

## TP Cyber DVWA

### Contexte :

A partir du lien <http://51.8.81.171:1664>, choisir 10 vulnérabilités, est rendre un livrable avec une description, comment utiliser cette faille, comment se protéger contre l'attaque puis montrer une exploitation sur le site web pour chaque vulnérabilité.

### Sommaire :

- 1) Attaque Brute Force
- 2) Attaque XSS (DOM)
- 3) Attaque XSS (Reflected)
- 4) Attaque XSS (Stored)
- 5) Attaque SQL Injection
- 6) Attaque Cross Site Request Forgery (CSRF)

## 1) Attaque Brute Force

### Description

Une attaque par force brute peut se manifester de différentes manières, mais consiste principalement en ce qu'un attaquant configure des valeurs prédéterminées, faire des requêtes à un serveur en utilisant ces valeurs, puis analyser la réponse. Par souci d'efficacité, un attaquant peut utiliser un dictionnaire attaque (avec ou sans mutations) ou une attaque par force brute traditionnelle (avec des classes de caractères données, par exemple : alphanumérique, spécial, casse (insensible). Compte tenu d'une méthode donnée, du nombre d'essais, de l'efficacité du système qui mène l'attaque et l'efficacité estimée du système attaqué, l'attaquant est capable de calculer approximativement combien de temps il faudra pour soumettre toutes les valeurs prédéterminées choisies.

### Outils offensifs

Une application Web peut être attaquée par force brute en prenant une liste de mots de pages connues, par exemple à partir d'un système de gestion de contenu populaire, et demander simplement chaque page connue puis analyser la réponse HTTP code pour déterminer si la page existe sur le serveur cible.

**DirBuster** est un outil qui fait exactement cela.

Les autres outils pour ce type d'attaque sont les suivants : **dirb**, **WebRoot**, **Burp Suite**

**dirb** est capable de : paramétrage des cookies - ajout de n'importe quel en-tête HTTP - en utilisant PROXY - objets mutants qui ont été trouvés - test des connexions http(s) - recherche de catalogues ou fichiers à l'aide de dictionnaires et modèles définis - et bien plus encore

### Outils défensifs

Détecteur d'attaque Php-Brute-Force.

Détectez vos serveurs Web analysés par des outils de force brute tels que **Wfuzz**, **OWASP DirBuster** et des scanners de vulnérabilités tels que **Nessus**, **Nikto**, **Acunetix**, etc. Cela vous aide à identifier rapidement les sondes probables en cas de mauvais acteurs qui veulent creuser d'éventuelles failles de sécurité.

### Démonstration DVWA

#### Vulnerability: Brute Force

##### Login

Username:

Password:

Login

## Code source low level DVWA

```
<?php

if( isset( $_GET[ 'Login' ] ) ) {
    // Get username
    $user = $_GET[ 'username' ];

    // Get password
    $pass = $_GET[ 'password' ];
    $pass = md5( $pass );

    // Check the database
    $query = "SELECT * FROM `users` WHERE user = '$user' AND password = '$pass'";

    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die(
    '<pre>' . ((is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mysqli_res =
mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );

    if( $result && mysqli_num_rows( $result ) == 1 ) {
        // Get users details
        $row = mysqli_fetch_assoc( $result );
        $avatar = $row["avatar"];

        // Login successful
        echo "<p>Welcome to the password protected area {$user}</p>";
        echo "<img src=\"{$avatar}\" />";
    }
    else {
        // Login failed
        echo "<pre><br />Username and/or password incorrect.</pre>";
    }

    ((is_null($__mysqli_res = mysqli_close($GLOBALS["__mysqli_ston"]))) ?
false : $__mysqli_res);
}

?>
```

## 2) Attaque XSS (DOM)

### Description

Basé sur DOM XSS (ou comme on l'appelle dans certains textes, "type-0 XSS") est une attaque XSS dans laquelle la charge utile de l'attaque est exécutée suite à la modification de « l'environnement » DOM dans le navigateur de la victime utilisé par le script côté client d'origine, de sorte que le code côté client se déroule de manière « inattendue ». Autrement dit, la page elle-même (le HTTP c'est-à-dire la réponse) ne change pas, mais le code côté client contenu dans la page s'exécute différemment en raison des modifications malveillantes qui se sont produits dans l'environnement DOM.

