Sécurité des sites Web
Pas un cours un recueil du net

INF340 Jean-François Berdjugin

Vulnérabilité

- Définition (wikipédia) : Dans le domaine de la sécurité informatique, une vulnérabilité est une faiblesse dans un système informatique permettant à un attaquant de porter atteinte à l'intégrité de ce système, c'est-à-dire à son fonctionnement normal, à la confidentialité et l'intégrité des données qu'il contient. On parle aussi de faille de sécurité informatique.

Architecture trois tiers

- Attaque basées sur le serveur web (XST Cross site tracing)
- Attaque basées sur le serveur d'application (XSS Cross Site Scripting) + failles liées au langage.
- Attaque basées sur le SGBD (SQL Injection, smuggling)
- Attaque basées sur le client (XSS)

Rem : restent aussi les attaques TCP et IP.

SQL

- SQL Injection
- SQL Smuggling
- Blind SQL Injection

SQL Injection (principe)

- Principe : exploiter une faille de sécurité en injectant du code SQL.
- Moyen : formulaire, URL, URL avec script PERL

SQL Injection (exemple)

- Select uid from users where login='login' and pass='condense du pass'
- Avec pour **login '--**

=> La suite devient commentaire

Select uid from users where login='login'--'
and pass='condense du pass'

SQL Injection (exemple)

- Plein de requêtes marrantes (source http://www.phpcs.com)
  - *select*
    - *' OR 1=1"); //*
    - *login'#*
  - *INSERT INTO membres (login,password,nom,prenom,email,level) VALUES ('$login','$password','$nom','$prenom','$email','$level')*
    - *Si le pirate écrit, dans le champ du email, le code suivant : monemail@monsite.com', '1'# la requête deviendra :*
    - *INSERT INTO membres (login,password,nom,prenom,email,level) VALUES ('Terri','MonPass','Terri','Terri',monemail@monsite.com',1'# ','$level')*
  - UPDATE membres SET password='$var_pass',nom='$var_nom',email='$var_email' WHERE id='$id'
    - champ du mot de passe le code suivant : ',level='3
    - UPDATE membres SET password='',level='3',nom='$var_nom',email='$var_email' WHERE id='$id'

SQL Injection (exemple)

- Composant Joomla (http://www.exploit-db.com/)

Un script perl (EXPLOIT) est fourni.
- Autre attaque (http://www.securityfocus.com)

SQL Injection (protection)

- Vous êtes le programmeur :
  - Des requêtes préparées
  - Utiliser des fonctions pour filtrer les variables :
    - Fonctions ctype_*()
    - Extension filter en php5
  - Contrôle des variables :
    - Liste blanche
    - Liste noire
    - Liste grise
    - Contrôle de format
    - Contrôle de type
  - Le moins d'info dans les cookies
- Vous êtes intégrateurs
  - Pas de béta version
  - CVS et mise à jour régulières
  - Produit reconnus

SQL Smuggling (principe, exemple, protection)

- Principe (http://www.comsecglobal.com): utiliser la traduction automatique des caractères non supporte
- Exemple : le caractère U+02BC peut-être traduit en U+0027 (la quote)
- Protection :
  - interdire la traduction automatique des caractères Unicode,
  - Liste blanche des caractères autorisés.
- Rem : WAF(Web Applucation Firewall), contournés.

BLIND SQL INJECTION

- Pour les scripts à réponse binaire par exemple une authentification
- Découvrir par essai successifs des informations sur les tables
- SELECT * FROM table WHERE champ = 'a' OR @@version > 3; Avec @@version une variable MySQL

…

XSS Cross Site Scripting (principe)

- Principe : déposer sur un site (application web) un script qui sera ensuite consulté par les autres utilisateurs, la consultation pouvant entrainer l'exécution par le navigateur du script qui pourra :
  - Voler des informations (cookies, session)
  - Effectuer une redirection éventuellement transparente
  - Afficher un contenu non autorisé
  - Planter le site
  - Réaliser des opérations sur le site

XSS Cross Site Scripting (principe)

- Trois types (http://www.webappsec.org, http://ha.ckers.org/) :
  - *Permanent* : Les données sont stockées (SGBD) puis réaffichées sans que les caractères spéciaux HTML n'aient été encodés ('<' et '>' par exemple).
  - *Non permanent* : Les données du client sont utilisée de manière brute par le serveur pour lui répondre. Où est le problème ? L'ingénierie sociale.
  - *Local* : le script écrit dans sa propre page pour générer du code (exécution de code à distance)

XSS Cross Site Scripting (exemple)

