

# Capture the Flag : jeu éducatif pour apprendre les bases du test d'intrusion web



<http://oc3.lddr.ch/Baurand/>

*Par : Yohann Baurand (3MG04)*

*Mentor : Christophe Bruchez*

*Avis au lecteur : Vous trouverez en bas de chaque niveau un lien vers une page de documentation. Il est fortement conseillé de se mettre au défi avec la documentation disponible sur le site avant de lire les réponses dans le TM. Il est possible de télécharger le code du site à partir du GitHub <https://github.com/Comete12/LDDR-CTF>. Pour un affichage optimal et éviter tout problème de compatibilité, le site doit être ouvert à partir de Google Chrome sur un écran 1080p. Veillez à vous créer un compte avant toute chose, et utilisez un mot de passe unique. Il se peut qu'en essayant de trouver une réponse, des erreurs inattendues surviennent. Si c'est le cas, essayez d'abord de recharger la page, sinon effacez le cache et relancez votre navigateur. Si par erreur vous modifiez une table au point de corrompre le site et le rendre inutilisable (ce qui n'est pas censé arriver à moins de vraiment le vouloir), contactez-moi par mail pour que je puisse restaurer une sauvegarde de la base de données.*

*[Yohann.Baurand@s2.rpn.ch](mailto:Yohann.Baurand@s2.rpn.ch)*

***Merci de ne pas appliquer ce qu'enseigne ce TM sur des sites non prévus à cet effet ou à des fins illégales.***

## Table des matières

Table des matières .....	3
Introduction.....	1
Cybersécurité.....	1
CTF.....	1
Origines .....	1
Actualité .....	1
Présentation du TM .....	2
Présentation du site .....	3
Structure.....	3
Inscription / connexion.....	8
Liste des niveaux .....	11
Attribution des points.....	12
Tableau des scores .....	13
Niveaux et failles .....	15
Niveau 1 : Source Code .....	15
Niveau 2 : Source Code .....	15
Niveau 3 : Obfuscation .....	16
Niveau 4 : Injection SQL .....	16
Niveau 5 : Injection SQL .....	17
Niveau 6 : Injection SQL .....	17
Niveau 7 : Injection XSS.....	18
Niveau 8 : Injection XSS.....	19
Niveau 9 : Injection XSS.....	19
Comment se protéger de ces failles ? .....	20
Conclusion.....	21
Sources.....	22

# Introduction

## Cybersécurité

La cybersécurité désigne tout ce qui concerne la défense des cyberattaques. Une cyberattaque peut avoir lieu sur un site web, une application, un logiciel, etc. et a comme principe de détourner certaines fonctionnalités pour les utiliser à un autre but, par exemple le vol de données, l'obtention de privilèges, l'usurpation d'identité, etc. L'objectif de la cybersécurité est de sécuriser ces systèmes pour empêcher une personne mal intentionnée d'en détourner le fonctionnement. Il existe de nombreuses manières de sécuriser un système, mais celle qui est probablement la plus connue est le pentesting, ou test d'intrusion en français. Cette technique consiste à essayer de pirater un site web, pour pouvoir repérer les différentes vulnérabilités et les corriger. On utilise parfois le terme « hacker-éthique » pour désigner une personne effectuant ce métier.

## CTF

Un CTF (Capture The Flag) est un exercice (ou un jeu pour certains) sous forme de compétition visant à exploiter une vulnérabilité présente dans un logiciel ou un site web. La faille n'est pas présente par erreur mais connue et codée par les développeurs du CTF, et est parfois très complexe à trouver et à exploiter selon le niveau de difficulté. Une fois la faille trouvée, le joueur recevra une clé (ou le « Flag ») sous forme d'une chaîne de caractères qui lui permettra d'obtenir des points. Ce jeu peut devenir extrêmement difficile et demande souvent de grandes connaissances en cybersécurité.

## Origines

Le concept du CTF est né en 1996 à Las Vegas lors de la Defcon annuelle, une des plus grandes conventions sur le hacking au monde. Cet événement a lieu chaque année depuis 1993 et regroupe des experts en cybersécurité, hackers, étudiants et journalistes du monde entier.

## Actualité

Depuis, le CTF a gagné énormément en popularité auprès des personnes s'intéressant au hacking et à la cybersécurité. Il existe notamment de nombreux sites proposant des CTF en ligne accessibles à tout le monde. De plus en plus de conventions et d'événements autour des CTF ont également vu le jour, attirant toujours plus de joueurs. Des équipes constituées d'experts s'affrontent régulièrement lors de grandes compétitions, on peut presque parler d'eSport.

