

Rapport 1  
Sécurité des Applications Web

HOTEL O

Groupe 1

# SOMMAIRE

<b>Fonctionnalités du site internet Hotel O</b>	<b>3</b>
Réserver des chambres	3
Laisser un message dans le livre d'or	3
S'inscrire	3
Se connecter	3
<b>Les 10 vulnérabilités OWASP</b>	<b>4</b>
Injection	4
Présentation de la faille	4
Implémentation de la faille sur notre site	4
Localisation de la faille	4
Code pour utiliser la faille	5
Violation de Gestion d'Authentification et de Session	7
Présentation de la faille	7
Implémentation de la faille sur notre site	7
Localisation de la faille	7
Code pour utiliser la faille	7
Prévention possible contre ce type d'attaque	8
Cross-Site Scripting	8
Présentation de la faille	8
Implémentation de la faille sur notre site	8
Localisation de la faille	8
Code pour utiliser la faille	9
Prévention possible contre ce type d'attaque	10
Références directes non sécurisées à un objet	11
Présentation de la faille	11
Implémentation de la faille sur notre site	11
Localisation de la faille	12
Code pour utiliser la faille	12
Prévention possible contre ce type d'attaque	12
Mauvaise configuration Sécurité	12
Présentation de la faille	12
Implémentation de la faille sur notre site	13
Localisation de la faille	13
Code pour utiliser la faille	13
Prévention possible contre ce type d'attaque	14

Exposition de données sensibles	15
Présentation de la faille	15
Implémentation de la faille sur notre site	15
Localisation de la faille	15
Code pour utiliser la faille	16
Prévention possible contre ce type d'attaque	16
Manque de contrôle d'accès au niveau fonctionnel	16
Présentation de la faille	16
Implémentation de la faille sur notre site	17
Localisation de la faille	17
Code pour utiliser la faille	17
Prévention possible contre ce type d'attaque	18
Falsification de requête intersite (CSRF)	18
Présentation de la faille	18
Implémentation de la faille sur notre site	18
Localisation de la faille	18
Code pour utiliser la faille	18
Prévention possible contre ce type d'attaque	20
Utilisation de composants avec des vulnérabilités connues	20
Présentation de la faille	20
Implémentation de la faille sur notre site	20
Localisation de la faille	20
Code pour utiliser la faille	21
Prévention possible contre ce type d'attaque	23
Redirections et renvois non validés	24
Présentation de la faille	24
Implémentation de la faille sur notre site	24
Localisation de la faille	24
Code pour utiliser la faille	24
Prévention possible contre ce type d'attaque	24
<b>III. Le Secret</b>	<b>25</b>

# I. Fonctionnalités du site internet Hotel O

Ce site internet est le site typique d'un hôtel. Il est utilisé par les clients et futurs clients pour découvrir l'établissement et par la même occasion réserver une chambre

## A. Réserver des chambres

Sur l'onglet "Nos chambres", l'utilisateur a la possibilité de regarder la liste des chambres disponibles, et ainsi de réserver celle qui lui plaît.

## B. Laisser un message dans le livre d'or

Sur l'onglet "Livre d'or", il est possible de laisser un message sur le site de l'hôtel et ainsi partager son expérience avec d'autres clients.

## C. S'inscrire

Lorsqu'un utilisateur n'a pas de compte, il a la possibilité de s'en créer un. Pour cela, il doit passer par l'onglet "Connexion", puis cliquer sur un bouton "S'inscrire", qui le redirige vers une page contenant un formulaire d'inscription. Cette page lance ensuite une requête à notre base de données pour y ajouter le nouveau compte de l'utilisateur.

## D. Se connecter

Sur l'onglet "Connexion", l'utilisateur a la possibilité de se connecter en utilisant un pseudo et un mot de passe. Pour cela, il doit avoir au préalable créé un compte. Une fois connecté, il a accès à la réservation de chambres.

## II. Les 10 vulnérabilités OWASP

### A. Injection

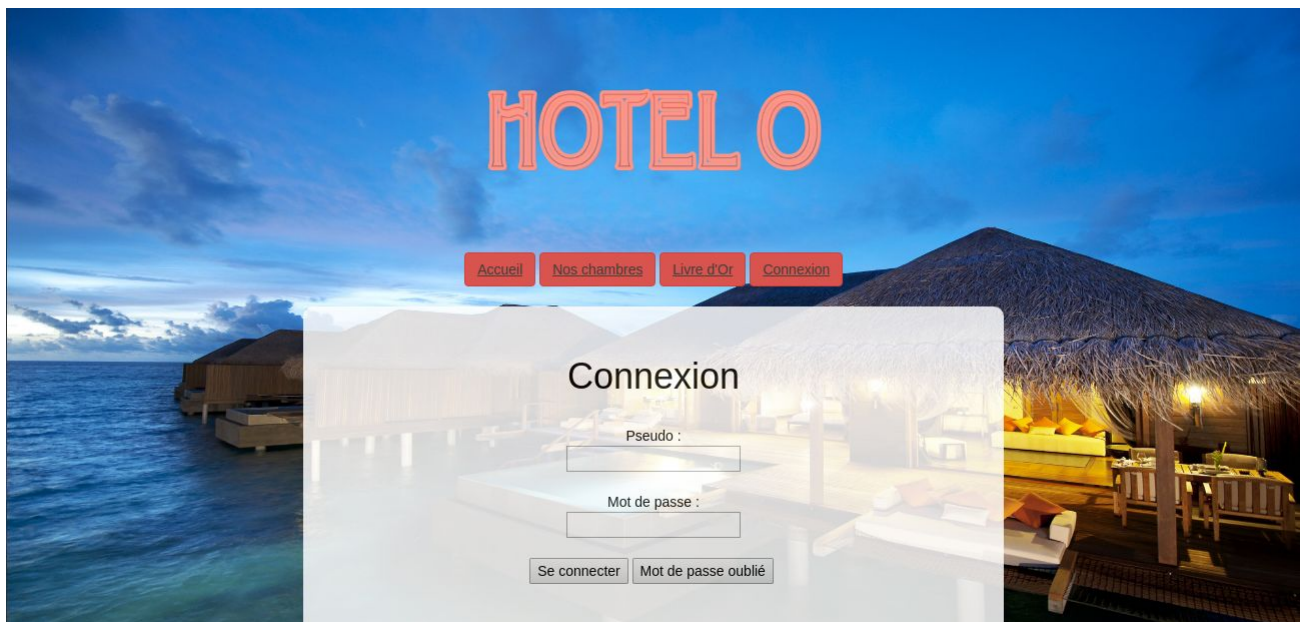
#### 1. Présentation de la faille

Une application Web peut être amenée à demander des informations à l'utilisateur afin d'agir en fonction de ces dernières. De ce fait, des données non-fiables peuvent être envoyées à l'interpréteur en tant qu'élément d'une commande (ou requête). Ainsi, l'interpréteur sera dupé et aura un comportement totalement différent de celui souhaité. Cela peut permettre à l'utilisateur d'accéder à des données non autorisées en exécutant les commandes le permettant.

#### 2. Implémentation de la faille sur notre site

##### a) Localisation de la faille

Lorsqu'un utilisateur cherche à se connecter au site et a oublié son mot de passe, il lui est possible de demander à récupérer son mot de passe, avec le bouton visible sur l'image ci-dessous:



*Page de connexion avec bouton de récupération de mot de passe*

Une fois que l'utilisateur a appuyé sur ce bouton, il est redirigé vers la page ci-dessous.



*Page de récupération de mot de passe*

Sur cette page, l'utilisateur doit saisir son pseudo, pour que le serveur lui renvoie le mot de passe associé. C'est à ce moment que l'attaquant peut exploiter la faille.

