

# Sécurité d'un site PHP

---

*Lundi 16 mai 2011*

*Jean-Ferdinand Susini*

# Les impératifs de sécurité

- Une application Web est une application distribuée sur Internet => problème de sécurité :
  - Les programmes de l'application manipulent-ils des données fiables ? (intégrité)
  - Les composants de l'application interagissent-ils avec les bons interlocuteurs ? (authentification)
  - Le secret des données échangées est-il préservé ? (confidentialité)

# Les impératifs de sécurité

- Une application Web est une application distribuée sur Internet => problème de sécurité :
  - Les programmes de l'application manipulent-ils des données fiables ? (intégrité)
  - ➔ **intégrité des données échangées, mais aussi des programmes**
  - Les composants de l'application interagissent-ils avec les bons interlocuteurs ? (authentification)
  - Le secret des données échangées est-il préservé ? (confidentialité)

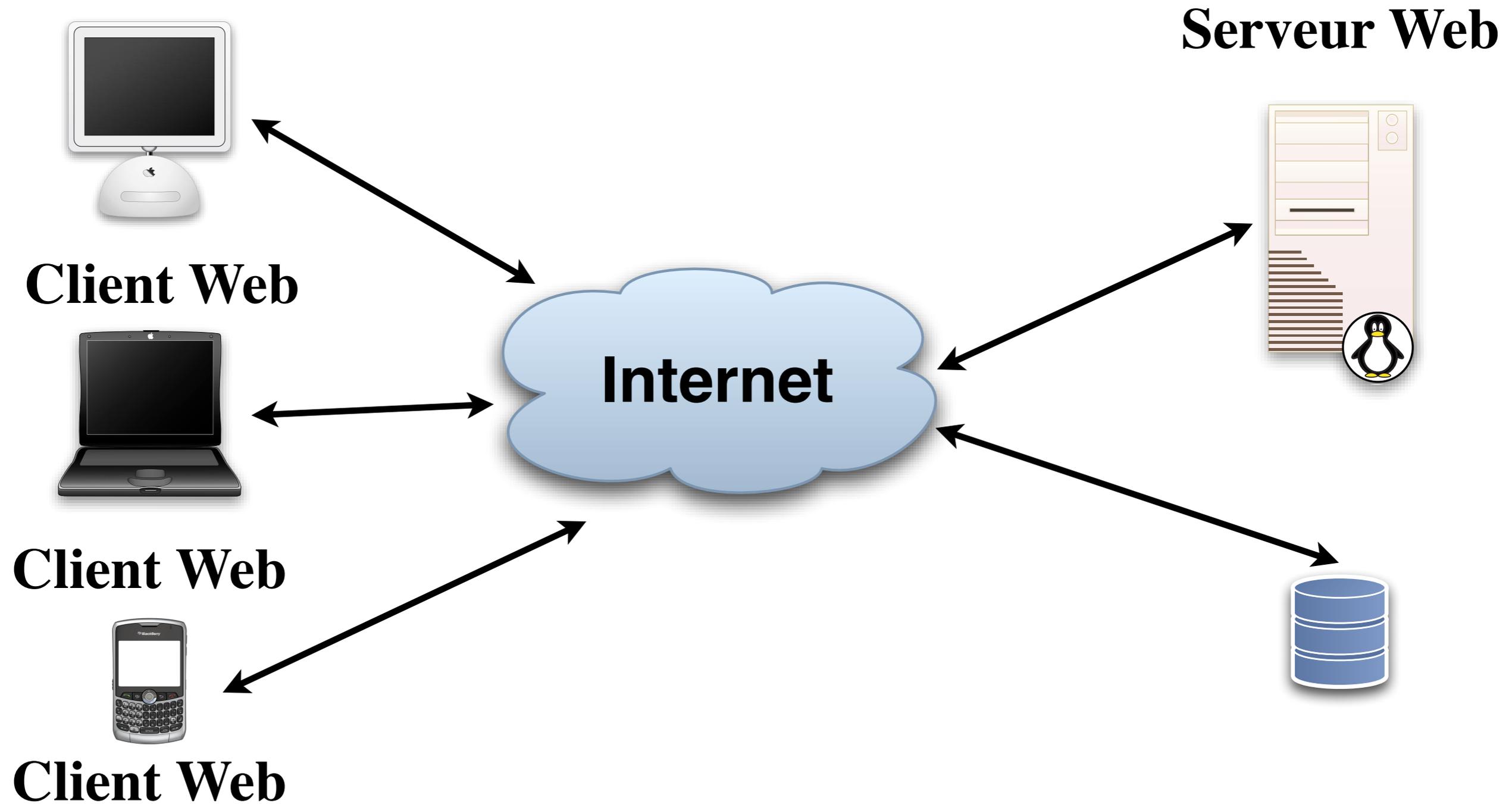
# Les impératifs de sécurité

- Une application Web est une application distribuée sur Internet => problème de sécurité :
  - Les programmes de l'application manipulent-ils des données fiables ? (intégrité)
  - ➔ **intégrité des données échangées, mais aussi des programmes**
    - Les composants de l'application interagissent-ils avec les bons interlocuteurs ? (authentification)
  - ➔ **de qui par rapport à qui ? (signature des données, pb du rejeu)**
    - Le secret des données échangées est-il préservé ? (confidentialité)