## **Présentation du TM**

L'objectif de ce travail de maturité est la conception et le développement d'un CTF de petite envergure sous forme de site web, accessible à toutes les personnes souhaitant s'y essayer, et guidant les joueurs pas-à-pas pour permettre aux personnes n'ayant pas de connaissances dans le domaine d'en acquérir et de pouvoir les mettre en pratique.

## Présentation du site

Le site est développé en HTML, CSS, Java Script, PHP et SQL, entièrement en anglais pour éviter toute barrière de langue. Il sera divisé en 3 parties principales : Un espace compte permettant de se créer un compte, se connecter et connaître son nombre de points, une liste des différents niveaux avec leur difficulté et le nombre de points qu'ils rapportent, permettant au joueur de se rendre sur le niveau sélectionné, et un tableau des scores affichant les joueurs avec le plus de points par ordre décroissant.

### Structure

Comme tout site utilisant du HTML, les premières lignes de code définissent le type de document et d'encodage, ainsi que la langue et le nom du site web. La balise `link` sert à lier une source, dans ce cas le CSS se trouvant dans le fichier `style.css`.


```
<link rel="stylesheet" type="text/css" href="style.css"/>
```

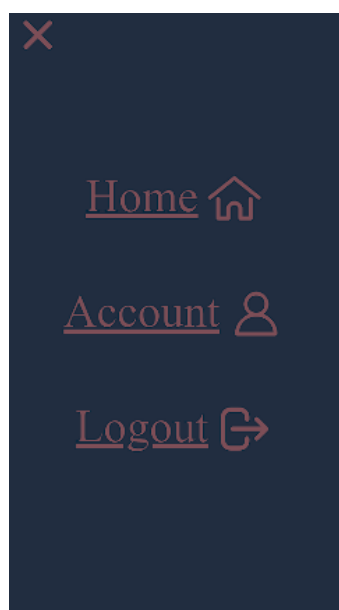
Le contenu du site web se trouve dans la balise `body`, et est séparé en plusieurs catégories, ce qui permet de leur appliquer un CSS différent en indiquant leur classe (par exemple : `wrapper`) ou le type de balise (par exemple : `nav`). Ainsi, en parlant de la div `wrapper`, on entend la balise `div` de classe `wrapper`.

```
<body>
  <div class="wrapper">
    <div class="off-screen-menu">...</div>
    <nav>...</nav>
    <div class="main">...</div>
    <footer>...</footer>
  </body>
```

La div `wrapper` sert à donner à l'ensemble des grandes catégories du site (`nav`, `main` et `footer`) la propriété de se répartir en colonne de haut en bas, avec l'attribut CSS `flex`.

```
.wrapper {  
    display : flex ;  
    flex-direction : column ;  
    . . .  
}
```

La div `off-screen-menu` contient le code du volet latéral accessible en appuyant sur la `burger-icon`  dans le coin supérieur gauche. « Off-screen » signifie hors écran car le menu se trouve à l'extérieur de la page et n'est donc pas visible tant qu'il n'est pas ouvert. Le contenu du menu est encadré par une balise de type `ul`, qui organise ses éléments (`li`) sous forme de liste. Ici, le premier est un simple lien vers la page d'accueil qui facilite la navigation sur le site. Ensuite, du code PHP est utilisé pour vérifier si un utilisateur est connecté avec la condition `if(isset($_SESSION['username']))`. Si la condition est vérifiée, le menu contiendra deux éléments supplémentaires : un lien vers une page `account` pour supprimer son compte, et un lien permettant de se déconnecter. Si elle n'est pas vérifiée, le menu contiendra un lien vers une page de connexion.



```

<ul>

    <li><a href="index.php">...</li>

    . . .

    If(isset($_SESSION['username'])) {

        echo "<li><a href='account.php'>...<a href='logout.php'>...</li>"

    } else {

        echo "<li><a href='login.php'>...</li>"

    }

</ul>

```

La balise `script` contient le code JavaScript nécessaire pour activer la balise `off-screen-menu` lors du clic sur la `burger-icon`. Le CSS définit la propriété `left` à 0 uniquement lorsque la balise est active, et permet de déplacer le menu dans la fenêtre pour qu'il devienne visible. L'attribut CSS `transition` permet comme son nom l'indique d'ajouter une transition animée pour un meilleur résultat.

```

<script>

    . . .

    burgerIcon.addEventListener("click", ( ) => {

        burgerIcon.classList.toggle("active");

        offScreenMenu.classList.toggle("active");

    });

</script>

```

```

.off-screen-menu {

    left : -350px;

    transition : 0.3s ease;

    . . .

}

.off-screen-menu.active {

    left : 0;

}

