Sécurité des Technologies Internet

Projet 2

Application de messagerie sécurisée

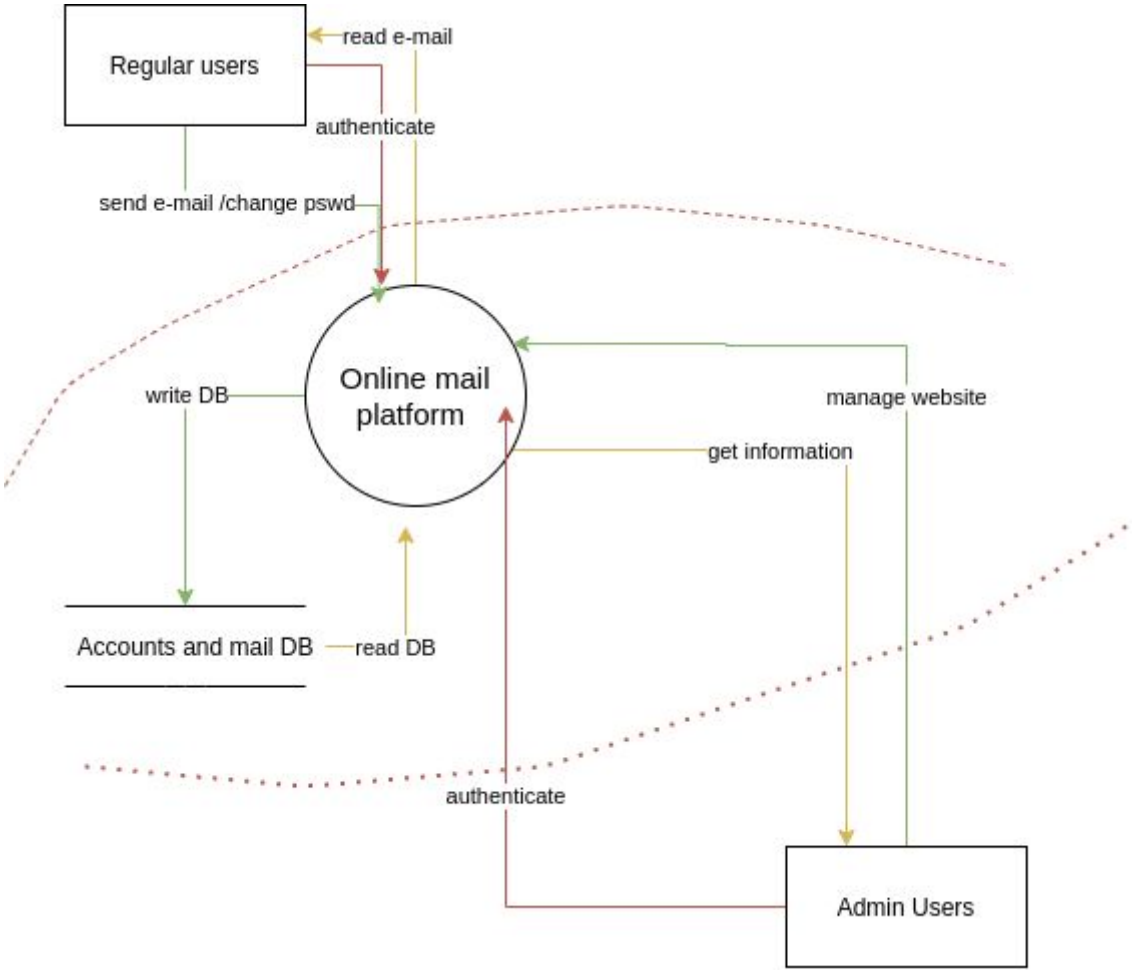
Auteurs : Fabio Marques et Gaëtan Daubresse

Introduction

L'objectif de ce projet consiste à analyser et sécuriser une application web. Pour ce faire nous allons d'abord effectuer une analyse des menaces afin de mieux identifier les éventuelles vulnérabilités présentes dans le code. Sur la base de cette analyse nous allons appliquer des correctifs sur le code afin de combler les vulnérabilités découvertes.

Décrire le système

DFD (Data-flow diagrams)



Identifier ses biens

- Base de donnée de l'application
    - Mails échangés confidentiels
    - Mot de passes utilisateurs confidentiels
- Infrastructure
    - Les mails envoyés ne peuvent pas avoir été modifié, intégrité

    - L’expéditeur est correcte, authenticité
    - Le système reste disponible

Définir le périmètre de sécurisation

Dans ce projet nous allons nous concentrer sur la sécurisation du code de l’application. Nous n’allons donc dans un premier pas considérer tous les éléments liés à l’infrastructure tels que le serveur Web ou les machines virtuelles.