Cela contraste avec les autres attaques XSS (stockées ou réfléchi), dans lequel la charge utile de l'attaque est placée dans la page de réponse (en raison d'un côté serveur défaut).

### Outils offensifs

Si vous êtes un utilisateur de Burp, vous pouvez utiliser DOM Invader. C'est une extension préinstallée dans le navigateur intégré de Burp. Elle rend la détection des DOM XSS plus facile et plus rapide. Il y a deux fonctionnalités majeures :

- \*La vision du DOM où l'on peut identifier toutes les sources contrôlables et les sinks de façon instantanée. Il est aussi très simple de retrouver où la charge utile est injectée dans le code côté client.

- \*La vision des Web Message (ou postMessage) qui permet de capturer, éditer et rejouer tous les web message qui passent par la page.

La première de ces deux fonctionnalités est la plus importante pour détecter une DOM-based XSS classique. L'extension est capable d'injecter une chaîne de caractère défini par l'utilisateur dans toutes les sources possibles ou seulement dans l'URL. Elle va ensuite signaler les sources et les sinks où cette chaîne de caractère se retrouve. Le plus de l'extension est que les caractères « ' « <> » sont ajoutés à la chaîne injectée pour savoir s'ils sont échappés ou encodés. Le second grand avantage de l'extension est que l'on a accès à la stack trace dans la console du navigateur.

```
-- DOM Invader: Logging stack trace VM47:1
  at Object.Dybod (<anonymous>:2:107949) VM47:1
  at _0xee80c9 (<anonymous>:2:305000)
  at Object.AVwra (<anonymous>:2:97369)
  at Object.YufrL (<anonymous>:2:247056)
  at _0x4d5221.<computed> (<anonymous>:2:247299)
  at window.onload (https://portswigger-labs.net/dom-invader/testcases/augmented-dom-eval/?x=
duxkscqe1%5C%3Cimg%20src=x%20onerror=alert(1)%3E%27%22%3A:12:13)
>
```

## Outils défensifs

1. Éviter la réécriture, la redirection ou la réécriture de documents côté client d'autres actions sensibles, en utilisant les données côté client. La plupart de ces effets peuvent être obtenus en utilisant des pages dynamiques (côté serveur).

2. Analyser et renforcer le côté client (Javascript) code. Référence aux objets DOM pouvant être influencés par l'utilisateur (attaquant) doivent être inspectés, y compris (mais sans s'y limiter) :

document.URL

document.URLUnencoded

document.emplacement (et plusieurs de ses propriétés)

document.référent

fenêtre.emplacement (et plusieurs de ses propriétés)

