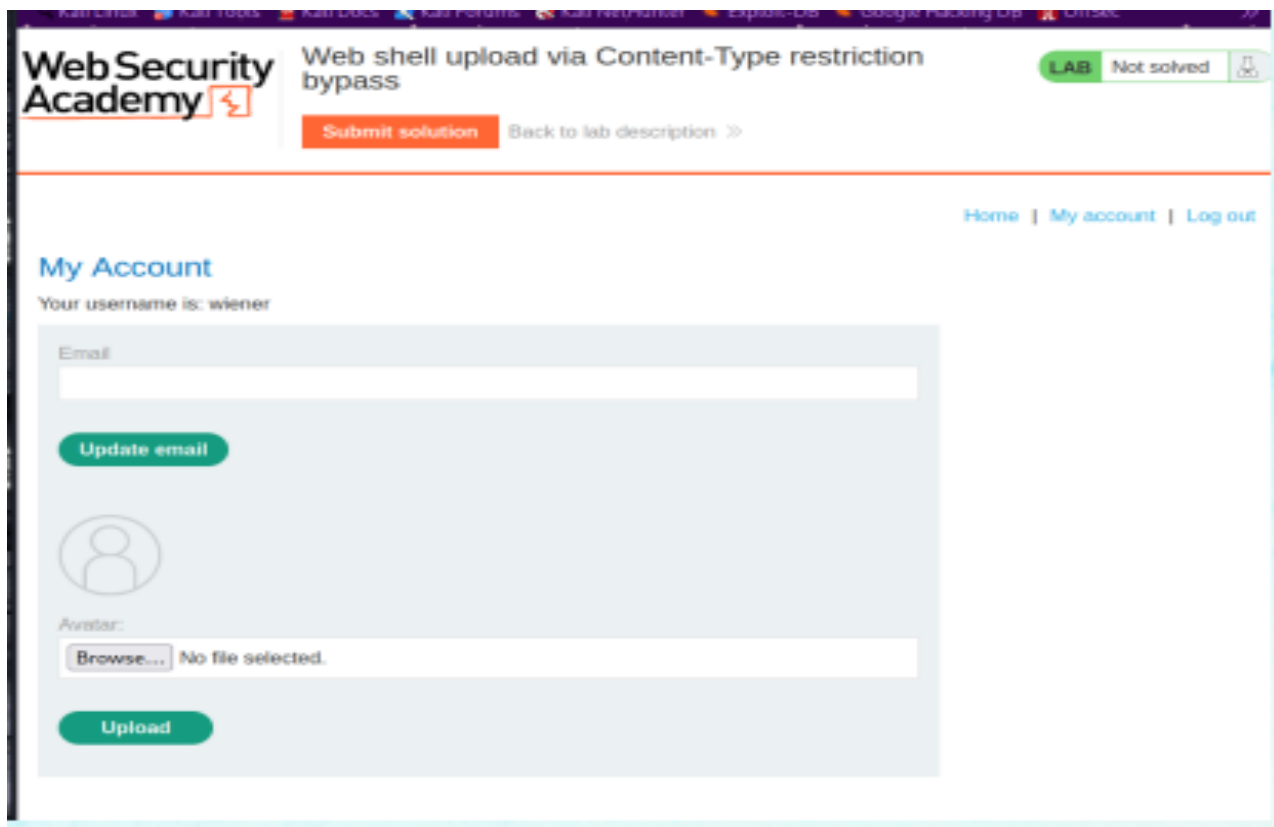
## **LFI (local file inclusion ) :**

Inclusion de fichiers locaux (Local File Inclusion - LFI): Lorsque cette vulnérabilité est présente, un attaquant peut inclure un fichier local, tel que des fichiers de configuration, des journaux d'accès ou même des fichiers sensibles contenant des informations d'identification. Un exemple de LFI serait l'ajout d'un chemin de fichier local à une requête Web pour accéder à un fichier système sur le serveur Web

