Activité 5 : Attaques de type XXE (XML External Entities)

DATE : 1 mars 2025

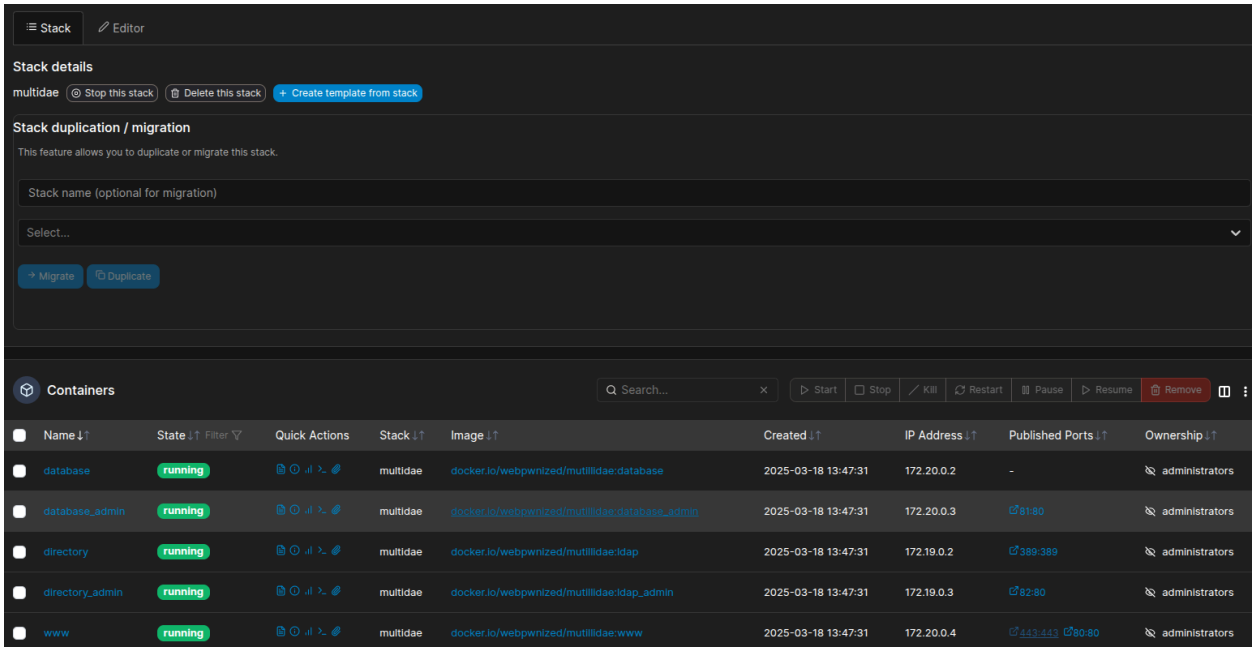
Mise en place de l'attaque XXE.

Question 1 :

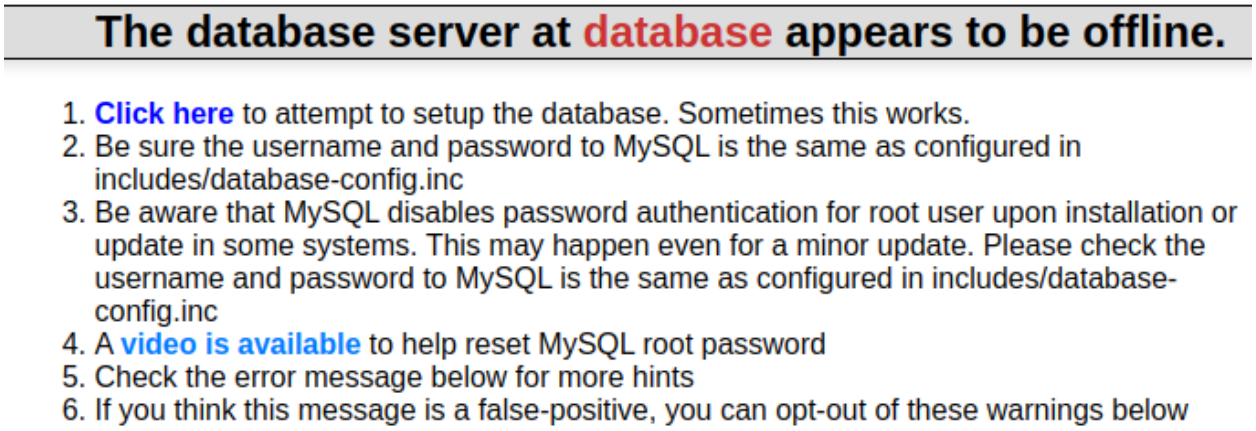
Commencer par préparer votre environnement de travail en démarrant le serveur Mutillidae ainsi que la machine cliente.

