

# Sécurité

Introductions aux Bases de Données  
Nathanaël Martel

# *Sécurité*

**Deux problèmes auxquels il faut faire attention :**

- **« Injection SQL »**
- **Stockage de données sensible**

# *Sécurité : Injection SQL*

**Les requête fabriqué en PHP prennent en compte des entrées utilisateur via l'url ou un formulaire :**

**« Injection SQL »**

# Sécurité : Injection SQL

Formulaire de commentaire avec deux champs «nom» et «commentaire» :

```
$query = 'INSERT INTO commentaires (nom,
commentaire, heure) VALUES ("'.$_POST['nom'].'",
"'.$_POST['commentaire'].'", NOW())';
```

```
$_POST['nom'] = 'Nathanaël' ;
```

```
$_POST['commentaire'] = 'cet article es génial' ;
```

```
$query = 'INSERT INTO commentaires (nom,
commentaire, heure) VALUES ("Nathanaël", "cet
article es génial", NOW())';
```

# *Sécurité : Injection SQL*

**Saisi malveillante :**

```
$_POST['nom'] = '";DROP DATABASE;';
```

```
$_POST['commentaire'] = ''
```

```
$query = ?
```

# Sécurité : Injection SQL

```
$query = 'INSERT INTO commentaires (nom,  
commentaire, heure) VALUES  
("";DROP DATABASE;", "", NOW())';
```

Soit trois requêtes SQL :

- INSERT INTO commentaires (nom, commentaire, heure) VALUES (""; → erreur, requête invalide
- DROP DATABASE; → requête valide !!
- ", "", NOW()); → erreur, requête invalide

# *Sécurité : Injection SQL*

**Il faut toujours « protéger » les entrées utilisateur :**

- **Les champs de formulaire (y compris login)**
- **Les données des URL**

**... et plus généralement tout ce qui dépend d'un tiers (application, utilisateur)**

# Sécurité : Injection SQL

Le driver PDO fournis une méthode « quote » à utiliser pour cela :

```
$nom = $pdo->quote($_POST['nom']);
```

```
$commentaire = $pdo->quote($_POST['commentaire']);
```

```
$query = 'INSERT INTO commentaires (nom, commentaire, heure) VALUES ('. $nom. ', '. $commentaire. ', NOW())';
```