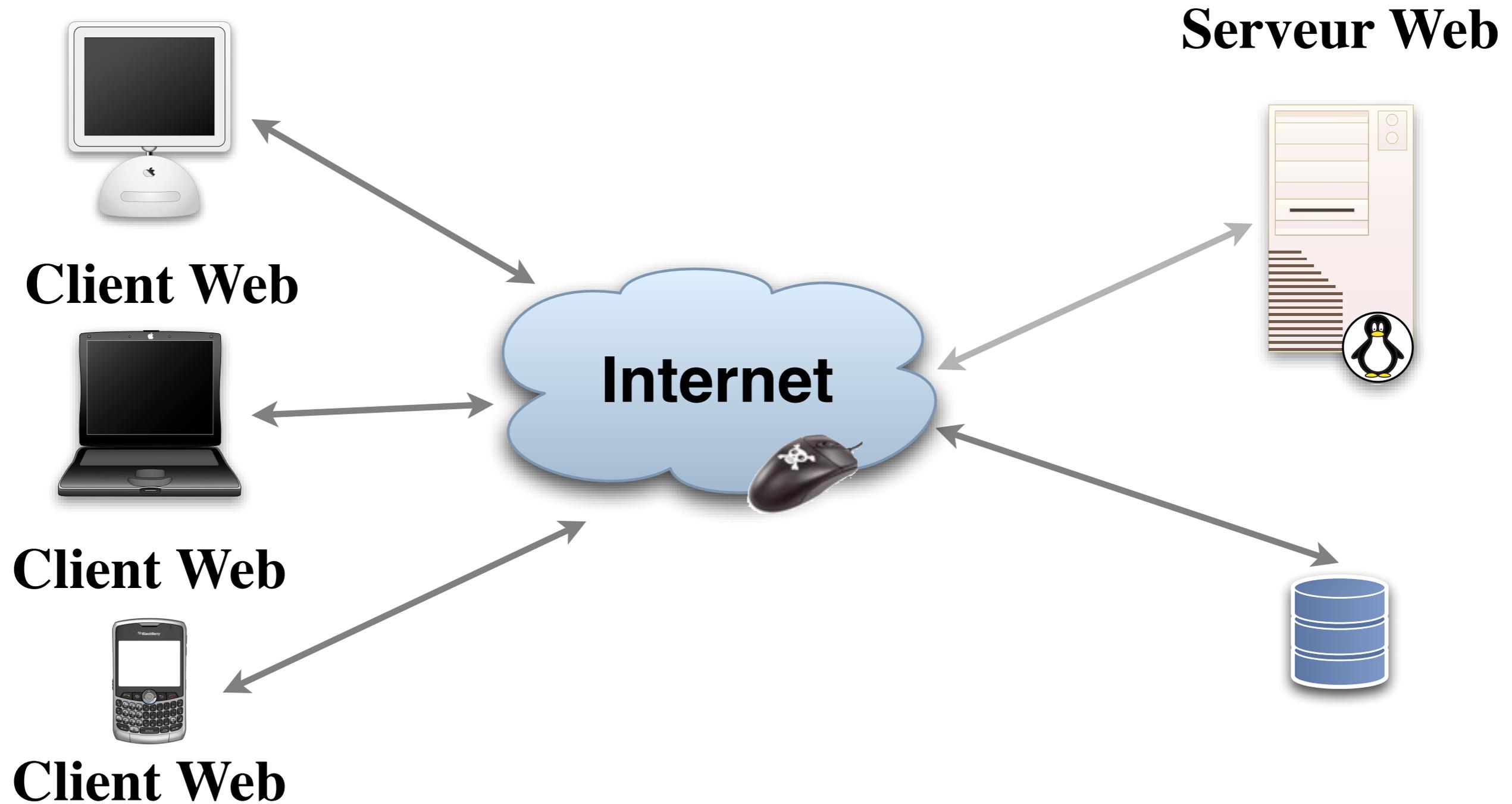
# Les impératifs de sécurité

- Une application Web est une application distribuée sur Internet => problème de sécurité :
  - Les programmes de l'application manipulent-ils des données fiables ? (intégrité)
  - ➔ **intégrité des données échangées, mais aussi des programmes**
    - Les composants de l'application interagissent-ils avec les bons interlocuteurs ? (authentification)
  - ➔ **de qui par rapport à qui ? (signature des données, pb du rejeu)**
    - Le secret des données échangées est-il préservé ? (confidentialité)
  - ➔ **par rapport à qui ?**

