La sécurité

Table des matières

I. Contexte

Durée : 1 h

Environnement de travail : Repl.it

Prérequis : Bases de PHP

Contexte

Lors de la création d'une application web, un point essentiel est à garder à l'esprit durant le développement : notre site est-il sécurisé ? Comment faire pour le sécuriser ? Quelles sont les conséquences s'il n'est pas assez sécurisé ? À partir du moment où les utilisateurs peuvent utiliser une ressource de l'application, celle-ci devient vulnérable au piratage. Lorsqu'une application web subit une brèche de sécurité, ce sont des millions de données utilisateurs qui sont compromises, allant des mots de passe à des numéros de cartes bancaires. C'est pourquoi la sécurité est un point central du développement et que nous devons lui apporter toute notre attention.

Dans ce cours, nous allons apprendre quelles sont les attaques les plus répandues et comment s'en prémunir.

II. Introduction à la sécurité

Objectifs

- Connaître le dogme de la sécurité
- Connaître OWASP

Mise en situation

On sait tous que la sécurité est importante sur une application, mais ce terme est souvent très abstrait. Pour beaucoup de développeurs, "protéger son site" reste une notion connue en théorie, mais très floue dans la pratique. Posons donc les bases de ce que la sécurité des applications web désigne vraiment et quelles sont les ressources à disposition des développeurs.

Never Trust User Input

Ne jamais faire confiance à une saisie de l'utilisateur est le mot d'ordre de tout développeur attentif à la sécurité de son site. Une donnée renseignée par un utilisateur est la première et principale source des failles de sécurité d'une application. Parfois, cette donnée peut être simplement erronée : sur un blog qui récupère un article selon le numéro de l'article, si une chaîne de caractères est indiquée à la place de ce nombre, l'application retournera une erreur pouvant éventuellement contenir des informations sur le serveur ou les fichiers utilisés.

Cette donnée peut aussi être malveillante : il est parfois possible d'injecter du code afin de prendre la main sur une application ou d'exposer des données sensibles, comme les informations d'une base de données. Il est nécessaire de toujours contrôler les données saisies par les utilisateurs.

OWASP

De nombreuses informations sur les vulnérabilités existent sur le Web. Différentes bases de données, principalement **CVE** (*Common Vulnerabilities and Exposures*) et **NVD** (*National Vulnerability Database*), répertorient ces vulnérabilités, allant des serveurs aux applications.

OWASP (*Open Web Application Security Project*) est une organisation mondiale et *open source* qui se consacre à l'amélioration de la sécurité des logiciels. Elle a pour mission principale de fournir des informations sur les différentes vulnérabilités du Web, ainsi que les préventions associées. L'un des projets de OWASP est le « Top Ten OWASP », un document implémentant un classement des risques les plus critiques pour les applications web, ainsi que certaines recommandations de prévention.

Remarque

D'autres moyens de prévention autres que ceux indiqués sur ce classement existent, ainsi que des vulnérabilités supplémentaires, répertoriées également pour certaines sur le site d'OWASP. Ce document peut ainsi être une base de sensibilisation pour la majorité des développeurs, mais il n'est pas suffisant pour répertorier l'ensemble des failles et leurs préventions possibles.

Exemple

Une application non protégée peut avoir de graves conséquences. Capital One Financial Corporation, une banque située aux États-Unis, s'est vue voler des données de plus de 100 millions de clients en 2019[1]. Yahoo[2], bien connu en tant que moteur de recherche, s'est fait voler à plusieurs reprises les informations des comptes clients (nom, adresse e-mail, mots de passe, etc).

Syntaxe **À retenir**

- Une application web est rapidement vulnérable aux interactions avec les utilisateurs. Il ne faut jamais faire confiance aux données utilisateurs et il faut les contrôler.
- De nombreuses vulnérabilités sont référencées dans des bases de données de référence et plusieurs organismes, tels que OWASP, permettent d'y apporter des informations complémentaires, des solutions, ou des outils les concernant.

Complément

OWASP [en][3]

OWASP Top Ten [en][4]

Attaque Capital One [fr][5]

Attaque Yahoo [en][6]

Exercice : Appliquez la notion

[solution n°1 p.21]

Pour répondre à ces questions, vous vous baserez sur le document de référence des 10 principaux risques de sécurité des applications web[7].

Exercice

1 https://www.latribune.fr/entreprises-finance/banques-finance/piratage-bancaire-comment-100-millions-de-personnes-ont-ete-touchees-par-un-vol-de-donnees-824884.html
2 https://www.nytimes.com/2017/10/03/technology/yahoo-hack-3-billion-users.html
3 https://owasp.org/
4 https://owasp.org/www-project-top-ten/
5 https://www.latribune.fr/entreprises-finance/banques-finance/piratage-bancaire-comment-100-millions-de-personnes-ont-ete-touchees-par-un-vol-de-donnees-824884.html
6 https://www.nytimes.com/2017/10/03/technology/yahoo-hack-3-billion-users.html
7 https://owasp.org/www-project-top-ten/

Quel type de faille est exploité lorsqu'un assaillant exécute du code dans une requête SQL ?

- ○ Injection
- ○ Broken Authentication
- ○ XML External Entities (XXE)
- ○ Insecure Deserialization

Exercice

Quel type de faille est exploité lorsqu'un assaillant accède à des données en clair stockées sur le serveur (mots de passe, numéros de carte bancaire) ?

- ○ Injection
- ○ Broken Authentication
- ○ Sensitive Data Exposure
- ○ Cross-Site Scripting XSS

IV. Modifications des données utilisateurs

Objectif

- Comprendre comment une application peut être vulnérable

Mise en situation

Les applications web peuvent très vite devenir vulnérables lorsque nous acceptons que nos utilisateurs puissent saisir des données. Mais comment cela se passe-t-il ? Nous allons voir cela ici, avec l'utilisation des variables en `GET` et `POST`.

