STI - Projet

Modélisation de menaces

WeChat

vers

WeChat Secure

Matthieu Chatelan & Loan Lassalle

13 janvier 2018

Table des matières

Table des figures

1 Introduction

Le but de ce document est de fournir une analyse de menaces auquels notre programme WeChat est exposé. Premièrement, une description du système est donnée fournissant ainsi une meilleure compréhension des différents composants de ce dernier, les différents rôles mis à disposition ou encore les biens qui seront à protéger.

Ensuite, une analyse des différentes sources de menaces auquels notre programme sera exposé. Les différentes sources potentielles sont énumérées ainsi que les compétences requises pour effectuer chacunes de ces attaques.

Finalement, plusieurs scénarios d'attaques seront donnés ainsi que les contremesures appropriées à mettre en place afin de les contrer.

2 Description du système

2.1 Objectifs du système

Les objectifs fixés du système étaient de développer une application web permettant d'envoyer et de reçevoir des messages textes entre collaborateurs au sein d'une entreprise. Le protocole de transmission SMTP ne devait pas être utilisé pour cette application.

Cette application serait par exemple utilisée entre employés d'une entreprise de développement de logiciels.

2.2 Exigences du système

Lors du développement de l'application, plusieurs contraintes ont été fixées par le client dans le cahier des charges. Dans les sous sections ci-dessous, ces dernières sont décrites et expliquées.

2.2.1 Exigences technologiques

Pour développer l'application, les différentes technologies suivantes ou contraintes ont dû être prises en compte :

- PHP
- SQLite
- Doit fonctionner sur l'environnement CentOS fournis

2.2.2 Exigences fonctionnelles

L'application devait être en mesure proposer deux rôles différents soit **collaborateur**, soit **administrateur**. De plus, un mécanisme d'authentification simple (utilisateur - mot de passe) doit permettre d'accéder aux fonctionnalités. Pour pouvoir se connecter, un utilisateur doit être définit comme étant "actif".

Une navigation aisée via des liens ou des boutons devait être mise en place.

Un **collaborateur** doit pouvoir effectuer les actions suivantes :

1. Lire les messages reçus sous forme de liste triée par ordre de date de réception avec plusieurs informations tels que :
   - la date de réception
   - l'expéditeur
   - le sujet
   - un bouton permettant de répondre au message (avec le sujet directement définit)
   - un bouton permettant de supprimer le message
   - un bouton permettant d'ouvrir un message et de voir le contenu du corps
2. Ecrire des messages à l'attention d'un autre utilisateur. Les informations suivantes doivent être fournies :
   - destinataire (unique)
   - sujet
   - corps du message
3. Changer le mot de passe de l'utilisateur connecté.

Un **administrateur** doit pouvoir effectuer les actions suivantes :

1. Avoir les mêmes actions qu'un collaborateur
2. Ajouter, modifier ou supprimer un utilisateur, représenté par les attributs suivants :
   - Un login (non modifiable)
   - Un mot de passe (modifiable)
   - Une validité actif/inactif (modifiable)
   - Un rôle (modifiable)

2.2.3 Exigences sécuritaires

Une authentification était demandée. Cette dernière ne devait pas permettre l'accès à toute autre page que celle de login lorsque l'utilisateur n'est pas authentifié.

Lors du développement initial, aucunes autre mesures de sécurité n'étaient demandées. De ce fait, aucunes protections contre des attaques XSS, CSRF, SQL Injection ou autre n'ont été mises en places.

Néanmoins, les mots de passes n'étaient pas stockés en clair dans la base de donnée, mais sous forme de hash salé pour empêcher une attaque par rainbow table sur ces derniers.

2.3 Constitution du système

Le système développé comprenait plusieurs acteurs et processus ainsi qu'une base de donnée centrale. Dans le diagramme de flux de données de l'application (Figure 2), un acteur est représenté par un rectangle alors qu'un processus est représenté par un cercle.

On notera que tous les processus qui se situent au dessus de la base de données dans le diagramme (5 au total) sont des processus que les deux acteurs peuvent utiliser. En revanche, les 3 processus au dessous de la base de données sont des processus réservés aux administrateurs.