## Démonstration DVWA

### Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

English ▼ Select

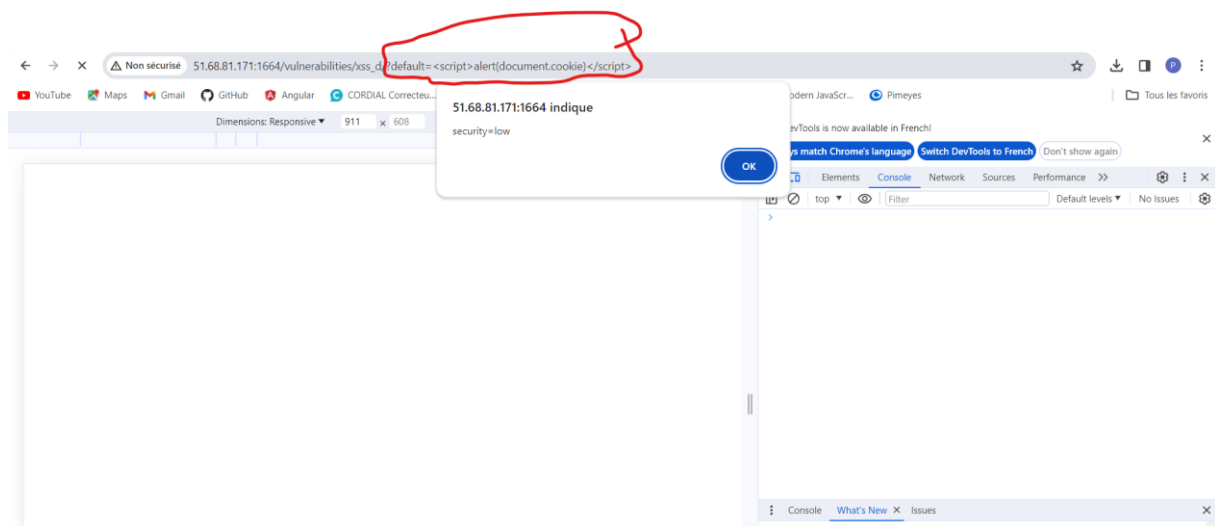
## Code source low level DVWA

```
<?php
```

```
# No protections, anything goes
```

```
?>
```

Au niveau Low Security il suffit d'insérer directement une balise `<script>` directement dans l'URL car la page est dynamiquement modifiée coté client sans aucune protection !



### 3) Attaque XSS (Reflected)

#### Description

Les reflected XSS sont les XSS où l'injection de code se situe généralement dans les paramètres d'une requête HTTP. L'attaquant va donc devoir forger un lien URL spécifique puis faire en sorte que la victime utilise le lien afin que le code injecté s'exécute dans son navigateur.

**Exemple** : `http://example.com/index.php?user=<script>alert(document.cookie)</script>`

Lorsque la victime cliquera sur le lien, le code JavaScript s'exécutera sur son navigateur et affichera une boîte de dialogue contenant ses cookies.

#### Outils offensifs

#### Outils défensifs

Les XSS sont de la famille des « Injection de code ». Comme toutes les vulnérabilités de cette famille, celles-ci sont dues à une absence de vérification des entrées utilisateurs.

Chaque entrée utilisateur doit systématiquement être vérifiée et le cas échéant échappée avant d'être insérée dans une page HTML. Cela vaut pour les paramètres venant de formulaires mais aussi pour les paramètres contenus dans les potentielles URL de l'application web. De nombreuses bibliothèques sont utilisables pour échapper les entrées utilisateurs en fonction de la technologie utilisée par l'application web.

En complément à cela, il peut être intéressant de mettre en place un pare-feu applicatif ou WAF. Le rôle de ce type de pare-feu est de détecter les tentatives d'exploitation de XSS sur l'application web et de les bloquer avant que l'injection de code ne soit exécutée sur le navigateur de la victime.

De plus, pour éviter que l'exploitation d'une XSS permette de voler le cookie des utilisateurs, il est préconisé de mettre le drapeau « HTTPOnly » sur les cookies de session. Cela empêchera l'accès aux cookies via le langage JavaScript.

Enfin, il est possible de paramétrer le serveur web de l'application pour retourner certains entêtes de protection contre les XSS :

**X-XSS-Protection** : active le filtre anti-XSS de certains navigateurs récents.

**Content Security Policy ou CSP** : le CSP définit les sources (images, fichiers CSS ou JavaScript) d'une application web autorisées à être chargées sur le navigateur des utilisateurs. Un CSP bien configuré empêchera un potentiel attaquant de charger des fichiers JavaScript externes par le navigateur des utilisateurs.

