



---

# **Distribution trouée Damn Vulnerable Web Application**

---

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

Le 25 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 Brute force</b>	<b>5</b>
2.1 Description . . . . .	5
2.2 Exploitation . . . . .	5
2.3 Contre-mesure . . . . .	5
<b>3 Command injection</b>	<b>5</b>
3.1 Description . . . . .	5
3.2 Exploitation . . . . .	5
3.3 Contre-mesure . . . . .	5
<b>4 CSRF</b>	<b>5</b>
4.1 Description . . . . .	5
4.2 Exploitation . . . . .	5
4.3 Contre-mesure . . . . .	5
<b>5 File Inclusion</b>	<b>6</b>
5.1 Description de la vulnérabilité . . . . .	6
5.1.1 Local File Inclusion . . . . .	6
5.1.2 Remote File Inclusion . . . . .	6
5.2 Exploitation de la vulnérabilité . . . . .	7
5.3 Contre-mesure . . . . .	7
<b>6 File Upload</b>	<b>7</b>
6.1 Description de la vulnérabilité . . . . .	7
6.1.1 Généralités . . . . .	7
6.1.2 Différentes vulnérabilités . . . . .	8
6.2 Exploitation de la vulnérabilité . . . . .	8
6.3 Contre-mesure . . . . .	8
<b>7 Insecure CAPTCHA</b>	<b>8</b>
7.1 Description de la vulnérabilité . . . . .	9
7.2 Exploitation de la vulnérabilité . . . . .	9
7.3 Contre-mesure . . . . .	9
<b>8 Injection SQL</b>	<b>10</b>
8.1 Description . . . . .	10
8.2 Exploitation . . . . .	10
8.2.1 DVWA - Security level "low" . . . . .	10
8.2.2 DVWA - Security level "Medium" et "High" . . . . .	13
8.3 Contre-mesures . . . . .	13
<b>9 Injection SQL aveugle</b>	<b>15</b>
9.1 Description . . . . .	15
9.2 Exploitation . . . . .	15
9.3 Contre-mesure . . . . .	15

<b>10 Attaques XSS</b>	<b>15</b>
10.1 Description . . . . .	15
10.2 Exploitation . . . . .	15
10.3 Contre-mesure . . . . .	15
<b>11 Attaques XSS enregistrées</b>	<b>15</b>
11.1 Description . . . . .	15
11.2 Exploitation . . . . .	15
11.3 Contre-mesure . . . . .	15
<b>12 Conclusion et perspectives</b>	<b>16</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.