```



La div `nav` se trouvant en haut de la page contient la `burger-icon`, et le score de l'utilisateur ou un lien vers une page de connexion dans le cas où l'utilisateur n'est pas connecté. De nouveau, la condition `if(isset($_SESSION['username']))` est utilisée pour vérifier si un utilisateur est connecté, et si c'est le cas le nombre de points contenu dans `$_SESSION['points']` est affiché.

```
<nav>

    <span class="burger-icon">

        <span></span>

        <span></span>

        <span></span>

    </span>

    . . .

    If(isset($_SESSION['username'])) {


        echo "<li>... . $_SESSION['points'] . </li>"

    } else {

        echo "<li><a href='login.php'>...</li>"

    }

</nav>
```

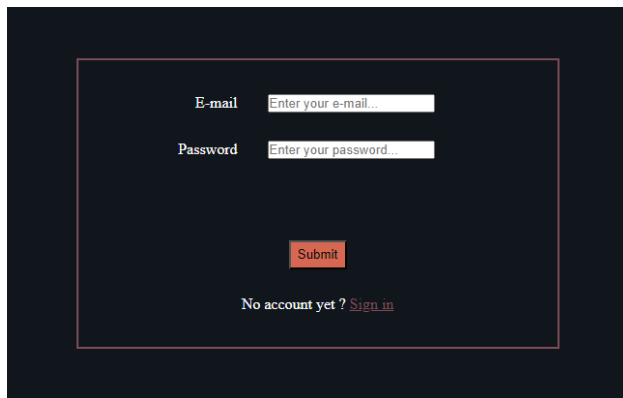
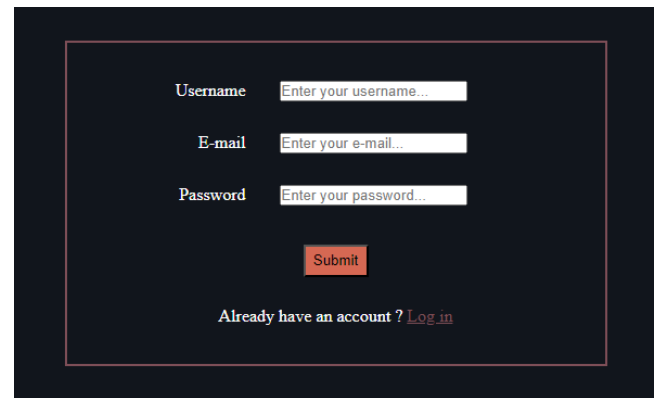
La balise `span` a plus ou moins la même utilité qu'une balise `div`, ici on en utilise trois qui seront mises en style grâce à du CSS pour créer les trois lignes de la `burger-icon`. Pour l'animation, lorsque la `burger-icon` est activée grâce au code JavaScript mentionné plus haut, on donne une opacité de 0 à la `span` du milieu pour la faire disparaître, on change la propriété `top` de la première ( `:nth-child(1)` ) et dernière ( `:nth-child(3)` ) `span` à 50% (qui était initialement à 25% et 75%), et on leur donne une rotation de 45 et -45 degrés. De cette manière, la `burger-icon` se transforme en une croix  avec une animation fluide grâce à l'attribut `transition`.

```
burger-icon span {  
    transition: 0.3s ease;  
    . . .  
}  
  
.burger-icon.active span :nth-child(2) {  
    opacity: 0;  
}  
  
.burger-icon span :nth-child(1) {  
    top: 25%;  
}  
  
.burger-icon.active span :nth-child(1) {  
    top: 50%;  
    transform: ... rotate(45deg);  
}
```

Finalement, le `footer` ou pied de page en français, est un espace en bas de page généralement utilisé pour y afficher des liens vers différents réseaux ou vers les conditions générales. Ici il sera possible d’y télécharger le code du site.

## Inscription / connexion

Pour la partie inscription / connexion, on envoie l'utilisateur sur une page dédiée (`login.php`) lorsqu'il clique sur le lien login dans le volet latéral ou dans le coin supérieur droit de la page. Ici, l'utilisateur a le choix entre se connecter s'il a déjà un compte, ou aller sur une page d'inscription (`signin.php`).

A dark-themed form for signing in. It contains two input fields: 'E-mail' with placeholder text 'Enter your e-mail...' and 'Password' with placeholder text 'Enter your password...'. Below these fields is an orange 'Submit' button. At the bottom, there is a link that says 'No account yet ? [Sign in](#)'.A dark-themed form for logging in. It contains three input fields: 'Username' with placeholder text 'Enter your username...', 'E-mail' with placeholder text 'Enter your e-mail...', and 'Password' with placeholder text 'Enter your password...'. Below these fields is an orange 'Submit' button. At the bottom, there is a link that says 'Already have an account ? [Log in](#)'.