#### Démonstration DVWA

### Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

# Adrien Pago BTS SIO

## Code source low level DVWA

```
<?php

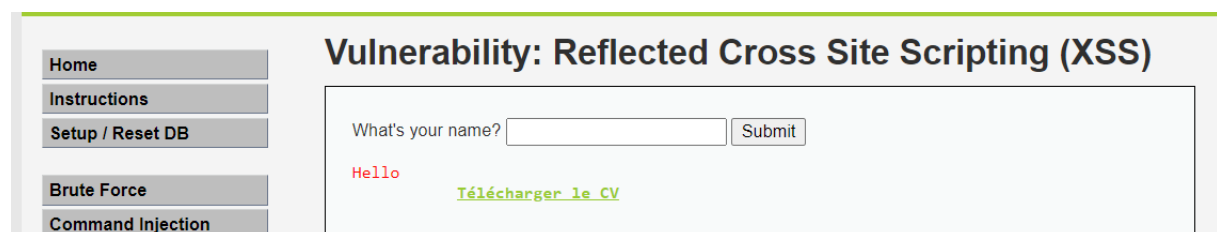
header ("X-XSS-Protection: 0");

// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}

?>
```

Pour cette attaque j'ai simplement introduit une <div> contenant un bouton qui charge un téléchargement sur le pc de la personne qui clique sur le bouton. Avec cette technique je peux donc introduire un virus à la place d'un simple pdf comme dans l'exemple

```
<div class="cv-btn">
    <a href="/Assets/CV Adrien PAGO.pdf" download="Adrien Pago BTS SIO.pdf">Télécharger le CV</a>
</div>
```



## 4) Attaque XSS (Stored)

### Description

Contrairement à la reflected XSS, la stored XSS ne nécessite pas une interaction forte de la part de la victime comme cliquer sur un lien. En effet, ici l'injection de code est directement stockée en base de données. Ainsi, lorsque la victime va charger la page vulnérable contenant l'injection de code, celle-ci va automatiquement s'exécuter sans aucune autre intervention de sa part.

Par exemple, prenons le cas d'un site de blog disposant d'un espace de commentaire sous chacun des articles. Si l'espace de commentaires permet d'exploiter une vulnérabilité de type stored XSS, l'attaquant va pouvoir injecter du code JavaScript dans un commentaire qu'il aura rédigé. Ce code JavaScript qui s'exécutera pour chacun des utilisateurs accédant à l'article. En effet, le commentaire malveillant étant stocké en base de données, aucune interaction avec la victime n'est nécessaire et le pirate aura juste à attendre que de nombreux cookies de session arrivent jusqu'à lui.

### Outils offensifs

### Outils défensifs

### Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text"/>
Message *	<input type="text"/>
<input type="button" value="Sign Guestbook"/> <input type="button" value="Clear Guestbook"/>	

Code source low level DVWA

```
<?php

if( isset( $_POST[ 'btnSign' ] ) ) {
    // Get input
    $message = trim( $_POST[ 'mtxMessage' ] );
    $name     = trim( $_POST[ 'txtName' ] );

    // Sanitize message input
    $message = stripslashes( $message );
    $message = ((isset($GLOBALS["__mysqli_ston"]) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $message ) :
(trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : "");

    // Sanitize name input
    $name = ((isset($GLOBALS["__mysqli_ston"]) &&
is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_real_escape_string($GLOBALS["__mysqli_ston"], $name ) :
(trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call!
This code does not work.", E_USER_ERROR)) ? "" : "");

    // Update database
    $query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message',
'$name' );";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die(
'<pre>' . ((is_object($GLOBALS["__mysqli_ston"])) ?
mysqli_error($GLOBALS["__mysqli_ston"]) : (($__mysqli_res =
mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );

    //mysqli_close();
}

?>
```



## Vulnerability: Stored Cross Site Scripting (XSS)

Name *	<input type="text" value="Hacker"/>
Message *	<input type="text" value="&lt;script&gt;alert(document.cookie)&lt;/script&gt;"/>
<input type="button" value="Sign Guestbook"/> <input type="button" value="Clear Guestbook"/>	



### Résumer des trois différentes attaque XSS (Dom, Stored, Reflected)

Les attaques XSS peuvent être classées en trois catégories principales : attaques XSS stockées, attaques XSS reflétées, attaques XSS basées sur le DOM.

