Vulnérabilités web (suite ISI)

- Compléments sur les injections SQL
- Failles CSRF
- Compléments sur les XSS
- Autres problèmes de protection de ressources
- Autres failles d’injection de code

Compléments injections SQL

Rappel ISI : Principe d'une injection SQL

- Une injection SQL est possible quand on créé une requête SQL partiellement à partir de données contrôlées par l'attaquant

Exemple (liste non exhaustive…) :

```
db.execute(« INSERT INTO machin VALUES(‘» + v + « ‘) »)

db.execute(« UPDATE machin SET toto=’ » + v + « ‘ )

db.execute(« UPDATE machin SET toto=’titi’ WHERE toto=’ » + v + « ‘)

db.execute(« SELECT trucs FROM machin WHERE toto=’ » + v + « ‘)
```

Compléments injections SQL

L'injection étendue à autre chose que du SQL...

- Ce principe d'injection n'est pas limité au SQL (ldap, injections shell, ...)

Exemple d'injection shell :

```
char host[32];
printf(« Entrez l'hote a pinguer? »);
fgets(host, 32, stdin);
char cmd[256];
snprintf(cmd, 256, « ping %s », host);
system(cmd);
```

Entrez l'hote a pinguer ? **google.com ; rm -rf /**

Compléments injections SQL

Comment les détecter sans le code source ?

- Comment détecter une vulnérabilité à une injection SQL si on ne dispose pas du source de l'application ? Essayer d'utiliser des caractères spéciaux pour voir si cela provoque une erreur…
  - ‘ (quote) ‘’ (guillemets) – (commentaire en SQL)…
  - Pour d'autres types d'injections (autres que SQL), utiliser des caractères significatifs dans le langage de requête ciblé…
- Parfois les messages d'erreur ne sont pas affichés...

Compléments injections SQL

Altération de conditions WHERE (exemple 1)

```
db.query(« SELECT * FROM users WHERE username=’ » + u + « ‘ AND
password=’ » + p + « ‘ »)
db.fetchall()

if db.rowcount > 0 :
   #authentification réussie !
```

- Username envoyé : **toto**
- Password envoyé : **’ OR ‘1’=’1**
- Requête construite :

  SELECT * FROM users WHERE username=’toto’ AND password=’’ OR ‘1’=’1’

Compléments injections SQL

Altération de conditions WHERE (exemple 2)

```
db.query(« SELECT * FROM users WHERE username=’ » + u + « ‘ AND
password=’ » + p + « ‘ »)
db.fetchall()

if db.rowcount == 1 :
   #authentification réussie !
```

- Username envoyé : **toto**
- Password envoyé : **’ OR 1=1 COUNT 1 --**
- Requête construite :

  SELECT * FROM users WHERE username=’toto’ AND password=’’ OR 1=1 COUNT 1

Compléments injections SQL

Altération de conditions WHERE (exemple 3)

```
db.query(« SELECT * FROM users WHERE (username=’ » + u + « ‘ AND
password=’ » + p + « ‘) »)
db.fetchall()

if db.rowcount == 1 :
   #authentification réussie !
```

- La facon d’exploiter dépend du code vulnérable…
  - Si on a le code source, c’est pratique.
  - Sinon, on peut s’aider des messages d’erreurs pour déduire la facon dont la requete est construite…
  - Sinon, on peut faire des hypothèses et tests...

Compléments injections SQL

Récupération d'informations

- Si des données résultantes de la requête sont affichées, il est facile de récupérer des données confidentielles :

```
db.query(« SELECT id,content FROM items WHERE id= '» + id + « ' »)
#on suppose que le script affiche ensuite le content à l'écran
```

- id : toto' UNION SELECT machin,chouette FROM autretable –
- Requete complete : SELECT id,content FROM items WHERE id='toto' UNION SELECT machin,chouette FROM autretable –'
- Attention au nombre de colonnes...

Compléments injections SQL

Récupération d'informations

- Si des données résultantes de la requête sont affichées, il est facile de récupérer des données confidentielles :

```
db.query(« SELECT id,content FROM items WHERE id= '» + id + « ' »)
#on suppose que le script affiche ensuite le content à l'écran
```

- On peut tâtonner si on ne connaît pas la structure exacte de la requête et qu'on n'a pas le code source…
- Il y a pas mal de fonctions utiles dans MySQL (ex : CONCAT)
- La base **information_schema** est très utile...

