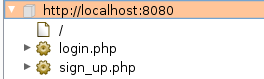
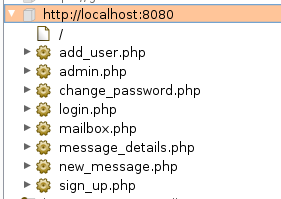
**Mapping the application**

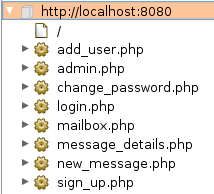
**Spidering**

Pour faire du spidering nous avons utilisé Burp suite Community Edition v1.x car elle permet plus facilement de faire un scan spidering. Nous avons testé en mode BlackBox, d’abord sans compte et ensuite avec un compte collaborateur. Puis nous avons testé en WhiteBox avec un compte administrateur.

Sans crédenciales :

  
  
Avec un utilisateur collaborateur :

  
  
Avec un compte admin :

  
On voit que pour un utilisateur collaborateur, il peut connaître le chemin pour des emplacements où il ne devrait pas aller. Les pages : “admin.php” et “add\_user.php” ne devraient pas être connues de l’utilisateur.   
En effet en testant l’application, si on tente d’accéder à cette page nous avons retour GET avec les permissions non autorisées :



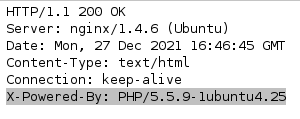
Afin de tester si on pouvait trouver d’autres fichiers cachés au niveau du serveur. Nous avons utilisé l’outil ZAP OWASP avec la worldist traditionnelle. Celui-ci n’a rien trouvé.

**Solutions :**

**...**

**Banner grabbing**

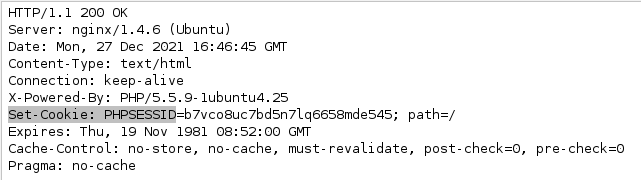
Lors du mapping nous avons constaté que les requêtes donnaient des informations sur le serveur. Nous savons maintenant quel langage nous utilisons et la version du serveur sur lequel tourne notre application.

  
**Solutions :**

Il faudrait faire un changement directement sur le serveur pour éviter de donner des informations dans les entêtes HTTP qui pourraient identifier notre infrastructure et trouver des vulnérabilités sur celle-ci.

**Token de session :**

On constate que le cookie est fait avec PHPSESSID (nous montre encore l’information du langage utilisé) et que celui-ci n’utilise pas de flag HttpOnly, ce qui veut dire que si on arrive à lancer un script malicieux, celui-ci pourra récupérer le cookie.

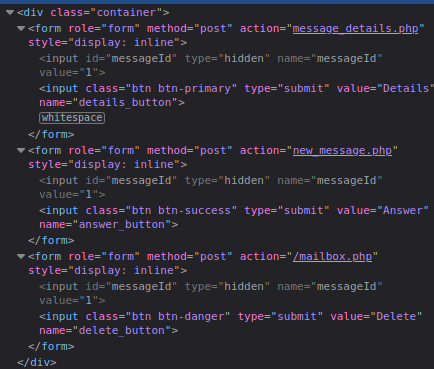
  
**Solutions :**

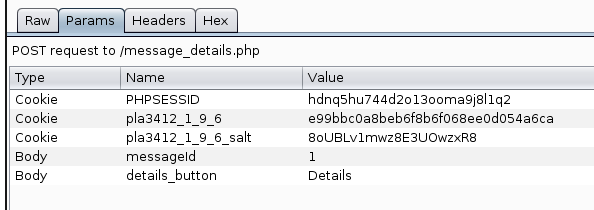
...

**Bypassing client-side control**

**Champs cachés**

Après plusieurs recherches dans le code avec un proxy (Burp) et en lisant le code source directement depuis la page web. Nous avons trouvé un champ caché. Ce champ caché est utilisé pour les boutons “Détails” “Delete” et “Answer” afin de pouvoir récupérer la bonne ID du message que l’on souhaite utiliser.

  
C ’est dangereux, car avec un proxy il est alors possible de changer ce paramètre et donc on va pouvoir lire et/ou supprimer un message qui ne nous appartient pas.

  
Nous avons d’autres champs cachés dans la partie Admin où on voit tous les utilisateurs et où on peut changer leur mot de passe, leur rôle, leur validité ou juste supprimer leur compte. Le champ caché contient le nom de l’utilisateur. Dans ce cas c’est moins dangereux car on va pouvoir supprimer ou modifier un autre utilisateur en changeant ce champ caché avec un autre nom d’utilisateur mais c’est déjà le fonctionnement de cette page.