#### Attaques XSS stockées (XSS persistant)

Un XSS stocké ou persistant est considéré comme l'attaque de cross-site scripting la plus dévastatrice. Elle se produit quand une page Web va stocker puis afficher un contenu envoyé par un pirate. Les points d'entrée pour les XSS stockés sont généralement des messages sur les forums, des commentaires sur des blogs, des profils utilisateurs et des champs de nom d'utilisateur. Un attaquant exploite généralement cette vulnérabilité en envoyant des charges utiles XSS sur des pages populaires ou en passant un lien à une victime, qui la redirigera vers une page contenant la charge utile XSS stockée. La victime visite la page et la charge utile est exécutée côté client par le navigateur Web de la victime.

#### Attaques XSS reflétées (XSS non persistant)

Les XSS reflétés représentent l'attaque de cross-site scripting la plus courante. Dans ce cas, la charge utile de l'attaquant doit faire partie de la demande envoyée par le serveur Web. Elle est ensuite renvoyée de manière à ce que la réponse HTTP inclue la charge utile de la demande HTTP. Les attaquants utilisent des liens malveillants, des emails de phishing et d'autres techniques d'ingénierie sociale pour inciter les victimes à effectuer une demande au serveur. La charge utile de l'attaque XSS reflétée est ensuite exécutée dans le navigateur de l'utilisateur.

Comme ce n'est pas une attaque persistante, le pirate doit délivrer la charge utile à chaque victime. Ces attaques sont souvent menées via les réseaux sociaux.

### Attaques XSS basées sur le DOM

Le XSS basé sur le DOM fait référence à une faille de cross-site scripting qui apparaît dans le DOM (Document Object Model) au lieu d'être dans une partie de l'HTML. Dans les attaques XSS reflétées et stockées, on peut voir la charge utile de la vulnérabilité dans la page de réponse, mais dans les attaques basées sur le DOM, le code source HTML et la réponse seront les mêmes. En somme, la charge utile ne se trouvera pas dans la réponse. Elle ne pourra être observée qu'au moment de l'exécution ou en examinant le DOM de la page.

Ce type d'attaque se produit généralement côté client et la charge utile malveillante n'est jamais envoyée au serveur. Cela rend la détection encore plus difficile pour les pare-feux d'applications Web (WAF) et les ingénieurs en sécurité qui analysent les journaux des serveurs, car ils ne voient jamais l'attaque. Les objets du DOM qui sont le plus souvent manipulés sont l'URL (document.URL), la partie d'ancrage de l'URL (location.hash) et le référent (document.referrer).

## 5) Attaque SQL injection

### Description

Une injection SQL attaque consiste en une insertion ou « injection » d'une requête SQL via les données d'entrée du client vers l'application. Un exploit d'injection SQL réussi peut lire des données sensibles à partir de la base de données, modifier les données de la base de données (Insérer/Mettre à jour/Supprimer), exécuter les opérations d'administration sur la base de données (telles que l'arrêt du SGBD), récupérer le contenu d'un fichier donné présent sur le système de fichiers du SGBD et dans certains cas, envoyez des commandes au système d'exploitation. Injection SQL Les attaques sont un type d'attaque par injection, dans lequel les commandes SQL sont injectés dans l'entrée du plan de données afin d'affecter l'exécution de commandes SQL prédéfinies.

### Outils offensifs

**Sqlmap** est un outil d'injection SQL open-source incroyablement efficace. **Sqlmap** « automatise le processus de détection et d'exploitation des failles d'injection SQL et de prise en charge des serveurs de bases de données », comme l'explique son site web. Sqlmap supporte toutes les cibles habituelles, y compris MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, Informix, HSQLDB et H2. Auparavant, les pirates devaient fabriquer eux-mêmes leur injection SQL. Aujourd'hui, **sqlmap** fait tout le travail à leur place.