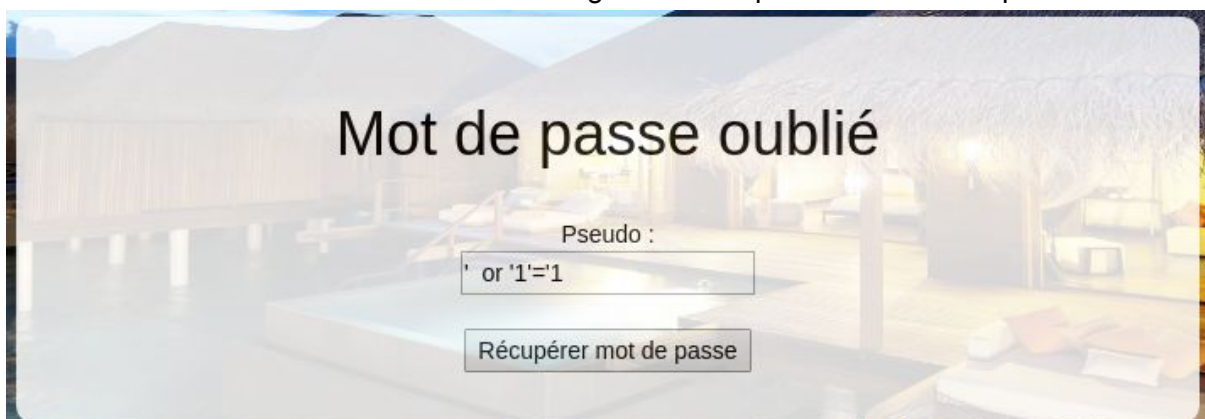
b) Code pour utiliser la faille

La requête à la base de données est réalisée à l'aide du code suivant:

```
$pseudo= $_GET["pseudo"];  
$password = $_GET["pwd"];  
$connexion = mysqli_connect("db4free.net", "rundle1", "booksmart2016", "booksmarttcd", "3306");  
$query="SELECT password FROM users WHERE pseudo='".$pseudo."'";
```

*Code pour la récupération du mot de passe*

Le paramètre pseudo de la requête étant récupéré depuis l'input dans lequel l'utilisateur saisit son pseudo, il n'y a aucune vérification de la part de notre code. Ainsi, l'attaquant peut utiliser la faille en saisissant une chaîne changeant le comportement de la requête:



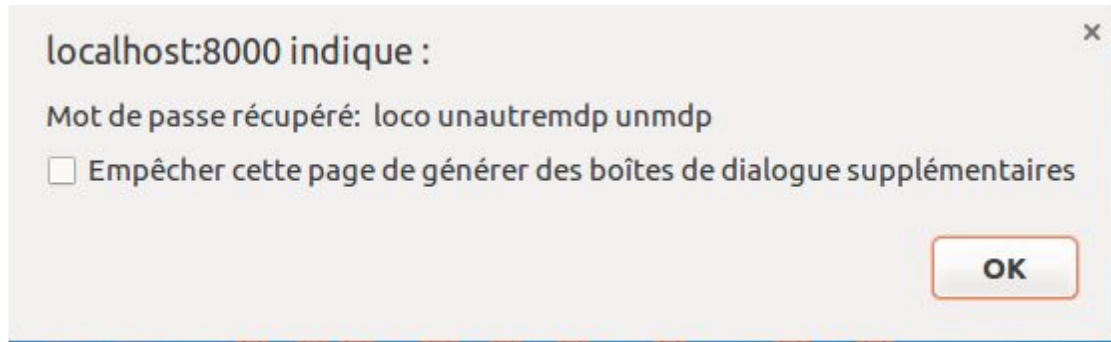
*Utilisation de la faille d'injection par l'attaquant*

En faisant cela, la requête SQL réalisée par le serveur voit son comportement changer totalement.

En effet, la requête devient:

“SELECT password FROM users WHERE pseudo=” or ‘1’=’1’ ”.

Ainsi, tous les mots de passe vont être renvoyés par la base de données, et affichés à l'utilisateur:



*Résultat de l'utilisation de la faille*

L'attaquant peut ensuite utiliser ces mots de passe de la façon qu'il veut, en les essayant tous par exemple pour se connecter au compte administrateur.

La vulnérabilité peut être exploitée différemment. En effet, l'attaquant peut terminer l'instruction de requête SQL, puis ajouter du code php à la suite, qui sera exécuté par le serveur, et enfin réécrire une nouvelle requête SQL fictive afin de garder la syntaxe en accord avec le code déjà existant.

Ainsi, la faille d'injection de notre site peut être utilisée en tant qu'injection SQL, mais également en tant qu'injection PHP.

#### c) Prévention possible contre ce type d'attaque

Afin de se protéger contre ce type d'attaque, il est possible d'utiliser les méthodes php permettant d'échapper les caractères spéciaux tels que les quotes, afin d'éviter que l'utilisateur change le comportement attendu de la requête. D'une façon plus générale, la prévention consiste à séparer les données non-fiables (dans notre cas, le pseudo saisi par l'utilisateur) des requêtes, en utilisant par exemple une API évitant l'intervention de l'utilisateur.

## B. Violation de Gestion d'Authentification et de Session

### 1. Présentation de la faille

Un système d'authentification ou de gestion de sessions est difficile à mettre en place de manière sécurisée. En effet, il existe de nombreuses failles concernant notamment la déconnexion, la gestion de mots de passe, l'expiration de session...

L'attaquant peut donc exploiter une telle faille et ainsi usurper l'identité d'un utilisateur ou encore compromettre plusieurs comptes dans le cas où il a accès à une base de données où les mots de passe ne sont pas bien chiffrés. A savoir que les attaquants cherchent en général à obtenir des comptes à privilèges.

### 2. Implémentation de la faille sur notre site

#### a) Localisation de la faille

Lorsqu'un utilisateur se connecte au site, une requête à un serveur MySQL est réalisée pour vérifier que l'utilisateur existe et que le mot de passe correspond. Dans ce cas, le token de session (enregistré dans la base de données) est utilisé pour créer un cookie qui sera conservé par le navigateur pendant une semaine, comme on peut le voir sur la capture d'écran ci-dessous.

```
<?php
$pseudo= $_GET["pseudo"];
$password = $_GET["pwd"];
$connexion = mysqli_connect('db4free.net', 'rundlel', 'booksmart2016', 'booksmarttcd', '3306');
$query = 'SELECT * FROM users WHERE pseudo="'. $pseudo. '" AND password="'. $password. '"';
$res = $connexion->query($query);
$row = $res->fetch_object();
if($row->pseudo) {
    $result = 'ok';
    setcookie("token", $row->token, time() + 7*24*60*60);
} else {
    $result = 'ko';
}
echo ($result);
?>
```

*Code exécuté lors de la connexion*

#### b) Code pour utiliser la faille

Il n'y a pas de code à exécuter pour exploiter la faille. Il suffit simplement qu'un utilisateur se soit connecté sur le site et qu'il ait oublié de fermer sa session avant de fermer le navigateur. Ainsi, en utilisant le même ordinateur après lui, il est possible d'aller sur le site et on verra que l'utilisateur est toujours connecté, et ce pendant une semaine. Ceci est dû au fait que notre code vérifie simplement l'existence du cookie pour permettre les fonctionnalités accessibles uniquement en tant qu'utilisateur connecté, et que le cookie n'est supprimé que lors d'une déconnexion.



c) Prévention possible contre ce type d'attaque

Dans notre cas particulier, l'une des solutions possibles serait de changer le timestamp du cookie, afin que celui-ci expire lorsque le navigateur sera fermé par exemple. En effet, le cookie est actuellement créé avec une durée d'une semaine, et est supprimé lors de la déconnexion de l'utilisateur connecté. En faisant en sorte que le cookie soit supprimé lors de la fermeture du navigateur, nous éviterions qu'un autre utilisateur aille sur notre site et soit déjà connecté avec le compte utilisé précédemment.

