STI - Application de messagerie sécurisée

Analyse de menaces

Auteur: Issolah Maude

Repo. GitHub: https://github.com/issolahma/STI-Projet2

Système

Application web de messagerie sur réseau privé (LAN).

Les utilisateurs enregistrés peuvent:

- envoyer des messages
- répondre à des messages
- supprimer des messages
- modifier leur mot de passe

Les utilisateurs avec un rôle administrateur peuvent en plus des actions précédentes:

- créer de nouveaux utilisateurs

- modifier les mots de passes de tous les utilisateurs
- supprimer (désactiver) des utilisateurs
- modifier certaines données des utilisateurs.

Objectifs du système

Cette application permet aux collaborateurs d'une entreprise de s'échanger des messages, dans l'idée des emails.

Hypothèses de sécurité

- Application déployée sur un réseau privé d'entreprise de confiance, et sécurisé.
- Administrateurs de confiance
- Système d'exploitation et serveur Web de confiance

Exigences de sécurité

- **Dans l'application:**
  - Les mots de passes ne doivent pas être en clair dans la base de données.
  - Les messages échangés ne doivent pas être en clair dans la base de données.
  - On ne doit pas pouvoir manipuler les valeurs GET
  - Les inputs des utilisateurs doivent être contrôlés.
- **Accès à l'application**
  - Le login doit différencier les utilisateurs, et appliquer les bons droits.
  - La déconnexion doit fonctionner
- **Usage de l'application**
  - Un utilisateur ne doit avoir accès qu'à ce qui lui appartient. Qu'aux données de son compte.
  - La base de donnée ne doit pas être en accès libre.

Eléments du système

- Application web
- Base de donnée SQLite

Rôles des utilisateurs

- **Administrateurs**

  Droit avancé de gestion de l'application.

  Droit en lecture/modification sur les données de tous les utilisateurs.

- **Utilisateurs**

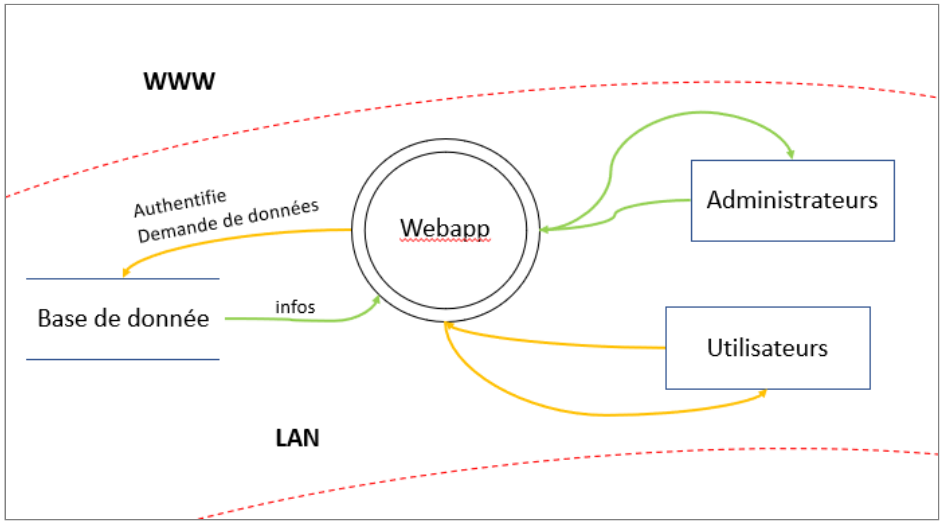
  Droit de base d'utilisation de l'application.

  Droits en lecture/modification uniquement sur ces propres données.

Actifs à haute valeur

- Base de donnée
    - Confidentialité (liste des utilisateurs)
    - Intégrité (modification des messages)
    - Un incident engendrerait une perte de revenu lié à la perte d'information, et au temps pour la recréer/retrouver.

Data flow diagramme



Sources de menaces

L'application n'étant utilisée que sur un réseau privé, les attaquants doivent commencer par accéder au LAN.

- **Hacker, script-kiddies**
    - Motivation: S'amuser, gloire
    - Cible: n'import quel élément / actif
    - Potentialité: faible
- **Cybercrime** (spam, maliciels)
    - Motivation: financière
    - Cible: vol de credentials, chiffrement des données
    - Potentialité: faible
- **Utilisateurs/employé malins**
    - Motivation: S'amuser, gloire, vengeance
    - Cible: vol de credentials, modification de messages, usurpation d'identité
    - Potentialité: moyenne

Scénario d'attaques

Scénario 1 - Manipulation *GET*

- **Impacte**: fort

  Un utilisateur enregistré peut obtenir des informations qui ne lui sont pas destinées sans avoir de rôle d'administrateur.