Variable GET

Pour récupérer des données utilisateurs, plusieurs méthodes existent. Une des plus connues et utilisées est la variable `$_GET`.

Exemple

Prenons l'exemple simple de la page d'un article dont les fonctionnalités d'administration sont accessibles seulement par un administrateur. Si on utilise un paramètre d'URL accessible en `GET` pour définir si c'est un administrateur ou non, on aurait l'URL `https://www.example.com/article.php?id=1` et le code suivant :

```php
if ($_GET['admin'] === '1') {
    // On affiche les fonctionnalités d'administration
}
```

La valeur de `$_GET['admin']` peut être facilement manipulée par un utilisateur, étant donné qu'elle est indiquée dans l'URL. Ainsi, si un utilisateur veut accéder aux fonctions d'administration, il suffit qu'il accède à l'URL suivante : `https://www.example.com/article.php?id=1&admin=1`.

Variable POST

Il est commun de penser que l'utilisation de données en `POST` est plus sécurisée, car elles ne sont pas accessibles dans l'URL. Ceci est entièrement faux. Plusieurs outils permettent effectivement de simuler une requête envoyée à une application en indiquant des données en `POST`. Pour cela, on peut utiliser une simple commande `cURL` ou des outils tels que Postman ou Insomnia. Tous trois permettent d'effectuer des requêtes HTTP.

Exemple

Imaginons que nous ayons une URL permettant de supprimer un article accessible via l'URL `https://www.example.com/article.php?id=1&action=delete`, et le code suivant :

```php
if ($_POST['admin'] === '1') {
    // On autorise la suppression
}
```

Si un utilisateur souhaite appeler cette page en `POST` avec la bonne valeur, il lui suffit d'exécuter cette requête `cURL` :

```
curl --request POST \
  --url 'https://www.example.com/article.php?id=1&action=delete' \
  --header 'content-type: application/x-www-form-urlencoded' \
  --data admin=1
```

Syntaxe **À retenir**

- Les données envoyées par un navigateur, tel qu'avec les méthodes `GET` et `POST`, peuvent être facilement manipulées par les utilisateurs. Elles peuvent engendrer des problèmes de sécurité, quelle que soit la façon dont elles sont récupérées. Il est donc primordiale de ne surtout pas les utiliser pour accorder des privilèges aux utilisateurs, mais seulement pour une utilisation simple, et de les sécuriser le cas échéant.

V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

Question

[solution n°2 p.21]

Nous sommes dans le cas d'une page permettant d'afficher une foire aux questions et de l'administrer. Voici le code PHP écrit.

Fichier `index.php` :

```php
<?php

$posts = [
    1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
    ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
    ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
];
?>
```

1 https://repl.it/

```
    <h1>Foire aux questions</h1>

<?php foreach ($posts as $id => $post) : ?>
    <h2>&laquo; <?= $post['title'] ?> &raquo; publié le <?= $post['createdAt']->format('d/m/Y') ?></h2>
    <p><a href="supprimer.php?id=<?= $id ?>&admin=1">Supprimer</a></p>
<?php endforeach;
```

Fichier `supprimer.php` :

```
<?php

if ($_GET['admin'] !== '1') {
    die('Vous n\'avez pas le droit de supprimer des articles !');
}

$posts = [
    1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
    ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
    ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
];

unset($posts[$_GET['id']]);

header('Location: index.php');
```

Faisons en sorte de sécuriser cette page selon les règles suivantes :

1. Modifiez l'URL de suppression d'un article pour ne plus passer le paramètre `admin` dans l'URL.
2. Modifiez la page `supprimer.php` pour utiliser les sessions, plutôt que `$_GET` (on partira du principe que la variable de session aura été correctement initialisée).

VI. Les erreurs de données

Objectifs

- Connaître les impacts possibles en cas de données non concordantes
- Savoir se prémunir des erreurs de données

Mise en situation

Comme nous l'avons vu, les données envoyées par un navigateur (et donc par un utilisateur) peuvent être manipulées et impacter la sécurité d'une application web. Sans aller jusqu'à imaginer une faille aussi grave qu'une élévation de privilège, nous pouvons être régulièrement confrontés à des données incohérentes par rapport à ce qui pourrait être attendu, comme une date de naissance mal formatée, par exemple. Mais quels peuvent en être les impacts et comment s'en prévenir ?

Remarque

Les exemples seront donnés dans le cas d'une variable passée dans l'URL et récupérée avec la superglobale `$_GET`, car c'est le plus simple à manipuler. Mais comme cela a été vu précédemment, toutes les méthodes de récupération de variables peuvent être modifiées.

Dépassement de valeurs

L'une des premières erreurs récurrentes est un manque de contrôle sur les dépassements de valeurs, comme des dates (32/02/22020), l'âge d'un utilisateur (-5 ans) ou encore l'affichage de données.

Exemple

Admettons qu'une page d'une application affiche une liste d'éléments par lot de 10, 20 ou 50. Ce nombre est défini directement dans un paramètre de l'URL `https://www.example.com/list?count=10`. Le code associé est le suivant :

```php
$count = $_GET['count'];
for ($i = 0; $i < $count; $i++) {
    echo 'Item n°'.$i.'<br>';
}
```

Un utilisateur peut modifier cette valeur et définir, par exemple, la valeur **1000**. Si chaque élément nécessite des requêtes vers une base de données ou d'autres actions, cela peut nuire à la performance de l'application. Il est donc nécessaire de gérer les cas où la valeur est incorrecte. Pour le cas de l'âge, nous pourrions vérifier si la valeur est supérieure à 0 ou, pour notre exemple actuel, voici une solution possible :

```php
$count = $_GET['count'];
$count = in_array($count, [10, 20, 50]) ? $count : 10;
for ($i = 0; $i < $count; $i++) {
    echo 'Item n°'.$i.'<br>';
}
```

Ici, on vérifie que la valeur du nombre d'éléments par page est bien comprise dans la liste des valeurs suivantes : 10, 20, 50.

