Atelier sécurité - deuxième partie

Failles en PHP et injection SQL

Victor Stinner

Club des utilisateurs de logiciels libres de l'UTBM

9 octobre 2005

Sommaire

Présentation de PHP

- PHP est un langage interprété écrit par *Rasmus Lerdorf* en 1994.
- Il est très utilisé aujourd'hui dans les sites Internet (25 millions de site en 2005).
- Il est apprécié pour sa facilité d'utilisation, la grande quantité des bibliothèques incluses, son aspect "logiciel libre", et enfin sa grande communauté d'utilisateurs (documentation).

Failles PHP

- PHP est souvent le premier langage appris par les concepteurs de site Internet.
- La sécurité est souvent négligée par manque d'expérience, ou simplement par manque d'intérêt.
- Les failles de sécurité sont plus ou moins graves, mais trop souvent on peut avec un peu d'expérience pénétrer dans un site sans grande difficulté.
- Vieil adage : « 90% des erreurs sont situées entre le clavier et l'écran » :-)

Exemples de failles

- Formulaire vérifiant de manière incomplète les entrées des utilisateurs.
- Outil de téléchargement permettant de télécharger n’importe quel fichier.
- Injection de SQL (détaillée plus tard).
- etc. (la liste est longue)

Limiter la casse

- Supposer que ce qui vient de l'extérieur provient d'une personne qui cherche à nous nuire.
- Ne pas expliciter les cas à proscrire, mais plutôt les cas à autoriser.
- Concevoir l'architecture du système pour isoler les parties sensibles.

Sommaire

Présentation de SQL

- SQL est un langage permettant d'interroger ou modifier un système de gestion de base de données (SGBD).
- Il est proche du langage naturel : sélectionne le prénom des personnes parmi tous les utilisateurs dont le nom est STINNER donne par exemple ...
- La requête SQL :

```
SELECT prenom FROM utilisateurs
WHERE nom='STINNER';
```

Injection de SQL

- Beaucoup de sites web privilégient l'utilisation d'un SGBD pour le stockage des données (plutôt que l'utilisation de fichiers).
- Malheureusement, l'écriture des "requêtes" SQL est souvent mal conçue et permet l'injection de SQL arbitraire.
- L'injection de SQL est très dangereuse car elle autorise l'accès en lecture et écriture à l'ensemble du SGBD, et dans certains cas donne même accès au système de fichier !

Exemples d'injection

- SELECT login FROM users WHERE login='$login' AND password='$password' ;
  où $login et $password sont saisis par l'utilisateur et n'ont subi aucun traitement.
- En utilisant le mot de passe « xxx' OR 'a'='a », on outrepasse la vérification du mot de passe.
- Requête résultante :
  SELECT login FROM users WHERE login='haypo' AND password='**xxx' OR 'a'='a**' ;

Requêtes à risque

- `SELECT (...) WHERE champ='$valeur';`
- `INSERT INTO table (a,b)`
  `VALUES('$valeura', '$valeurb');`
- Négligence des apostrophes (les pires) :
  `SELECT titre, description`
  `FROM livre WHERE id=$id;`

Faire parler les failles

- Utiliser une chaîne vide, une chaîne très courte / longue.
- Utiliser un nombre nul, négatif, trop grand / petit.
- Souvent, une simple apostrophe permet de trouver une faille.

Exploiter les failles

- Outrepasser un test :
  « SELECT ... WHERE id='**xxx' OR 'a'='a**' ; »
- Injecter une requête avec UNION :
  « `SELECT ... WHERE id='`**`xxx' UNION SELECT ... FROM table#'`**`;` »
- Fichiers :
  « `SELECT ...` **`INTO OUTFILE '/tmp/exploit'`**`;` »
  « `SELECT` **`LOAD_FILE('/etc/passwd')`**`;` »

Apostrophe et magic quote

- L'option magic_quotes de PHP (actif par défaut) ajoute un anti-slash devant les caractères suivants : « ' », « " », et les caractères ayant un code ASCII inférieur à 32.
- Surtout ne pas penser être protégé avec cette seule option. D'ailleurs, souvent un stripslashes est utilisé avant un INSERT ou UPDATE, et les requêtes du type « ... WHERE id=$id ; » sont toujours exploitables.
- On peut utiliser une écriture hexadécimale pour écrire une chaîne de caractère sans apostrophe. Exemple : 0x2F6574632F706173737764 représente "/etc/passwd".

Renforcer ses requêtes

- Utiliser des fonctions génériques qui vont générer les requêtes plutôt que de les écrire à la main. En plus, il sera d'autant plus facile d'utiliser un autre type de SGBD.
- Utiliser une fonction qui échapper les caractères spéciaux. Exemple : `mysql_real_escappe_string()` du module MySQL de PHP, ou `qstr()` de la bibliothèque AdoDB.
- Rappel : considérer que ce qui vient de l'extérieur est susceptible de venir d'un pirate, et indiquer les cas valides plutôt que lister les cas invalides.

Sommaire

Contre exemple du "lister les cas autorisés"

- On veut interdire les requêtes contenant certains "mots-clés" tels que "/bin/sh" ou "su root".
- On fera une liste des commandes à proscrire. On essayera d'être exhaustif : "/bin/sh", "/bin/bash", "su - root", etc.
- Mais on peut facilement autrepasser ces interdictions : "/usr/bin/bash", "su $(echo root)", "su r00ˆHˆHoot ou carrément :

```
echo "su ">> plop; echo "root">> plop; ./plop
```

À propos de l'architecture

- Il faut isoler au maximum le code sensible et c'est celui qui sera le mieux pensé / relu.
- Pour isoler le code, on peut utiliser des modules. Il faut alors bien renforcer la fonction qui va charger ou non le module.
- Beaucoup de démons Unix/BSD (ex : Apache) utilisent la séparation des droits : le serveur est lancé en tant qu'utilisateur root, mais dès qu'un client se connecte, une nouvelle instance est lancée en tant qu'utilisateur normal.

Fonctions dangereuses

- Appels systèmes comme `system` (PHP/Perl), `passthru` (PHP), etc.
- Fonctions permettant d'évaluer un code arbitraire comme `eval` (PHP) ou `compile`/`exec` (Python)