Lellipse traitillée en orange représente la frontière de confiance de l'application. Cela signifie que nous considérons que les communications entres processus et la base de données sont sécurisés. Toutes entrées utilisateur doivent être contrôlés avant de pénétrer cette zone de confiance.

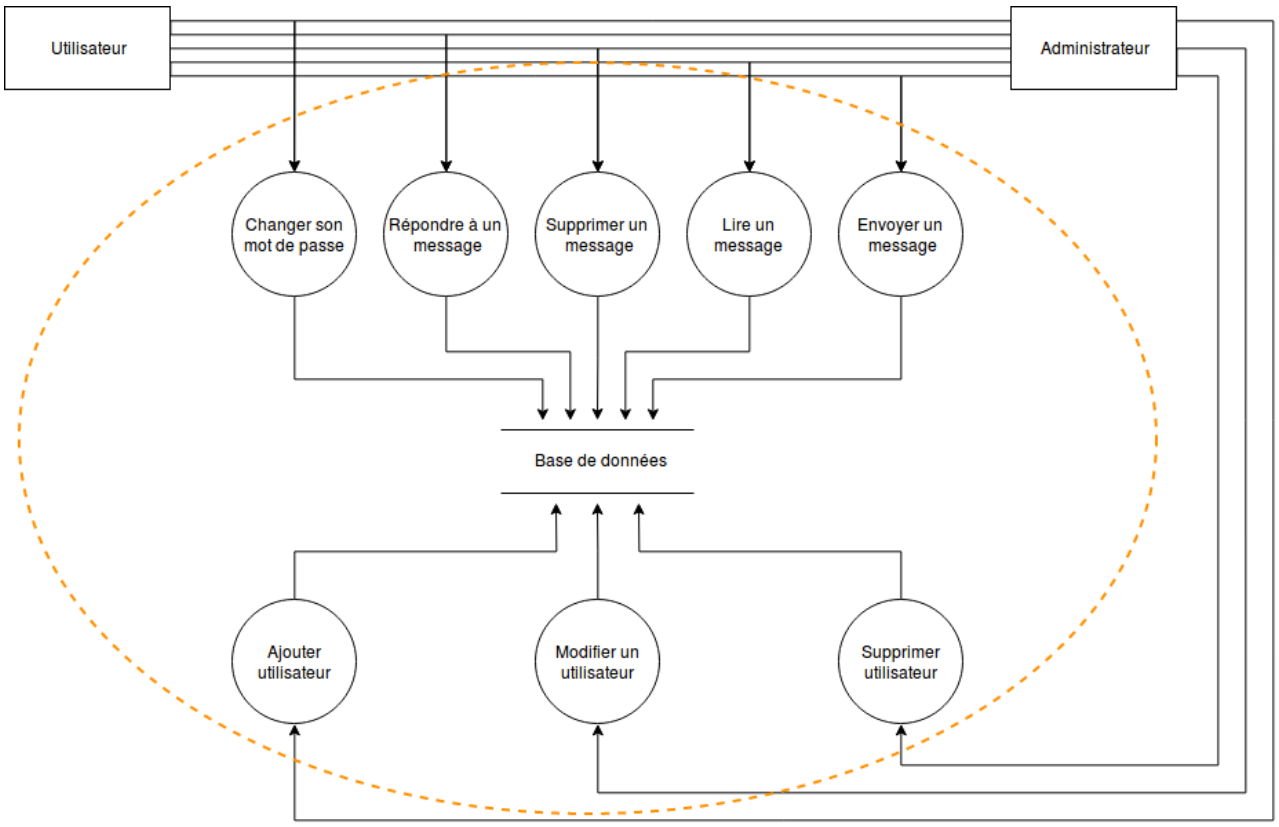

FIGURE 1 – Diagramme de flux de données de l'application

2.4 Biens

Les biens à protéger sont multiples dans une telle application. Ci-dessous, une liste non exhaustive de ces derniers

- Les messages
- Les comptes utilisateurs (leurs mots de passes)
- L'accès à l'administration des comptes