Problèmes de typage

Une seconde problématique récurrente est le typage des données. Certaines fonctionnalités attendent un type précis de données : un identifiant d'article attendra par exemple un nombre, un nom d'utilisateur une chaîne de caractères, etc. En cas d'incohérence avec le type de données attendu, l'application pourrait renvoyer une erreur. Il sera donc nécessaire de vérifier le typage des données. Pour ce faire, il existe plusieurs solutions :

- Typer les paramètres de fonction. Depuis PHP 7, il est possible de typer les données d'entrée et de retour des différentes fonctions.

```php
public function getEmail(): ?string
public function setEmail(?string $email): self
```

- Vérifier le type de données avec l'utilisation de fonctions (`is_numeric`, fonctions `ctype_*`, etc).

```php
is_numeric($_GET['count']);
ctype_digit($_GET['count']);
```

Absence de données

La dernière erreur récurrente sur les données concerne l'existence ou l'absence des données. L'utilisateur pourrait modifier la valeur passée en paramètre pour passer une valeur inexistante ou tout simplement pas de valeur du tout.

Exemple

Reprenons l'URL `https://www.example.com/list?count=10` et supprimons le paramètre `count` entièrement : un message d'erreur s'affiche.

```
Notice: Undefined index: count in [...][...] on line 3
```

La fonction `isset()` permet de vérifier si une donnée existe. Il convient ensuite de traiter cette problématique par un message d'erreur ou une valeur par défaut.

```php
$count = $_GET['count'] ?? 10;
$count = in_array($count, [10, 20, 50]) ? $count : 10;
for ($i = 0; $i < $count; $i++) {
    echo 'Item n°' . $i;
}
```

Syntaxe **À retenir**

L'opérateur ??1 (Opérateur de coalescence des nuls) est un opérateur de comparaison entre deux valeurs, celle de gauche et celle de droite. Il renvoie la valeur de droite si la valeur de gauche est dite nullish (null ou undefined), sinon, renvoie la valeur de gauche :

```
<code>
<?php
$var;  //variable undefined
null ?? 'test'; //renvoi test
$var ?? 'test'; //renvoi test
1234 ?? 'test'; //renvoi 1234
</code>
```

Remarque

En programmation, il existe un moyen de gérer les erreurs php : les exceptions. En effet, si une exception est détectée, cette instruction va récupérer l'erreur, le traitement en cours sera suspendu et celle-ci tentera de gérer l'erreur. Dans le cas contraire, le code sera exécuté normalement et le bloc catch n'interviendra pas.

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Operators/Nullish_coalescing_operator

```php
  <?php
try {
   // Essayer de faire quelque chose
}
catch(Exception $e) {
   // Si une erreur se produit, on arrive ici
}
```

- Les données doivent être vérifiées avant utilisation. Cela consiste à vérifier l'existence, le typage et enfin les valeurs possibles.

VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

1 https://repl.it/

Question

[solution n°3 p.22]

Nous sommes dans le cas d'une page permettant de récupérer un article et de l'afficher. Voici le code PHP écrit :

```php
<?php

$posts = [
  1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
  ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
  ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
];

$id = $_GET['id'];

$post = $posts[$id];

echo 'L\'article "'.$post['title'].'" a été publié le '.$post['createdAt']->format('d/m/Y').'.';
```

Faisons en sorte de sécuriser cette page selon les règles suivantes :

1. La valeur de `id` doit être un entier strictement supérieur à 0.
2. L'article doit exister.

Dans ces deux cas, on affichera un simple message *Une erreur est survenue. Merci de réessayer.*.

VIII. Les attaques XSS et CSRF

Objectif

- Savoir ce que sont les attaques XSS et comment s'en protéger
- Savoir ce que sont les attaques CSRF et comment s'en protéger

Mise en situation

Lorsque nous sommes amenés à afficher du contenu saisi par un utilisateur, il n'est pas toujours possible de vérifier les données. Dans le cadre de l'affichage de commentaires sur un blog, par exemple, il n'est pas possible de vérifier toutes les possibilités d'affichage. Comment cela peut-il rendre une application vulnérable ?

Attaques XSS

Lors de l'affichage de données utilisateurs non contrôlées, l'application peut être sujette à une faille dite **XSS**. Une faille XSS est simplement une injection de code HTML visant à être affichée dans le navigateur d'un internaute pour effectuer des opérations à son insu.

Exemple

L'exemple suivant, une fois exécuté, appliquera la balise HTML `strong` au texte :

```php
$commentaireUtilisateur = '<strong>Mon commentaire</strong>';
echo $commentaireUtilisateur;
// Affichera "Mon commentaire" en gras.
```

Risques possibles

Dans l'exemple précédent, le risque n'est pas grand. Cependant, en prenant cet exemple-ci :

```php
$commentaireUtilisateur = "<script>alert('Une faille XSS')</script>";
echo $commentaireUtilisateur;
```

À l'exécution, un message d'alerte apparaît. Le risque de cette faille est donc l'exécution de code sur les navigateurs des utilisateurs visitant la page. Ici, ce n'est qu'un simple **alert**, mais il est possible d'exécuter du code complexe allant de la récupération de cookies à une redirection vers un site de *phishing* permettant de récupérer des coordonnées bancaires.