Pour l'inscription, on utilise un formulaire (`form`) avec 3 champs (`input`) qui demande de renseigner un nom d'utilisateur, un email et un mot de passe. L'input de type `submit` permet l'envoi du formulaire grâce à la méthode `POST` qui va envoyer les valeurs de chaque champ vers le code contenu dans `process.php`. Ici, on commence par se connecter à la base de données (BDD) qui va servir à stocker les informations de chaque utilisateur. On fournit les informations de connexion nécessaires et on essaie une connexion à la BDD avec `try`. On utilise `catch` pour intercepter et renvoyer toute erreur de connexion. Une fois la connexion vérifiée, on récupère les valeurs envoyées par le formulaire avec `extract` qui va associer chaque valeur à une variable du même nom. Par exemple, l'input du nom d'utilisateur à un attribut `name="username"`, le nom d'utilisateur sera donc stocké dans la variable du même nom `$username`. On vérifie ensuite que tous les champs sont bien remplis, et on insère les valeurs dans la table `users` avec la requête SQL `INSERT`. Pour rappel, une table est une manière de stocker des informations dans une BDD de façon organisée. La table contient cinq colonnes : `username`, `email`, `password`, `points` et `rank`. On associe chaque colonne à la variable contenant les informations de l'utilisateur, sauf pour les points qui sont définis à 0 et `rank` qui sera défini dans le code du `leaderboard`. Les informations de l'utilisateur sont maintenant stockées dans la BDD et seront utilisées pour qu'il puisse se connecter et conserver ses points.

```
<form method="POST" action="process.php">

    . . .

    <input name="username" ...>

    <input name="email" ...>

    <input name="password" ...>

    <input type="submit" ...>

    . . .

</form>
```

```
$servername="..."

. . .

try{

    $conn = new PDO("mysql:host=$servername; ...")

    . . .

}

catch(PDOException $e){

    echo "Erreur : ". $e->getMessage( );

    exit( );

}

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

    extract($_POST);

    . . .

    $req = $conn->prepare("INSERT INTO users VALUES
(:username, ...)

    $req->execute(["username" => $username, ...])

    . . .

}
```

Pour la connexion, le principe est le même : l'utilisateur remplit un formulaire contenant son email et son mot de passe qui est envoyé grâce à un `submit` et à la méthode `POST`, on effectue ensuite la connexion à la BDD comme pour l'inscription et on récupère les valeurs avec `extract`.

On vérifie que les champs `email` et `password` ne soient pas vides, et on recherche un email avec un mot de passe correspondant dans la BDD grâce à la requête SQL `SELECT`. Si une ligne qui correspond est trouvée (avec la condition `if ($rep)`), l'email, le nom d'utilisateur et le nombre de points sont stockés dans des variables `$_SESSION`, qui vont garder leurs valeurs entre les différentes pages tant que le navigateur n'est pas fermé ou que l'utilisateur ne se déconnecte pas grâce au lien vers `logout.php` (qui est tout simplement un `session_destroy()`). De ce fait, le navigateur pourra garder en mémoire quel utilisateur est connecté et utiliser ces informations pour créditer les points obtenus au bon utilisateur, et l'afficher sur le tableau des scores. L'utilisateur est ensuite redirigé vers la page de laquelle il vient ou vers la page d'accueil si aucune variable `$_SESSION['redirection']` (qui sera défini plus bas) n'existe.

```
$reqt = $conn->prepare("SELECT * FROM users WHERE email = :email AND
password= :password");

$req->execute(['email' => $email, 'password' => $password]);

$rep = $req->fetch();

if($rep){

    $points = $rep['points'];

    $_SESSION['points'] = $points;

    . . .

    if($_SESSION['redirection']){

        echo "<script>window.location.href = '" .
$_SESSION['redirection'] . "';</script>";



    } else{

        echo "<script>window.location.href = 'index.php';</script>";

    }

}
```

## Liste des niveaux

La liste des niveaux est un tableau (balise `table`) composé de balises `tr` (lignes), elles-mêmes composées de `td` (colonnes). Ici c'est donc un tableau 3x4 avec quatre `tr` contenant chacune trois `td`. Chaque balise `td` a un identifiant propre au niveau qu'elle contient, par exemple la `td` du niveau 1 aura l'attribut `id="lvl1"`. Toutes les cases du tableau ont un nom et numéro de niveau, un bouton `PLAY` pour y accéder, la difficulté et le nombre de points qu'il rapporte. Quelques lignes de code PHP permettent de vérifier si le niveau a déjà été complété, une requête SQL vérifie si une ligne dans la table `completed_level` avec l'email du joueur et le numéro du niveau existe. S'il y a une ligne correspondante, on change le bord inférieur de la `td` en vert et on change l'image de crâne par une autre image de crâne  avec une coche dessus. 

```
$level = 1;

$email = $_SESSION['email'];

$stmt = $conn->prepare("SELECT * FROM completed_levels WHERE email = :email
AND level = :level");

$stmt->execute(['email' => $email, 'level' => $level]);

if ($stmt->rowCount() == 0) {

    echo "<img src='skull.png'>";

}else{

    echo "<img
src='skull2.png'>...document.getElementById('lvl1').style='border-bottom:
green; '...";

}
```