Identifier les sources de menaces

- hackers, script-kiddies
    - Motivation: s’amuser
    - Cible: n’importe quel actif
    - Potentialité: haute
- Cybercriminels
    - Motivation: financière
    - Cible: Vol de credentials
    - Potentialité: Moyenne
- Utilisateur malin
    - Motivation: Lire des message d’autres utilisateurs
    - Cible: Escalation de privilèges
    - Potentialité: faible
- Concurrent
    - Motivation: éliminer ces concurrents
    - Cible: Déni de service
    - Potentialité: faible

Identifier les scénarios d’attaques

Scénario(s) d’attaque

XSS Réinitialisation mot de passe

Placer un script dans le mail afin qu’il s’exécute chez le destinataire. Lorsque le destinataire ouvrira le mail, le script sera chargé dans la page HTML.
Motivation: financière
STRIDE: Information disclosure, Spoofing, Tampering

XSS Envoi mail depuis autre compte

Un attaquant peut placer un script dans le mail afin que lorsque le destinataire le reçoive, le script envoie un e-mail depuis le compte du destinataire.
Motivation: challenge, amusement
STRIDE: Spoofing

XSS Obtention des droits admin

Si un attaquant connaît le nom d'un compte admin, il peut envoyer un mail sur ce compte avec un script permettant de lui donner les droits administrateurs. Si le script supprime le mail dans lequel il était contenu après s'être exécuté, l'administrateur pourrait ne pas se rendre compte de l'attaque.
Motivation: challenge, amusement
STRIDE: Elevation of privileges

XSS Suppression boîte mail

Un attaquant pourrait également, à l'aide d'un script placé dans le mail, supprimer la totalité de la boîte mail de la cible.
Motivation: challenge, amusement
STRIDE: Tampering

Fuite de mot de passe

En faisant un dump de la base de données à l'aide de sqlmap l'attaquant peut lire les mots de passe de chaque utilisateur.
Motivation: financière
STRIDE: Information disclosure

Bypass authentification injection

A l'aide d'un formulaire de connexion vulnérable aux injections sql un attaquant peut utiliser sqlmap pour bypasser l'authentification et se connecter a l'application.
Motivations: Amusement, Gloire, usurpation d'identité
STRIDE: Spoofing

Bypass authentification brute force liste

A l'aide d'un formulaire de connexion, un attaquant peut utiliser une liste de user/mdp pour essayer de rentrer sur le compte d'un utilisateur qui utilise le même mot de passe partout.
Motivation: Amusement, Gloire, usurpation d'identité
STRIDE: Spoofing

Bypass authentication brute force mdp faible

A l'aide d'un formulaire de connexion, un attaquant peut essayer plusieurs mot de passe faibles pour se connecter sur le compte d'un utilisateur qui utilise des mots de passe faibles.
Motivation: Amusement, Gloire, usurpation d'identité
STRIDE: Spoofing

Scénario d'attaque mis en pratique

Attaque de type XSS

Pour tester si cette attaque fonctionne nous pouvons dans un premier temps essayer de placer le script suivant dans le contenu du mail à envoyer :

```
<script>alert('test')</script>
```

Si le popup s'affiche lorsque le destinataire lit le mail cela veut dire que le script est correctement exécuté. Nous pouvons ensuite réfléchir à des scripts plus poussés permettant par exemple de réinitialiser le mot de passe de l'utilisateur, supprimer son compte ou encore envoyer des mails.
Afin de découvrir la façon dont le mot de passe est modifié, nous analysons d'abord les paquets envoyés lorsque nous changeons le mot de passe sur notre propre compte. Nous voyons qu'il s'agit de l'envoi d'un formulaire à l'aide d'une méthode POST. Nous essayons donc de reproduire cet envoi au travers du script suivant.

```
<script>
document.body.innerHTML += '<form id="dynForm"
action="http://localhost:8080/controllers/users/change_password.php"
method="post"><input type="hidden" name="password"
value="pwned"></form>';
document.getElementById("dynForm").submit();
</script>
```

En analysant les paquets lorsque nous changions le mot de passe sur notre compte nous avons récupéré l'adresse ou le formulaire était envoyé ainsi que le nom de la variable contenant le mot de passe. Ceci est suffisant pour envoyer un faux formulaire grâce au script ci-dessus que nous allons pouvoir insérer dans un e-mail.

Attaque de type CSRF

Si on crée un site malicieux contenant du code javascript qui fait une requête pour changer le mot de passe d'un utilisateur de notre site, exemple de code en dessous.

```
<script>
document.body.innerHTML += '<form id="dynForm"
action="http://localhost:8080/controllers/users/change_password.php"
method="post"><input type="hidden" name="password"
value="pwned"></form>';
document.getElementById("dynForm").submit();
</script>
```

Et que un utilisateur qui est connecté à notre site web va sur ce site web malicieux, sur un autre onglet par exemple, alors le code du site malicieux sera exécuté à l'insu de la victime et elle se retrouvera avec son mot de passe modifié. Dans ce cas, le problème est que notre application n'a aucun moyen de vérifier si c'est un vrai utilisateur ou un "robot" qui se fait passer pour nous. Il faut donc ajouter une protection csrf pour éviter ce genre d'attaques.

