# MISE EN OEUVRE D'ATTAQUES

## 1 Environnement de travail

Nous allons avoir besoin d'un outil de développement (IDE) qui va accélérer notre production de code. Il existe sur le marché de nombreux IDE, comme *Eclipse* (préférable si vous ne connaissez pas les autres), *Netbeans*, *PHPStorm* ou encore *Sublime Text*. A vous de choisir l'outil qui correspond le mieux à vos attentes.

Nous allons avoir besoin d'un serveur http. Celui-ci est disponible sur chaque machine de TP. Les fichiers qui seront placés dans votre répertoire `~/public_html/` seront accessibles par le serveur par l'URL `http:localhost//~prenom.nom/`. Attention, il faudra vérifier que les fichiers sont accessibles en lecture par le serveur. Si votre nom est Robert Duchmol et que vous avez placé le fichier `monScript.php` dans votre répertoire `~/public_html/` alors vous pourrez exécuter votre script avec l'URL suivante `http://localhost/~robert.duchmol/monScript.php`.

### Exercice 1 :

**1.1.** A l'aide de votre IDE créez un projet PHP `Todolist`.

**1.2.** Créer un fichier `testPHP.php` contenant :

```
<?php
        phpinfo();
?>
```

et tester que le script s'exécute sur le serveur http.

Il est possible d'utiliser le serveur web de base proposé par l'interpréteur php en interne. La commande suivante permet de répondre à des requêtes locales.

```
php -S localhost:8080 -t .
```

Après avoir tapé la commande précédente dans un terminal, Si vous utilisez l'URL `http://localhost:8080/testPHP.php` dans votre navigateur vous devez obtenir le même résultat qu'avec le serveur apache de votre machine.

**1.3.** Faites le test.

## 2 Architecture du projet

Dès que le nombre de classes impliquées dans un projet devient important, il est préférable d'utiliser la notion de *namespace* et d'utiliser un outil d'auto inclusion (`autoload_register()`).

Pour cela nous allons utiliser l'outil `composer` couramment utilisé dans la communauté des développeurs PHP qui va nous permettre de gérer automatiquement l'auto inclusion et de plus va nous permettre d'inclure facilement dans notre projet des librairies développés par des tiers.

La commande `composer` utilise un fichier de configuration : `composer.json`, en utilisant la commande

```
composer init
```

vous allez créer de manière interactive le fichier de base.

```
This command will guide you through creating your composer.json config.

Package name (<vendor>/<name>) [hemery/td_tp3]:
Description []: Premiere application web
Author [Hemery Fred <fred.hemery@univ-artois.fr>, n to skip]:
Minimum Stability []:
Package Type (e.g. library, project, metapackage, composer-plugin) []: project
License []:

Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]? no
Would you like to define your dev dependencies (require-dev) interactively [yes]?
    ➥no

{
    "name": "hemery/td_tp3",
    "description": "Premiere application web",
    "type": "project",
    "authors": [
        {
            "name": "Hemery Fred",
            "email": "fred.hemery@univ-artois.fr"
        }
    ],
    "require": {}
}

Do you confirm generation [yes]?
```

crée le fichier `composer.json` dans le répertoire courant.

## Intégration de votre code dans le projet

La commande `composer` permet d'intégrer rapidement des librairies tiers dans votre projet, mais il permet aussi d'intégrer votre code. Il faut pour cela utiliser la directive `autoload`

```
  "autoload": {
    "psr-4": {"IutLens\\" : "app/"}
  }
```

`psr-4` spécifie que dans votre projet vous allez respecter la norme `psr-4` (`http://www.php-fig.org/psr/psr-4/`). `IutLens` correspond à la notion de *vendor* dans la norme `psr-4`, c'est le nom du paquetage racine de votre projet. Toutes les autres classes doivent se trouver dans un paquetage sous le paquetage `IutLens`. `app` est le chemin relatif qui indique où se trouve votre code.

Si j'ai une classe `ClasseA` définie dans le paquetage `IutLens`, elle doit se situer dans le répertoire `app`.

Si j'ai une classe `ClasseB` définie dans le paquetage `IutLens\Model`, elle doit se situer dans le répertoire `app/Model`.

Dans la suite, vous choisirez un nom de *vendor* (`SuperProjet`, `YaPasMieux`, ...) qui sera votre marque de fabrique. De cette façon votre code pourra éventuellement être déplacé pour être utilisé dans un autre projet sans provoquer de conflits de noms.