Solutions

Afin de se prémunir contre une faille XSS, la protection est simple. Il est nécessaire de toujours traiter ce qui va être affiché en filtrant les caractères spéciaux HTML.

Plusieurs outils sont proposés pour cela en PHP :

- La fonction PHP native `htmlspecialchars()` permet de convertir les caractères HTML spéciaux afin qu'ils ne soient pas directement interprétés par les navigateurs.
- L'utilisation d'un moteur de template tel que Twig permet de sécuriser par défaut l'affichage des données.

Attention **Valeurs par défaut de la fonction htmlspecialchars()**

La fonction `htmlspecialchars()` de PHP accepte en tout quatre paramètres :

- La chaîne à encoder,
- Un masque permettant d'indiquer comment les guillemets sont gérés (on peut utiliser des constantes pour le décrire),
- L'encodage de la chaîne (depuis PHP 5.4.0, c'est UTF-8),
- Un booléen indiquant si on veut encoder les entités HTML existantes (on peut l'omettre le plus souvent).

Toutes les valeurs conviennent parfaitement à l'échappement de caractères HTML, hormis pour la seconde. En effet, la valeur par défaut du masque, soit `ENT_COMPAT | ENT_HTML401`, n'encode pas les simples guillemets `'`. Pour les prendre en charge, il ne faudra pas oublier de changer le masque pour `ENT_QUOTES`.

En indiquant le masque `ENT_QUOTES`, par exemple, le filtrage permet de convertir les caractères spéciaux comme ceci :

Caractère	Remplacement
&	&amp;
"	&quot;
'	&#039;
<	&lt;
>	&gt;

```php
$commentaireUtilisateur = "<strong>Mon commentaire</strong>";
echo htmlspecialchars($commentaireUtilisateur, ENT_QUOTES);
// Affichera "<strong>Mon commentaire</strong>" et non "Mon commentaire" en gras.
```

Différence entre htmlentities() et htmlspecialchars()

La fonction `htmlspecialchars()` convertit les caractères spéciaux d'une chaîne en entités HTML. Imaginons que vous ayez une balise < br > dans votre chaîne, htmlspecialchars()la convertira en br. Les caractères n'ayant aucune signification en HTML, quant à eux, ne sont pas pris en compte. Par ailleurs, la fonction htmlentities(), elle, convertira tous les caractères spéciaux d'une chaîne en entités HTML, même ceux n'ayant aucun rapport avec le langage HTML.

Syntaxe **À retenir**

- Les attaques XSS ont lieu lors de l'affichage de données non protégées. Le principe est d'exécuter du code JavaScript sur les navigateurs.
- Les moteurs de template ou certaines méthodes natives PHP permettent de s'en protéger.

Attaques CSRF

Le CSRF, qui signifie Cross Site Request Forgery est un type d'attaque assez courant sur internet. L'attaque nécessite qu'un utilisateur (innocent) soit connecté, le pirate va envoyer une requête http à l'utilisateur innocent, la requête peut être cachée dans un formulaire, une image, un mail, etc. Cette requête effectuera des actions internes non désirées sur un site par les utilisateurs sans qu'ils s'en aperçoivent. L'attaque ne marche uniquement si une session utilisateur est ouverte !

Mise en situation

Vous êtes connecté à votre compte bancaire lorsque vous recevez un mail de votre conseiller urgent ! Dans le mail, une simple image inoffensive, mais cette image contient une requête http qui va utiliser votre session utilisateur car vous êtes resté connecté. Et que pourrait bien contenir cette requête ? Et bien c'est très simple, elle contient un code qui va effectuer un petit virement de votre compte à celui qui attaque, et cela va sûrement vider le compte en question… Mais attention, le pirate peut aussi attaquer votre conseiller qui est administrateur, et donc pirater plusieurs comptes en même temps. Mais bien évidemment toutes les banques sont au courant de cette attaque et en sont protégées, nous allons justement voir comment faire.

Solution

Pour s'en protéger, il y a plusieurs solutions possible :

- Contrôler le comportement des cookies de session avec SameSite
- Ajouter une clé d'authentification unique (token anti-CSRF)
- Ajouter la double authentification / demander vérification par mail

Complément

Owasp XSS [en][1]

1 https://owasp.org/www-community/attacks/xss/

IX. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°4 p.22]

Nous sommes dans le cas d'une page permettant d'afficher une foire aux questions. Voici le code PHP écrit.

Fichier `index.php` :

```php
<?php


$posts = [
    1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
    ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
    ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
    ['title' => 'Que se passe-t-il lorsqu\'on fait un <script>alert(\'Raté\')</script> ?', 'createdAt' => new \DateTime('2020-01-31')],
];
?>


<h1>Foire aux questions</h1>

<?php foreach ($posts as $id => $post) : ?>
    <h2> <i> <?= $post['title'] ?> </i> publié le <?= $post['createdAt']->format('d/m/Y') ?> </h2>
<?php endforeach;
```

Faites en sorte de sécuriser cette page selon la règle suivante :

1. Protégez l'affichage du titre de la question contre la faille XSS.

X. L'envoi de fichier

Objectif

- Connaître les risques et préventions liés au téléchargement de fichiers

Mise en situation

Les fichiers sont un type particulier de données venant des utilisateurs. Ils ne sont pas gérés de la même façon par les serveurs et nécessitent un traitement particulier.

Risques

Lors du téléchargement de fichiers sur une application web, ces derniers entraînent un risque de vulnérabilité important. Les fichiers peuvent comporter du code malveillant qui sera exécuté dans l'application afin de modifier sa logique et son bon fonctionnement ou encore de récupérer des données.