- http://blog.developpez.com, utilisation de $_SERVER['PHP_SELF']
- Le script :
  -
```
<html>
<body>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>">
<input type="hidden" name="submitted" value="1" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```
- Le résultat sur http://localhost/phpself.php :
  -
```
<html>
<body>
<form action="/phpself.php">
<input type="hidden" name="submitted" value="1" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```
- Le résultat sur http://localhost.com/phpself.php/on_injecte_ici
  -
```
<html>
<body>
<form action="/phpself.php/on_injecte_ici">
<input type="hidden" name="submitted" value="1" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```
- http://localhost/phpself.php/%22%3E%3Cscript%3Ealert('xss')%3C/script%3E%3Cdata
  -
```
<html>
<body>
<form action="/phpself.php"><script>alert('xss')</script><data">
<input type="hidden" name="submitted" value="1" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

XSS Cross Site Scripting (protection)

- Filtrer les variables pas de '<' ou de '>'
- Penser aussi à filtrer en sortie

XSS Cross Site Scripting (des bébés en exemple)

- XSRF Cross Site Request Forgeries
- XST Cross site tracing

XSRF Cross-Site Request Forgeries (principe et exemple)

- Principe : un utilisateur devient complice sans en être conscient. Le problème est lié a une authentification à un instant donné et non une autorisation pour une action donnée.
- Exemple : Alice n'a pas de pouvoir mais Bob en a.
  - Alice envoie l'url de l'action qu'elle ne peut faire à bob, dans une IMG par exemple.
  - Le navigateur de Bob suit le lien et fourni le mot de passe
  - Le navigateur n'ayant pas récupéré d'image, il n'affiche rien.

XSRF Cross-Site Request Forgeries (protection)

- Pas GET pour une action sinon Alice trouve le lien.
- Utiliser des jetons (durée de vie), comme cela Alice doit agir vite.
- Utiliser le champ referer (HTTP) pour vérifier l'adresse du script appelant, si ce n'est pas celui utilisé par Bob alors il y a un problème.
- Demander des confirmations à l'utilisateur, comme cela Bob sait ce qu'il fait.

XST Cross site tracing (principe et exemple)

- Principe (http://*ww.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf*): utilisation de la méthode trace HTTP. La méthode TRACE HTTP renvoie au client l'entête avec donc les données d'authentification et les cookies.
- Exemple : L'attaqué doit exécuter un script qui invoque TRACE, avec par exemple XMLHttpRequest, ce qui permet de récupérer les cookies de l'utilisateur. Puis le script doit renvoyer les cookies à l'attaquant.

XST Cross site tracing (protection)

- Protection : interdire les trace
  - Dans un .htaccess
    - RewriteEngine On
      RewriteCond %{REQUEST_METHOD} ^TRACE
      RewriteRule .* - [F]

Faille dans la logique application

- Php pour l'exemple (http://www.phpsecure.info) :
  - Les failles de PHP
    - Vulnérabilité 'escape shell' (system())
    - Fonction include() avec de paramètre pour accéder à des fichiers 'interdit'
    - Fonction mail() (passage de l'entête du mail => mail anonyme, multiples, …)
    - Les fichiers de logs de Apache et autre fichiers système en lecture (include).
    - Script d'upload (upload d'un .php)
  - PHP & MySQL
    - Requetes MySQL multiples
    - Fakes posts
    - Stupid DoS
    - Bypasser une authentification

  => Utiliser un modèle de sécurité => utiliser un framework

Faille dans l'application (exemple)

- Des outils (www.insecure.org) :
  - Libre : WebInspect, SAINT, Nesus, …
- Les failles du moment :
  - http://www.certa.ssi.gouv.fr/
- Un bon forum :
  - http://www.w3.org/Security/faq/www-security-faq.html

Les firewall applicatif

- Les firewall classique + IDS (Intrusion Detection System) protègent efficacement des attaques réseaux.

⇒ Il faut pour le méchant passer aux couches supérieures (la couche application)

⇒ Il faut se protéger du méchant en filtrant au niveau applicatif

Les firewall applicatifs

- Imposés maintenant par certaines normes, par exemple pour le commerce électronique.
- Deux architectures :
  - Sécurité positive
    - Tout est fermé
    - On ouvre spécifiquement (=> apprentissage + mise à jour du modèle conceptuel)
  - Sécurité négative
    - On ouvre tout
    - On ferme spécifiquement (=> mise à jour permanantes des règles de fermeture)
- Adaptation pour SSL :
  - Il faut que le firewall puisse déchiffrer le SSL
    - Intégré au serveur
    - A la responsabilité du chiffrement

Les firewall applicatifs

- Protocoles pris en charge ?
- Méthodes d'authentification ?
- Consigne (log) ?
- Matériel/Logiciel ?

=> Imposés pour certaine applications critiques mais beaucoup de contrainte.

Conclusion

- Choisir un Framework et des outils supportés
- Maintenir à jour les applications
- Filtrer les entrées et les sorties