D'une manière plus générale, pour se protéger contre ce type d'attaque, la sécurisation de la base de données est un moyen efficace, ne permettant pas à un attaquant de récupérer les informations d'un utilisateur pour usurper son identité lors d'une session.

## C. Cross-Site Scripting

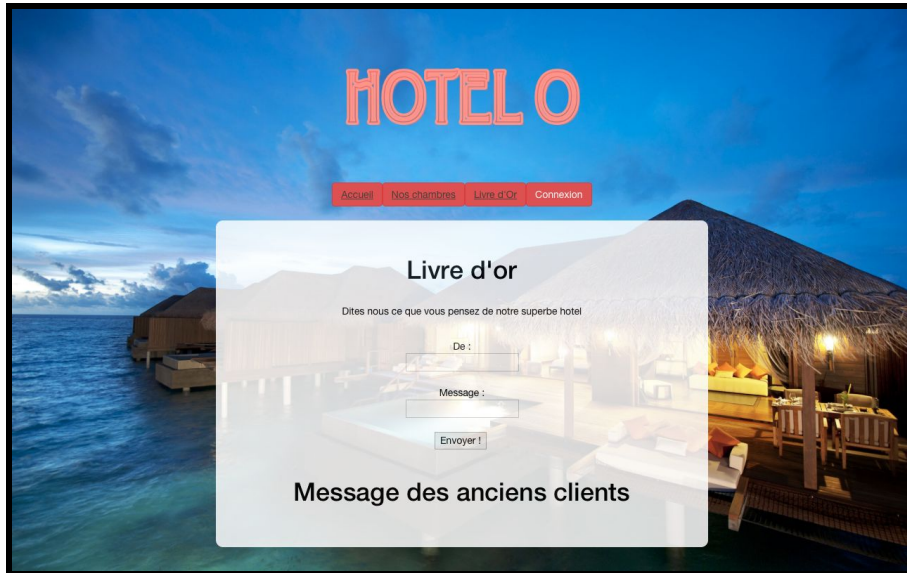
### 1. Présentation de la faille

Les failles XSS permettent l'acceptation des données non fiables et les envoie à un navigateur web sans être validées. XSS permet à des attaquants d'exécuter du script (principalement Javascript) dans le navigateur de la victime dans le but, d'altérer des sites web, détourner des sessions utilisateur ou encore rediriger la victime vers des sites malicieux. Dans notre cas, il s'agit d'une faille XSS stockée (ou permanente) car elle sera stockée sur le serveur de notre application. Il est intéressant de noter que cette faille constitue aussi une injection de code qui est un autre type de faille présent dans le OWASP TOP Ten.

### 2. Implémentation de la faille sur notre site

a) Localisation de la faille

L'implémentation de cette faille a été faite au travers de la page "Livre d'or". En effet, le site internet offre la possibilité aux clients de donner leur avis sur les services de l'hôtel, comme visible sur l'image ci-dessous.



*Page du livre d'or de notre site internet*

Chaque message est ensuite enregistré dans un fichier txt et est affiché à la suite sur la page en dessous du titre "Message des anciens clients"



*Affichage du message d'un client sur la page du Livre d'or*

#### b) Code pour utiliser la faille

Les messages sont affichés grâce à la balise script suivante :

```
<div id= show >  
<script>  
    document.write("<?php echo $messages;?>");  
</script>  
</div>
```

où la variable php \$messages contient l'ensemble des messages.

Il existe donc plusieurs exemples de code possibles pour utiliser cette faille. Le modèle de code est à écrire dans les champs "de" et "message". Le code est divisé en trois parties:

- Une fermeture de fonction "); permettant de fermer la fonction document.write
- Le code à exécuter
- Une fonction Javascript de fin pour terminer sans erreur l'exécution du script qui possède le ");. Celui-ci servait initialement à fermer le paramètre de la fonction document.write

Pattern utilisable :

`");codeàexecuter;functionJavascript("`

De plus, il faut remarquer que l'affichage des messages se fait sous la forme :

*Message Auteur*

En effet, le code associé au stockage du message est le suivant :

```
$message= $_GET["message"]. " <i>".$_GET["client"]. "</i><br/><br/>" ;  
$file="messages.txt";  
file_put_contents($file,$message,FILE_APPEND);
```

Il est intéressant de remarquer l'ajout des balises `<i>`, `</i>`, `<br/>` et l'ordre de stockage du message et de l'auteur.

Il faut donc commencer le code dans le champs "Message" soit le deuxième champs. De plus, l'auteur est encadré par les chevrons `<i>` pour permettre l'affichage en italique. Il faut donc que le code à exécuter ouvre une string à la fin du champs Message et la referme dans auteur.



*Exemple de code pour injecter du script dans le site internet*

Le résultat de cette attaque sera l'affichage de plusieurs messages d'alerte "popup".

Il est possible que l'attaquant, en essayant d'injecter du code, paralyse la page du livre d'or. C'est pourquoi, il a été décidé d'ajouter une manière de supprimer l'ensemble des messages rapidement pour être capable de remettre en marche sans notre intervention le livre d'or. Il s'agit d'entrer delete dans l'input auteur et delete dans l'input message. Cette astuce est en commentaire dans la page html du livre d'or.

### c) Prévention possible contre ce type d'attaque

Il est possible d'éviter ce genre d'attaques en utilisant certains dispositifs.

Il existe certaines fonctions PHP et Javascript qui permettent de filtrer le contenu des données renseignées par les utilisateurs. Il est possible d'utiliser :

- La fonction `htmlspecialchars` qui convertit les caractères spéciaux en codage HTML. Par exemple '`<`' deviendra `&lt;`;
- La fonction `htmlentities()` est très proche de `htmlspecialchars` mais elle convertit aussi tous les codage html et javascript.
- La fonction `strip_tags()` permet de supprimer les balises HTML et PHP des strings en paramètres

Il est possible aussi d'implémenter ses propres fonctions javascript pour sanitiser les inputs des utilisateurs.

Il est possible aussi de mettre en place la "Content-Security-Policy" qui permettrait ainsi de ne pas exécuter le code script inline.

## D. Références directes non sécurisées à un objet

### 1. Présentation de la faille

Un référence directe non sécurisée à un objet décrit la présence d'un accès par l'utilisateur à des variables qui, en modifiant leur valeur, permettent l'accès à des données protégées. Cela peut permettre à un utilisateur malveillant d'utiliser ces variables pour contourner le bon fonctionnement du site (modification du prix de la réservation).

### 2. Implémentation de la faille sur notre site

Lors de la réservation d'une chambre, le prix de la chambre est identifié :

Prix à la semaine : `<span id="prixConfort">850</span>€.<br />`

Cette identification est réutilisée lors de la réservation en JavaScript :

```
function reserverConfort(){  
    var prix = document.getElementById("prixConfort").innerText;  
    // Modifications visuelles de la page...  
    validateReservation(prix);  
}
```

Puis il est utilisé lors de la mise en place de la réservation sur le serveur :

```
function validateReservation(prix) {  
    // Gestion de la fonction callback...  
    var url = window.location.origin;  
    url += "/bookRoom.php?prix=" + prix;  
    var xmlhttp = new XMLHttpRequest();  
    xmlhttp.open('GET', url, true);  
    xmlhttp.onreadystatechange = callback;  
    xmlhttp.send(null);  
}
```

La page bookRoom.php envoyée en httprequest insère la réservation dans la base de données. Cependant elle est appelée avec la méthode PHP GET, avec en paramètre le prix.

a) Localisation de la faille

Cette faille est indirectement présente dans les fichiers JavaScript utilisés sur la page de réservation des chambres.

b) Code pour utiliser la faille

En observant les événements réseau lors de l'utilisation du site, l'utilisateur malveillant peut observer cet transmission résultante du processus décrit précédemment :

	bookRoom.php?prix=850	200 OK	xhr	<a href="#">vulneChambres.js:32</a> Script	1.3 KB 2.8 KB	110 ... 110 ...
---	-----------------------	-----------	-----	---	------------------	--------------------

*En observant la timeline network, on peut observer que le script a appelé la page bookRoom.php avec un paramètre GET accessible à l'utilisateur, et donc modifiable.*

Cette page envoyée en XHR, n'a pas son contenu visible à l'utilisateur mais peut permettre tout de même d'effectuer des réservations en modifiant le prix. Ainsi un utilisateur pourrait appeler la page bookRoom.php?prix=1 par divers moyens (nouveau XHR, modification du code JavaScript, modification de la requête avant envoi.

c) Prévention possible contre ce type d'attaque

Pour prévenir de cette attaque, il est nécessaire de lier la réservation ainsi que ses informations avec la base de données. Les prix seraient alors stockés dans la base plutôt que dans le code HTML. L'utilisateur n'aurait alors plus un accès direct aux variables sensibles du système.