1 https://repl.it/

L'exemple suivant est un fichier PHP scannant les fichiers d'un répertoire. Il ne fait que scanner, mais pourrait récupérer des données, modifier des portions de code, ou même supprimer tous les fichiers, rendant l'application non fonctionnelle :

```
<?php
        $dir = '/tmp'; //Nom de votre dossier
        $files = scandir($dir);//Retourne tous les dossiers et les fichiers contenus dans un tableau
        print_r($files);//Affiche le contenu du dossier
    ?>

```

Solutions

Tout d'abord, il convient de valider qu'il n'y a pas d'erreur dans la superglobale `$_FILES` concernant le fichier :

```
function isUploadSuccessful(array $uploadedFile): bool {
    return isset($uploadedFile['error']) && $uploadedFile['error'] === UPLOAD_ERR_OK;
}

if (!isUploadSuccessful($_FILES['uploaded_file'])) {
    throw new RuntimeException('Error while uploading file.');
}
```

Ensuite, nous pouvons vérifier que la taille du fichier ne dépasse pas la taille maximale :

```
function isUploadSmallerThan2M(array $uploadedFile): bool {
    return $uploadedFile['size'] < 2000000;
}

if (!isUploadSmallerThan2M($_FILES['uploaded_file'])) {
    throw new RuntimeException('File is too big.');
}
```

La gestion des fichiers se fait avec un contrôle rigoureux du type de fichier autorisé, afin d'interdire le téléchargement de fichiers dangereux comme des fichiers PHP ou shell. Pour cela, il faut se baser sur le type MIME du fichier, il ne faut pas faire confiance à son extension ou à la valeur de `$_FILES['uploaded_file']['type']`.

```
function isMimeTypeAuthorized(array $uploadedFile): bool {
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $mimeType = $finfo->file($uploadedFile['tmp_name']);

    return in_array($mimeType, ['image/jpeg', 'image/png', 'image/gif'], true);
}


if (!isMimeTypeAuthorized($_FILES['uploaded_file'])) {
    throw new RuntimeException('Invalid file mime type.');
}
```

Nous pouvons aussi vérifier que notre fichier se télécharge bien en générant un nom de fichier unique (il faut se méfier de la valeur de `$_FILES['uploaded_file']['name']`):

```
function getExtensionFromMimeType(string $mimeType): ?string {
    switch ($mimeType) {
        case 'image/jpeg':
            return 'jpg';
        case 'image/png':
            return 'png';
        case 'image/gif':
            return 'gif';
```

```
        default:
            return null;
    }
}

function moveUploadedFile(array $uploadedFile): bool {
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $mimeType = $finfo->file($uploadedFile['tmp_name']);

    return move_uploaded_file(
        $uploadedFile['tmp_name'],
        sprintf('./uploads/%s.%s',
            sha1_file($uploadedFile['tmp_name']),
            getExtensionFromMimeType($mimeType)
        )
    );
}

if (!moveUploadedFile($_FILES['uploaded_file'])) {
    throw new RuntimeException('Impossible to upload file.');
}
```

Syntaxe **À retenir**

- Le téléchargement de fichiers sans contrôle est dangereux pour les applications. Pour les sécuriser, il faut vérifier plusieurs critères tels que les erreurs d'upload, le type MIME du fichier, son poids, et faire en sorte de pouvoir stocker le fichier sur le serveur, quel que soit son nom.

Complément

Fonctions fileinfo [fr][1]

XI. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[2]

Question

[solution n°5 p.23]

À vous de mettre en place la fonction d'upload de fichiers qui va vérifier son bon téléchargement. Voici le code non sécurisé :

```
function moveUploadedFile(array $uploadedFile): bool {
    move_uploaded_file(
        $uploadedFile['tmp_name'],
        sprintf('./uploads/%s', $_FILES['upfile']['name'])
    );
}

moveUploadedFile($_FILES['uploaded_file']);
```

1 https://www.php.net/manual/fr/ref.fileinfo.php
2 https://repl.it/

```
 9
10 echo 'Upload OK';
```

Pensez à vérifier que :

1. L'upload n'a pas d'erreurs
2. Le fichier ne pèse pas plus d'un mégaoctet
3. Le fichier est bien un fichier PNG
4. Le fichier est correctement déplacé dans le répertoire d'upload avec un nom sécurisé

XII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°6 p.24]

Exercice

Laquelle de ces propositions est vraie ?

- ○ On peut toujours faire confiance aux saisies d'un utilisateur
- ○ On peut faire confiance aux saisies d'un utilisateur lorsqu'on utilise un framework
- ○ On ne peut jamais faire confiance aux saisies d'un utilisateur

Exercice

Quels sites pouvez-vous utiliser pour vous informer sur les failles de sécurité des applications web ?

- ☐ OWASP
- ☐ CVE
- ☐ NVD
- ☐ Capital One Financial

Exercice

Comment pouvez-vous récupérer le rôle d'un utilisateur de façon sûre ?

- ☐ Avec la superglobale `$_GET`
- ☐ Avec la superglobale `$_POST`
- ☐ Avec la superglobale `$_SESSION`
- ☐ Directement depuis la base de données en récupérant l'identifiant de l'utilisateur connecté depuis la superglobale `$_GET`

Exercice

Citez une fonction PHP qui permet de vérifier qu'une chaîne est seulement composée de chiffres.

Exercice

Comment pouvez-vous définir une valeur par défaut pour `$id` dans le cas où `$_GET` ne contient pas de clé `id` ?

- [ ] `$id = !empty($_GET['id']) ? $_GET['id'] : 1;`
- [ ] `$id = isset($_GET['id']) ? $_GET['id'] : 1;`
- [ ] `$id = $_GET['id'] ?: 1;`
- [ ] `$id = $_GET['id'] ?? 1;`

Exercice

Quels types d'attaques peut-on mener avec la faille XSS ?