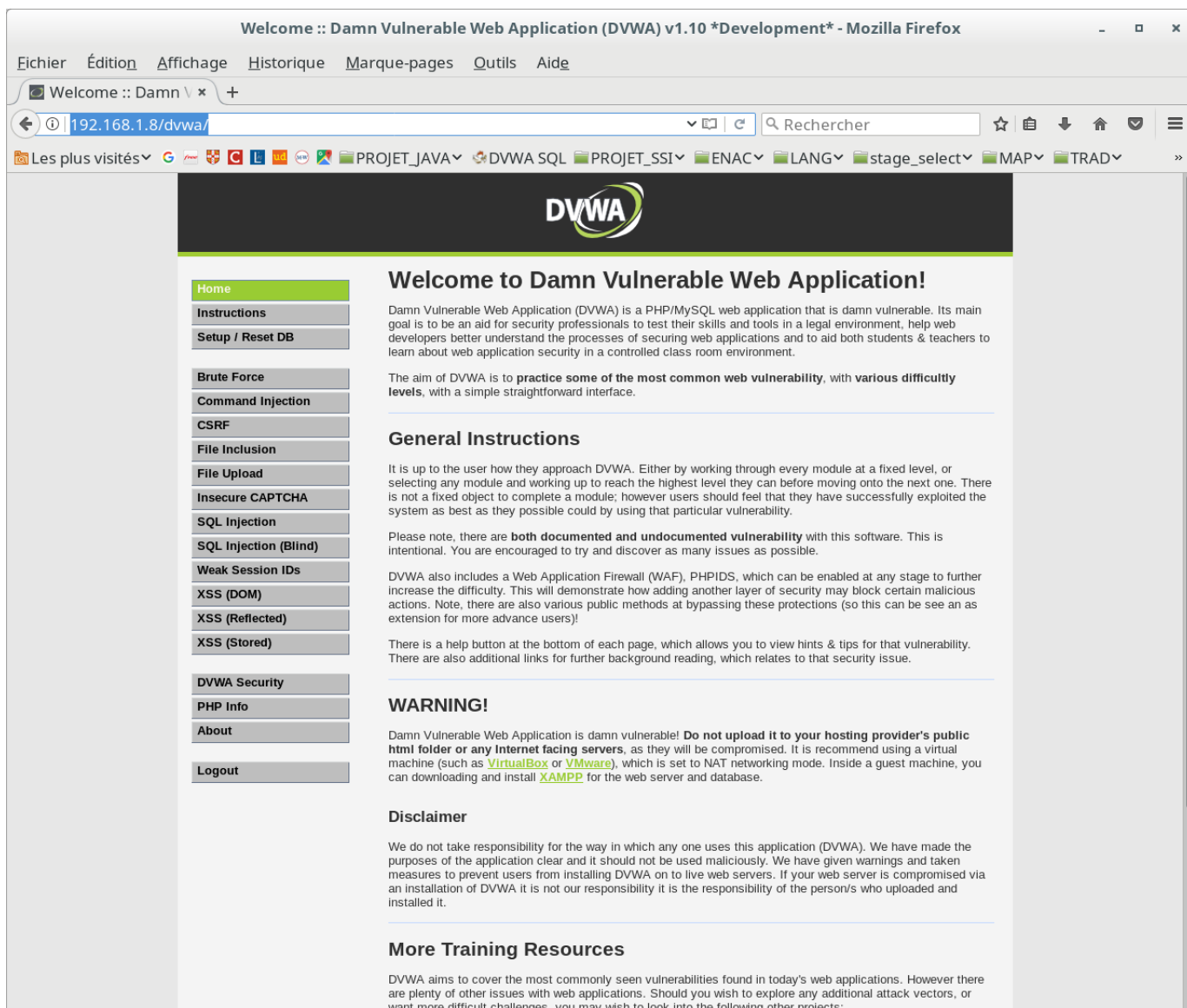


FIGURE 2 – Accès à DVWA à partir du navigateur de la machine hôte. Le site web DVWA est installé sur une machine virtuelle kali linux.

## **2 Brute force**

### **2.1 Description**

### **2.2 Exploitation**

### **2.3 Contre-mesure**

## **3 Command injection**

### **3.1 Description**

### **3.2 Exploitation**

### **3.3 Contre-mesure**

## **4 CSRF**

### **4.1 Description**

### **4.2 Exploitation**

### **4.3 Contre-mesure**

## 5 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.

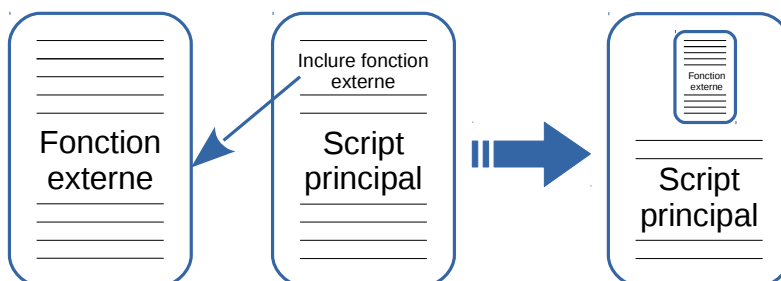


FIGURE 3 – Mécanisme d'inclusion d'un fichier

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.

### 5.1 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 5.1 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).