**Exercice 2 :** Modification du fichier `composer.jcon`.

**2.1.** Ajouter la directive `autoload` pour que votre code soit pris en compte dans votre projet.

**2.2.** Créer une classe Hello dans le répertoire src avec le contenu suivant:

```
 <?php
namespace IutLens; // adapter en fonction de votre nom de vendor
class Hello {
    public static function  Bonjour() {
        return "<h3>Bonjour Monde !</h3>";
    }
}
```

**2.3.** A la racine de votre projet créer un fichier `index.php` avec le contenu suivant :

```
 <?php>
 require "vendor/autoload.php";

use \IutLens\Hello;

echo Hello::bonjour();
```

**2.4.** Pour que cela fonctionne, on doit initialiser la fonction de chargement automatique. Exécuter dans un terminal, à la racine de votre projet, la commande

```
 composer install
```

le répertoire `vendor` a été créé. Il contient le fichier `autoload.php` qui gère pour vous la recherche des différentes classes qui seront utilisées dans votre projet.

**2.5.** Tester que votre projet fonctionne.

## Intégration d'une librairie tiers dans le projet

En partant du principe qu'il ne faut pas réinventer la roue . . . , il est important de pouvoir intégrer facilement des librairies tiers dans votre projet. La commande `composer` permet de définir des dépendances pour votre projet. Dès que vous avez ajouté une dépendance à votre projet, la commande `composer` va s'assurer que celle-ci est satisfaite et sinon va tenter de rechercher cette dépendance sur le web. La commande `composer` va stocker pour vous la librairie au bon endroit (dans le répertoire `vendor`) et inclure la librairie dans la liste des répertoires de recherche des classes utilisées.

Pour illustrer notre propos nous allons mettre en place un mécanisme de gestion de trace d'exécution (*Log*). La librairie que nous avons choisi est `monolog/monolog`(`https://github.com/Seldaek/monolog`) très utilisée dans les frameworks comme Symfony2, Laravel, . . .

Cette librairie a de nombreuses possibilités à découvrir par vous même, dans la suite une configuration donne quelques possibilités d'utilisation.

Pour ajouter une dépendance au projet avec `composer` on utilise la commande

```
composer require monolog/monolog
```

qui provoque la récupération de la librairie sur le web et modifie le fichier `composer.json`.

**Exercice 3 :** Configuration du mécanisme de trace.

**3.1.** Nous allons créer une classe `Config` pour initialiser le mécanisme de trace. Créer le fichier `Config➥.php` dans le répertoire `src` avec le contenu suivant:

Listing 1: src/Config.php

```php
<?php
namespace IutLens;

use Monolog\Logger;
use Monolog\Handler\FirePHPHandler;
use Monolog\Formatter\LineFormatter;
use Monolog\Handler\StreamHandler;
use Monolog\Processor\IntrospectionProcessor;


class Config {
    public static $logger;

    /**
     * Config constructor.
     */
    public function __construct() {
        date_default_timezone_set('Europe/Paris');
        Define("DIR", "/");

        $dateFormat = "[D j/m/y H:i:s]";
        $output = "%datetime% %channel%.%level_name%: %message% %context% %extra
            ➥%\n";
        $formatter = new LineFormatter($output, $dateFormat);

        // Create the logger
        self::$logger = new Logger('trace');
        // Now add some handlers
        $stream =new StreamHandler(__DIR__.'/../'.'/td_tp3.log', Logger::DEBUG);
        $stream->setFormatter($formatter);
        self::$logger->pushHandler($stream);
        self::$logger->pushHandler(new FirePHPHandler());
        self::$logger->pushProcessor(new IntrospectionProcessor());

        //configuration pour la base de données
        define('DB_TYPE', 'mysql');
        define('DB_HOST', 'localhost');
        define('DB_NAME', 'hemery');
        define('DB_USER', 'hemery');
        define('DB_PASS', 'tolkien');

        define ('ASSETS', DIR.'public/');
    }

    public static function getLogger() {
        return self::$logger;
    }
}
```

**3.2.** Modifier le fichier `index.php`

Listing 2: index.php

```php
<?php
require "vendor/autoload.php";
use IutLens\Config;
```

```php
use IutLens\Hello;

new Config();
echo Hello::bonjour();
```