- [ ] Rediriger un utilisateur vers un site malveillant
- [ ] Voler un cookie de session pour usurper l'identité d'un utilisateur connecté
- [ ] Afficher une publicité à un utilisateur contre le gré du site web
- [ ] Enregistrer les frappes clavier d'un utilisateur, telles qu'un numéro de carte bancaire (*key logging*)

Exercice

Quelle fonction PHP native permet de se protéger contre la faille XSS ?

Exercice

Quels types d'attaques peut-on mener avec la faille **Unrestricted File Upload** ?

- [ ] Récupérer des informations sensibles directement depuis la base de données
- [ ] Supprimer des fichiers du serveur
- [ ] Récupérer le contenu de fichier de configuration, tel que des mots de passe
- [ ] Récupérer des informations sensibles sur un utilisateur

Exercice

Sur quoi peut-on se baser pour définir le type d'un fichier uploadé (image, PDF...) ?

- [ ] La valeur de `$_FILES['uploaded_file']['type']`
- [ ] L'extension de la valeur de `$_FILES['uploaded_file']['tmp_name']`
- [ ] L'extension de la valeur de `$_FILES['uploaded_file']['name']`
- [ ] Le résultat de `(new finfo(FILEINFO_MIME_TYPE))->file($_FILES['uploaded_file']['tmp_name'])`

Exercice

Quelle méthode calcule le hash SHA1 du contenu d'un fichier ?

B. Exercice : Défi

Vous disposez du code suivant : il s'agit d'un formulaire permettant de gérer un upload d'images pour une photothèque.

En l'état, ce code n'est pas suffisamment sécurisé.

```php
<?php

@mkdir('./upload', 0644);

$message = null;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {//"===" vérifie que les deux termes sont égaux en valeur et en typage
    $filename = $_POST['filename'];
    $uploadedFile = $_FILES['uploaded_file'];
    $authorizedMimeTypes = explode(',', $_POST['authorized_mime_types']);

    if (!in_array($uploadedFile['type'], $authorizedMimeTypes)) {
        $message = 'Type de fichier non supporté';
    } else {
        move_uploaded_file(
            $uploadedFile['tmp_name'],
            sprintf('./upload/%s', $filename)
        );

        $message = 'Upload réussi';
    }
}
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Upload de fichier</title>
    </head>
    <body>
        <form method="POST" enctype="multipart/form-data">
            <?php= $message ? sprintf('<p>%s</p>', $message) : '' ?>
            <label>
                Nom de l'image
                <input type="text" name="filename">
            </label>
            <label>
                Fichier à télécharger
                <input type="file" name="uploaded_file">
            </label>
            <label>
                <input type="hidden" name="authorized_mime_types" value="image/png,image/jpeg,image/gif">
            </label>
            <input type="submit" value="Envoyer">
        </form>
    </body>
</html>
```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

Question

[solution n°7 p.26]

Il va falloir :

1. Définir des valeurs par défaut pour les champs de formulaire (au cas où un utilisateur utiliserait un client HTTP pour soumettre son fichier)
2. Supprimer l'information des types MIME autorisés et les stocker en dur dans une variable, car c'est une information sensible
3. Effectuer les vérifications d'usage sur le bon téléchargement et le format du fichier
4. Vérifier que le fichier ne pèse pas plus de 2 Mo
5. Vérifier que le nom du fichier stocké dans `$_POST['filename']` ne contient que des caractères alphanumériques, des tirets ou des *underscores*
6. Stocker le fichier dans le répertoire `uploads/` avec un nom unique (ici, on utilisera le nom fourni par l'utilisateur en y ajoutant un hash et l'extension du fichier en fonction de son type MIME, par exemple **mon-image.a58f535ebc.jpg**)

Indice :

Vous pouvez utiliser la fonction `preg_match('/^[\w-]+$/', $filename)` pour le point 5.

Solutions des exercices

1 https://repl.it/

Exercice p. 4 Solution n°1

Exercice

Quel type de faille est exploité lorsqu'un assaillant exécute du code dans une requête SQL ?

- (●) Injection
- ( ) Broken Authentication
- ( ) XML External Entities (XXE)
- ( ) Insecure Deserialization

Exercice

Quel type de faille est exploité lorsqu'un assaillant accède à des données en clair stockées sur le serveur (mots de passe, numéros de carte bancaire) ?

- ( ) Injection
- ( ) Broken Authentication
- (●) Sensitive Data Exposure
- ( ) Cross-Site Scripting XSS

p. 6 Solution n°2

Fichier `index.php` :

```php
<?php

$posts = [
    1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
    ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
    ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
];
?>
<h1>Foire aux questions</h1>

<?php foreach ($posts as $id => $post) : ?>
    <h2>&laquo; <?= $post['title'] ?> &raquo; publié le <?= $post['createdAt']->format('d/m/Y') ?></h2>
    <!-- 1 -->
    <p><a href="supprimer.php?id=<?= $id ?>">Supprimer</a></p>
<?php endforeach;
```