3 Sources de menaces

Initiateur	Motivations	Cible(s)	Potentialité
Utilisateur	Fun, Revanche, Curiosité	Tout éléments accessibles	Haute
Administrateur	Revanche, Curiosité	Tout éléments accessibles	Moyenne
Concurrent	Secrets d'entreprise	Base de données	Moyenne
Hackers	Gloire, Argent, Destruction	Tout éléments accessibles	Faible
Cyber-criminel	Vol d'informations, Spam, DDoS	Base de données	Faible
Etat	Vol d'informations, Profit	Tout éléments accessibles	Faible

Parmi les différents initiateurs décrits ci-dessus, les utilisateurs représentent la plus grande menace et la menace la plus probable d'arriver. En effet, comme le programme est destiné à un usage interne à l'entreprise, a priori seuls les employés y auront accès.

Un administrateur est par défaut une source de menace implicite car ce dernier à un accès total à l'application. Il sera dans notre cas en mesure d'ajouter des utilisateurs ou d'en supprimer, ou encore de les désactiver ou changer leur rôle. Si il a un accès à la base de données, il aura accès à toutes les informations contenues dans cette dernière.

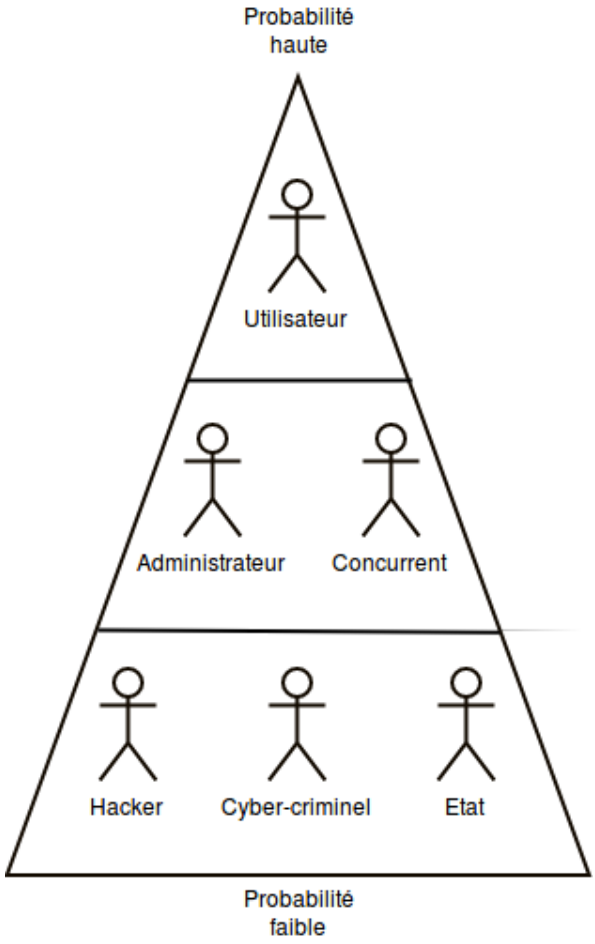

FIGURE 2 – Pyramide des sources de menace

4 Scénarios d'attaques

Dans cette section, plusieurs scénarios d'attaque sont décrits. Grâce à ces scénarios, il est possible d'imaginer les différents points vulnérables de l'application et ainsi permet de mettre en place des contremesures appropriées.

Ces différents scénarios sont regroupés par impacte et non par vulnérabilité. Il se peut qu'un impacte soit causé par de multiples failles de sécurité.

#	**Impacte**	**Source de menace**	**Bien ciblé**
4.0.1	Perte de confidentialité	Utilisateur, Concurrent, Hacker, Cyber-criminel	BDD
4.0.2	Perte de disponibilité	Utilisateur, Concurrent, Hacker, Cyber-criminel	Tout élément actif
4.0.3	Perte d'intégrité	Hacker, Concurrent	BDD
4.0.4	Accès à des ressources non autorisées	Utilisateur, Hacker, Concurrent	Application, BDD

Les différents scénarios d'attaques et leurs sources de menaces dépendent de comment l'application de chat est déployée et de comment elle est utilisée par les employés. En effet, si l'application est accessible par l'extérieur de l'entreprise, il sera facile pour des intervenants externes tels que des cyber-criminels ou des hackers d'accèder directement aux biens.