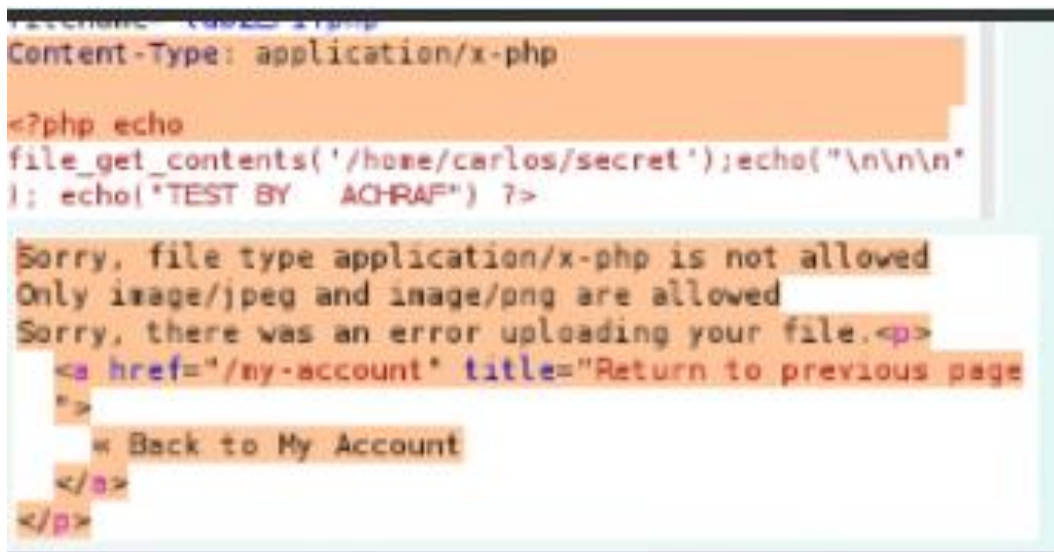
## **RFI (Remote file inclusion ) :**

Inclusion de fichiers distants (Remote File Inclusion - RFI): Lorsque cette vulnérabilité est présente, un attaquant peut inclure un fichier distant sur un serveur Web tiers, qui peut contenir du code malveillant. Un exemple de RFI serait l'ajout d'un chemin de fichier distant à une requête Web pour inclure un fichier hébergé sur un autre serveur