Fichier `supprimer.php` :

```php
<?php

if ($_SESSION['admin'] !== '1') {
    die('Vous n\'avez pas le droit de supprimer des articles !');
}

$posts = [
```

```
    1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
    ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
    ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
];

unset($posts[$_GET['id']]);

header('Location: index.php');
```

p. 10 **Solution n°3**

```
<?php

$posts = [
  1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
  ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
  ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
];

// 1
if (!ctype_digit($_GET['id']) || $_GET['id'] <= 0) {
    die('Une erreur est survenue. Merci de réessayer.');
}

$id = (int) $_GET['id'];

// 2
if (!isset($posts[$id])) {
    die('Une erreur est survenue. Merci de réessayer.');
}

$post = $posts[$id];

echo 'L\'article "'.$post['title'].'" a été publié le '.$post['createdAt']->format('d/m/Y').'.';
```

p. 13 **Solution n°4**

Fichier `index.php` :

```
<?php

$posts = [
    1 => ['title' => 'Comment sécuriser mon application ?', 'createdAt' => new \DateTime('2020-04-06')],
    ['title' => 'Comment récupérer un paramètre de mon URL ?', 'createdAt' => new \DateTime('2020-03-12')],
    ['title' => 'Comment récupérer le contenu de mon formulaire ?', 'createdAt' => new \DateTime('2020-02-04')],
    ['title' => 'Que se passe-t-il lorsqu\'on fait un <script>alert(\'Raté\')</script> ?', 'createdAt' => new \DateTime('2020-01-31')],
];
?>

```

```
<h1>Foire aux questions</h1>

<?php foreach ($posts as $id => $post) : ?>
    <h2> <i> <?= htmlentities($post['title']) ?> </i> publié le <?= $post['createdAt']->format('d/m/Y') ?> </h2>
<?php endforeach;
```

p. 15 **Solution n°5**

```
# 1
function isUploadSuccessful(array $uploadedFile): bool {
    return isset($uploadedFile['error']) && $uploadedFile['error'] === UPLOAD_ERR_OK;
}

# 2
function isUploadSmallerThan1M(array $uploadedFile): bool {
    return $uploadedFile['size'] < 1000000;
}

# 3
function isMimeTypeAuthorized(array $uploadedFile): bool {
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $mimeType = $finfo->file($uploadedFile['tmp_name']);

    return $mimeType === 'image/png';
}

# 4
function getExtensionFromMimeType(string $mimeType): ?string {
    switch ($mimeType) {
        case 'image/png':
            return 'png';
        default:
            throw new RuntimeException('Unsupported mime type');
    }
}

function moveUploadedFile(array $uploadedFile): bool {
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $mimeType = $finfo->file($uploadedFile['tmp_name']);

    return move_uploaded_file(
        $uploadedFile['tmp_name'],
        sprintf('./uploads/%s.%s',
            sha1_file($uploadedFile['tmp_name']),
            getExtensionFromMimeType($mimeType)
        )
    );
}

# 1
if (!isUploadSuccessful($_FILES['uploaded_file'])) {
    throw new RuntimeException('Error while uploading file.');
}

# 2
if (!isUploadSmallerThan1M($_FILES['uploaded_file'])) {
```

```
49      throw new RuntimeException('File is too big.');
50  }
51
52  # 3
53  if (!isMimeTypeAuthorized($_FILES['uploaded_file'])) {
54      throw new RuntimeException('Invalid file mime type.');
55  }
56
57  # 4
58  if (!moveUploadedFile($_FILES['uploaded_file'])) {
59      throw new RuntimeException('Impossible to upload file.');
60  }
61
62  echo 'Upload OK';
```

Exercice p. 16 **Solution n°6**

Exercice

Laquelle de ces propositions est vraie ?

- ○ On peut toujours faire confiance aux saisies d'un utilisateur
- ○ On peut faire confiance aux saisies d'un utilisateur lorsqu'on utilise un framework
- ◉ On ne peut jamais faire confiance aux saisies d'un utilisateur

Il faut toujours se méfier des saisies d'un utilisateur, même lorsqu'on utilise un framework.

Exercice

Quels sites pouvez-vous utiliser pour vous informer sur les failles de sécurité des applications web ?

- ☑ OWASP
- ☑ CVE
- ☑ NVD
- ☐ Capital One Financial

Exercice

Comment pouvez-vous récupérer le rôle d'un utilisateur de façon sûre ?

- ☐ Avec la superglobale `$_GET`
  Il ne faut jamais utiliser `$_GET` pour passer des informations sensibles à vos scripts PHP.
- ☐ Avec la superglobale `$_POST`
  Il ne faut jamais utiliser `$_POST` pour passer des informations sensibles à vos scripts PHP.
- ☑ Avec la superglobale `$_SESSION`
  `$_SESSION` peut être une solution si le rôle est déjà stocké en session.

- [ ] Directement depuis la base de données en récupérant l'identifiant de l'utilisateur connecté depuis la superglobale `$_GET`
  Encore une fois, il ne faut jamais passer d'informations sensibles à vos scripts PHP par la superglobale `$_GET`, telles que l'identifiant de l'utilisateur connecté. Vous pourriez le stocker en session au moment de la connexion, puis le réutiliser pour récupérer le rôle depuis la base de données.

Exercice

Citez une fonction PHP qui permet de vérifier qu'une chaîne est seulement composée de chiffres.

ctype_digit

Exercice

Comment pouvez-vous définir une valeur par défaut pour `$id` dans le cas où `$_GET` ne contient pas de clé `id` ?

- [ ] `$id = !empty($_GET['id']) ? $_GET['id'] : 1;`
  La valeur par défaut fonctionnera bien si `$_GET['id']` n'existe pas, mais si elle vaut 0, la valeur sera définie à 1, ce qui n'est pas correct.

- [x] `$id = isset($_GET['id']) ? $_GET['id'] : 1;`
  Cette solution fonctionne.

- [ ] `$id = $_GET['id'] ?: 1;`
  *La valeur par défaut fonctionnera bien si `$_GET['id']` n'existe pas, mais il y aura une erreur de type **notice**, et si elle vaut 0, la valeur sera définie à 1, ce qui n'est pas correct.*

- [x] `$id = $_GET['id'] ?? 1;`
  C'est la solution idéale.