Une fois sur la page du niveau, on vérifie tout d'abord comme pour la page d'accueil si un utilisateur est connecté pour afficher ses points dans la barre de navigation en haut de l'écran. Cette fois, si aucun utilisateur n'est connecté, à la place de simplement lui proposer un lien vers la page de login, on l'y redirige avec `window.location` en mettant le lien du niveau dans la variable `$_SESSION['redirection']`. De cette manière, l'utilisateur est obligé de se connecter avant de poursuivre et sera automatiquement redirigé vers le niveau une fois connecté.

```
$_SESSION['redirection']= 'lien_vers_le_niveau.php';

echo "<script>window.location.href='login.php';</script>"
```

## Attribution des points

Les points sont attribués à la fin de chaque niveau si la condition requise est validée, c'est-à-dire si le mot de passe correspond à celui dans la base de données comme pour la page de connexion. Ensuite, il faut vérifier si le niveau a déjà été complété par l'utilisateur comme dans la liste des niveaux. Si c'est le cas, les points sont mis à jour avec `UPDATE` dans la ligne où email correspond à l'email de l'utilisateur connecté, c'est-à-dire `$_SESSION['email']`.

```
$stmt=$conn->prepare("UPDATE users SET points = points + 5 WHERE email = :email")
```

Ensuite, on récupère le total des points de l'utilisateur avec un `SELECT` pour mettre à jour `$_SESSION['points']` et pouvoir afficher les points de l'utilisateur une fois le niveau réussi.

```
$stmt=$conn->prepare ("SELECT points FROM users WHERE email = :email")
$pointsResult = $stmt->fetch();
if ($pointsResult) {
    $_SESSION['points'] = $pointsResult['points'];
}
```

Et finalement on ajoute dans la table `completed_levels` une ligne avec l'email de l'utilisateur et le numéro du niveau avec `INSERT INTO`, pour éviter que l'utilisateur récupère plusieurs fois les points et pour pouvoir changer le CSS dans la liste des niveaux.

```
$stmt=$conn->prepare ("INSERT INTO completed_levels (email, level) VALUES
(:email, :level)")
```

## Tableau des scores

Le tableau des scores (ou `leaderboard`) sert à afficher les dix joueurs ayant le plus de points sur la page d'accueil. Pour ça on utilise un `SELECT` et `ORDER BY ... DESC` pour classer les résultats dans l'ordre décroissant en fonction des points. Les résultats sont ensuite retournés ligne par ligne dans un tableau associatif avec `fetch(PDO::FETCH_ASSOC)` et tant que ce tableau contient des données, on met à jour le rang (ou `rank`) avec la valeur de `$rank` qui augmente de 1 à chaque itération (ainsi on pourra récupérer sa valeur à tout moment car elle sera disponible dans la table `users`), et on retourne le `rank` ainsi que toutes les données demandées par la requête et stockées dans `$row` (d'où le `foreach`) des dix premiers utilisateurs dans un tableau.

```
$stmt = $conn->query("SELECT username, points FROM users ORDER BY
points DESC");

$rank = 1;

while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    $username = $row['username'];

    $updateStmt = $conn->prepare("UPDATE users SET rank = :rank
WHERE username = :username");

    $updateStmt->execute([':rank' => $rank, ':username' =>
$username]);

    f($rank<=10){
        echo "<tr><td>{$rank}</td>";
        foreach ($row as $key => $value) {
            echo "<td>{$value}</td>";
        }
        echo "</tr>";
    }

    $rank+=1;
}
```



Pour la fonction de recherche, on utilise toujours la même méthode avec un `POST` et un `SELECT`, mais cette fois à la place de préparer la requête et d'assigner les valeurs après coup pour la sécuriser on assigne directement `$username` dans la requête SQL pour permettre une injection SQL (pour le niveau 6). On récupère ensuite toutes les valeurs retournées par la BDD par le biais d'un tableau associatif comme pour le code mentionné plus haut. Pour finir, on retourne le résultat en haut du tableau avec un fond rouge pour plus d'impact visuel.

```
$stmt = $conn->query ("SELECT rank, username, points FROM users WHERE  
username='$username'");  
  
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {  
    echo "<tr>";  
    foreach ($row as $key => $value) {  
        echo "<td style='background-color: red;'>{$value}</td>";  
    }  
    echo "</tr>";  
}
```