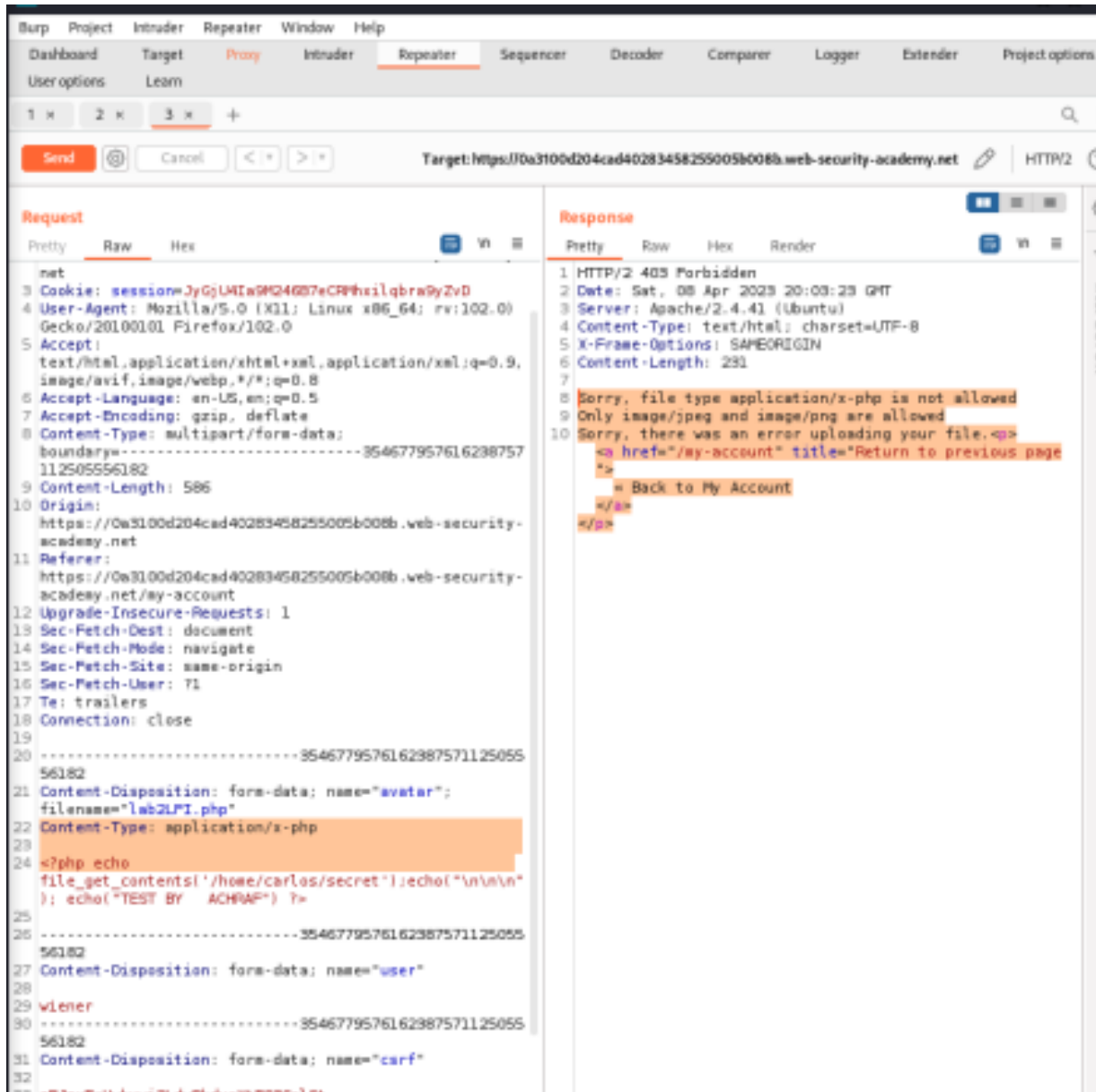
## **PRATIQUE**



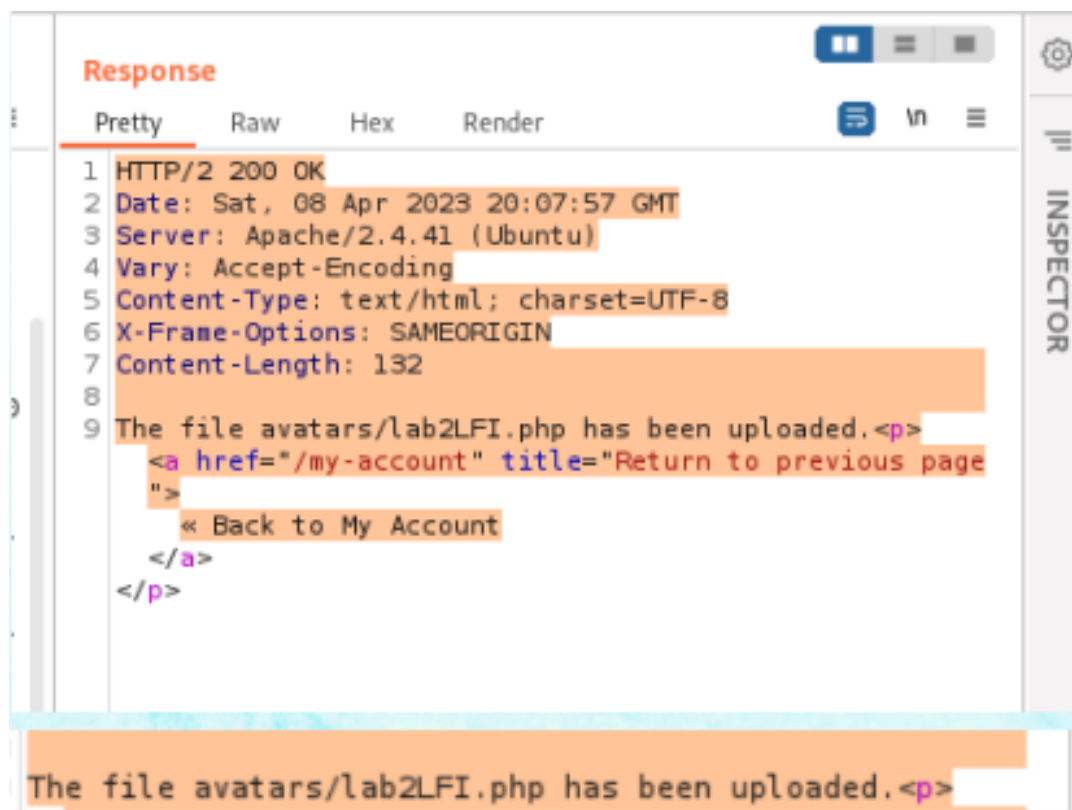
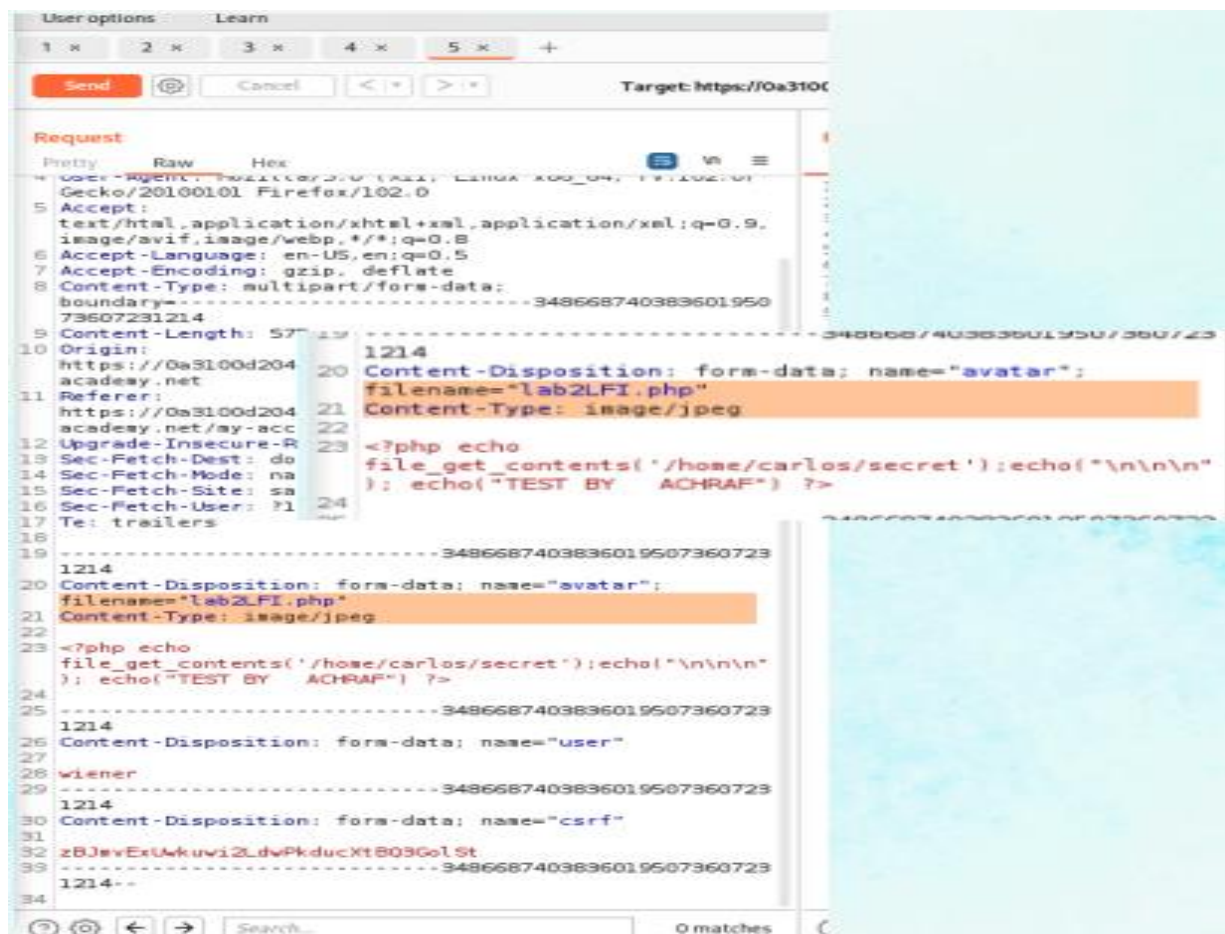
Ce site offre la fonctionnalité d'ajout d'un fichier. Nous allons essayer d'injecter notre fichier PHP pour afficher le contenu d'un dossier qui contient le flag (solution du lab)



