

La sécurité des applications web

La sécurité des applications web: cette question est d'autant plus cruciale que les attaques informatiques deviennent de plus en plus sophistiquées et peuvent avoir des conséquences désastreuses pour les entreprises et les utilisateurs finaux. Nous examinerons les failles fréquentes telles que l'injection SQL, le Cross-Site Scripting (XSS) et les défauts de contrôle d'accès, qui représentent des vecteurs d'attaques courants exploités par les cybercriminels pour compromettre la sécurité des applications web.

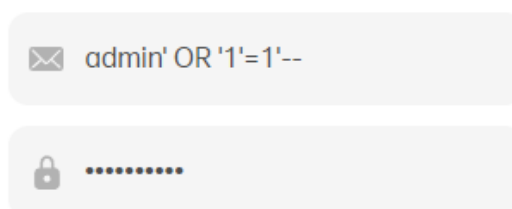
Failles fréquentes

Injection SQL

Les attaques par injection SQL se produisent lorsque des données malveillantes sont injectées dans une base de données via une application web. Cela peut conduire à des fuites de données, à la modification ou à la suppression de données.

On peut prendre l'exemple d'une page d'authentification, l'authentification marche avec cette requête SQL.

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

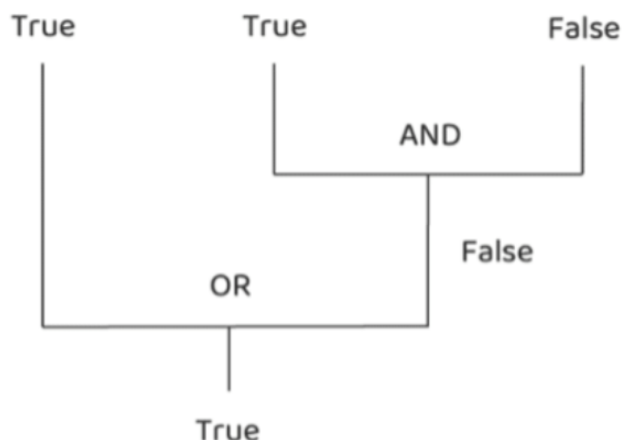


The image shows a simplified login interface. It consists of two rounded rectangular input fields. The top field has an envelope icon on the left and contains the text 'admin' OR '1'='1'--'. The bottom field has a padlock icon on the left and contains a series of dots representing a password.

[Mot de passe oublié ?](#)

`1=1` renvoie toujours TRUE.

Les `--` permet de mettre en commentaire du code SQL.



Dans ce cas, la requête SQL va être :

```
SELECT * FROM users WHERE username='admin' OR 1=1;
```

Exemple :

CVE-2023-34362 : **exploitation d'une vulnérabilité dans MOVEit Transfer**

Il s'agit d'un logiciel de transfert de fichier or, on ne peut pas avoir plus d'information concernant les détail techniques.

Cross-Site Scripting (XSS)

Cette attaque est au même titre que l'injective SQL, mais ici

Les attaques XSS injectent des scripts malveillants dans les pages web vues par d'autres utilisateurs, ce qui peut entraîner le vol de cookies, de sessions ou de données personnelles.

On peut prendre l'exemple d'un formulaire de contact.

 ✓

Dans le cas de PHP, le code de renvoie peut être

```
console.log("Bonjour, ".$name);
```

Si dans notre case on écrit

Nom	✓
Lea; cat ./controller/controllerForm.php	

On va donc nous afficher le fichier "controllerForm.php".

Contrôle d'accès défaillant

Accès non autorisé a des données et/ou modification des ses données.

On peut prendre l'exemple de ce code qui permet d'accéder à d'autre compte utilisateurs depuis l'url

```
<?php
$user_id = $_GET['user_id'];
$user = getUserDetails($user_id);
echo $user;
?>
```

l'url va contenir l'id de l'utilisteur, donc si l'on change son id, on peut accéder à n'importe quel compte utilisateur.

`index.php?user_id= 2`

Si le compte admin est associé à l'id 1, avec cet url on peut accéder à son compte:

`index.php?user_id= 1`

Des données peuvent etre modifiées si on accède à compte utilisateur autre, on peut modifier ses données si le script comporte ce code

```
<?php
$user_id = $_GET['user_id'];
$new_details = $_POST['new_details'];
updateUserDetails($user_id, $new_details);
?>
```

avec l'url joint ci-dessus, on nous permet encore de faire une requete POST, PUT ou DELETE.

Solution a ses failles

Injection SQL

- Requête paramétrées

Afin de créer une requête paramétrée on lui passe une collection de paramètres dont le type et la taille ont été définis.