## Niveaux et failles

### Niveau 1 : Source Code

Le niveau 1 est une introduction et présente l'une des failles de sécurité les plus faciles à repérer. Les développeurs ont la possibilité d'ajouter des commentaires dans le code HTML, une ligne de texte qui ne sera pas interprétée comme une commande mais qui peut servir de note pour aider à comprendre le code. Parfois, il arrive qu'un développeur oublie d'enlever une note importante du code en le publiant, et que n'importe quel utilisateur puisse y avoir accès. Ici, une simple inspection du code source de la page avec cliquer droit > inspecter suffit à découvrir qu'un commentaire contenant le mot de passe a été oublié dans le code. Il suffit alors au joueur d'entrer ce mot de passe dans le champ prévu à cet effet pour terminer le premier niveau et être crédité en points.

```
<!-- The password is : verystrongpassword -->
```

### Niveau 2 : Source Code

Le niveau 2 comporte une autre faille assez classique. Il existe un attribut `disabled` pouvant être donné à un bouton ou à un `submit`, qui empêche l'utilisateur d'appuyer dessus. Cet attribut peut être utilisé pour restreindre une fonction qui est en cours de développement et empêcher un utilisateur d'y avoir accès. Le problème de cet attribut est qu'il se trouve côté client, c'est-à-dire que son code peut être modifié par l'utilisateur lui-même. Bien que la modification n'ait lieu que pour l'utilisateur, il suffit que l'action associée à cet `input` n'ait pas été retirée par le développeur par maladresse et une requête censée ne pas être utilisable se retrouve envoyée au serveur. Pour ce niveau, un `submit` servant à se connecter au compte admin avec la méthode `POST` est désactivé. Le joueur peut alors supprimer l'attribut dans le code de la page et réussir à se connecter. Il est alors crédité en points.

```
<form method="POST">
  <p>Welcome back admin !</p>
  <input disabled="true" type="submit" value="Log in" name="submit">
</form>
<script>
```

### Niveau 3 : Obfuscation

Le niveau 3 demande des connaissances de base en JavaScript et un peu de logique. Il arrive parfois que pour sécuriser un site web, un développeur fasse recours à l'obfuscation. Le but est de rendre le code incompréhensible en le cryptant pour empêcher une personne mal intentionnée de détourner le fonctionnement du code. Cette méthode est maintenant très peu utilisée car de nombreux outils permettent de décrypter ce genre de code. Ici, le mot de passe est crypté en code ASCII, augmenté de 3 pour éviter de pouvoir simplement le traduire grâce à un outil en ligne, et stocké dans une liste visible par l'utilisateur dans la balise `script`. Une fonction de décodage est également présente pour permettre au joueur de décrypter la liste et trouver le mot de passe. Il faudra donc utiliser l'onglet console et utiliser la fonction `decode(h)` sur la liste `L` avec la commande `decode(L)`. L'utilisateur pourra ensuite utiliser le mot de passe que la console a renvoyé, et être crédité en points.

```
L = [60, 55, 40, 36, 50, 125, 53, 107, 59]

function decode(h) {
  let pass = ""
  for(let i = 0; i<h.length; i++){
    pass += String.fromCharCode(h[i]-3)
  }
  return pass
}
```

### Niveau 4 : Injection SQL

Les injections SQL sont des failles de sécurité plutôt communes et très redoutées par les développeurs. Le principe consiste à utiliser un champ de saisie, souvent celui d'une page de connexion, pour envoyer ou modifier les requêtes SQL destinées à la base de données. Il est facile d'utiliser une injection pour contourner un mot de passe, à condition de connaître la structure d'une requête SQL, qui prend généralement cette forme :

```
SELECT * FROM table WHERE username = '$username' AND password='$password'
```

Les variables `$username` et `$password` contenant les valeurs renseignées par l'utilisateur. Il est maintenant possible d'imaginer que l'utilisateur renseigne un `'` dans le champ nom d'utilisateur, et en utilisant le principe de la mise en commentaire (`--`) il est possible de modifier la requête de telle sorte que le mot de passe ne soit pas vérifié :

```
SELECT * FROM table WHERE username = 'admin' -- ' AND password='$password'
```

La suite de la commande est alors mise en commentaire et l'utilisateur n'a plus besoin de connaître le mot de passe pour se connecter. La requête est donc exécutée comme suit :

```
SELECT * FROM table WHERE username = 'admin'
```

Il suffit donc d'entrer la commande `admin' --` (sans oublier les espaces avant et après les `-`) dans le champ utilisateur, et un mot de passe quelconque (car il ne peut pas être vide) dans le champ mot de passe pour réussir le niveau et être crédité en points.