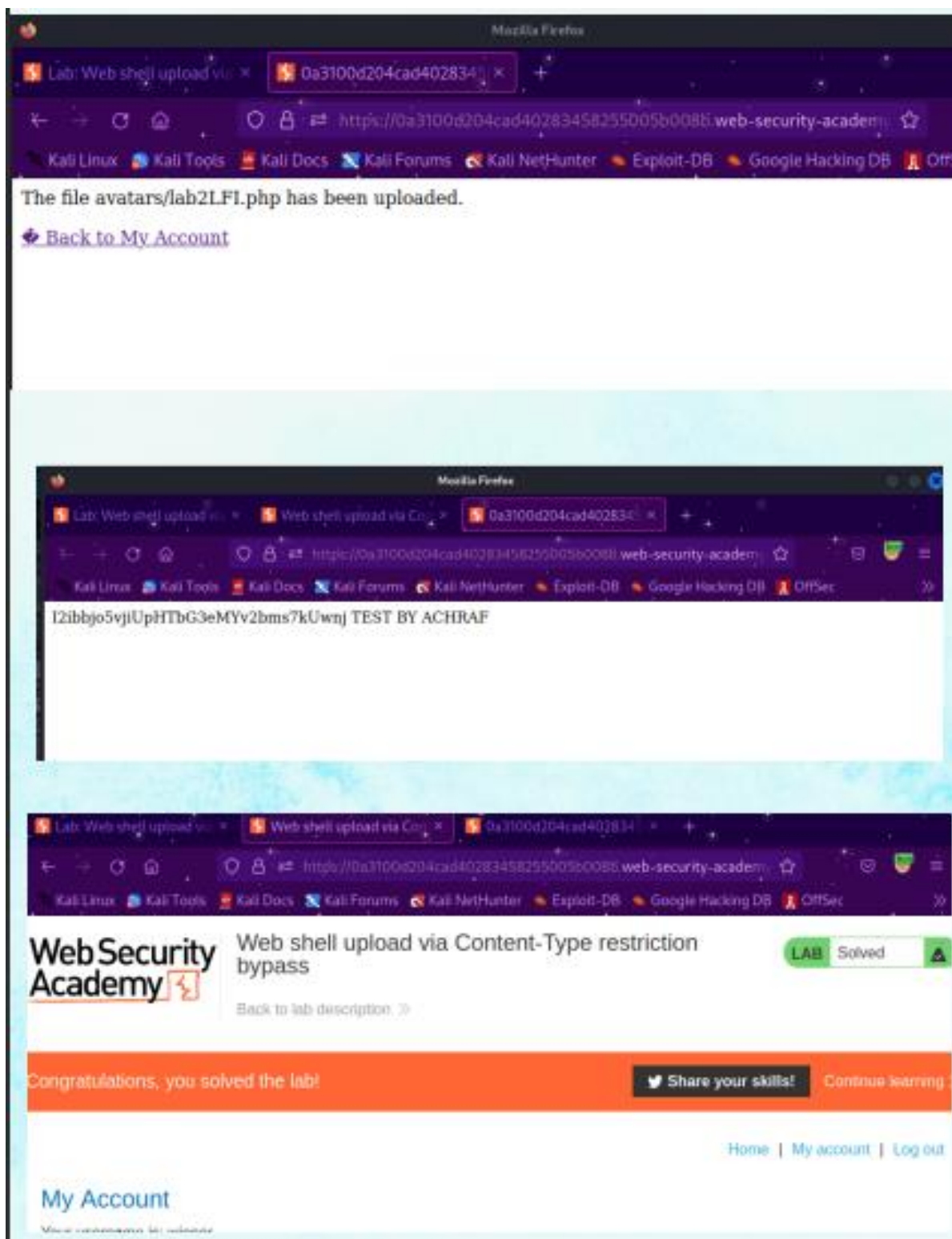
**Utilisant burpsuite on remarque se type MIME filtre le contenu qu'il doit être télécharger dans la page web donc il faut éviter le filtrage par cette méthode car un attaquant peut changer sa valeur**