Exemple :

```
SELECT nom FROM Clients WHERE Nom = @nomClient
```

Exemple en C# .Net

```
String query = "SELECT account_balance FROM user_data WHERE user_name =  
try {  
    OleDbCommand command = new OleDbCommand(query, connection);  
    command.Parameters.Add(new OleDbParameter("customerName", CustomerName)  
    OleDbDataReader reader = command.ExecuteReader();  
    // ...  
} catch (OleDbException se) {  
    // error handling  
}
```

- Utiliser des comptes de bdd avec des droits spécifique
Eviter de mettre tous les droits sur le compte qui communique avec la bdd depuis le site.
Et limiter au stricte minimum les privilèges

Cross-Site Scripting (XSS)

- Filtrer les données reçu et sorties

On peut retrouver des fonctions comme `htmlentities()` ou `htmlspecialchars()` qui vont venir encoder les caractères. Ces fonctions encodent tous les caractères spéciaux (<, >, ", etc.) mais aussi les caractères accentués (é, è, à, ù, etc.). Cependant, `htmlentities()` encode également les simples guillemets, ce qui peut être problématique si vous utilisez la chaîne vulnérable dans un attribut d'une balise HTML qui est sous la forme `attribut='valeur'`. Pour éviter cela, vous pouvez ajouter le paramètre `ENT_QUOTES` à la fonction `htmlentities()` ou `htmlspecialchars()`

Exemple d'utilisation

```
$userInput = "<script>alert('XSS');</script>";  
$safeOutput = htmlentities($userInput, ENT_QUOTES, 'UTF-8');
```

- Mettre des mots clés interdits

vous pouvez utiliser la méthode `replace()` avec une expression régulière pour supprimer les balises `<script>`

Exemple

```
var userInput = "<script>alert('XSS');</script>";
var safeOutput = userInput.replace(/<script\b[^\>](?:?!<\script>)<[^\>]
```

- limiter les entrée

En fonction de ce qu'il doit être renseigné, limiter les entrée avec `maxlength()` et `minlength()` et autorisé seulement les caracteres dont l'utilisateur va avoir besoin avec `addEventListener()`.

Exemple pour entrer un numéro de téléphone :

```
document.getElementById("phone").addEventListener("input", function (event) {
    var value = event.target.value;
    var pattern = /^[0-9]+$/; // Only allow numbers
    if (!pattern.test(value)) {
        event.target.value = value.replace(/^[0-9]/g, "");
    }
});
```

Contrôle d'accès défaillant

Pour commencer, un identifiant unique pour les utilisateurs qui se connectent est important afin qu'ils soient d'autant plus difficile à trouver.

Utiliser des cookies sécurisés, avec `HttpOnly` qui empêche le code JS d'accéder au cookie, de plus, avec `secure` , les données sont envoyées seulement dans le cas où il s'agit de requêtes HTTPS.

```
// Démarre la session
session_start();

// Génère un identifiant de session aléatoire
$session_id = bin2hex(random_bytes(32));

// Définit l'identifiant de session sécurisé dans un cookie
session_set_cookie_params([
    'lifetime' => 3600, // Durée de vie de la session en secondes
    'path' => '/',
    'domain' => 'example.com',
    'secure' => true, // Le cookie ne sera envoyé que via HTTPS
    'httponly' => true, // Le cookie est accessible uniquement via HTTP
```

```

    'samesite' => 'Strict' // Protection contre les attaques CSRF
]);

// Définit l'identifiant de session
session_id($session_id);

// Démarre la session avec l'identifiant généré
session_start();

```

Afin d'éviter des incidents liés au contrôle d'accès défaillants il est important d'attribuer un rôle et de bien séparer en fonctions des droits liés au rôle. Il faut régulièrement vérifier si l'utilisateur a les droits d'accès en fonctions des actions qu'il fait.

```

// Fonction pour vérifier les autorisations basées sur le rôle
function check_permission($required_role) {
    session_start();
    if (!isset($_SESSION['role']) || $_SESSION['role'] !== $required_role) {
        // Redirige l'utilisateur vers une page d'erreur ou une page d'accès refusé
        header("Location: access_denied.php");
        exit();
    }
}

```

conclusion

En conclusion, la sécurité des applications web est un enjeu majeur pour les professionnels de l'IT et nécessite une attention constante. Les failles telles que les injections SQL, le Cross-Site Scripting et les contrôles d'accès défaillants représentent des risques significatifs pour l'intégrité des systèmes d'information. Heureusement, des solutions existent pour mitiger ces risques, telles que l'utilisation de requêtes paramétrées, le filtrage des entrées et sorties, et l'implémentation de politiques de sécurité strictes.

Bibliographie

<https://www.cert.ssi.gouv.fr/alerte/CERTFR-2023-ALE-005/>

blog.qualys.com

<https://www.vaadata.com/>

<https://www.cert.ssi.gouv.fr/>

https://owasp.org/Top10/A00_2021_Introduction/

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html