**3.3.** Modifier la classe `Hello` pour ajouter une message de trace.

Listing 3: src/Hello.php

```php
<?php
namespace IutLens; // adapter en fonction de votre nom de vendor
class Hello {
    public static function Bonjour() {
        Config::getLogger()->addInfo('Voici un exemple de message stocké dans le
            ➥ fichier log');
        $i = 18;
        Config::getLogger()->addInfo('Pourquoi ?', array('i' => $i));

        return "<h3>Bonjour Monde !</h3>";
    }
}
```

**3.4.** Vérifier que le message est stocké dans le fichier de trace.

## **Exercice 4 :** Utilisation de langage *Markdown*

Nous allons utiliser un outil de formatage de texte (`Markdown`) qui est largement utilisé par les développeurs web pour communiquer dans les sites *wiki*. Les fichiers au format `Markdown` (fichiers avec une extension `md`) seront placés dans le répertoire `public/md`.

**4.1.** Récupérer la librairie qui convien à l'aide des commandes suivantes :

```
composer require erusev/parsedown
composer require erusev/parsedown-extra
```

**4.2.** Insérer la constante `ASSETS` dans le fichier `Config.php` (déjà fait).

**4.3.** Créer la classe Welcome dans le répertoire `src` avec le contenu suivant :

Listing 4: src/Welcome.php

```php
<?php
namespace IutLens;


class Welcome {
    public static function welcome($file) {
        $file = file_get_contents(__DIR__.'/..'.ASSETS.'md/'.$file.'.md');
        $parsedown = new \ParsedownExtra();
        $html = $parsedown->text($file);
        if ($file) {
            return $html;
        } else {
            return __DIR__."<br>";
        }
    }
}
```

**4.4.** Créer le fichier `welcome.php` à la racine avec le contenu suivant :

Listing 5: welcome.php

```php
<?php
require "vendor/autoload.php";
new \IutLens\Config();
echo \IutLens\Welcome::welcome('exemple');

?>
```

**4.5.** Tester la page `welcome.php` dans votre navigateur.

# 3 Vérification des entrées

## Vérifier la liste des variables en entrée

Le pirate peut modifier la liste des variables d'entrée dans un formulaire. Pour sécuriser cette éventualité, il suffit d'ajouter dans le script qui traite le formulaire une liste exhaustive des champs qui seront traités.

Listing 6: ex3.php

```php
<?php
$expected = array( 'nom', 'prenom', 'profession' );
foreach( $expected AS $key ) {
    if ( !empty( $_POST[ $key ] ) ) {
        ${$key} = $_POST[ $key ];
    }
    else {
        ${$key} = NULL;
    }
}
?>
```

## Type, Longueur et Format des variables en entrée

Listing 7: ex4.php

```php
<?php
// Chaîne de caractères
if ( isset( $value ) && $value !== NULL ) {
// $value est une (vide éventuellement) string en PHP
}

// Entier
// 1
$age = $_POST['age'];
if ( !is_int( $age ) ) exit ( "$age n'est pas une valeur valide pour l'age!" );

// 2
if ( gettype( $age ) != 'integer' ) {
exit ( "$age n'est pas une valeur valide pour l'age!" );
}

// 3
$age = intval( $_POST['age'] );
```

```php
$age = ( int ) $_POST['age'];
if ( !settype ( $age, 'integer' ) ) {
    exit ( "$age n'est pas une valeur valide pour l'age!" );
}

// is_numeric($age) accepte zéro et les réels
is_numeric(3.14); // renvoie True

//La longueur
if ( strlen( $age ) > 3 ) exit ( "$age n'est pas une valeur valide pour l'age!" )
    ➥;
```

Un script qui vérifie champs récupéré d'un formulaire

Listing 8: ex5.php

```php
<?php

// set up array of expected values and types
$expected = array( 'carModel'=>'string', 'year'=>'int',
'imageLocation'=>'filename' );

// check each input value for type and length
foreach ( $expected AS $key=>$type ) {
    if ( empty( $_GET[ $key ] ) ) {
        ${$key} = NULL;
        continue;
    }
    switch ( $type ) {
        case 'string' :
            if ( is_string( $_GET[ $key ] ) && strlen( $_GET[ $key ] ) < 256 ) {
                ${$key} = $_GET[ $key ];
            }
            break;
        case 'int' :
            if ( is_int( $_GET[ $key ] ) ) {
                ${$key} = $_GET[ $key ];
            }
            break;
        case 'filename' :
// limit filenames to 64 characters
            if ( is_string( $_GET[ $key ] ) && strlen( $_GET[ $key ] ) < 64 ) {
// escape any non-ASCII
                ${$key} = str_replace( '%', '_', rawurlencode( $_GET[ $key ] ) );
// disallow double dots
                if ( strpos( ${$key}, '..' ) === TRUE ) {
                    ${$key} = NULL;
                }
            }
            break;
    }
    if ( !isset( ${$key} ) ) {
        ${$key} = NULL;
    }
}
```