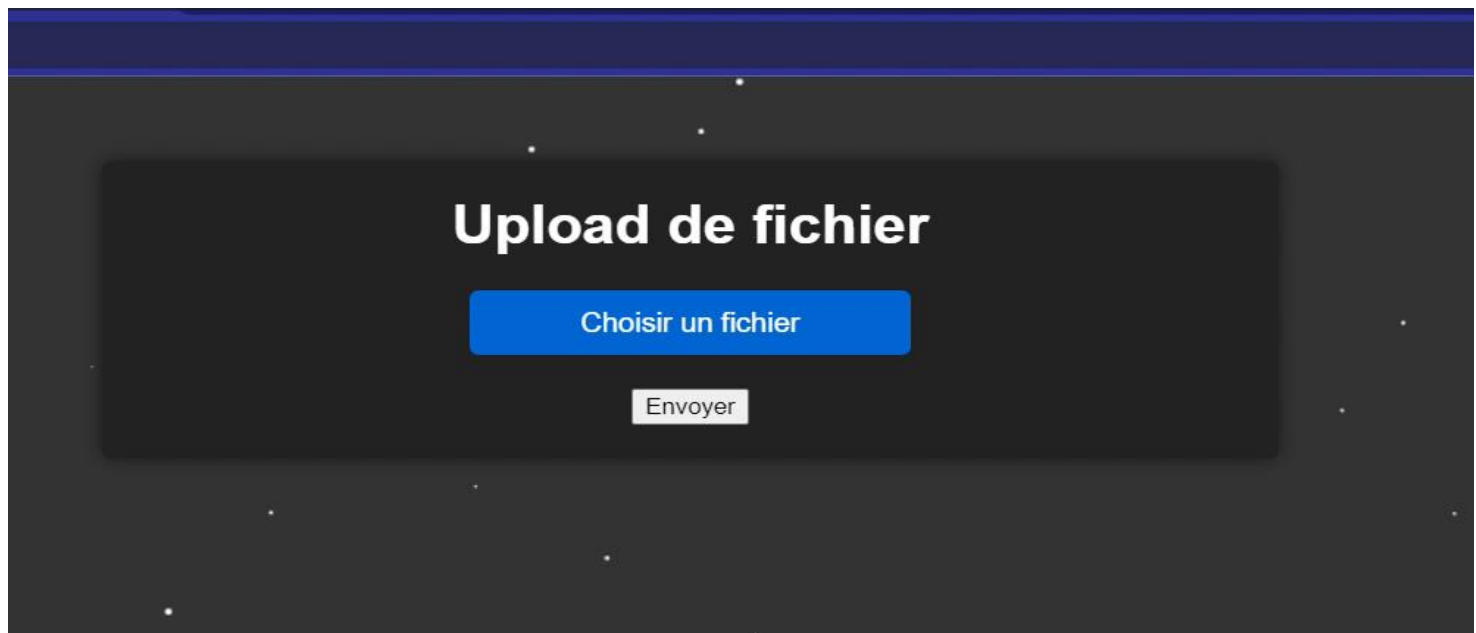


**On va tester de changer la valeur du type content et on va voir**



Il semble que le fichier a été déposé sur le serveur. Maintenant, nous allons visualiser son contenu