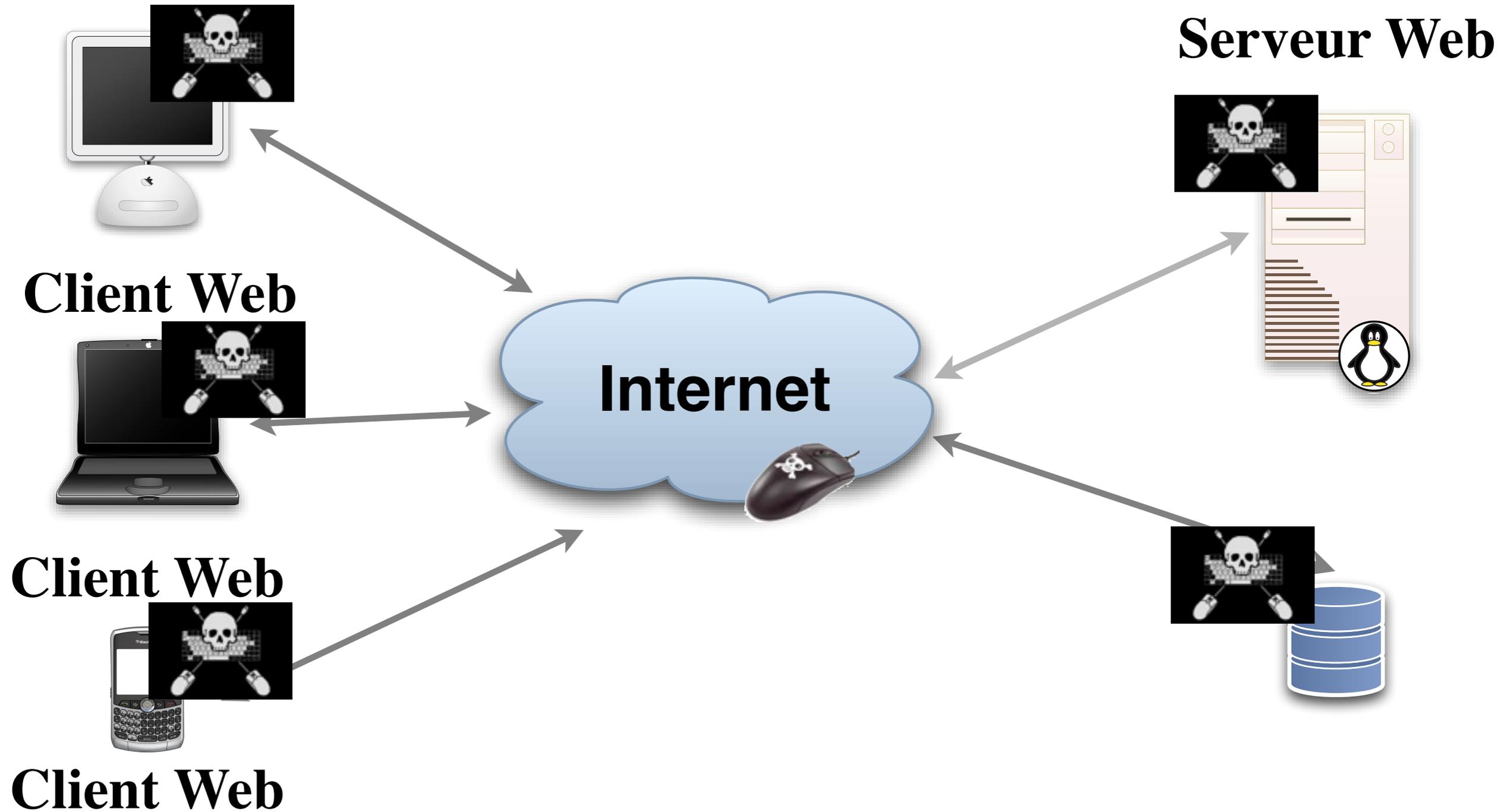
# Une application Web



# Une application Web



# Une application Web



# Les impératifs de sécurité

- Un équilibre à trouver :
  - entre le “coût” du développement de programmes ayant des propriétés de sécurité satisfaisantes et la valeur intrinsèque des données ainsi que la maintenance de l’application ou des serveurs
  - entre la “souplesse” d’utilisation et les contraintes induites sur l’utilisation de l’application des mesures de sécurité
- => Intégrer le plus tôt possible ces contraintes dans la conception de l’application pour en appréhender les “coûts”.

# Les attaques

- Considérer que l'application distribuée s'exécute dans un monde hostile :
  - Empêcher le déroulement correcte de l'application par brouillage des données
  - Désorganiser les échanges et le fonctionnement de l'application
  - Usurper l'identité d'un interlocuteur pour utiliser l'application pour soi et/ou récupérer des données confidentielles
  - Prendre le contrôle partiel ou complet du système plus ou moins discrètement
  - ...

# Mesures de sécurité

- Angle d'attaque du problème de la sécurité :
  - Fiabiliser les données et le logiciel échangés (les programmes circulent aussi sur le réseau !)
  - Identifier les auteurs des échanges et en particulier le traitement des formulaires
  - Sécuriser l'accès aux bases de données
  - Sécuriser la gestion des sessions
  - Prendre en compte le recours à des serveurs partagés
  - Protéger l'accès au fichiers
  - ...

# Le Safe Mode

- Lorsque l'hébergement des pages Web est mutualisé, le module PHP (par exemple associé au serveur Apache) a les mêmes droits pour tous les utilisateurs => risques d'attaques par des sites co-hébergés.
- Contre-mesure : le mode sécurisé dans la configuration du module PHP : `php.ini` (`safe_mode`). Permet de sur ajouter des droits d'accès aux différents fichiers, interdire l'accès à des variables d'environnement sensibles (`PATH`, `LD_LIBRARY_PATH`, ...).

# Register Globals

- Cette directive permet de copier la valeur d'un paramètre de requête directement dans une variable globale de même nom. Cela évite le recours aux super globales `$_GET` ou `$_POST`