### Le traitement des méta-caractères

Pour les rendre inopérant :

- Placer un symbole '\' devant le méta-caractère
- Placer le méta-caractère entre quote (')
- Remplacer le méta-caractère par son code `%nn` à l'aide de `urlencode()`

### Exercice 5 :

### Questions

**5.1.** Ecrire un formulaire qui permet de saisir `nom`, `prenom`, `profession`, `age`, `email` et `fichier`

**5.2.** Ecrire un script qui affiche uniquement les valeurs des variables `nom`, `prenom` et `profession` d'un formulaire à l'exclusion de toutes autres qui pourraient apparaitre dans un formulaire.

**5.3.** Compléter le script pour qu'il protège le programme contre les méta-caractères.

## 4 Injection SQL

Soit la table SQL

```
CREATE TABLE IF NOT EXISTS `comptes` (
   `id` int(11) NOT NULL AUTO_INCREMENT COMMENT 'identifiant',
   `nom` varchar(30) NOT NULL COMMENT 'nom d''utilisateur',
   `motdepasse` varchar(41) NOT NULL,
   `typecarte` varchar(30) NOT NULL COMMENT 'type de carte',
   `numerocarte` varchar(30) NOT NULL COMMENT 'numéro de carte',
   PRIMARY KEY (`id`), UNIQUE KEY `nom` (`nom`)
) ENGINE=InnoDB DEFAULT CHARSET=UTF8 AUTO_INCREMENT=1 ;
```

La requête SQL qui permet de renvoyer le numéro de carte si le nom et le mot de passe sont corrects est la suivante :

```
SELECT `numerocarte` FROM `comptes`
  WHERE `nom` = 'duchmol' AND `motdepasse` = PASSWORD( 'secret' );
```

Pour tester l'injection nous allons créer une table dans une base de données et le but sera d'obtenir plus d'informations.

### Exercice 6 :

### Questions

**6.1.** Imaginez les données que peut saisir l'utilisateur pour afficher l'ensemble des informations stockées dans la table `comptes`.

**6.2.** Réécrire la requête pour ne plus avoir ce type de problème.

**6.3.** Réaliser le test de l'attaque.

# 5 Cross-Site Scripting (XSS)

Nous allons mettre en place une attaque XSS. Pour cela nous allons utiliser deux tables :

```
CREATE TABLE `SW_messages` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `msg` text,
  `valide` char(1) DEFAULT 'T',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
CREATE TABLE `SW_des_cookies` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cookie` text,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8
```

Nous allons utiliser ces tables pour mettre en oeuvre une attaque XSS. Un écran de saisie va permettre de créer un message, lorsque l'on affichera le message on pourra vérifier qu'il est facile de faire une attaque. Même chose si l'on stocke les messages dans la base de données.

### Exercice 7 :

### Questions

**7.1.** Ecrire la classe AccesBD qui crée la table `SW_messages` et `SW_cookies`.

**7.2.** Ecrire le code php pour la saisie d'un message

**7.3.** Imaginer une attaque

**7.4.** Modifier le code php pour interdire cette attaque

# 6 Vol de session

Nous allons mettre en oeuvre une attaque CSRF (Cross-Site Request Forgery) et tenter de trouver une parade.

### Exercice 8 :

### Questions

**8.1.** Ecrire une page `ex8.php` qui permet de créer une session (stockage dans la session d'un nom d'utilisateur).

**8.2.** Ecrire une page `ex8-protege.php` qui permet d'exécuter une action uniquement si le nom d'utilisateur stocké dans la session est 'Bob'

**8.3.** Ecrire une page `ex8-malveillant.php` qui fait exécuter à Bob la page `ex8-protege.php` sans qu'il le sache

**8.4.** Modifier le code php de la page `ex8-protege.php` pour interdire cette attaque.