## E. Mauvaise configuration Sécurité

### 1. Présentation de la faille

Cette faille de sécurité est liée à la mauvaise configuration de certains éléments du serveur web ainsi que la mauvaise configuration / des oublis sur la configuration du site web en lui même.

Ceci se représente notamment par des comptes de bases non modifiés et donc exploitables par toute personne connaissant les identifiants à la création, des fichiers d'installations non supprimés, qui permettent ainsi la modification des propriétés du site web, voir même sa réinstallation complète, permettant ainsi de détourner entièrement son utilisation.

Les autres possibilités moins conséquentes sont l'oubli de fichier d'informations / anciennes versions du programme qui permettraient d'accéder à des informations compromettantes comme la liste des fichiers du serveur, des informations sur la configuration actuelle du système, qui donneraient donc à l'attaquant des informations sur les failles exploitables liées au système.

Ce problème est réellement un problème transverse à toutes les couches du site-web et doit être étudié par tous les partis (les failles pouvant aussi bien se faire au niveau du serveur global, que du site web, que de la base de données, que des comptes liées à des API externes ...).

De plus, si le serveur ou site ne dispose pas d'outil de tracking de connexions, il serait alors impossible de savoir si un attaquant ne dispose pas de tous les droits et n'est pas en train d'espionner / exploiter les données du site internet.

## 2. Implémentation de la faille sur notre site

### a) Localisation de la faille

Une première faille est la possibilité de lister les fichiers sur le serveur. Il faut pour cela se servir d'une faille d'injection PHP décrite dans la partie A du rapport (en adaptant la syntaxe afin de finir la requête SQL et de pouvoir exécuter du script PHP)

Le fichier d'installation du serveur n'a pas été supprimé ( install.php , trouvable simplement sur le lien <http://secuweb.neoskai.com/install.php> qui n'a pas été caché), et permet d'accéder à des informations critiques telles que les informations php ainsi que la configuration actuelle du serveur (qui contient le nom de compte administrateur par défaut du site).

### b) Code pour utiliser la faille

L'exploitation de cette faille peut passer par plusieurs intermédiaires.

Dans un premier temps, l'utilisateur peut exploiter l'injection de code afin de lister tous les fichiers sur le serveur, et ainsi découvrir que l'administrateur n'a pas supprimé la page de configuration de celui-ci.

Pour cela, il lui suffirait d'injecter le code suivant grâce à une faille d'injection PHPt:

```
$dir = '.';  
$files1 = scandir($dir);  
print_r($files1);
```

Une fois cette étape faite, le serveur va lui renvoyer la liste de la totalité des fichiers sous ce format:

```
Array ( [0] => . [1] => .. [2] => .htaccess [3] => assets [4] => chambres.php [5] => includes.inc.php [6] =>  
index.php [7] => install.php [8] => livredor.php [9] => livredorleavemessage.php [10] => management.php [11] =>  
menu.inc.php [12] => messages.txt [13] => nettoyerlivredor.php [14] => pictures )
```

Il peut ainsi se rendre compte qu'il existe encore une page install.php (numéro 8), qui n'a pas été supprimée.

Le fichier install.php étant commun à de nombreux sites disposant d'installations manuelles / graphiques (tels que les CMS), il s'agit d'une route facilement testable lors de l'arrivée sur un nouveau site.

Le hacker n'a alors pas besoin de passer par la première faille.



Il lui suffit donc de se rendre sur cette page et à partir de celle-ci accéder à diverses opérations:

- Accéder au fichier phpinfo, qui présente toutes les informations de configuration de php du serveur, et ainsi connaître toutes les failles exploitables liées à la version et à la configuration du serveur php.
- Accéder à la page de configuration
- Activer la page d'affichage de la configuration qui ne l'est pas par défaut

De base lorsque l'on tente d'accéder à la page de management, l'accès est refusé car la page n'est pas activée:



Il suffit alors d'activer la fonctionnalité, qui va modifier le fichier de configuration afin d'activer la fonctionnalité et ainsi pouvoir afficher la page de configuration, qui dispose du nom de compte admin, nécessaire à la découverte du secret.

#### c) Prévention possible contre ce type d'attaque

Une fois le site web installé, il est essentiel de vérifier que toutes les fichiers / identifiants liés à la configuration du serveur sont bien supprimés.



Ceci est d'autant plus vrai lorsque l'on utilise des CMS web qui permettent la réinstallation complète du site, et ainsi de générer un nouveau site sur lequel nous n'aurions potentiellement plus la main.

Il faut également modifier tous les mots de passes et comptes utilisateurs générés par défaut, qui sont des failles très facilement exploitables sur un site (compte administrateur laissé avec les identifiants admin/password).

Il faut enfin vérifier qu'aucun code mort ne tourne sur le serveur, qui pourraient être des failles exploitables par n'importe quel attaquant. Si une partie du code n'est pas nécessaire au bon fonctionnement de l'application, il est alors inutile de l'avoir sur le serveur de production.

Enfin, vérifiez que les éventuelles erreurs données par le système, ne révèlent pas des informations compromettantes sur celui-ci, telles que l'architecture de vos données si elles sont invalides etc...

Le dernier élément, qui est préférable mais non applicable tout le temps, est de tenter de tenir au mieux à jour sa plateforme avec les dernières versions sorties de chaque élément, qui corrigent en générale les éventuelles failles de sécurité des versions précédentes.

## F. Exposition de données sensibles

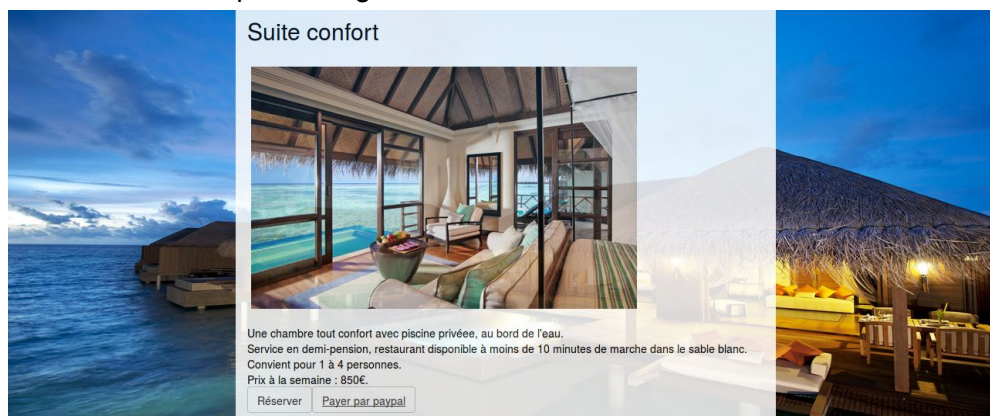
### 1. Présentation de la faille

Cette faille regroupe toutes les vulnérabilités liées à la protection des données sensibles. Les objectifs peuvent notamment être d'accéder à des données confidentielles, ou d'usurper une identité.