- L'emploi de cette directive est largement déconseillé. Bien que très pratique elle présente un haut risque d'interférence aisément exploitable par un attaquant
- Cette faille met en évidence la nécessité pour le programmeur de toujours explicitement initialiser toutes ses variables avant de les utiliser, même si une initialisation est supposée (trop dépendant de la correction du fichier `php.ini`)

# Vérifier et filtrer les données

---

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
- Il faut donc vérifier que les données reçues sont légitimes
- Les données doivent être contrôlées et ces contrôles doivent-être incontournables

# Vérifier et filtrer les données

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
  - Il faut donc vérifier que les données reçues sont légitimes
  - Les données doivent être contrôlées et ces contrôles doivent-être incontournables
- ➔ Mis en place d'un point de passage unique maîtrisé

# Vérifier et filtrer les données

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
- Il faut donc vérifier que les données reçues sont légitimes
- Les données doivent être contrôlées et ces contrôles doivent-être incontournables
- ➔ **Mis en place d'un point de passage unique maîtrisé**
  - un point d'entrée unique

# Vérifier et filtrer les données

- PHP n'est pas un langage fortement typé ce qui engendre des risques bien plus importants
- Il faut donc vérifier que les données reçues sont légitimes
- Les données doivent être contrôlées et ces contrôles doivent-être incontournables
- ➔ **Mis en place d'un point de passage unique maîtrisé**
  - un point d'entrée unique
  - un module unique appliqué systématiquement à tous les points d'entrée de l'application

# Vérifier et filtrer les données

- Vérification du type des données par des programmes génériques extrayant les données des variables super globales
- Initialisation des variables au démarrage du script
- Configuration soignée de la gestion des erreurs (php.ini) :
  - `error_reporting` : `E_ALL` pour détecter les variables non initialisées
  - `display_errors` : à utiliser systématiquement en phase de développement et masquer en production
  - `log_errors/error_log` : pour avoir des retours en production