Lien pour utilisation avancé de **Sqlmap**

<https://connect.ed-diamond.com/MISC/misc-062/utilisation-avancee-de-sqlmap#:~:text=Sqlmap%20est%20un%20outil%20open,h%C3%A9bergeant%20la%20base%20de%20donn%C3%A9es.>

## Outils défensifs

Lors du développement de votre site web ou de votre application web, vous pouvez intégrer des mesures de sécurité qui limitent votre exposition aux attaques par injection SQL. Par exemple, les mesures de sécurité suivantes sont la manière la plus efficace de prévenir les attaques par injection SQL :

**Utilisation d'instructions préparées et de requêtes paramétrées.** Lorsqu'un développeur utilise des requêtes paramétrées, il doit définir l'intégralité du code SQL, puis passer chaque paramètre. Il est donc impossible pour un attaquant de modifier l'objet de la requête ultérieurement.

**Utilisation de procédures stockées.** Lorsqu'un développeur utilise une procédure stockée, il définit des instructions SQL à l'aide des paramètres stockés dans la base de données et appelés depuis l'application. Cette technique, qui constitue une alternative à l'utilisation d'instructions préparées, est tout aussi efficace contre les attaques par injection SQL.

**Utilisation de la validation des saisies dans la liste d'autorisation.** Dans certains cas, les développeurs peuvent définir une valeur attendue, comme le nom autorisé d'un tableau ou d'une colonne. Cette validation permet d'empêcher l'ajout d'une saisie utilisateur non approuvée à une requête.

**Application de l'échappement aux saisies utilisateur avant leur intégration à une requête.** Cette technique consiste à appliquer l'échappement de caractères aux saisies utilisateur afin que celles-ci ne soient pas confondues avec le code SQL du développeur. Les autres méthodes décrites ici sont préférables à celle-ci dans la mesure où elles offrent une meilleure protection. Cependant, les entreprises recourent parfois à cette technique lorsqu'elles modernisent le code d'ancienne génération si la validation des saisies utilisateur s'avère trop onéreuse.

**Installer les versions des logiciels et les correctifs de sécurité les plus récents dès leur publication**

**Octroyer les privilèges minimums requis aux comptes qui se connectent à la base de données SQL**

**Ne pas utiliser de comptes de base de données partagés pour différents sites web et applications web**

**Configurer une procédure de signalement des erreurs plutôt que d'envoyer des messages d'erreur au navigateur web client**

En plus d'intégrer des mesures de sécurité lors du développement et de la modernisation d'un site web ou d'une application web, vous pouvez utiliser un logiciel pour surveiller activement les menaces de sécurité. Des outils tels que les logiciels de détection et d'intervention managées et de **Threat Hunting** managé peuvent vous aider à mettre en place des mesures de prévention efficaces. Ces outils peuvent également vous aider à bloquer les menaces en cas d'attaque.

## Démonstration DVWA

### Vulnerability: SQL Injection

User ID:

## Code source low level DVWA

```
<?php
if( isset( $_REQUEST[ 'Submit' ] ) ) {
    // Get input
    $id = $_REQUEST[ 'id' ];

    switch ( $_DVWA[ 'SQLI_DB' ] ) {
        case MYSQL:
            // Check database
            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
            $result = mysqli_query($GLOBALS["__mysqli_ston"], $query ) or die( '<pre>' .
((is_object($GLOBALS["__mysqli_ston"])) ? mysqli_error($GLOBALS["__mysqli_ston"]) :
(($__mysqli_res = mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // 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>";
            }

            mysqli_close($GLOBALS["__mysqli_ston"]);
            break;
        case SQLITE:
            global $sqlite_db_connection;

            $sqlite_db_connection = new SQLite3($_DVWA['SQLITE_DB']);
            $sqlite_db_connection->enableExceptions(true);

            $query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
            #print $query;
            try {
                $results = $sqlite_db_connection->query($query);
            } catch (Exception $e) {
                echo 'Caught exception: ' . $e->getMessage();
                exit();
            }

            if ($results) {
                while ($row = $results->fetchArray()) {
                    // 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>";
                }
            } else {
                echo "Error in fetch " . $sqlite_db->lastErrorMsg();
            }
            break;
    }
}

?>
```

On remarque sur le code une protection contre l'injection de caractère sql comme «-- » pour mettre en commentaire du code en sql.