Compléments injections SQL

Récupération d'informations (attaque en aveugle)

```
db.query(« SELECT * FROM users WHERE username=' » + u + « ' AND
password=' » + p + « ' »)
db.fetchall()

if db.rowcount > 0 :
    #authentification réussie !
```

- Parfois, les résultats ne sont pas affichés… On peut avoir par ex, seulement deux types de réponses (ex : « OK » / «echec»)
- On peut extraire en principe n'importe quelle information par dichotomie.

Compléments injections SQL

Récupération d'informations (attaque en aveugle)

```
db.query(« SELECT * FROM users WHERE username=' » + u + « ' AND
password=' » + p + « ' »)
db.fetchall()

if db.rowcount > 0 :
   #authentification réussie !
```

- Savoir le nombre d'entrées?
  SELECT * from users WHERE username='toto' AND password='' OR '1'='1' LIMIT 100,1
- L'authentification marche ou échoue suivant qu'il y a plus ou moins de 100 entrées. Ensuite : on fait par dichotomie
- On combine AND '1'='0' et UNION pour accéder à une autre table (cf slides précédents...)

Compléments injections SQL

Récupération d'informations (attaque en aveugle)

```
db.query(« SELECT * FROM users WHERE username=' » + u + « ' AND
password=' » + p + « ' »)
db.fetchall()

if db.rowcount > 0 :
   #authentification réussie !
```

- Récuperer du contenu ?
  SELECT * from users WHERE username='toto' AND password='' OR '1'='1' UNION SELECT machin FROM chouette WHERE machin LIKE « A % » LIMIT 100,1
- Permet de savoir le nombre d'entrées commencant par 'A' … on teste au cas par cas
- C'est long mais facilement automatisable

Compléments injections SQL

Récupération d'informations (timing-based)

```
db.query(« SELECT * FROM users WHERE email=' » + u + « ' »)
db.fetchall()
if db.rowcount == 1 :
   #envoi du mail de récupération
print « Nous vous avons peut-être envoyé un mail... »
```

- Parfois, on n'a aucune réponse du tout (meme pas « OK / echec »)… Solution : chronometrer
- Il y a une fonction SLEEP(s) dans MySQL
- Evaluation booléenne de gauche à droite :
  - SELECT 1 AND SLEEP(1) → 1 seconde
  - SELECT 0 AND SLEEP(1) → immédiat

Compléments injections SQL

Récupération d'informations (timing-based)

```
db.query(« SELECT * FROM users WHERE email=' » + u + « ' »)
db.fetchall()
if db.rowcount == 1 :
    #envoi du mail de récupération
print « Nous vous avons peut-être envoyé un mail... »
```

- SELECT * FROM users WHERE email=" OR 1=1 AND 0<(select count(*) FROM machin WHERE chouette LIKE « A % ») AND SLEEP(1) ;
  - Attends N secondes (avec N == nombre d'entrées dans users) si au moins une entrée commencant par « A » dans la table « machin » existe, sinon immediat.
  - Il existe aussi une fonction IF(cond,then,else)

Failles CSRF

Exemple député confus : Attaques CSRF

- CSRF : Cross-Site Request Forgery
- l'attaquant :
  - construit un lien de soumission du formulaire
  - envoie le lien à la victime

Formulaire vulnérable :

```
<form ...>
    <input action="/bar.php" type="text" name="foo" />
    <input type="submit" value="OK" />
</form>
```

Lien : `http://vulnerable.com/bar.php?foo=something`

Failles CSRF

Exemple député confus : Attaques CSRF

- la victime :
  - est authentifiée sur le site cible
  - suit le lien (peut-etre automatiquement, ex : <img...)
  - « valide » le formulaire à son insu
- cela fonctionne car le browser de la victime a autorité sur le site cible (député confus)

Requête envoyée par la victime :

```
GET /bar.php?foo=something HTTP/1.1
Host: vulnerable.com
Cookies: session=XXXXXXXXXX; …
```