#### 5.1.1 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.

#### 5.1.2 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

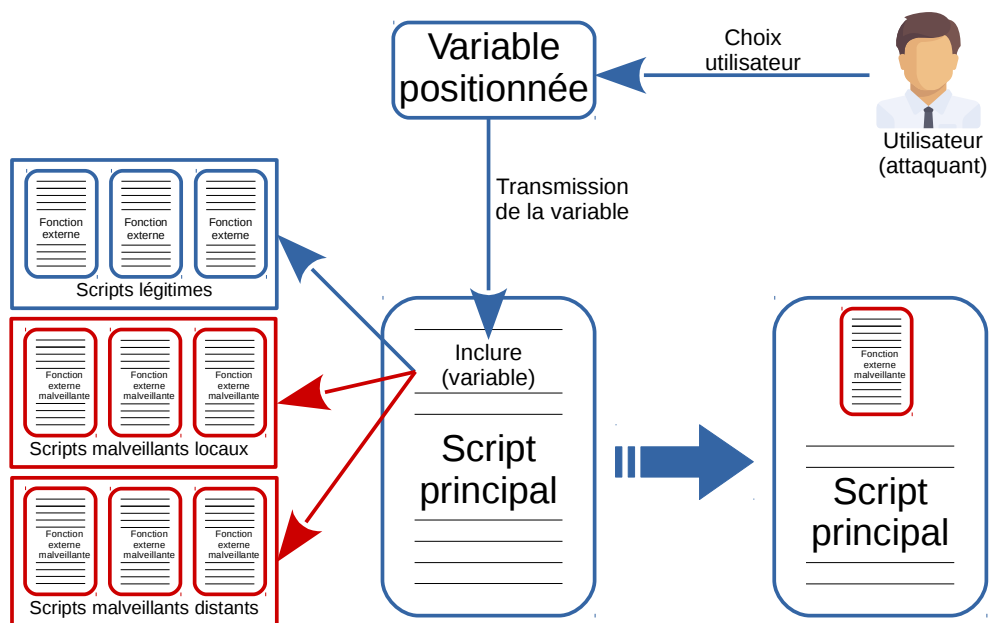


FIGURE 4 – Vulnérabilité d'inclusion d'un fichier

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.

## 5.2 Exploitation de la vulnérabilité

## 5.3 Contre-mesure

# 6 File Upload

De très nombreuses applications web laissent la possibilité à l'utilisateur de charger 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.

## 6.1 Description de la vulnérabilité

### 6.1.1 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 une possibilité claire à un attaquant de charger 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.

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 alors nécessaire de restreindre le chargement à certains types de fichiers ou d'effectuer certains contrôles.