# Vulnérabilité XSS

- Les “Cross-site scripting” exploitent des vulnérabilités mettant en œuvre des informations partagées par des sites différents : des sites affichant des données externes
- Si les données affichées ne sont pas correctement filtrées elle peuvent contenir des codes malicieux qui s’exécuteront sur le client
- Contre-mesures : utilisation des fonctions de filtrage (ne pas réinventer la roue) : `htmlentities`, `htmlspecialchars`,...
- objectif : filtrer les données pour empêcher l’interprétation de contenu html ou javascript.

# Construire de fausses requêtes

- Les dangers de l'utilisation de la méthode GET :
- Le requête engendré par la balise :
- ``
- ne peut être distingué du côté du serveur de la requête engendrée par l'utilisation du formulaire :
- ```
<form action="/action.php">
<select name="task">
  <option value="delete">delete</option>
  <option value="view">view</option></select>
<input type="text" name="num" />
</form>
```

# Vulnérabilités CSRF

- Consiste à forger de fausses requêtes à partir d'URL authentifiées et de pousser le client à exécuter des actions sans le savoir.
- Pour fonctionner, il faut que l'authentification ait été réalisée et que l'action du formulaire ne demande pas d'authentification ou de confirmation
- On forge alors la requête en dissimulant un formulaire ou en utilisant une balise image contrôlée par exemple par du JavaScript

# un formulaire, un traitement

- Il faut s'assurer qu'une demande de traitement est bien issue d'un formulaire géré par l'application
- Utilisation de mécanisme cryptographique pour "identifier les formulaires utilisés avec un numéro unique
  - `$token = uniqid(rand(), true);`
- et ajouter un champ invisible au formulaire :
  - `<input type="hidden" name="token" value="<?php echo $token; ?>" />`
- ne pas oublier de stocker (dans une session, dans une base de données ou dans un fichier le token)

# Injection SQL

- Lorsque l'on construit une requête SQL directement à partir des champs d'une requête POST ou GET, on fournit à un attaquant la possibilité de pratiquer de l'injection de code malicieux
- ```
<?php
$sql = "INSERT INTO users
(reg_username,reg_password,reg_email)
VALUES ( '{$_POST['reg_username']}' ,
'$reg_password' , '{$_POST['reg_email']}' )";
?>
```
- par exemple en entrant dans le champ utilisateur :
- `bad_guy' , 'mypass' , '' ) , ('good_guy`
- utilisation de filtres : `mysql_real_escape_string`, ...

# Injection SQL

- Lorsque l'on construit une requête SQL directement à partir des champs d'une requête POST ou GET, on fournit à un attaquant la possibilité de pratiquer de l'injection de code malicieux

- ```
<?php  
$sql =  
(reg_us  
VALUES  
'$reg_p  
?>
```

**Attention ne pas s'ahbriter  
derrière un php.ini "bien  
configuré"**

- par exemple en entrant dans le champ utilisateur :
- `bad_guy', 'mypass', '' ), ('good_guy`
- utilisation de filtres : `mysql_real_escape_string`, ...

# Protéger les données

- Stocker les données de façon cryptées

// Stockage du mot de passe hashcodé

```
$query = sprintf("INSERT INTO Login.users(name,pwd) VALUES  
( '%s', '%s' );", addslashes($username), md5($password));  
$result = sql_query($connection, $query);
```

// interroger le serveur pour comparer le mot de passe soumis

```
$query = sprintf("SELECT 1 FROM users WHERE name='%s' AND pwd='%s' ;",  
                addslashes($username), md5($password));  
$result = sql_query($connection, $query);
```

# Le vol de session

- `start_session` démarre une session mais c'est à votre code PHP de vérifier si la session existe déjà ou pas. Un attaquant malveillant peut fixer un identifiant de session si vous ne le vérifiez pas (`session.use_only_cookies`)
- Rajout d'information qui assurent que l'on reçoit bien des requêtes de la même session (en particulier le chemin path en environnement partagé)
- Utilisation d'identificateurs non prédictibles
- Utilisation de l'interactions utilisateur pour éveiller l'attention sur quelque chose de suspect
- Éviter les identifiants de session stockés dans les URLs (perte compatibilité). Éviter les fuites (HTTP Referer)...