En revanche, si l'application est déployée sur un réseau local non accessible par l'extérieur, un attaquant externe devra s'introduire dans le réseau avant de pouvoir attaquer l'application et accèder aux biens. De l'ingénierie sociale peut être utilisée pour inciter un employer à visiter un lien ou à installer une application vulnérable.

4.0.1 Perte de confidentialité

Dans ce scénario d'attaque, les différents acteurs listés en tant que source de menace tenteront d'accèder à des informations qui ne leurs sont pas destinées causant ainsi une perte de confidentialité.

Pour parvenir à ses fins, un attaquant pourra essayer de multiples vecteurs d'attaque tels que :

- Manipulation de l'URL
- Injection SQL dans les divers champs de saisie vulnérables
- Utilisation d'une faille de XSS stocké afin de récupérer des cookie de session

Sans des connaissances poussées en informatique, on pouvait très rapidement remarquer que lors de la lecture d'un message, l'id de ce dernier était donnée dans l'url. Etant donné qu'aucunes vérification n'était faite du côté serveur pour s'assurer que la personne lisant le message était bien le destinataire de ce dernier, il était possible d'écrire un script python qui testait de manière incrémentale l'accès à tous les message en incrémentant l'id de ce dernier. Si en réponse une erreur 404 est obtenue, on sait que le message n'existe pas. En revance, si un code 200 est retourné, on sait qu'un message existe et on peut automatiquement l'extraire et le stocker dans un fichier ou le parser afin de tenter de trouver des informations confidentielles tels que des adresses mail ou des mots de passes.

Avec peu de recherches sur internet et quelques tentatives, un attaquant avec de faibles connaissances sera en mesure de récupérer des données confidentielles tels que des mails ou encore des

mots de passes en utilisant une injection SQL. Une fois les mots de passes récupérés, une étape de cassage hors ligne est nécessaire étant donné que mots de passes sont stocké sous forme hachée et salée.

Avec une attaque plus élaborée, cet attaquant pourrait récupérer des cookie de session en se servant d'une attaque par XSS. Pour récupérer des informations sur les conversations d'un utilisateur, une solution serait d'envoyer un message avec du code javascript en tant que sujet du message. Ce dernier sera stocké sur le serveur et lorsque l'utilisateur cible se connectera à sa boîte de réception, le code malicieux sera exécuté car les sujets sont affichés. Tous les destinataires ou sujets des messages reçus affichés par la victime peuvent être envoyés à l'attaquant (tant que notre message est toujours visible dans la liste et donc que le script est exécuté).

L'exemple de code ci-dessous à insérer dans le champ sujet du message permet de récupérer facilement tout le contenu html de la page après un délai de une seconde (pour laisser le temps aux autres informations de s'afficher dans le navigateur de la cible). Ici l'exemple ne transmet pas les informations à l'attaquand mais les affichent à la cible.

```
<script>
  setTimeout(() => {alert(document.documentElement.innerHTML);}, 1000);
</script>
```

4.0.2 Perte de disponibilité

Ce scénario concerne la destruction d'informations par les différentes sources de menaces citées dans le tableau. Pour parvenir à ses fins, un attaquant aura plusieurs approches possibles tels que :

- Manipulation de l'URL
- Attaques XSS
- Attaques CSRF
- Injections SQL

Comme ennoncé précédement, comme les paramètres des actions sont passées dans l'url en clair, il serait possible pour un utilisateur de supprimer un message dont il n'est pas le destinataire légitime. En effet, tous les messages de la base de données peuvent être effacés par un seul utilisateur qui lancera un script parcourant tous les id possibles et émettant la requete de suppression pour cet id.

On pourrait imaginer une action similaire lancée par un script javascript inséré dans un corps de message (attaque XSS), permettant par exemple "l'autodestruction" du message envoyé après un certain temps après la lecture. Le script serait donc exécuté lors de l'ouverture du message, et récupérerait l'id du message dans l'url. Après quelques secondes, on exécute la suppression de ce message en contactant le bon end-point avec cet id.