On a uploadé plusieurs fichiers dans ce site comme vous voyez ces fichiers contient des mots de passe et des informations critiques a propos de l'application xamp donc on voire les différents types de filtrage possible



## Index of /fileup


<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">apache-info.txt</a>	2023-05-02 23:41	2.9K	
 <a href="#">file.php</a>	2023-05-02 18:27	6.0K	
 <a href="#">password.txt</a>	2023-05-02 23:41	543	
 <a href="#">test_file-inclu.png</a>	2023-05-02 23:41	8.3K	

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



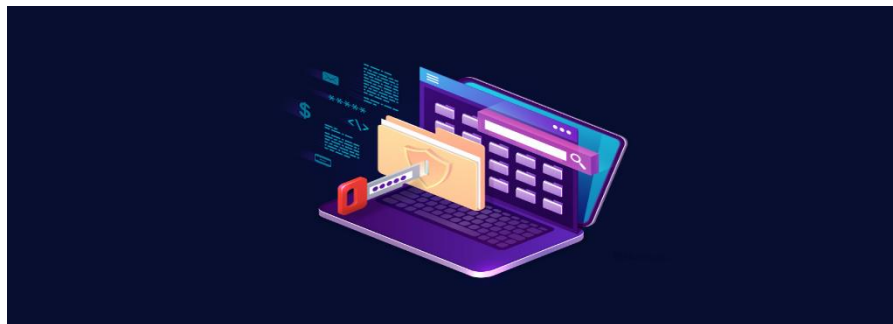
Si un attaquant peut entrer des fichiers malveillants, il peut accéder à des informations auxquelles il n'a pas le droit, et ces informations peuvent être critiques

## Index of /fileup

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 <a href="#">Parent Directory</a>		-	
 <a href="#">apache-info.txt</a>	2023-05-02 23:41	2.9K	
 <a href="#">file.php</a>	2023-05-02 18:27	6.0K	
 <a href="#">password.txt</a>	2023-05-02 23:41	543	
 <a href="#">test_file-inclu.png</a>	2023-05-02 23:41	8.3K	
 <a href="#">virus.php</a>	2023-05-02 23:41	652	

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

Dans cet exemple, l'attaquant a injecté un code malveillant qui va lister le contenu de tous les fichiers qui se trouvent avec lui dans le même répertoire. Le fichier s'appelle virus.php. Lorsqu'on clique sur le fichier, ce dernier commence à s'exécuter, mais dans d'autres cas, l'attaquant peut injecter un code très dangereux pour prendre le contrôle total de la machine de la victime







```

if(isset($_FILES['file'])) {
 $target_dir = "C:/xampp/htdocs/fileup/";
 $target_file = $target_dir . base-
name($_FILES["file"]["name"]);
 $uploadOk = 1;
 $imageFileType = strtolower(pathinfo($target_file,PATH-
INFO_EXTENSION));

 // Vérifie si le fichier est une image ou non
 if(isset($_POST["submit"])) {
 $check = getimagesize($_FILES["file"]["tmp_name"]);
 if($check !== false) {
 echo "Le fichier est une image - " . $check["mime"] .
".";
 $uploadOk = 1;
 } else {
 echo "Le fichier n'est pas une image.";
 $uploadOk = 0;
 }
 }
}

```

Ce code est vulnérable car il ne filtre pas les fichiers entre par utilisateur

```

-----filtrage cote client (vulnérable défavorable)

```

Le filtrage des fichiers côté client est effectué sur le navigateur de l'utilisa-  
 teur avant que les fichiers ne soient téléchargés sur le serveur.

Les technologies telles que JavaScript peuvent être utilisées pour valider le  
 type de fichier et les extensions autorisées. Cependant, le filtrage

Côté client peut être contourné par des utilisateurs malveillants qui peuvent  
 modifier le code JavaScript pour télécharger des fichiers non autorisés.

```

-----filtrage cote serveur (fort favorable et peut vulnérable)

```

Le filtrage des fichiers côté serveur est effectué sur le serveur avant de per-  
 mettre le téléchargement des fichiers. Les langages de programmation tels que  
 PHP, Python et Ruby peuvent être utilisés pour valider le type de fichier et les  
 extensions autorisées. Le filtrage côté serveur est généralement plus sûr car  
 il est impossible pour les utilisateurs de contourner la validation.