### 6.1.2 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 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.

## 6.2 Exploitation de la vulnérabilité

## 6.3 Contre-mesure

# 7 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.

## 7.1 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.

## 7.2 Exploitation de la vulnérabilité

## 7.3 Contre-mesure

## 8 Injection SQL

### 8.1 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.

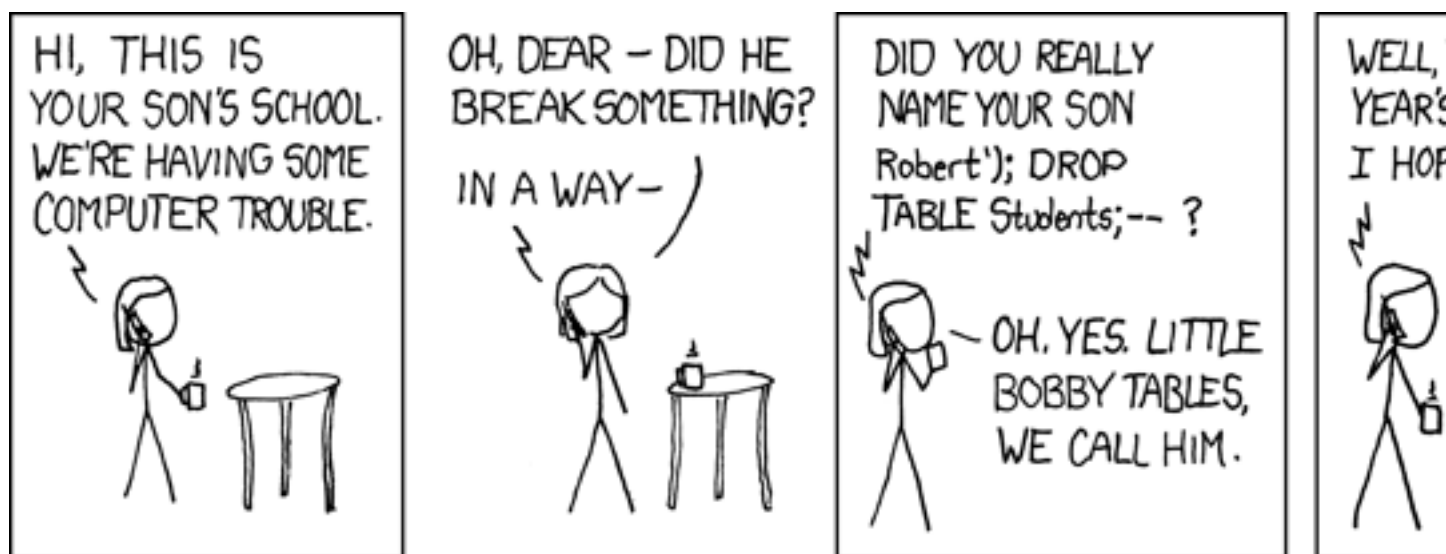


FIGURE 5 – source : <https://xkcd.com/327/>

### 8.2 Exploitation

#### 8.2.1 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 :*

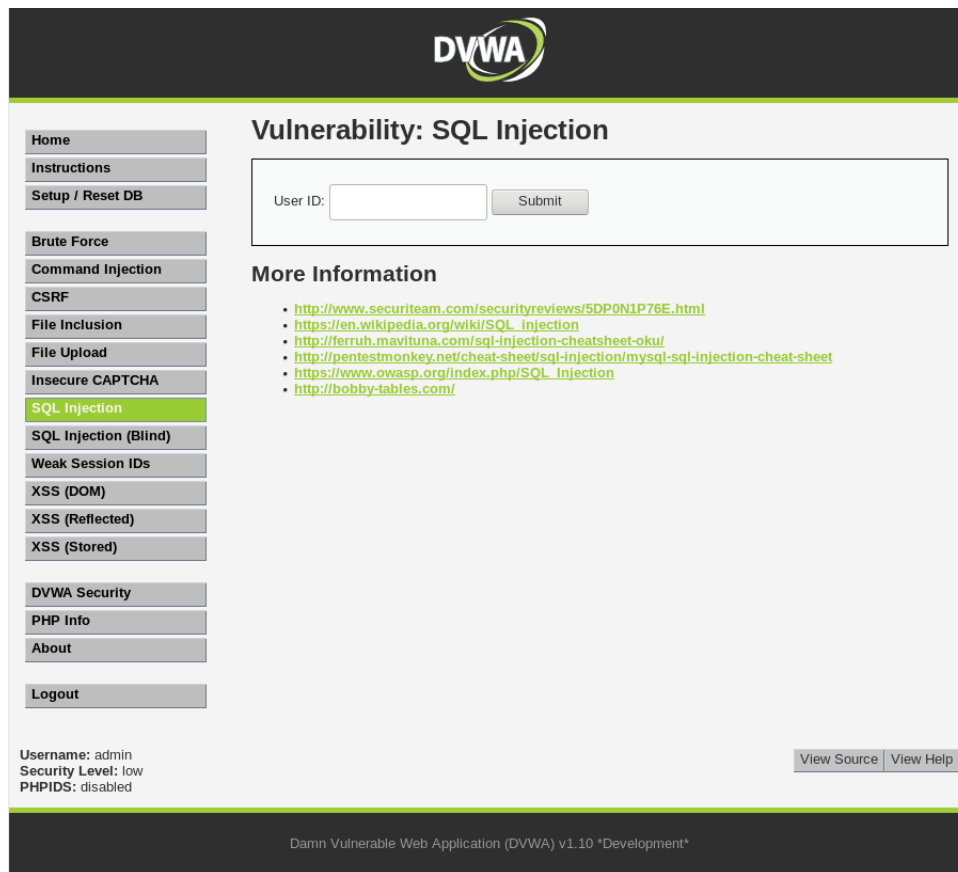


FIGURE 6 – Les saisies incorrectes donnent lieu à un message d’erreur SQL qui informe le hacker potentiel de l’absence de protection contre les SQLi. D’autres informations importantes sont dévoilées comme le type de base de donnée, ici MariaDB, version libre de MySQL rachetée par Oracle.