Même si une simple verification que seul le propriétaire du message peut le supprimer, un attaque par CSRF est toujours possible si aucunes autres mesures de protection ne sont mises en oeuvre. Un attaquant pourra envoyer un lien déguisé (short google url ou autre) à la cible. Si lors du clic sur ce dernier la victime est connectée et authentifiée sur le site, l'attaquant pourra forcer la victime à supprimer un message sur son compte.

Une injection SQL peut permettre à un attaquant de complètement détruire la base de donnée (drop) ou simplement arrêter le serveur avec une commande "shutdown".

4.0.3 Perte d'intégrité

Dans ce scénario d'attaque, un attaquant pourrait se servir de certaines attaques dans le but de corrompre les données présentes dans la base de données produisant ainsi une perte d'intégrité.

Si l'utilisateur utilisé pour faire la requête SQL officiellement prévue a les droits d'écriture dans la base de donées, un attaquant pourrait se servir d'une injection SQL pour effectuer plusieurs actions sur la base de données. Par exemple, au lieu de simplement lire une entrée dans la BDD, il pourrait aussi effectuer une autre action comme ajouter un utilisateur avec les droits administrateurs. De plus, il pourrait corrompre tous les messages. Dans une première phase, il pourrait récupérer les messages, puis dans un second temps les chiffrer / modifier ou les supprimer. Finalement, il pourrait renvoyer ces messages modifiés dans la base de données.

4.0.4 Accès non autorisé à des ressources

Avec la version actuelle du programme, un attaquant pourrait effectuer une énumération des ressources accessibles sur le serveur web. Grâce à un programme comme dirbuster et un dictionnaire des noms de dossiers et de ressources les plus courantes, ce dernier pourra énumérer chacunes des ressources et tenter d'y accèder sans que ces dernières ne soient référencées dans les pages accessibles par les clients.

Dans le cas de cette application, un dossier *docker* est à la racine du serveur web. Un utilisateur ne devrait pas avoir accès à ce dossier. Avec dirbuster, il se peut qu'un utilisateur trouve cette ressource.

5 Contremesures

tableau récap et montrer new implémentation

#	**Vulnérabilité**	**Contremesure**
5.1	Manipulation de l'URL	Effectuer un contrôle côté serveur, paramètres non prédictibles
5.2	Attaque XSS	Filtrer les entrées utilisateur stockées lors de l'affichage
5.3	Injection SQL	Filtrer les entrées utilisateur
5.4	Attaque CSRF	Passage des paramètres en POST
5.5	Path exploration	Filtrer les accès aux ressources
5.5	Bruteforce login	Limiter le nombre de tentatives

5.1 Contremesure

Afin d'éviter l'accès à certains répertoires non autorisés, nous avon smis en place des règles au sein du fichier httpd.conf :

```
<Directory /var/www/html/wechat/>
  AllowOverride None
  Order allow,deny
  Require all granted
  Options -Indexes
  ErrorDocument 403 /wechat/errors/forbidden.php
  <FilesMatch "\.php|\.css|\.html">
    Allow from All
  </FilesMatch>
</Directory>
```

Avec la définition de ces règles au sein du serveur Apache, l'accès au sous-répertoires du site Internet ne seront pas accessible par les utilisateurs et ils seront redirigés vers la page d'accueil ou d'authentification.

5.2 Contremesure

Afin d'éviter les attaques XSS, nous avons mis en place la fonction htmlentities :

```
<td>'.htmlentities($mail['from'], ENT_QUOTES | ENT_HTML5, 'UTF-8').'</td>
<td>'.htmlentities($mail['subject'], ENT_QUOTES | ENT_HTML5, 'UTF-8').'</td>
```

Elle permet de convertir tous les caractères éligibles en entités HTML. Nous l'avons mis en place lorsque l'application affiche des informations de la base de données ou de la saisie utilisateur.