Exercice

Quels types d'attaques peut-on mener avec la faille XSS ?

- [x] Rediriger un utilisateur vers un site malveillant
  C'est possible grâce à la balise `script` et à JavaScript.

- [x] Voler un cookie de session pour usurper l'identité d'un utilisateur connecté
  C'est possible grâce à la balise `script` et à JavaScript.

- [x] Afficher une publicité à un utilisateur contre le gré du site web
  C'est possible grâce à la balise `img` par exemple.

- [x] Enregistrer les frappes clavier d'un utilisateur, telles qu'un numéro de carte bancaire (*key logging*)
  C'est possible grâce à la balise `script` et à JavaScript.

Exercice

Quelle fonction PHP native permet de se protéger contre la faille XSS ?

htmlspecialchars

Exercice

Quels types d'attaques peut-on mener avec la faille **Unrestricted File Upload** ?

- [x] Récupérer des informations sensibles directement depuis la base de données
  Un script PHP uploadé peut accéder à la base de données en utilisant une connexion PDO existante.

- [x] Supprimer des fichiers du serveur
  Un script PHP uploadé peut agir sur le système de fichiers du serveur.
- [x] Récupérer le contenu de fichier de configuration, tel que des mots de passe
  Un script PHP uploadé peut lire des fichier sur le système de fichiers.
- [x] Récupérer des informations sensibles sur un utilisateur
  Un script PHP uploadé à la place d'une image peut récupérer des informations d'un utilisateur tentant de l'afficher en tant que telle.

Exercice

Sur quoi peut-on se baser pour définir le type d'un fichier uploadé (image, PDF...) ?

- [ ] La valeur de `$_FILES['uploaded_file']['type']`
  Le type MIME peut ne pas correspondre au type réel du fichier.
- [ ] L'extension de la valeur de `$_FILES['uploaded_file']['tmp_name']`
  L'extension peut ne pas correspondre au type réel du fichier.
- [ ] L'extension de la valeur de `$_FILES['uploaded_file']['name']`
  L'extension peut ne pas correspondre au type réel du fichier.
- [x] Le résultat de `(new finfo(FILEINFO_MIME_TYPE))->file($_FILES['uploaded_file']['tmp_name'])`
  Ce résultat est sûr, car c'est le serveur qui détermine le type MIME en fonction du fichier, tel qu'il existe sur le serveur.

Exercice

Quelle méthode calcule le hash SHA1 du contenu d'un fichier ?

sha1_file

p. 19

Solution n°7

```php
<?php

function isUploadSuccessful(array $uploadedFile): bool {
    return isset($uploadedFile['error']) && $uploadedFile['error'] === UPLOAD_ERR_OK;
}

function isUploadSmallerThan2M(array $uploadedFile): bool {
    return $uploadedFile['size'] < 2000000;
}

function isMimeTypeAuthorized(array $authorizedMimeTypes, array $uploadedFile): bool {
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $mimeType = $finfo->file($uploadedFile['tmp_name']);

    return in_array($mimeType, $authorizedMimeTypes, true);
}

function getExtensionFromMimeType(array $authorizedMimeTypes, array $uploadedFile): string {
    $finfo = new finfo(FILEINFO_MIME_TYPE);
    $mimeType = $finfo->file($uploadedFile['tmp_name']);

    if ($extension = array_search($mimeType, $authorizedMimeTypes, true)) {
```

```
        return $extension;
    }

    throw new RuntimeException('Le type MIME n\'est lié à aucune extension');
}

function moveUploadedFile(array $uploadedFile, string $filename, string $extension): bool {
    return move_uploaded_file(
        $uploadedFile['tmp_name'],
        sprintf('./uploads/%s.%s.%s',
            $filename,
            sha1_file($uploadedFile['tmp_name']),
            $extension
        )
    );
}

@mkdir('./uploads', 0644);

// 2
$authorizedMimeTypes = [
    'png' => 'image/png',
    'jpg' => 'image/jpeg',
    'gif' => 'image/gif',
];
$message = null;

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    try {
        // 1
        $filename = $_POST['filename'] ?? null;
        $uploadedFile = $_FILES['uploaded_file'] ?? [];

        // 3
        if (!isUploadSuccessful($uploadedFile)) {
            throw new RuntimeException('Le téléchargement a échoué');
        }

        if (!isMimeTypeAuthorized($authorizedMimeTypes, $uploadedFile)) {
            throw new RuntimeException('Le type de fichier n\'est pas supporté');
        }

        // 4
        if (!isUploadSmallerThan2M($uploadedFile)) {
            throw new RuntimeException('Le fichier dépasse les 2 Mo');
        }

        // 5
        if (!preg_match('/^[\w-]+$/', $filename)) {
            throw new RuntimeException('Le nom du fichier ne doit pas être vide et ne
contenir que des lettres, des chiffres, des tirets ou des underscores');
        }

        if (!moveUploadedFile($uploadedFile, $filename,
getExtensionFromMimeType($authorizedMimeTypes, $uploadedFile))) {
            throw new RuntimeException('Le téléchargement a échoué');
        }

        $message = 'Upload réussi';
```

```php
    } catch (RuntimeException $e) {
        $message = $e->getMessage();
    }
}
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Upload de fichier</title>
    </head>
    <body>
        <form method="POST" enctype="multipart/form-data">
            <?php= $message ? sprintf('<p>%s</p>', $message) : '' ?>
            <label>
                Nom de l'image
                <input type="text" name="filename">
            </label>
            <label>
                Fichier à télécharger
                <input type="file" name="uploaded_file">
            </label>
            <!-- 2 -->
            <input type="submit" value="Envoyer">
        </form>
    </body>
</html>
```