Fuite de mot de passe

Les mots de passé n'étaient pas hachés dans la base de données ce qui aurait permis à un attaquant qui met la main sur la base de données de les récupérer sans effort. Par contre on a pas réussi à dumper la base de données à l'aide d'une injection sql

Identifier les contre-mesures

En fonction des scénarios d'attaques

Attaque de type XSS

Pour se protéger de ce genre d'attaque, il est important d'assainir les entrées et sorties des applications. Nous allons donc essayer d'appliquer un filtre au moment de l'envoi des e-mail et à leur lecture afin d'autoriser seulement l'usage de certaines balises HTML.
La méthode `htmlspecialchars()` permet d'encoder les chaînes de caractères en HTML. Appliquer cette méthode sur chaque chaîne de caractères envoyées ainsi qu'à leur lecture pourrait prévenir l'attaque XSS décrite plus haut.

Lors de la sauvegarde du message dans le fichier `store_message.php` nous encodons les chaînes de caractères comme-ceci :

```
$query->execute([
            // string encoded in HTML to avoid XSS attacks
            htmlspecialchars($_POST['subject']),
            htmlspecialchars($_POST['content']),
            time(),
            $auth_user,
            $_POST['sendTo']
]);
```

Nous constatons que lorsque nous choisissons de répondre à un mail, c'est un autre bout de code qui s'occupera de stocker le message dans la base de donnée, il est donc important d'appeler la méthode `htmlspecialchars()` également dans le fichier `reply_message.php`

Si nous appliquons uniquement ce correctif, nous remarquons que l'encodage se fait du côté client. Un attaquant pourrait donc utiliser un outil comme burp afin d'intercepter la requête POST permettant d'enregistrer le mail et y placer le script qui ne sera alors pas encodé.

C'est pourquoi il est également important de sanitizer le message lors de sa lecture. Nous appliquons à nouveau la méthode `htmlspecialchars()` pour la lecture du sujet du mail ainsi que son contenu. A noter que nous avons également encodé l'expéditeur du mail puisque celui-ci aurait pu manuellement modifier ce champ lors de l'envoi de l'email.

Nous pouvons constater qu'il n'est pas évident de se protéger contre les attaques de type XSS, en effet il faut veiller à ce que chaque champ écrit et lu dans la base de donnée puisse être encodé afin que les scripts potentiels ne puissent être exécutés.

Attaque de type CSRF

Pour se protéger de cette attaque il faut ajouter une valeur à chaque requête sensible qui permet d'identifier si la requête a bien été lancée par l'utilisateur ou par un code malicieux. Pour ce faire, on commence par créer une fonction qui permet de générer un "csrf_token" et l'enregistrer dans la session en cours de l'utilisateur. Ensuite pour chaque requête sensible on va inclure ce token dans un input caché. Il ne reste plus qu'à vérifier en backend si le token de la requête correspond bien à celui de l'utilisateur connecté.

Fuite de mot de passe

La seule solution pour garantir que les mot de passe ne seront pas compromis lors d'un vol de données c'est de les hacher avec un algorithme sûr et ne jamais les stocker en clair.

Injection sql

Il n'y a actuellement aucune vulnérabilité qui permettrai de faire une injection sql sur le site. Pour rappel, afin d'éviter les injections sql, il ne faut jamais concatener directement les entrées utilisateur dans une requête sql. Il faut donc préparer chaque requête qui contient des entrées utilisateur. Dans l'idéal il faudrait aussi nettoyer les données même dans le cas où on utilise la méthode prepare car dans certains cas il est quand même possible de faire des injections sql dans des requêtes préparées.

Bypass authentification brute force liste

La solution la plus simple consiste à obliger les utilisateurs à modifier leur mot de passe régulièrement pour réduire la probabilité que le tuple user/mdp soit présent dans une liste, par contre le gros désavantage c'est que ça risque d'énerver les utilisateurs qui n'aiment pas qu'on leur impose des choses.

Une deuxième solution consiste à mettre en place un système qui permet de bloquer les machines/IPs qui font trop de tentatives de connexion.

Bypass authentication brute force mdp faible

La solution c'est d'obliger l'utilisateur à créer un mot de passe fort par exemple: contenant des caractères spéciaux et des majuscules.

Conclusion

Du côté de l'authentification l'application était bien sécurisée, toute la gestion de la session ce fait au travers de la variable $_SESSION ce qui empêche de pouvoir modifier des informations d'état depuis le client (ex: passer la valeur $_SESSION['admin'] à true et se faire passer par un admin) et toutes les requêtes à la base de données sont preparées. Le seul point vulnérable c'était que les mots de passe étaient stockés en clair par contre il n'y avait aucun moyen de dumper la base de données. On a aussi augmenté la sécurité des mots de passe en exigeant des mots de passe plus complexes pour éviter les attaques par bruteforce.

En ce qui concerne les emails il y avait une grosse faille XSS ajoutée au fait qu'il n'y avait pas de protection CSRF qui faisait qu'on pouvait faire des grosses attaques comme par exemple modifier le mot de passe de la victime. On a pu corriger la faille XSS en nettoyant simplement les entrées utilisateur à l'aide de la fonction php **htmlspecialchars.** Il n'est donc plus possible d'injecter du code javascript dans l'application.

Pour finir tout ce qui concerne la gestion des utilisateurs il n'y avait rien à redire car tout était sécurisé de ce côté.