5.3 Contremesure

Afin d'éviter les attaques CSRF, nous avons mis en place la génération de token sur la page d'accueil :

```
$token_mail = Utils::get_instance()->random_str(32);
$_SESSION['token_mail'] = $token_mail;

```

```
$token_user = Utils::get_instance()->random_str(32);
$_SESSION['token_user'] = $token_user;
```

```
$token = isset($_SESSION['token_mail']) && isset($token) ? $token : "";

if ($_SESSION['token_mail'] === $token) {
    // Delete mail
}

// Redirect to home
```

```
$token = isset($_SESSION['token_user']) && isset($token) ? $token : "";

if ($_SESSION['token_user'] === $token) {
    // Delete user
}

// Redirect to home
```

5.4 Contremesure

Afin d'éviter les attaques CSRF, nous avons mis en place une vérification du referer header :

```
public function redirect_if_is_not_correct_file_origin($files_origin) {
  $http_referer_file = substr($_SERVER['HTTP_REFERER'],
      strrpos($_SERVER['HTTP_REFERER'], '/') + 1);
  $ask_pos = strrpos($http_referer_file, '?');

  if ($ask_pos !== false) {
      $http_referer_file = substr($http_referer_file, 0, $ask_pos);
  }

  foreach ($files_origin as $file_origin) {
      if ($http_referer_file === $file_origin) {
          return;
      }
  }

  header('location:/wechat/home.php');
  exit();
}
```

5.5 Contremesure

Afin d'éviter les injections SQL, nous avons mis en place des requêtes préparées :

```
/**
 * Retrieves user's ID with digest
 */
public function get_id() {
    $query = "SELECT id
                    FROM users
                    WHERE digest=:digest;";
    $parameters = array(new Parameter(':digest', $_SESSION['digest'],
        PDO::PARAM_STR));
    $array = self::$_database->query($query, $parameters);

```

```
    return count($array) >= 1 ? $array[0]['id'] : null;
}
```

```
/**
 * Get the result of a query
 */
public function query($sql, $parameters) {
    if (!isset($this->_pdo)) {
        $this->connection();
    }

    $stmt = $this->_pdo->prepare($sql);

    if (isset($parameters)) {
        foreach($parameters as $parameter) {
            $stmt->bindParam($parameter->get_name(), $parameter->get_value(),
                $parameter->get_pdo_type());
        }
    }

    $stmt->execute();

    return $stmt->fetchAll(PDO::FETCH_ASSOC);
}
```

5.6 Contremesure

Afin d'éviter les attaques brute force, nous avons mis en place une liste noire d'adresse IP :

```
/**
 * Retrieves blacklisted attempt
 */
public function get_attempt() {
    $blacklist_id = $this->get_id();
    $query = "SELECT attempt
                FROM blacklist
                WHERE id=:id;";
    $parameters = array(new Parameter(':id', $blacklist_id, PDO::PARAM_STR));
    $array = self::$_database->query($query, $parameters);

    return count($array) >= 1 ? $array[0]['attempt'] : null;
}
```

5.7 Contremesure

Afin d'éviter les erreurs d'exécution, nous avons mis en place la vérification des saisies utilisateurs :

```
if (isset($from) && isset($to) && isset($subject) && isset($body)) {
    $len_from = strlen($from);
    $is_correct_from = $len_from >= Database::USERNAME_MIN &&
                    $len_from <= Database::USERNAME_MAX;

    $len_to = strlen($to);
    $is_correct_to = $len_to >= Database::USERNAME_MIN &&
                    $len_to <= Database::USERNAME_MAX;

```

```
10      $len_subject = strlen($subject);
11      $is_correct_subject = $len_subject >= Database::PHP_STR_MIN &&
12                      $len_subject <= Database::PHP_STR_MAX;
13
14      $len_body = strlen($body);
15      $is_correct_body = $len_body >= Database::PHP_TEXT_MIN &&
16                      $len_body <= Database::PHP_TEXT_MAX;
17
18    if ($is_correct_from &&  $is_correct_to && $is_correct_subject &&
          $is_correct_body) {
19      // Do something
20    }
21  } else {
22   // Inout incorrect
23  }
```