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

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

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

```
$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.

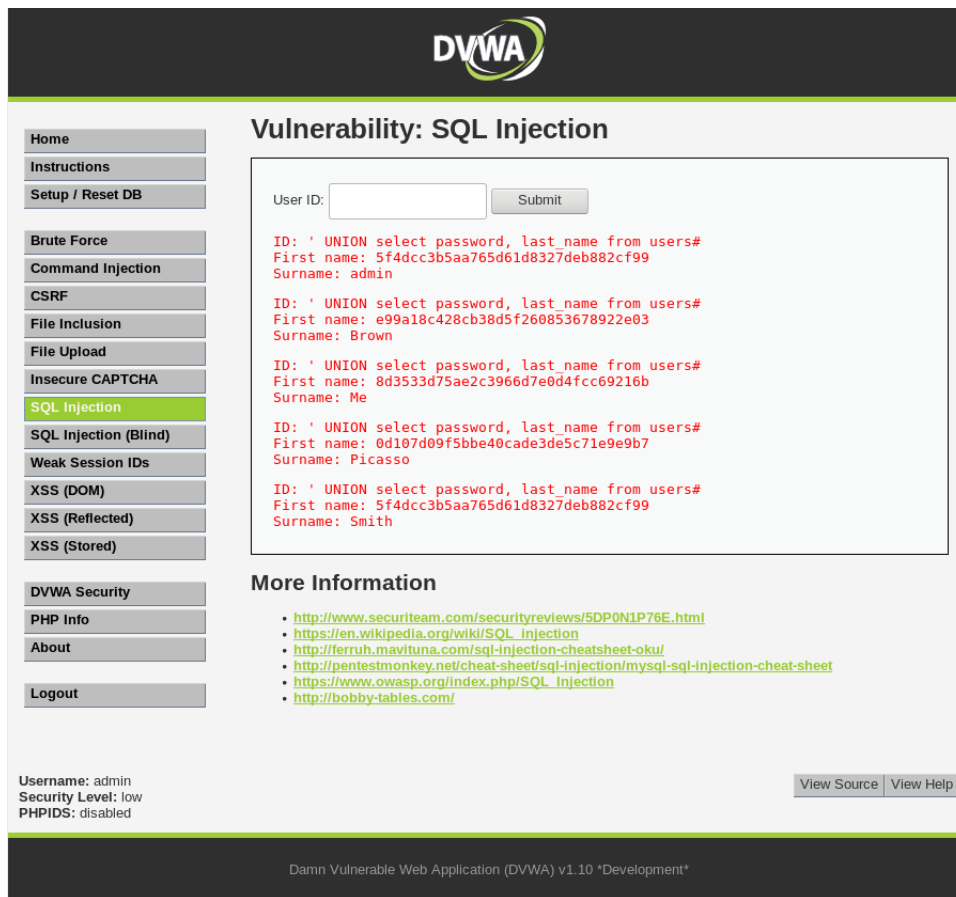


FIGURE 7 – Vol des mots de passe par l'injection SQLi sur site non protégé.

*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.*

*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 # ]
```

*Si l'on part à la recherche des méthodes et fonctions que l'on peut utiliser en PHP afin de se protéger de certains caractères, on peut croiser le chemin de la fonction PHP `mysql_real_escape_string`. Celle-ci est souvent utilisée pour protéger les requêtes SQL intégrant des paramètres envoyés par l'utilisateur. Néanmoins nous allons voir que cette fonction peut être déjouée, c'est d'ailleurs le but dans ce niveau DVWA.*

*Si l'on schématise, il faut donc faire passer des caractères comme des guillemets ou des apostrophes, sans inscrire dans l'URL les caractères en question. Un joyeux défi ! Pour ceux qui ont l'habitude de travailler sur les technos web, vous avez peut être déjà entendu parlé de l'encodage, et plus spécifiquement de l'encodage HTTP ou "HTML URL Encoding". Pour ceux qui ne voient pas de quoi il s'agit, je vous conseille de faire un tour sur cette page : [http://www.w3schools.com/tags/ref\\_urlencode.asp](http://www.w3schools.com/tags/ref_urlencode.asp)*

*L'HTML URL Encoding permet entre autre de faire passer au travers une URL des caractères spéciaux ou des lettres avec accent qui pourraient ne pas être interprétés correctement par les navigateurs ou les serveurs web. Ainsi, l'utilisation d'un encodage basé sur les caractères ASCII est utilisé (notamment la valeur "Hex" des lettres : <http://table-ascii.com/>)*