- **Source de la menace**:

  Hacker/utilisateurs malins

- **Motivation**:

  Challenge

- **Cible de l'attaque**:

  L'application

- **Scénario d'attaque**:

  Modification du paramètre de l'url.

- **Controls**
  - Valeurs de paramètres aléatoires, non liés directement a la donnée.
  - Utiliser des cookies au lieu de *GET*

Scénario 2 - Vol de données

- **Impacte**: fort
- **Source de la menace**:

  Cybercrime/utilisateurs malins

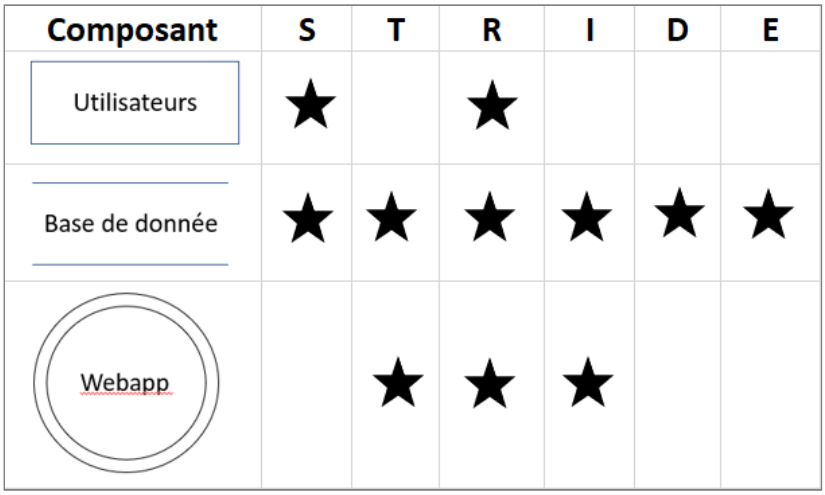
- **Motivation**:

  Financière/challenge

- **Cible de l'attaque**:

  La base de donnée

- **Scénario d'attaque**:
  - Injection de code
  - Vol de credentials
- **Controls**:
  - Chiffrement des données en base de données.
  - Contrôle des accès
  - Log des connexions et alerte en cas d'utilisation inhabituelle.
  - Politique de mot de passe forte
  - Contrôle des entrées utilisateurs



Contremesures

Données

- Contrôle des entrées utilisateurs
  - Toutes les entrée utilisateurs sont filtrées avant utilisation par l'application.
- Mot de passes
  - Seul le hash est enregistré en base de données
  - Le hash est calculé sur le poste client et le mot de passe n'est jamais envoyé au serveur.
- Messages
  - Les message sont chiffrés avant d'être envoyé à la base de donnée.
- Données de l'application (variables)
  - Remplacement de la méthode GET par l'utilisation de cookies.

Implémentation

Mots de passes

Création du mot de passe.

Lors de la création d'un utilisateur ou la modification d'un mot de passe, ce dernier est hashé avant d'être enregistré dans la base de donnée.

L'utilisation de la méthode `password_hash()` ajoute automatiquement un sel au hash, ce qui permet d'avoir des hash différents pour un même mot de passe.

Le paramètre `PASSWORD_DEFAULT` utilise algorithme le plus fort disponible, actuellement `bcrypt`, mais est sujet à changement si un meilleur algorithme est choisis par les développeurs.

```
$hashedPwd = password_hash('password', PASSWORD_DEFAULT);
```

Validation du mot de passe.

La validation se fait avec la méthode `password_verify()`.

```php
$validePwd = password_verify('password', 'hash');
```

Traitement des entrées utilisateurs

Pour éviter toute manipulation de l'application par un utilisateur malveillant via les entrées utilisateurs, ces dernières sont traitées avant d'être utilisées lors de requêtes sql.

La méthode `filter_var` est utilisée.

```php
$email = filter_var('email', FILTER_SANITIZE_EMAIL);
```

Lors de la validation du login, l'input du mot de passe n'est pas contrôlé, car il n'est ni envoyé à la base de donnée ni utilisé plus loin.

L'email est utilisé pour valider existence de l'utilisateur dans la base de donnée, et si celui-ci existe, son mot de passe est validé côté client avec la méthode `password_verify` et le hash reçu de la base de donnée.

```php
$valideUser = password_verify($_POST['password'], $user[0]['password']);
```

Remplacement de la méthode GET

La méthode *GET* utilisée pour passer des données d'une page à l'autre avec des valeurs sensibles (email, id) est manipulable pour obtenir des informations qui ne nous étaient pas destinée (ex: modification de l'id ou de l'adresse mail).