**Solutions :**

Une solution est de protéger ce paramètre pour qu’il ne soit pas modifiable lors d’une requête.

**HTTP Cookies, URL parameters et referer :**

Les cookies ne transmettent pas de données. Aucune donnée n’est transmise depuis l’URL, seulement les messages d’erreur. Il n’y a pas de Referer dans les entêtes HTTP.

**Attacking authentification**

**Mot de passe et nom d’utilisateur prédictibles :**

Nous utilisons un compte admin avec un mot de passe admin. Il faut qu’il soit plus fort. De plus le username admin est très prédictible donc il faudrait le changer.

**Solution :**

1. Forcer la complexité du mot de passe lors de la création du compte ou du changement de mot de passe.
   1. Une longueur minimum de 15.
   2. Doit contenir au moins un symbole spécial (&, /, \*,”, +, …)
   3. Doit contenir au moins un numéro.
   4. Doit contenir au moins une lettre majuscule.
   5. Doit contenir au moins une lettre minuscule.
2. Obliger le changement de mot de passe après 180 jours en envoyant des E-Mail de notification.
3. Empêcher les nouveaux mots de passe d’être des mots de passe déjà utilisé par le passé.

**Stockage mot de passe :**

Si on va sur la page de la base de données, on constate que nos mots de passe sont stockés en clair.

  
Si un attaquant tombe sur la base de données, il aura complétement accès à tous les comptes. Il faut hasher ces mots de passe afin d’avoir une seconde ligne de défense si par malheur la base de données est rendue visible.

**Solution :**

Selon les recommandations OWASP, il faut utiliser un algorithme de hashage dans cette liste :

* Argon2id
* Bcrypt
* Scrypt
* PBKDF2

Et il faut considérer utiliser un pepper pour ajouter encore plus de défense.

**Transmission vulnérable de crédenciales**

Nous utilisons HTTP en local donc la transmission sera toujours non sécurisée. Une attaque MITM pourrait récupérer les crédenciales lors du login, du sign up et du changement de mot de passe.

**Solution :**

Il faudrait utiliser un moyen de transmission chiffré (SSL/TSL) surtout lorsqu'on manipule des données sensibles.

**Timing attack et message d’erreur trop verbeux :**

Nous avons fait plusieurs tests afin de tester s’il est possible de faire une “Timing attack”. Voici les résultats :

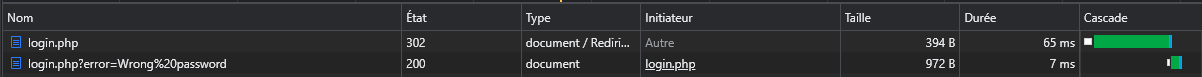
En testant des utilisateurs qui n’existent pas :





On reste dans les 50-65 ms.

En testant avec le nom de compte admin (un compte existant) :

On reste également dans les 50-65 ms.

Parfois les données étaient plus basses dans les deux cas. Ce n’est pas une science exacte. On pourrait utiliser un outil pour faire plus de tests afin d’être sûr. Cependant dans notre cas c’est normal qu’il n’y a pas vraiment de différence car nous ne faisons pas de hashage. Donc il n’y a pas plus de temps qui est pris pour la vérification.

Cependant, on constate qu’il y a un message d’erreur, donc même pas besoin de “Timing attack”. Nous sommes sur un cas de message d’erreur trop verbeux.

**Solutions :**

Quand nous allons corriger le système de stockage de mot de passe, il est nécessaire de faire attention à eviter une “Timing attack” en faisant du code constant. C’est à dire que quand on vérifiera le nom d’utilisateur et que celui-ci n’existe pas on fera un hash du mot de passe pour perdre du temps.

Pour le problème de message d’erreur trop verbeux, il faut juste faire un message d’erreur générique. Comme “Wrong credentials”. Afin de donner le moins d’informations possible.

**Changement de mot de passe :**

Pour le changement de mot de passe il manque certaines étapes afin de sécuriser le changement.

**Solutions :**

Il faut demander l’ancien mot de passe, pour qu’un attaquant qui arrive à se connecter via cookie par exemple ne puisse pas changer le mot de passe.

Il faut notifier le changement de mot passe par un mail, pour qu’une victime puisse être au courant si son compte a été compromis.

**Attacking session management**

**Attacking access controls**

**Contrôle d’accès sur pages :**

Nous avons testé toutes les pages accéssible uniquement par l’administrateur. Elles ont toutes un contrôle de permissions.

Page admin.php :



Page add\_user.php :



Si on n’est pas connecté aucune page n’est accessible sauf celle de login et de sign up :



Si on notre compte est “disabled”, aucune page n’est accessible sauf celle de login et de sign up :



Et il n’y a aucune vulnérabilité IDOR (Insecure Direct Object Reference)

**Attacking data store**

**Attacking backend components**

**Attacking application logic**

**Attacking users: Cross-site scripting**