### 2. Implémentation de la faille sur notre site

#### a) Localisation de la faille

Dans notre site, la faille est présente dans la page "Nos chambres". En effet, lorsque l'utilisateur veut effectuer une réservation, le prix de la réservation est récupéré sur la page sans aucune mesure de sécurité, ce qui va permettre à l'utilisateur malveillant de modifier le prix à sa guise avant d'effectuer la réservation.





b) Code pour utiliser la faille

Afin d'exploiter cette faille, l'utilisateur doit ouvrir le code source de la page, modifier le champ contenant le prix de la suite à réserver, et enfin réserver cette dernière avec le nouveau prix.

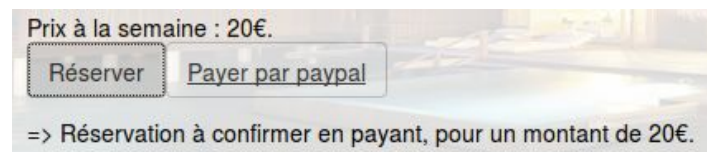
Illustration :

```
Prix à la semaine :  
<span id="prixConfort">850</span>  
€.
```

On modifie le prix :

```
Prix à la semaine :  
<span id="prixConfort">20</span>  
€.
```

Puis on va pouvoir réserver :



c) Prévention possible contre ce type d'attaque

Afin de se protéger contre ce type d'attaque, plusieurs mesures peuvent être prises.

Dans le cas présent, utiliser une méthode de chiffrement fort afin d'empêcher la modification du prix paraît être la solution la plus adaptée.

De manière plus générale, il faut éviter de stocker des informations que l'on a pas besoin de stocker. Il faut également éviter de collecter des informations qui pourraient être entrées ou modifiées par l'utilisateur.

De même, il faut identifier les données sensibles et s'assurer qu'elles sont correctement cryptées lors de la transmission et lors du stockage.

On peut également effectuer un test d'intrusion sur notre application web afin de vérifier de n'avoir rien oublié.

## G. Manque de contrôle d'accès au niveau fonctionnel

### 1. Présentation de la faille

En général, les applications web ne permettent pas à tous les utilisateurs d'accéder à l'intégralité des fonctionnalités qui la composent. Cela peut se traduire par des menus plus ou moins fournis dans l'interface graphique. Lorsque le seul contrôle d'accès à une fonctionnalité est fait côté client, l'application est vulnérable : un attaquant peut aisément déjouer la vérification et utiliser le programme à sa guise. Toutefois la plupart des applications web vérifient les droits d'accès avant de rendre visible dans l'interface une fonctionnalité restreinte mais oublient d'effectuer les mêmes vérifications de contrôle d'accès sur le serveur lors de l'accès à chaque fonction. Si les demandes ne sont pas vérifiées, les attaquants seront en mesure de forger des demandes afin d'accéder à une fonctionnalité non autorisée.

## 2. Implémentation de la faille sur notre site

### a) Localisation de la faille

Sur notre site, la faille est accessible depuis toutes les pages comportant le menu (la barre avec les boutons “Accueil”, “Nos chambres”...).

### b) Code pour utiliser la faille

Pour utiliser la faille, il suffit d’inspecter le code source de n’importe quelle page. Si on regarde uniquement le menu de manière graphique, on voit qu’il comporte quatre boutons: “Accueil”, “Nos chambres”, “Livre d’Or” et “Connexion”.



Mais en faisant attention dans le code source, on voit que le menu comporte en réalité cinq boutons, mais que le dernier est caché.

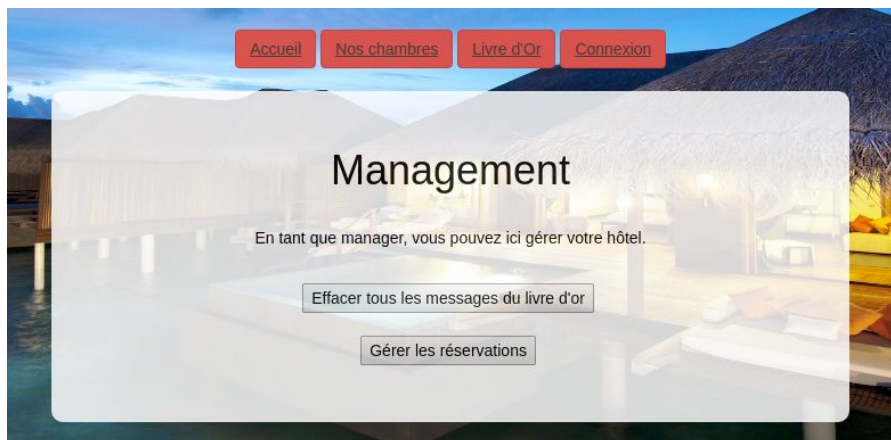
```
<a class="btn btn-danger" type="button" href="index.php">Accueil</a>  
<a class="btn btn-danger" type="button" href="chambres.php">Nos chambres</a>  
<a class="btn btn-danger" type="button" href="livredor.php">Livre d'Or</a>  
<a class="btn btn-danger" type="button" href="connexion.php">Connexion</a>  
<a id="btnManagement" class="btn btn-danger" type="button" href="management.php" style="display: none;">Management</a>
```

*Code source accessible depuis la console de Google Chrome*

En effet, le dernier bouton “Management” est caché uniquement côté client en JavaScript (jQuery) :

```
$(document).ready(function() {  
    $("#btnManagement").hide();  
});
```

L’attaquant peut donc voir dans la console ce bouton, et il voit qu’il redirige vers une page “management.php”. S’il essaye d’accéder à la page management, il n’y a pas de vérification et il peut donc effectuer quelques fonctions qui devraient être uniquement réservées pour des managers ou administrateurs.



*Page management.php*

c) Prévention possible contre ce type d'attaque

Pour se protéger contre cette faille, il faudrait tout d'abord ne pas simplement cacher le bouton en JavaScript mais plutôt gérer différents types d'utilisateurs et donc avoir des pages différentes suivant les utilisateurs (par exemple invité, client, administrateur...). Il faut également gérer l'authentification d'une meilleure manière, déjà en interdisant par défaut tous les droits d'accès et ensuite autoriser de manière explicite certains rôles pour les fonctionnalités restreintes. Enfin on pourrait effectuer une nouvelle vérification à chaque fois qu'une fonction supposée restreinte est exécutée, pour s'assurer que la personne l'utilisant est bien autorisée à le faire.

## H. Falsification de requête intersite (CSRF)

### 1. Présentation de la faille

L'objet de l'attaque CSRF est de forcer des utilisateurs à exécuter une ou plusieurs requêtes non désirées sur un site donné, forgées par un utilisateur malveillant. La victime choisie aura les privilèges nécessaires à l'exécution de la requête, voire une session encore active. Forger une requête revient à créer / falsifier une requête HTTP (par le biais d'une URL ou d'un formulaire principalement) pointant sur une action précise interne au site qui sera certainement néfaste pour la victime.

### 2. Implémentation de la faille sur notre site

#### a) Localisation de la faille

Dans notre site, la faille est notamment présente sur la page de réservation des chambres (chambres.php). Un exemple de mise en place de la faille est présenté sur la page index.php.