Cette méthode a été remplacée par l'utilisation de cookies. La méthode de création du cookie a été ajoutée dans le fichier `html/assets/js/sti.js`.

```javascript
function setCookie(cname, cvalue) {
    document.cookie = cname + "=" + cvalue + "; Max-Age=60" + "; SameSite=Strict";
}
```

Le cookie est créé lors du click sur un lien.

```php
<a href='editUser.php' onclick=\"setCookie('email','".$row['email']."')\">Edit</a>
```

Sécurité du cookie

- **Expiration**

La date d'expiration du cookie est fixé a 60 seconde après sa création, car il n'a pas d'autre utilité que de transférer une donnée d'une page a l'autre.

- **Suppression**

  Même si le cookie a une durée de vie courte, dans le doute, il sera supprimé en fin de session dans le fichier *logout.php*.

  ```
  setcookie ("email", "", time() - 3600);
  ```

- **HTTPS**

  Pour éviter qu'un cookie ne soit envoyé sur une connexion non sécurisée, il faut utiliser l'option `secure`. Dans notre cas le site n'est pas sécurisé, et cette option ne sera donc pas utilisée.

- **XSS**

  L'option `httponly` interdit la modification du cookie par un script javascript. Le cookie est créé depuis un script javascript, cette option n'est donc pas utilisable.

- **Domaine**

  Pour limiter l'utilisation du cookie uniquement sur le site, l'option `SameSite=Strict` est utilisable. L'option `domain` n'est pas utilisée, car elle demande un nom précis, et avec docker sur localhost, le nom peut différer (localhost/0.0.0.0).

Autres modifications

Beaucoup de modification mineures ont été faite. Voici la liste des principales.

- **Formulaires**:

  Ajout de la valeur `required` aux champs obligatoires des formulaires.

- **HTML/PHP**:
  - Séparation du traitement des données de formulaires *HTML* vers un autre fichier exclusivement *PHP*.

    Dans certain cas, la valeur du `$_POST` était traitée directement dans le fichier du formulaire, ce qui rendait le tout peu lisible.
  - Suppression de fichier *PHP* inutiles. Certaine traitement de formulaires passaient par plusieurs fichiers *PHP*, ce qui n'est pas nécessaire.

- **Bouton home**

  Ajout de boutons *home* et *logout* sur toutes les pages pour faciliter la navigation.

- **Validation du rôle utilisateur**

  Ajout de la validation du rôle en début de fichier, pour empêcher l'accès à la page.

  Dans certain cas, il était possible d'accéder la page, mais le contenu n'était pas affiché, car le contrôle se faisait en milieu de fichier.

- **Base de donnée**

  Les méthode de connexion et requêtes *sql* se trouvent dans un fichier séparé: *Db.php*.

- **Fichiers**
  - **editUser.php**

Le formulaire proposait de modifier le rôle avec deux boutons radio *admin*, et *user*. Ces deux boutons était sélectionnables en même temps, et il modifiaient la même valeur en base de donne.

Le bouton *user* a été supprimé, ce qui est équivalent. L'utilisateur est admin, ou ne l'est pas.

- **getAllUsers.php**

  Cette page affichait la liste des utilisateurs pour tous, avec des informations complémentaires pour les utilisateurs *admin*.

  Un utilisateur de base n'a aucune raison d'avoir la liste de tous les utilisateurs, il y a donc un contrôle du rôle en début de fichier pour l'accès a cette page.

  Pour le rôle *admin*, les mots de passes étaient affichés avec les autres données des utilisateurs. Cette donnée sensible, même si hachée n'est pas une information utile a un administrateur, contrairement au rôle, ou email, cette information a donc été supprimée de l'affichage.

Reste à faire

- **Messages**
  - Chiffrement des messages en base de données.
- **Utilisateurs**
  - Les utilisateurs sont identifiés lors des requêtes à la base de donnée via leur adresse mail, et non leur id. Cela rend la modification de cette dernière impossible.

    Il faudrait remplacer la gestion des utilisateurs en utilisant leur id.
  - Les utilisateurs n'ont pas de nom/prénom, ce qui faciliterait la validation de l'existence d'un utilisateur lors des requêtes *sql* en retournant une valeur non sensible tel que le prénom.
- Page ***editUser.php***
  - Il n'y a pas de possibilité de modifier l'adresse mail de l'utilisateur.
  - Les données de l'utilisateur ne sont pas reportées dans le formulaire, ce qui serait utile, et faciliterait la requête *sql* de modification.
- Page ***addUser.php***

  Il n'est pas possible de définir le rôle du nouvel utilisateur lors de sa création.
- **Gestion des erreurs**

  Il manque globalement (sauf login) l'affichage de message d'erreur lorsque la requête a échoué, ou si un des champs n'est pas valide.
- **Navigation**

  Les possibilités de navigation sur le site sont peu présentes, notamment pour revenir au menu précédent.