Failles CSRF

Exemple député confus : Attaques CSRF

- « Contre-mesures » non efficaces :
  - CSRF fonctionne même sur un formulaire POST
  - CSRF fonctionne même en https
  - Etc. (liste non-exhaustive)
- Contre-mesure efficace : le CSRF-token

Formulaire sécurisé:

```
<form ...>
   <input action="/bar.php" type="text" name="foo" />
   <input type="hidden" name="token"
value="59ab5edf189c42a7" />
   <input type="submit" value="OK" />
</form>
```

Lien : http://vulnerable.com/bar.php?
foo=something&token=59ab5edf189c42a7

Failles CSRF

Exemple député confus : Attaques CSRF

- Bonnes propriétés du CSRF-token
  - imprévisible
  - lié à la session de l'utilisateur
  - ne pas oublier de le vérifier
  - ne pas le passer dans un cookie (!)

Compléments XSS

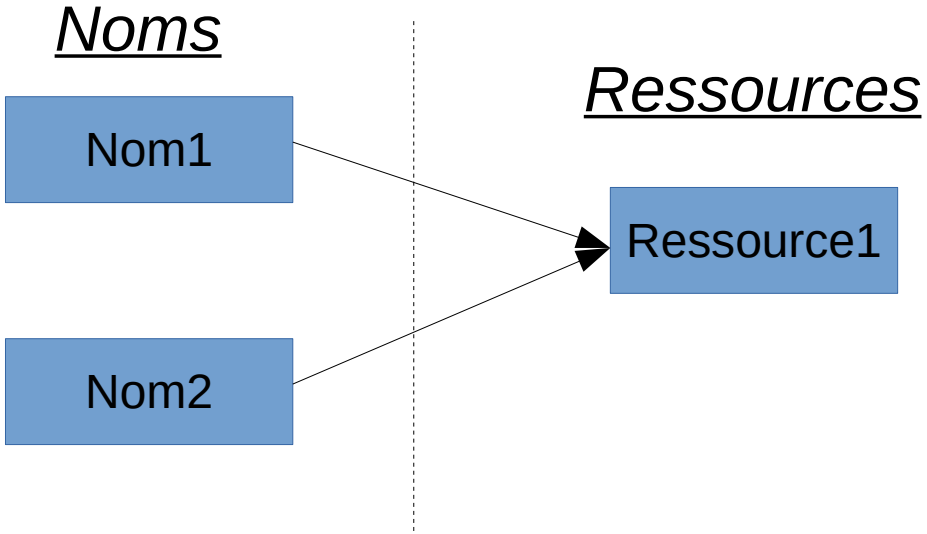
XSS : injection de code JS dans une page
Deux types de XSS :

- XSS persistent (comme vu en ISI)
  - Le JS injecté va être stocké (ex : dans une base de données)
- XSS volatile / réfléchi
  - Le JS à injecter n'est pas stocké, il est encodé dans une URL envoyée à la victime
  - Ex : http://blah.com/show.php?text =<script>….</script>
- Conséquences : phishing, vol de cookies, vol de CSRF token ...

Pb protection ressources

Plusieurs noms pour une même ressource

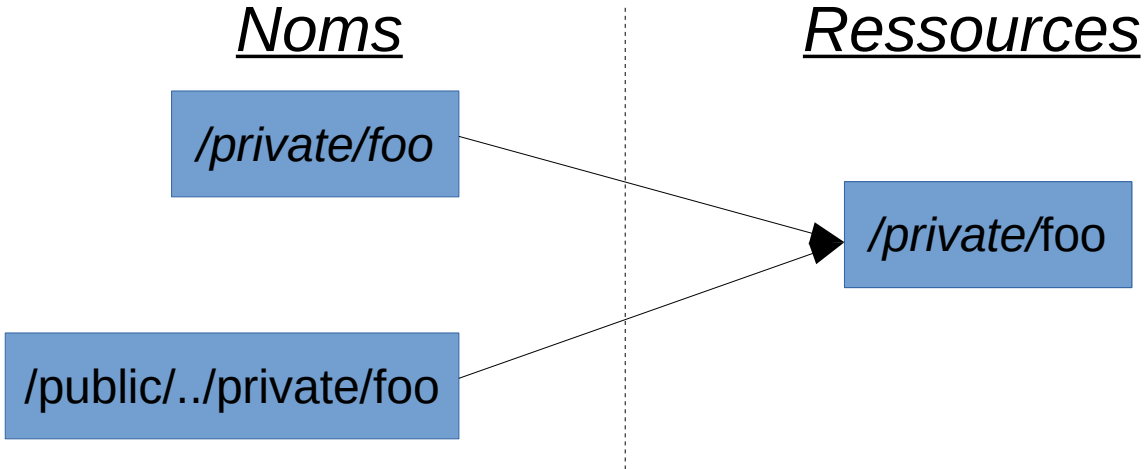
- Ressources soumises à contrôle d'accès
  - Plusieurs ressources, avec des noms (chemins, ...)
  - Plusieurs noms → une même ressource
  - Contrôle d'accès basé sur le nom de la ressource