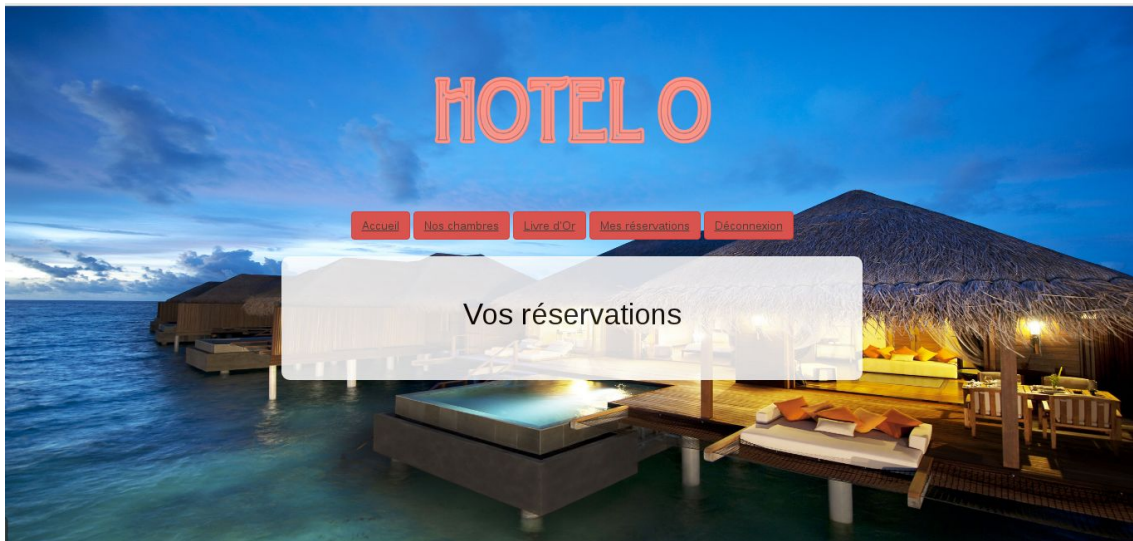
#### b) Code pour utiliser la faille

Pour réserver une chambre l'utilisateur doit se connecter sur notre application web. Lors de la connexion, en cas de succès un cookie contenant un identifiant est généré dans le navigateur de l'utilisateur, pour une semaine. Il peut alors se rendre sur la page de réservation et réserver en cliquant sur le bouton correspondant.

Une requête est adressée à la page bookRoom.php. Si l'on regarde la première instruction,

on vérifie si l'utilisateur possède un cookie puis on regarde dans la base de données si l'identifiant est connu. Dans le cas où la réponse est positive, la réservation est enregistrée. L'utilisateur peut en avoir la confirmation en se rendant sur la page "Mes Réservations". On s'aperçoit donc que le mécanisme réservation est vulnérable à une attaque CSRF. En effet après que l'utilisateur légitime est obtenu un cookie d'identification, ce dernier peut subir une requête forgée.

Un exemple d'utilisation de la faille est donné sur la page d'index. Dans l'exemple, on suppose que je suis l'utilisateur avec comme nom d'utilisateur "hory". Je n'ai fait aucune réservation comme le montre ma page de réservation ci dessous.



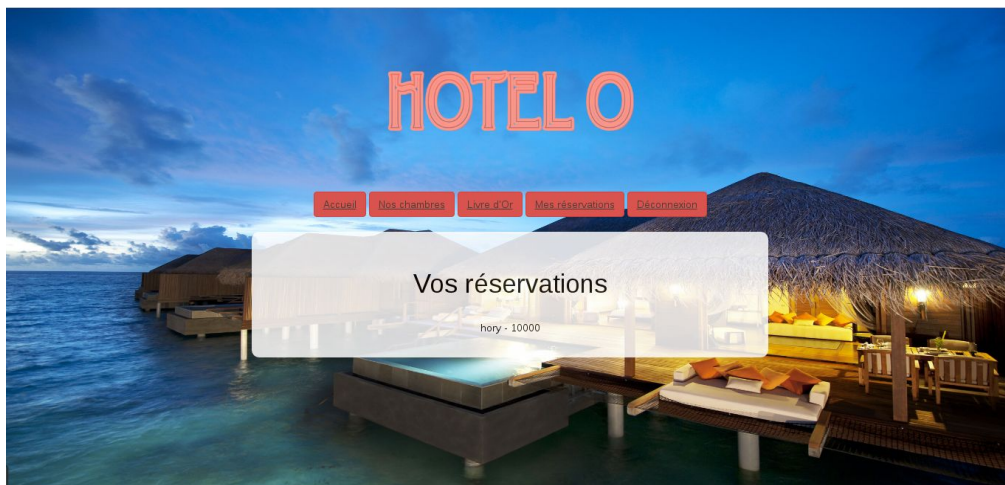
*Page de réservation avant l'attaque CSRF*

On imagine ici que l'attaquant a réussi à injecter une balise image sur la page d'accueil. Voici le code ajouté :

```
<img src='http://secuweb.neoskai.com/bookRoom.php?prix=10000' />
```

Ainsi dès que je vais me rendre sur cette page une requête de réservation sera envoyée au serveur avec mon cookie d'identification. Ce qui va conduire à la réalisation d'une réservation à mon nom alors que ce n'est pas moi qui l'ai faite.

En me rendant sur la page de mes réservations, je peux noter que j'en ai une nouvelle qui est apparue avec la valeur du prix passée dans l'URL.



*Page de réservation après l'attaque CSRF*

c) Prévention possible contre ce type d'attaque

Pour éviter cette attaque, la principale prévention est l'utilisation d'un jeton en plus du cookie d'identification. Lors de la connexion, le jeton sera envoyé au navigateur de l'utilisateur qui le renverra à chaque requête qui sera vérifié en plus de la valeur du cookie. Du coup, la requête créée par l'attaquant ne fonctionnera plus car lors de l'envoi, le jeton ne sera pas joint et la vérification côté serveur rejettera la "fausse requête".

Une autre solution pour éviter ce genre d'attaque est de demander une nouvelle preuve d'authentification lors de la réalisation d'une requête critique, comme par exemple faire une nouvelle demande de mot de passe.

Enfin, côté client il est préférable de supprimer les cookies d'identifications dès que l'on quitte le site web ce qui rend impossible l'attaque.

## I. Utilisation de composants avec des vulnérabilités connues

### 1. Présentation de la faille

L'utilisation de composants à vulnérabilités connues consiste à utiliser des composants tiers (bibliothèques externes, frameworks, modules ...) pour une application web, dans une optique de faciliter le développement. Cependant, comme tout logiciel, ces composants tiers sont vulnérables et exposés à des attaques web. Ils peuvent présenter des failles d'injection SQL, XSS, ou des failles d'authentification par exemple.

Plus un composant devient populaire, et plus il devient la cible d'attaques.

Lorsqu'une faille de sécurité est découverte sur l'un de ces composants tiers, la vulnérabilité est rendue connue du public, et une mise-à-jour du composant est développée dans le but de corriger les failles de sécurité pour qu'elles ne soient plus disponibles dans la nouvelle version. Bien-sûr, les attaquants restent à l'affût lorsque les nouvelles versions apparaissent pour tenter de trouver d'autres failles sur ces versions mises à jour.

### 2. Implémentation de la faille sur notre site

#### a) Localisation de la faille

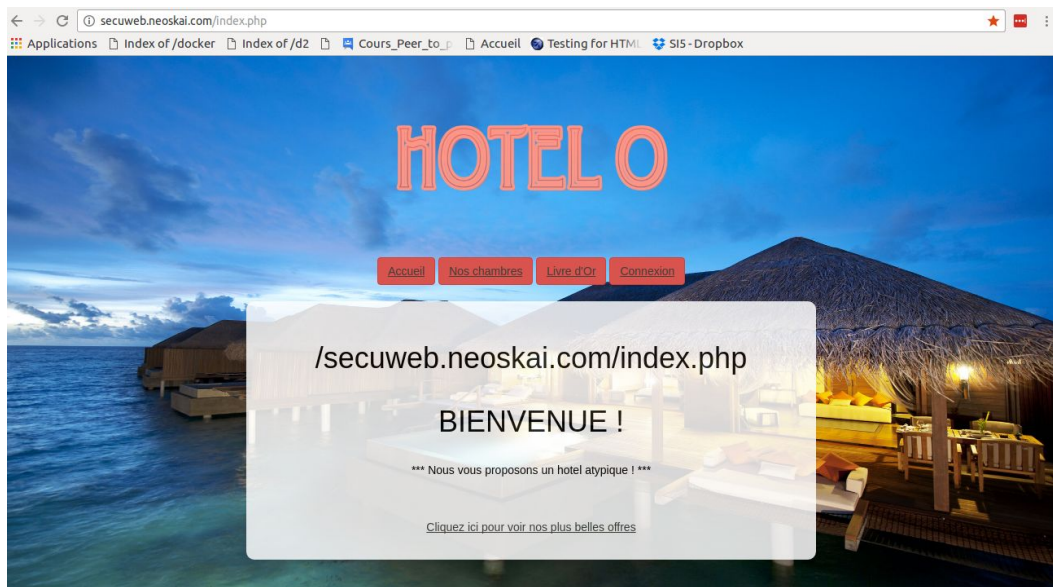
La faille se situe sur la page d'accueil de notre site web.