J'ai essayé la fameuse formule 'or 1=1' —e pour test d'injecter du sql et j'ai eu la page suivante :


```
Fatal error: Uncaught mysqli_sql_exception: 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 in
/var/www/html/vulnerabilities/sqli/source/low.php:11 Stack trace: #0 /var/www/html/vulnerabilities/sqli/source/low.php(11): mysqli_query(Object(mysqli), 'SELECT first_na...') #1 /var/www/html/vulnerabilities/sqli/index.php(34):
require_once('/var/www/html/v...') #2 {main} thrown in /var/www/html/vulnerabilities/sqli/source/low.php on line 11
```



[Home](#)  
[Instructions](#)  
[Setup / Reset DB](#)  
  
[Brute Force](#)  
[Command Injection](#)  
[CSRF](#)

### Vulnerability: SQL Injection

User ID:    
  
ID: 1  
First name: admin  
Surname: admin



[Home](#)  
[Instructions](#)  
[Setup / Reset DB](#)  
  
[Brute Force](#)  
[Command Injection](#)

### Vulnerability: SQL Injection (Blind)

User ID:    
  
User ID exists in the database.

## [Attaque Cross Site Request Forgery \(CSRF\)](#)

### Description

CSRF (Cross-Site Request Forgery) est une attaque qui usurpe l'identité d'un utilisateur de confiance et envoie des commandes non désirées sur un site web. Cela peut être réalisé, par exemple, en ajoutant des paramètres malveillants dans une URL associée à un lien qui prétend aller quelque part ailleurs.

### Outils offensifs

### Outils défensifs

Pour se protéger contre les attaques CSRF, une mesure de sécurité courante consiste en l'implémentation de jetons CSRF.

Les jetons CSRF sont des valeurs uniques générées aléatoirement côté serveur et envoyées au client. Ces valeurs étant uniques pour chaque requête, elles permettent de renforcer la sécurité d'une application en rendant difficile (voire impossible) pour un attaquant de les deviner et donc d'exploiter une vulnérabilité CSRF.

# Adrien Pago BTS SIO

Par ailleurs, certains frameworks, tels que **Django** (Python) et **Laravel** (PHP), intègrent des middlewares activés par défaut assurant une protection contre les attaques CSRF.

## Démonstration DVWA

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

Change your admin password:

Test Credentials

New password:

Confirm new password:

Change

## Code source low level DVWA

```
<?php
```

```
if( isset( $_GET[ 'Change' ] ) ) {
    // Get input
    $pass_new = $_GET[ 'password_new' ];
    $pass_conf = $_GET[ 'password_conf' ];

    // Do the passwords match?
    if( $pass_new == $pass_conf ) {
        // They do!
        $pass_new = ((isset($GLOBALS["__mysqli_ston"]) &&
is_object($GLOBALS["__mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["__mysqli_ston"],
$pass_new ) : ((trigger_error("[MySQLConverterToo] Fix the mysql_escape_string() call! This
code does not work.", E_USER_ERROR)) ? "" : ""));
        $pass_new = md5( $pass_new );

        // Update the database
        $current_user = dvwaCurrentUser();
        $insert = "UPDATE `users` SET password = '$pass_new' WHERE user = '" . $current_user .
        "'";
        $result = mysqli_query($GLOBALS["__mysqli_ston"], $insert ) or die( '<pre>' .
        ((is_object($GLOBALS["__mysqli_ston"])) ? mysqli_error($GLOBALS["__mysqli_ston"]) :
        (($__mysqli_res = mysqli_connect_error()) ? $__mysqli_res : false)) . '</pre>' );

        // Feedback for the user
        echo "<pre>Password Changed.</pre>";
    }
    else {
        // Issue with passwords matching
        echo "<pre>Passwords did not match.</pre>";
    }
}

((is_null($__mysqli_res = mysqli_close($GLOBALS["__mysqli_ston"]))) ? false :
$__mysqli_res);
?>
```