Pb protection ressources

Plusieurs noms pour une même ressource

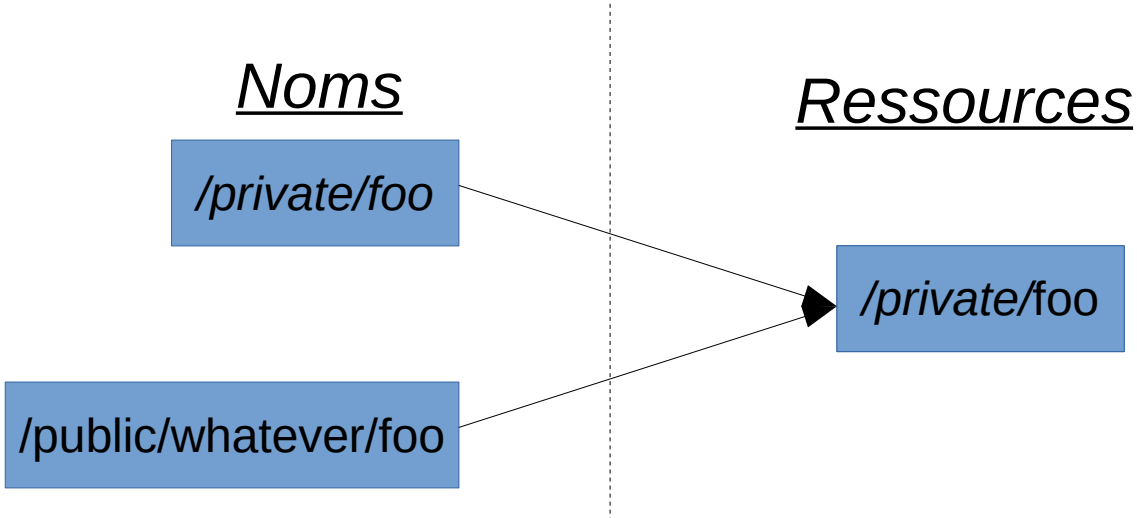
- Exemple : path traversal
  - Dossier « /public » autorisé, « /private » interdit
  - On vérifie que l'objet demandé commence par « /public »
  - Problème : « */public/../private/* »



Pb protection ressources

Plusieurs noms pour une même ressource

- Exemple : lien symbolique
  - Dossier « /public » autorisé, « /private » interdit
  - On vérifie que l'objet demandé commence par « /public »
  - Problème : présence d'un lien symbolique de « */public/whatever* » *vers* « /private/ »



Pb protection ressources

Problèmes de spécification et d'intégration

- requêtes sont parsées par le serveur et proxy
  - requête ambiguë ou malformée ? différentes interprétations possibles

Interprétation 1

```
GET public HTTP/1.0
Host: www.toto.com
Content-Length: 43
Transfer-Encoding: chunked
0
GET /private HTTP/1.0
Host: www.toto.com
[...]
```

Légende :

Première requête

Seconde requête

Pb protection ressources

Problèmes de spécification et d'intégration

- requêtes sont parsées par le serveur et proxy
  - requête ambiguë ou malformée ? différentes interprétations possibles

Interprétation 2

```
GET public HTTP/1.0
Host: www.toto.com
Content-Length: 43
Transfer-Encoding: chunked
0
GET /private HTTP/1.0
Host: www.toto.com
[...]
```

Légende :

Première requête
Seconde requête

Injection de code

Problèmes d'injection de code (PHP)

- Dossier d'upload avec exécution du code
- Remote inclusion
- … etc …