Plus précisément, elle se situe dans l'URL de la page d'accueil.



# Rapport 1 - Groupe 1

## Sécurité des Applications Web



Page d'accueil du site web

### b) Code pour utiliser la faille

Pour implémenter la faille correspondant à "l'Utilisation de composants à vulnérabilités connues", nous avons décidé d'utiliser une vieille librairie jQuery : la librairie jQuery 1.6.3, qui est vulnérable pour la plupart des navigateurs.

jQuery Version	Is it Vulnerable?
jQuery 2.0.3	Safe
jQuery 2.0.2	Safe
jQuery 2.0.1	Safe
jQuery 2.0.0	Safe
jQuery 1.10.2	Safe
jQuery 1.10.1	Safe
jQuery 1.9.1	Safe
jQuery 1.10.0	Safe
jQuery 1.9.0	Safe
jQuery 1.5.1	Safe
jQuery 1.4.2	Vulnerable
jQuery 1.6.1	Vulnerable
jQuery 1.8.3	Vulnerable
jQuery 1.6.2	Vulnerable
jQuery 1.6.3	Vulnerable
jQuery 1.7.2	Vulnerable
jQuery 1.3.1	Vulnerable

Liste de versions jQuery safe et vulnérables

(<https://domstorm.skepticfx.com/modules?id=529bbe6e125fac0000000003>)

Voici le code que nous avons utilisé pour implémenter notre faille:

```
<script  
src="https://code.jquery.com/jquery-1.6.3.js"  
integrity="sha256-m6oQ4cVjDD3Nm7Rr8AkTzJSzhV1YyUWa6YSD0cVm6Xs=" "  
crossorigin="anonymous"></script>
```

*Importation de la librairie vulnérable*

```
<script type="text/javascript">  
if (window.location.href.indexOf("pseudo=")) {  
    var pseudo_position = location.href.indexOf("pseudo=");  
    var pseudo = location.href.substring(pseudo_position+7);  
    $("#bienvenue").html(pseudo + "<br><br> BIENVENUE !");  
    $("#cadrePrincipal").append('<br><a target="_blank" href="http://secuweb.neoskai.com/chambres.php">Cliquez ici pour voir nos plus  
        belles offres</a>');  
}  
</script>
```

*Script "inoffensif"*

Avec cette faille, des attaques de type "jQuery Selector Injection" sont possibles (<http://www.mjcblog.net/2011/06/jquery-selector-injection/>). En réalité, pour exploiter cette faille, l'importation de l'ancienne librairie jQuery 1.6.3 vulnérable suffit quasiment.

L'explication de la faille est la suivante : jQuery autorise à créer des éléments HTML à la volée grâce à la méthode jQuery() (symbolisée par \$( ) ici). Le fait de créer de nouveaux éléments HTML en utilisant cette technique est commun dans le monde de jQuery. Des problèmes dus à de l'injection de code du côté client peuvent potentiellement émerger si un input malicieux est passé dans cette méthode. Par exemple si on considère le code \$("img='src' onerror=alert('xss attack');>") : le code du côté client sera exécuté, à savoir une alerte qui affiche le message "xss attack". Grâce à ce vecteur, il est donc possible d'exécuter du code malveillant du côté client.

Démonstration:



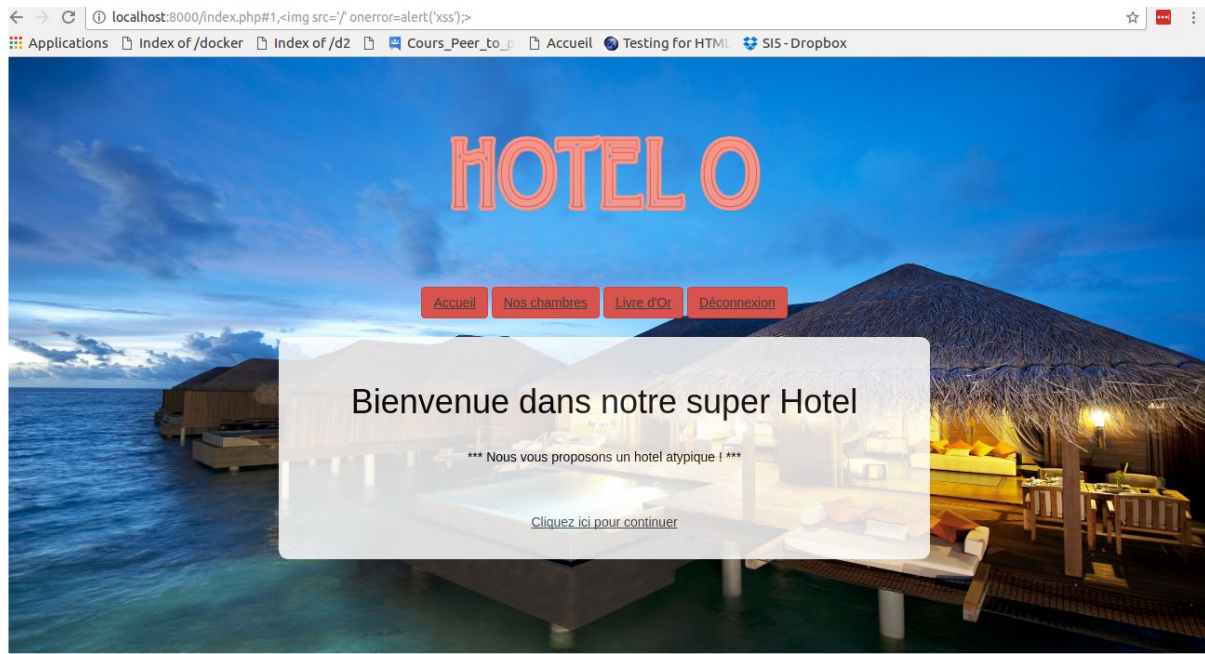
*jQuery Selector Injection*

En se plaçant sur la page d'accueil et en ajoutant à la suite de l'URL :

**#1,<img src='/' onerror=alert('xss');>**

l'attaque d'injection de sélecteur jQuery se produit et affiche une alerte avec le message xss.

On peut constater que cette vulnérabilité n'est plus exploitable lorsque la version de la librairie jQuery est bien supérieure à la 1.6.3 :



*Tentative d'attaque avec un jQuery de version supérieure à 1.6.3*

#### c) Prévention possible contre ce type d'attaque

Pour prévenir ce type d'attaque, il y a plusieurs possibilités.

Maintenir une liste des composants externes utilisés avec leur version correspondante est important si l'on veut éviter ce genre de faille : cela permettra d'identifier les composants vulnérables plus rapidement et d'agir en conséquence efficacement. Surveiller les nouvelles versions disponibles et les vulnérabilités connues est important dans la prévention de ce type de faille. En résumé, en utilisant les dernières versions de tous les composants tiers utilisés, la prévention de cette faille est possible.

Une autre solution serait de ne pas utiliser des composants que nous n'avons pas écrits nous-même, mais cela est coûteux en temps et n'est fait que très rarement ...