### *III) Solution*

```

// Vérifie la taille du fichier
// ici c'est une precaution important il faut limiter
// la taille des fichier entre par l'utilisateur
if ($_FILES["file"]["size"] > 500000) {
 echo "Désolé, votre fichier est trop volumineux.";
 $uploadOk = 0;
}

// Autorise certains formats de fichier
// ici on ajoute des extension des fichier qu'on veut
que l'utilisateur entre
if($imageFileType != "jpg" && $imageFileType != "png" &&
$imageFileType != "jpeg"
&& $imageFileType != "gif" && $imageFileType != "php"
&& $imageFileType != "txt") {
 echo "Désolé, seuls les fichiers JPG, JPEG, php , PNG &
GIF sont autorisés.";
 $uploadOk = 0;
 // on essaie de supprimer tous les extension executable
 if($imageFileType != "jpg" && $imageFileType != "png" &&
$imageFileType != "jpeg"
&& $imageFileType != "gif" && $imageFileType != "txt")
{
 echo "Désolé, seuls les fichiers JPG, JPEG, PNG & GIF
sont autorisés.";
 $uploadOk = 0;
 /*
 generalement il faut chercher tous les version des
scriptes php php6
 pour essayer de filter tous les autres extension du php
(le langage utilise dans notre cas)
 */
 // il existe un autre type de filtrage c'est le fil-
trage par le nombre magic du fichier voici un exemple mais
attention ce dernier est vulnérable
/
Vérifie si le fichier est une image en utilisant son nombre
magique

$info = finfo_open(FILEINFO_MIME_TYPE);
$mime = finfo_file($info, $_FILES["file"]["tmp_name"]);

```

```
if(strpos($mime, "image") !== 0) {
 echo "Désolé, seules les images sont autorisées.";
 $uploadOk = 0;
}
finfo_close($finfo);
```

Ce code est destiné à vérifier la validité d'un fichier téléchargé par un utilisateur avant de le télécharger sur un serveur. Il effectue plusieurs vérifications pour s'assurer que le fichier est conforme aux attentes.

La première vérification limite la taille du fichier téléchargé. Si la taille du fichier est supérieure à 500 000 octets, un message d'erreur est affiché et le téléchargement est refusé.

La deuxième vérification vérifie l'extension du fichier téléchargé. Seuls les fichiers avec les extensions .jpg, .jpeg, .png, .gif, .php et .txt sont autorisés. Si une extension différente est trouvée, un message d'erreur est affiché et le téléchargement est refusé.

Il y a également une tentative de suppression des extensions de fichier exécutables telles que .php. Enfin, un dernier filtre est effectué en utilisant le nombre magique du fichier pour vérifier si le fichier est une image. Si le fichier ne semble pas être une image, un message d'erreur est affiché et le téléchargement est refusé.

Ces vérifications sont importantes pour éviter les téléchargements de fichiers malveillants ou potentiellement dangereux sur un serveur. Cependant, il est important de noter que ces mesures ne garantissent pas à 100% la sécurité contre les attaques, et il est donc recommandé d'utiliser des techniques de sécurité supplémentaires pour renforcer la sécurité du téléchargement de fichiers sur un serveur.

## Astuce pour éviter (RFI LFI)

Utiliser une liste blanche (whitelist) : au lieu d'interdire des fichiers spécifiques, autorisez uniquement les fichiers qui sont nécessaires pour votre application. Cela peut être fait en vérifiant le nom du fichier, le type de fichier ou le chemin du fichier.

Restreindre l'accès aux fichiers inclus : placez les fichiers qui doivent être inclus dans un répertoire protégé et restreignez l'accès à ce répertoire. Vous pouvez également utiliser des règles d'accès au niveau du serveur pour empêcher l'accès direct aux fichiers inclus.

Utiliser des chemins absolus : utilisez des chemins absolus plutôt que des chemins relatifs pour inclure des fichiers. Les chemins absolus spécifient le chemin complet à partir de la racine du serveur, ce qui rend plus difficile pour un attaquant de modifier le chemin.

Utiliser une fonction d'inclusion sécurisée : utilisez des fonctions d'inclusion de fichiers sécurisées telles que `require_once()` ou `include_once()`. Ces fonctions effectuent des vérifications de sécurité avant d'inclure des fichiers, ce qui peut aider à prévenir les inclusions de fichiers malveillants.

Ne pas faire confiance aux entrées de l'utilisateur : vérifiez et nettoyez toutes les entrées de l'utilisateur avant de les utiliser dans une inclusion de fichier. Assurez-vous que les entrées sont valides et qu'elles ne contiennent pas de caractères malveillants.