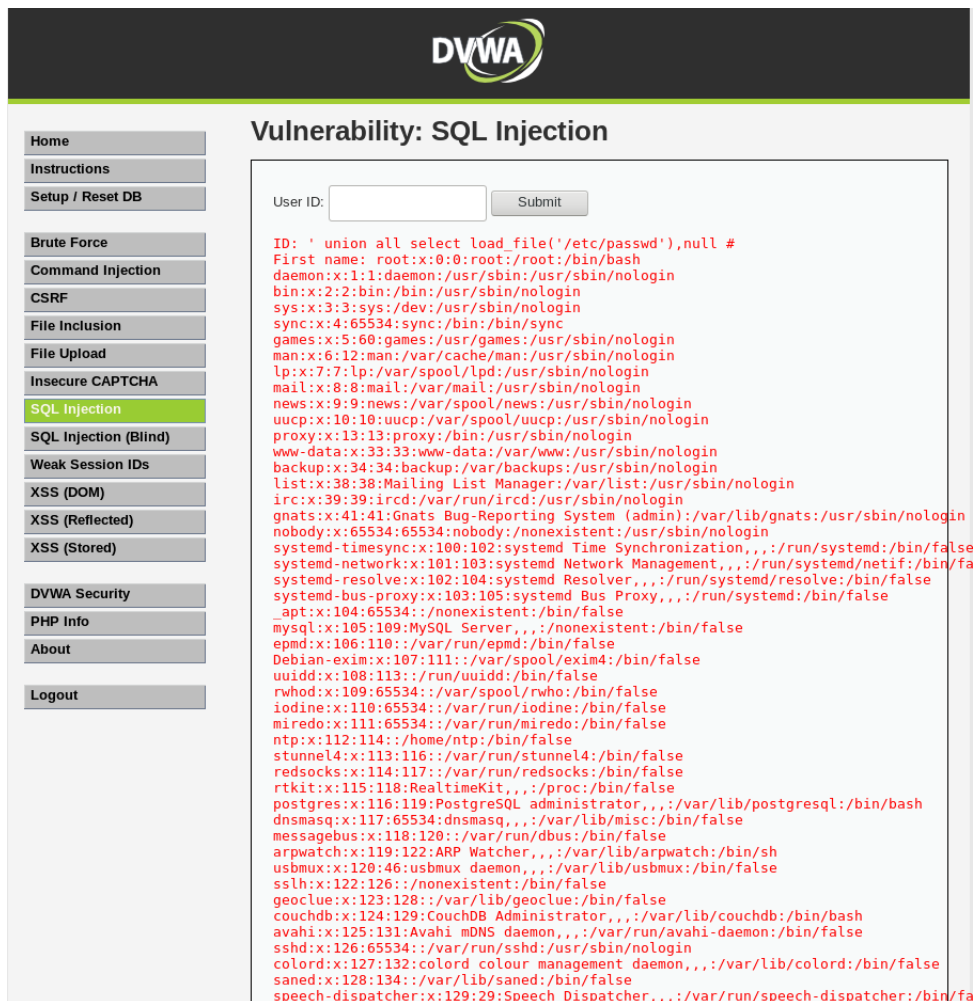


FIGURE 8 – Récupération du fichier /etc/passwd via la commande load\_file :

En utilisant l'HTTP URL Encoding, un espace devient un %20 dans l'URL, un "!" devient %21. Cela nous permet donc ici de faire passer une guillemet ou une apostrophe de façon encodée.

### 8.2.2 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
```

## 8.3 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,
- 👉 Restreindre au minimum les privilèges des comptes utilisés,
- 👉 Supprimer les comptes utilisateurs non utilisés, notamment les comptes par défaut,
- 👉 Utiliser un firewall Applicatif de type mod\_security
- 👉 Désactiver l'option Load\_File.

Plus spécifiquement, il est possible d'utiliser des "requêtes préparées" comme dans l'exemple en PHP tiré de DVWA - level "impossible"; le code et les données sont alors entièrement séparées. Une référence web utile se trouve sur le site [requetes-preparees](#).

```
<?php
```

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

    // 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 id=:id' );
        $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();

?>
```

## 9 Injection SQL aveugle

### 9.1 Description

*Les injections SQL aveugles, ou "blind SQL" en anglais, qu'on peut nommer BSQLi, 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 renvoie les erreurs SQL :*

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

### 9.2 Exploitation

### 9.3 Contre-mesure

## 10 Attaques XSS

*Reflected XSS*

### 10.1 Description

*désignées au choix par les acronymes CSS ou XSS et qui seront Un site web qui fournit d'une part un service de recueil d'informations via des formulaires et d'autre part un service de publication sur le site de ces mêmes informations*

### 10.2 Exploitation

### 10.3 Contre-mesure

## 11 Attaques XSS enregistrées

### 11.1 Description

### 11.2 Exploitation

### 11.3 Contre-mesure



## 12 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** et deux principes généraux : toujours **filtrer les saisies des utilisateurs**, par exemple en utilisant les expressions rationnelles ou une API et penser à **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.*

*De façon plus générale, que ce soit dans le cadre du développement web ou dans tout autre langage, 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 théorique. C'est pourquoi, il est préférable d'utiliser des API, bibliothèques et framework web éprouvés et patchés.*

## Bibliographie

- [ACI15] ACISSI. Sécurité informatique - Ethical Hacking. *ENI Editions*, 2015. <http://www.acissi.net/>.
- [Dor15] Mickael Dorigny. Blog sur la sécurité de l'information et le pentest. 2015. <https://www.information-security.fr/>.
- [GA10] Philippe Oechslin Gildas Avoine, Pascal Junod. Sécurité informatique 2ème édition. *Vuibert*, 2010.



*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.*