## Niveau 5 : Injection SQL

Pour ce niveau, on utilise le même principe que pour le niveau 4 mais cette fois le champ d'utilisateur n'est plus disponible car le nom d'utilisateur est déjà renseigné dans la requête.

```
SELECT * FROM table WHERE username = 'admin' AND password='$password'
```

On peut alors utiliser la commande OR suivi d'une vérité générale pour se connecter sans connaître le mot de passe :

```
SELECT * FROM table WHERE username = 'admin' AND password='1234' OR 1=1 -- '
```

La commande sera donc interprétée comme :

```
SELECT * FROM table WHERE username = 'admin' AND password='1234' OR 1=1
```

`1=1` étant toujours vraie, même si le mot de passe n'est pas le bon, la connexion sera de toute façon effectuée. La commande que le joueur doit donc entrer dans le champ mot de passe pour être crédité en points est donc `' OR 1=1 -- '`.

## Niveau 6 : Injection SQL

Cette fois, l'objectif est de récupérer le mot de passe contenu dans la table *admintable*, comme il est possible de le voir dans un commentaire du code source. Pour effectuer cette injection, à la place d'utiliser le champ mot de passe qui lui est sécurisé (il sera donc impossible d'injecter du code dedans), il faut utiliser le champ de recherche au-dessus du leaderboard sur la page d'accueil. La requête utilisée pour rechercher un utilisateur et afficher ses points et son rank est la suivante :

```
SELECT * FROM table WHERE username='$username'
```

Sachant que le résultat de cette requête sera automatiquement affiché sur le leaderboard, on peut utiliser l'opérateur UNION qui sert à joindre une autre requête du même type à la précédente :

```
SELECT * FROM table WHERE username='username' UNION SELECT * FROM admintable -- '
```

La commande sera donc interprétée comme :

```
SELECT * FROM table WHERE username='username' UNION SELECT * FROM admintable
```

et tout le contenu de *admintable* sera affiché dans le leaderboard. Il faut ensuite copier le mot de passe et retourner sur le niveau pour valider la réponse et obtenir les points. A noter que cela fonctionne uniquement si le nombre de colonnes dans le leaderboard correspond exactement au nombre de colonnes dans *admintable*. Si ce n'était pas le cas, l'injection serait un peu plus complexe mais toujours réalisable.

## Niveau 7 : Injection XSS

Les injections XSS sont, tout comme les injections SQL, un moyen de faire exécuter du code arbitraire par injection. Pour faire une injection XSS (ou cross-site scripting), il faut qu'une variable soit retournée sur le site sans protection. Par exemple, si une input de connexion assigne son contenu à une variable `$username`, et que le site renvoie une erreur du style :

```
echo "Désolé {$username}, mot de passe incorrect."
```

Il sera possible d'injecter du code JavaScript à la place du nom d'utilisateur et, par exemple, récupérer les cookies du site dans une `alert` :

```
echo "Désolé {<script>alert(document.cookie)</script>}, mot de passe incorrect."
```

Le code sera donc interprété comme :

```
echo "Désolé <script>alert(document.cookie)</script>, mot de passe incorrect."
```

Et une `alert` contenant les cookies (dont un contenant le mot de passe) s'affichera à l'écran. On peut alors l'utiliser pour récupérer les points en utilisant `admin` comme nom d'utilisateur.

## Niveau 8 : Injection XSS

Cette fois, il n'y a pas de champ dans lequel entrer un nom d'utilisateur, mais des options. On peut remarquer qu'en changeant d'option et en essayant des mots de passe, l'URL (le lien) du site change. La valeur de l'input est envoyée avec la méthode GET au lieu de POST, utilisant l'URL comme référence (une telle méthode peut être utilisée dans un site web, par exemple, pour permettre de partager le lien en gardant en mémoire les options précédemment sélectionnées et permettre à la personne qui le reçoit de retrouver la même page que la personne qui la lui a envoyée). On peut donc voir les valeurs qu'on a fournies sous ce format :

```
oc3.lddr.ch/Baurand/xssinjection2.php?username=admin&password=1234&submit=Log+in
```

Le problème se trouve dans le retour de l'erreur, qui récupère la valeur de ce GET sans être protégé des injections XSS. On peut alors utiliser l'URL pour modifier le nom d'utilisateur et injecter du code JavaScript. Le code sera donc exécuté et les cookies (dont celui du mot de passe) affichés dans une alert.

```
...php?username=<script>alert(document.cookie)</script>&password=...
```