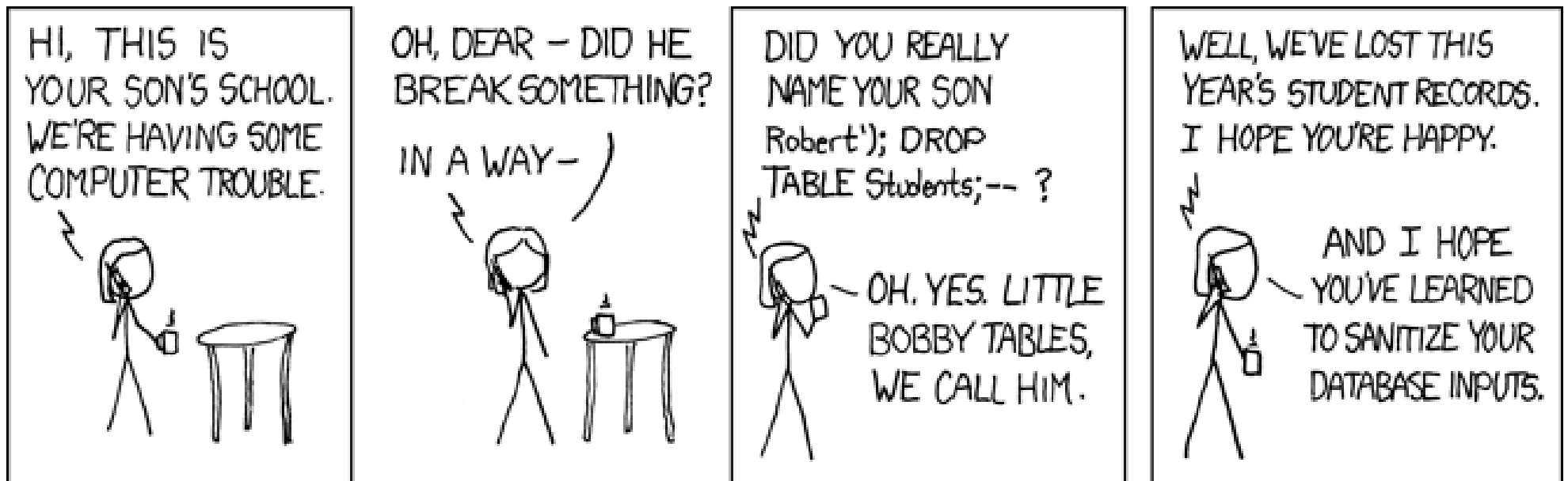
# *Sécurité : Injection SQL*

**Ce qui donne dans nos exemples :**

```
INSERT INTO commentaires (nom, commentaire,  
heure) VALUES ('Nathanaël', 'cet article es  
génial', NOW());
```

```
INSERT INTO commentaires (nom, commentaire,  
heure) VALUES ('\';DROP DATABASE;', '',  
NOW());
```

# Sécurité : Injection SQL



# Sécurité : Injection SQL



# *Sécurité : stockage de données sensibles*

**Quand les données d'une base sont rendu publique ça peut poser problème, notamment pour :**

- **Numéro de carte bleue**
  - Interdiction de stocker le cryptogramme
- **Les mots de passe**
- **Données personnelles**
  - Attention à la GDPR (ou RGPD)
  - General Data Protection Regulation (  
Règlement général sur la protection des données)

# ***Sécurité : stockage de données sensibles***

**Pour que les utilisateurs puisse s'authentifier, nous avons besoins de pouvoir comparer son mot de passe.**

**Si nous stockons le mot de passe dans la base, il peut se retrouver visible par des personnes mal intentionné (piratage, administrateur mal intentionné...).**

**Comment faire ?**

# *Sécurité : stockage des mots de passe*

**Pour que les utilisateurs puisse s'authentifier, nous avons besoins de pouvoir vérifier son mot de passe.**

**Comment faire ?**

# ***Sécurité : stockage des mots de passe***

**Stocker le mot de passe en claire dans la base.**

- **Une personne accède à la base (par injection sql par exemple)**
- **Une personne accède à une sauvegarde, accède à la machine...**

**Jamais**  
**Skyrock, reddit...**

# ***Sécurité : stockage des mots de passe***

**Stocker un hash du mot de passe (fonction à sens unique)**

```
SELECT * FROM user WHERE login='login' AND pass=md5(password);
```

- **Si la base de données est exposé, seul les hashes sont visible et pas les mots de passes**
- **Il est possible de faire une «rainbow table», une table de correspondance entre tous les mots de passe possible et leur hash...**

**Pas terrible...**



# *Sécurité : stockage des mots de passe*

## Stocker un hash du mot de passe et du sel

```
SELECT * FROM user WHERE login='login' AND  
pass=md5(password.'some_salt');
```

- La «rainbow table» est plus difficile à faire car on ne connaît pas le sel... enfin sauf si la personne malveillante à aussi accès au code
- md5 a été cassé, il en existe d'autre : sha1 (cassé aussi), sha2, sha3, sha256, haval160,4...

Bien, mais peu mieux faire

# *Sécurité : stockage des mots de passe*

## Stocker un hash du mot de passe et du sel

```
SELECT * FROM user WHERE login='login' AND  
pass=md5(password.'some_salt');
```

- La «rainbow table» est plus difficile à faire car on ne connaît pas le sel...
- md5 a été cassé, il en existe d'autre : sha1 (cassé aussi), sha2, sha3, sha256, whirlpool, haval160,4...

Bien, mais peu mieux faire

# *Sécurité : stockage des mots de passe*

Stocker un hash du mot de passe et du sel « constant » et du sel « variable »

```
$Hash = hash('sha256', $password.$login.$salt);
```

- La «rainbow table» est beaucoup plus difficile à faire (deux utilisateur différent n'ont pas le même hash même s'ils ont le même mot de passe)... enfin sauf si la personne malveillante à aussi accès au code

Ça commence à être pas mal...

# ***Sécurité : stockage des mots de passe***

**Ralentir la génération du hash.**

**Utilisation de bcrypt (Blowfish)**

- **L'utilisateur, ne fera pas la différence entre 0.000001 et 0.1 seconde**
- **Par contre la «rainbow table» devient très longue (et coûteuse) à faire**

**C'est le mieux que l'on sache faire ?**

# *Sécurité : stockage des mots de passe*

## Ne pas stocker de mot de passe

- **Utilisation d'un outil externe :**
  - facebook connect, Google, Open ID, Oauth
  - Mail, SMS

# Conclusion

Il faut toujours protéger les entrées utilisateurs