## J. Redirections et renvois non validés

### 1. Présentation de la faille

La faille de redirections et renvois non validés consiste à modifier les redirections présentes sur un site internet. Le but étant de renvoyer les utilisateurs d'un site vers un site malicieux type phishing afin de lui soutirer des informations si l'utilisateur ne fait pas attention au site qu'il visite.

### 2. Implémentation de la faille sur notre site

#### a) Localisation de la faille

La faille se situe au niveau du paiement par paypal lors de la réservation de chambre. Le système redirige alors vers <https://paypal.com> qui est indiqué en paramètre du lien.

#### b) Code pour utiliser la faille

Pour utiliser cette faille, l'attaqueur doit réussir à modifier ce paramètre du lien pour rediriger vers un site de phishing de paypal sans que l'utilisateur s'en aperçoive. Il pourrait par exemple, en exploitant une faille d'injection de code ou pour les utilisateurs naïfs par self-XSS, insérer le code suivant :

```
$('#cadrePrincipal a.btn').each(function($el) { $(this).attr('href', 'http://evilpaypal.com') });
```

Ce code va modifier tous les liens qui normalement pointent vers paypal.com vers le site evilpaypal.com

#### c) Prévention possible contre ce type d'attaque

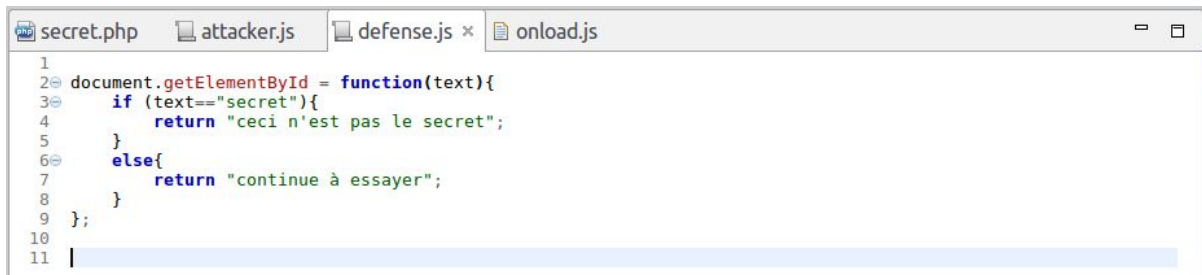
Il existe plusieurs type de préventions contre ce type d'attaques :

1. On peut tout simplement éviter de faire des redirections.
2. Eviter d'utiliser des paramètres utilisateurs pour calculer le lien de destination
3. Si on ne peut pas éviter d'utiliser des paramètres utilisateur, il faut au minimum protéger et vérifier ce/ces paramètre(s) pour s'assurer d'un lien de destination valide et non dangereux.
4. Ne surtout pas utiliser de lien en paramètre, c'est à dire que les paramètres utilisateurs doivent être des variables pour une fonction de calcul et non pas d'une adresse internet complète.

### III. Le Secret

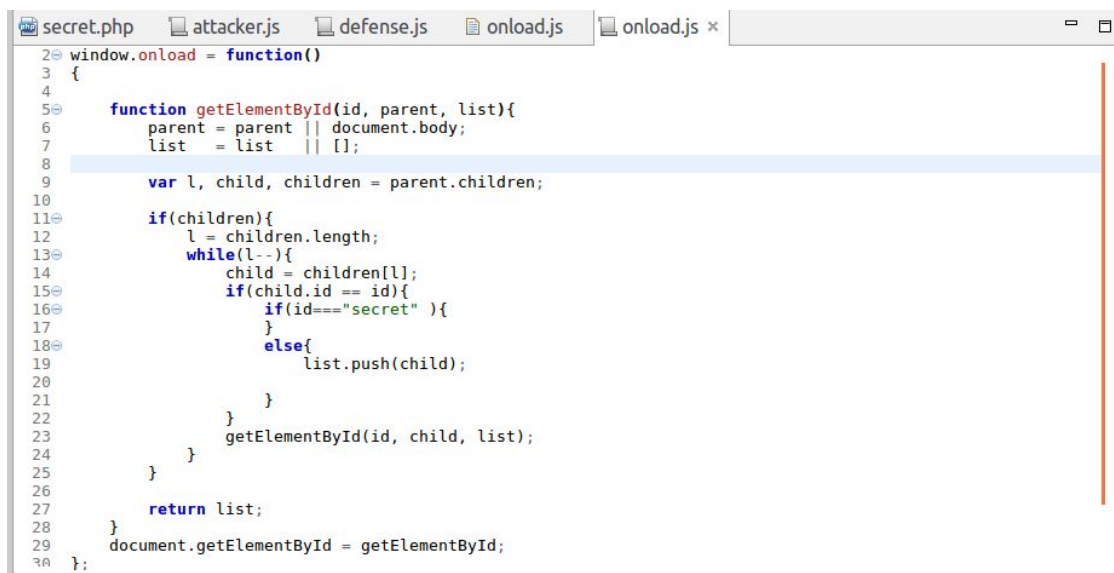
Afin de pouvoir obtenir le secret caché dans notre site, il faut d'abord, récupérer le mot de passe de l'utilisateur "secretholder" et se connecter à nos services. Pour cela, un appel à notre base de données est réalisé afin de comparer le token créé à la connexion et celui stocké pour le compte "secretholder" dans notre base de données. En effet le secret sera caché à l'intérieur d'une div avec l'identifiant "secret" qui ne sera pas affichée sur le code source car elle possède la propriété *css, display* qui vaut "none". Cependant pour y accéder, il faudra faire un peu plus que juste récupérer l'élément avec la méthode *getElementById*. Le code de cette méthode a été redéfini deux fois.

La première dans le fichier "defense.js" de la manière suivante:



```
secret.php  attacker.js  defense.js x  onload.js
1
2 document.getElementById = function(text){
3     if (text=="secret"){
4         return "ceci n'est pas le secret";
5     }
6     else{
7         return "continue à essayer";
8     }
9 };
10
11
```

Dans un premier temps, la fonction *getElementById* ne permettra pas d'accéder aux éléments de la page et toute tentative de l'attaquant sera infructueuse. Par contre, cette dernière sera redéfinie une deuxième fois, lorsque la page sera complètement chargée. Ainsi, cette nouvelle implémentation permettra de faire la recherche des objets, à condition que l'ID soit différent de "secret". Le nouveau code pour cette fonction est défini de la manière suivante:



```
secret.php  attacker.js  defense.js  onload.js  onload.js x
2 window.onload = function()
3 {
4
5     function getElementById(id, parent, list){
6         parent = parent || document.body;
7         list = list || [];
8
9         var l, child, children = parent.children;
10
11         if(children){
12             l = children.length;
13             while(l--){
14                 child = children[l];
15                 if(child.id == id){
16                     if(id=="secret" ){
17                         }
18                     else{
19                         list.push(child);
20                     }
21                 }
22                 getElementById(id, child, list);
23             }
24         }
25     }
26
27     return list;
28 }
29 document.getElementById = getElementById;
30
```

Pour que l'attaquant puisse accéder à l'information contenu dans la div, il faut qu'il exécute son code après que la nouvelle implémentation de `getElementById` soit en place. Dans la mesure où nous vérifions si le paramètre est égal à `"secret"`, il suffit à l'attaquant de créer un nouvel objet qui implémente la méthode `toString` car lorsque la comparaison est faite, JavaScript fait un appel implicite à la fonction `toString` de l'objet.

Finalement, la commande pour que l'attaquant puisse récupérer le secret est la suivante:

```
$ var obj = {toString:function(){return "secret",}};  
$ setTimeout(function(){window.alert(document.getElementById(obj)[0].innerHTML);},2000);
```

Après l'exécution de ce code, une fenêtre affichera, après 2 secondes, le contenu de la div avec l'id `"secret"`.