## Niveau 9 : Injection XSS

Les failles XSS deviennent beaucoup plus dangereuses quand la variable est retournée à un autre utilisateur, par exemple pour un site de messagerie, où on pourrait faire exécuter du script de son côté. On peut alors le rediriger sans son autorisation vers un site programmé pour voler ses cookies, et les utiliser pour usurper son identité. Malheureusement, la programmation d'un exercice de ce genre implique de créer un faux utilisateur (bot) qui lit automatiquement une messagerie, mais ce n'est possible qu'en installant Node.js sur le serveur du site web, ce qui demande des droits d'accès que je n'ai pas. Une démonstration en local pourra être effectuée durant la soutenance. Les fichiers du bot sont disponibles sur le GitHub. Pour essayer en local, installez Node.js sur votre ordinateur, et ouvrez le fichier START.bat dans le dossier bot. Rendez-vous ensuite sur la page du niveau 9 et vérifiez qu'un message « read by admin ! » soit bien affiché après l'envoi d'un message.

## Comment se protéger de ces failles ?

Pour certaines, la réponse peut paraître évidente comme vérifier de ne pas laisser de commentaires dans le code, ou enlever intégralement les fonctions qui ne sont pas censées être utilisables. Néanmoins, pour les injections SQL et XSS la réponse est un peu plus complexe. Il y a deux moyens principaux de faire une requête SQL : insérer les variables dans la requête, et préparer la requête avant d'y insérer les variables. C'est la première méthode qui est vulnérable et qui n'empêche pas les injections. Pour s'en protéger, il faut donc passer de :

```
$req = $conn->query("SELECT * FROM table WHERE username = '$username'")
```

à :

```
$req = $conn->prepare("SELECT * FROM table WHERE username = :username")  
$req->execute(['username' => $username])
```

Une ligne de code de plus qui enlève tout risque potentiel. Pour les injections XSS, il faut utiliser une fonction spécifique appelée `htmlspecialchars( )` pour transformer tous les caractères spéciaux comme « >, <, ", \, ... » en retournant une variable et éviter ainsi que du code soit exécuté.

```
echo "Désolé {htmlspecialchars($username)}, mot de passe incorrect."
```

Il est également recommandé de sécuriser les cookies avec l'attribut `httponly` lorsqu'ils sont déclarés pour empêcher d'y avoir accès avec du code JavaScript.

## Conclusion

Ce TM a donc pour but d'instruire (avec la documentation disponible) et d'exercer (sous forme de niveaux) aux bases du test d'intrusion web, notamment avec les failles d'injections SQL et XSS. C'est un exercice qui demande beaucoup de logique et de persévérance. Ça permet également aux joueurs de voir la programmation sous un angle plus ludique, et se mettre dans la peau d'un expert en pentesting. Le principe d'apprendre à attaquer pour mieux se protéger est parfaitement illustré par ce TM, mais il est important de préciser que les vulnérabilités mentionnées ici sont très connues par les développeurs et qu'il devient rare d'en trouver, surtout s'il s'agit d'un site de grande envergure. Ce qui aurait pu être apporté au site pour le rendre meilleur est, d'une part, un ajout de failles et de niveaux plus complexes même si trop pousser la difficulté pourrait rendre les niveaux irréalisables pour la plupart des joueurs, et d'une autre part un accès à plus de fonctionnalités sur le serveur comme l'installation de scripts ou la possibilité de jouer avec ses paramètres. Il existe encore de très nombreuses failles qui ne sont pas mentionnées dans ce TM, et il est impératif de faire appel à un expert en cybersécurité si un site nécessite d'être sécurisé.

**Gardez en mémoire que la plus grande vulnérabilité reste l'humain.**



## Sources

Journée de stage d'informatique à l'EPFL – le 24.11.2023

<https://www.w3schools.com/> - consulté fréquemment durant toute la durée du TM

[https://en.wikipedia.org/wiki/Capture\\_the\\_flag\\_\(cybersecurity\)](https://en.wikipedia.org/wiki/Capture_the_flag_(cybersecurity)) – consulté le 8.10.2024

[https://en.wikipedia.org/wiki/DEF\\_CON](https://en.wikipedia.org/wiki/DEF_CON) – consulté le 8.10.2024

<https://portswigger.net/web-security/sql-injection> - consulté le 15.5.2024

<https://owasp.org/www-community/attacks/xss/> - consulté le 12.10.2024

<https://portswigger.net/web-security/cross-site-scripting> - consulté le 12.10.2024

<https://www.root-me.org/> - consulté fréquemment durant toute la durée du TM

<https://github.com/Ruulian/BotNodeJSChallenge> - bot (Niveau 9)