## ***PORTSWIGGER PLATEFORME***



## ***DVWA PLATEFORME***



## ***BURPSUITE***



# ***Conclusion :***

*Dans le cadre de notre projet de Pentesting Web, nous avons réalisé des tests approfondis pour évaluer la sécurité des applications Web en utilisant différentes techniques et en ciblant des vulnérabilités courantes telles que XSS (Cross-Site Scripting), l'injection SQL (SQL Injection), l'authentification défectueuse (Broken Authentication), l'exploitation des entités externes (XXE - XML External Entity), l'injection de commandes (Command Injection), le contrôle d'accès défectueux (Broken Access Control), la force brute (Brute Force), les identifiants de session faibles (Weak Session ID) et le contournement de la politique de sécurité du contenu (CSP Bypass).*

*Nos tests ont révélé plusieurs vulnérabilités et faiblesses dans les applications Web évaluées, soulignant l'importance de la sécurité des applications dans le paysage numérique actuel. Les constatations les plus significatives de notre projet incluent :*

*XSS (Cross-Site Scripting) : Nous avons identifié des vulnérabilités XSS, ce qui signifie que les applications ne filtrent pas correctement les entrées utilisateur, permettant aux attaquants d'injecter du code malveillant et de l'exécuter sur le navigateur des utilisateurs. Cela peut conduire à des attaques de vol de session, à la divulgation de données sensibles et à la manipulation du contenu affiché.*

*Injection SQL : Nous avons découvert des vulnérabilités d'injection SQL, révélant que les applications ne valident pas suffisamment les entrées utilisateur avant de les incorporer dans les requêtes SQL. Cela expose les applications à des attaques permettant aux attaquants d'interagir directement avec la base de données et de compromettre la confidentialité et l'intégrité des données.*

*Authentification défectueuse : Nous avons constaté des problèmes liés à l'authentification défectueuse, tels que des politiques de mots de passe faibles, des sessions non expirées ou des mécanismes de réinitialisation de mot de passe*

***vulnérables. Ces vulnérabilités peuvent permettre à des attaquants de s'approprier des comptes d'utilisateur, d'accéder à des informations confidentielles et de compromettre la sécurité des utilisateurs.***

***Exploitation des entités externes (XXE) : Nous avons identifié des vulnérabilités XXE, ce qui signifie que les applications ne gèrent pas correctement les entrées XML externes, permettant aux attaquants de lire des fichiers sensibles, d'effectuer des attaques par déni de service ou d'accéder à des ressources critiques sur le serveur.***

***Injection de commandes : Nous avons repéré des vulnérabilités d'injection de commandes, indiquant que les applications ne filtrent pas correctement les entrées utilisateur avant de les exécuter comme des commandes système. Cela peut permettre à des attaquants d'exécuter des commandes malveillantes sur le serveur, entraînant des risques de compromission du système et de fuite de données.***

***Contrôle d'accès défectueux : Nous avons constaté des lacunes dans le contrôle d'accès, où des utilisateurs non autorisés peuvent accéder à des ressources sensibles ou effectuer des actions pour lesquelles ils n'ont pas les droits nécessaires. Ces vulnérabilités peuvent entraîner des fuites d'informations confidentielles, des atteintes à la vie privée des utilisateurs et compromettre l'intégrité du système.***

***Force brute : Nous avons identifié des vulnérabilités de force brute, ce qui signifie que les applications ne mettent pas en place des mécanismes adéquats pour limiter le nombre de tentatives d'authentification. Cela facilite les attaques par force brute, où les attaquants peuvent essayer de deviner les mots de passe en utilisant des techniques d'automatisation, augmentant ainsi le risque de compromission des comptes utilisateur.***

***Identifiants de session faibles : Nous avons constaté l'utilisation d'identifiants de session faibles ou prévisibles, ce qui peut faciliter les attaques d'usurpation de session. Les attaquants pourraient exploiter ces vulnérabilités pour accéder à***

***des comptes d'utilisateurs légitimes, effectuer des actions non autorisées et compromettre la confidentialité et l'intégrité des données.***

***En conclusion, notre projet de Pentesting Web a mis en évidence un certain nombre de vulnérabilités et de faiblesses dans les applications évaluées. Ces résultats soulignent l'importance de réaliser des tests de sécurité réguliers et de mettre en place des mesures de protection appropriées pour atténuer les risques. Il est essentiel de mettre en œuvre des pratiques de développement sécurisé, de valider et de filtrer rigoureusement les entrées utilisateur, d'adopter des politiques de mots de passe solides, de gérer correctement les sessions et de mettre en place des contrôles d'accès robustes. En prenant ces mesures, les organisations peuvent renforcer la sécurité de leurs applications Web, protéger les données sensibles et prévenir les attaques potentielles.***