Vérifier que vos deux machines communiquent à l'aide de la commande ping. Puis, positionner le niveau de sécurité de Mutillidae à 0.

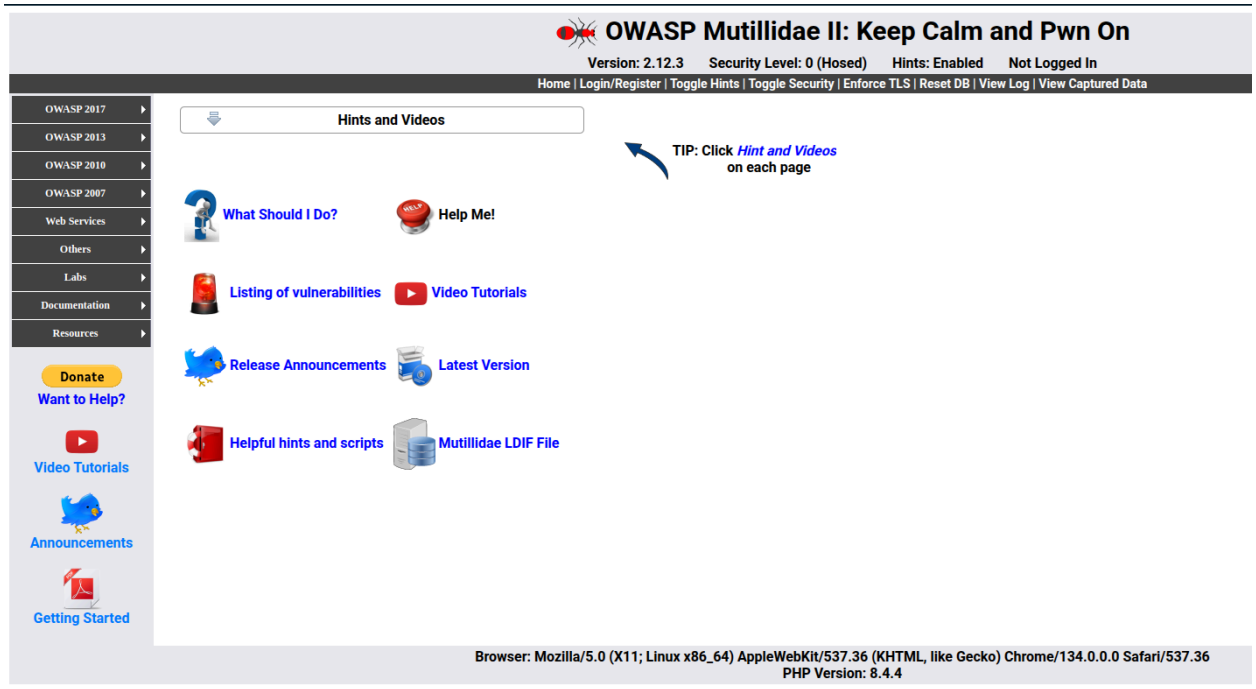
Pour accéder au site web de Mutillidae nous devons démarrer le stack :



Ceci fait, nous devons accéder à L'URL du site web donc (OWASP Mutillidae II) :



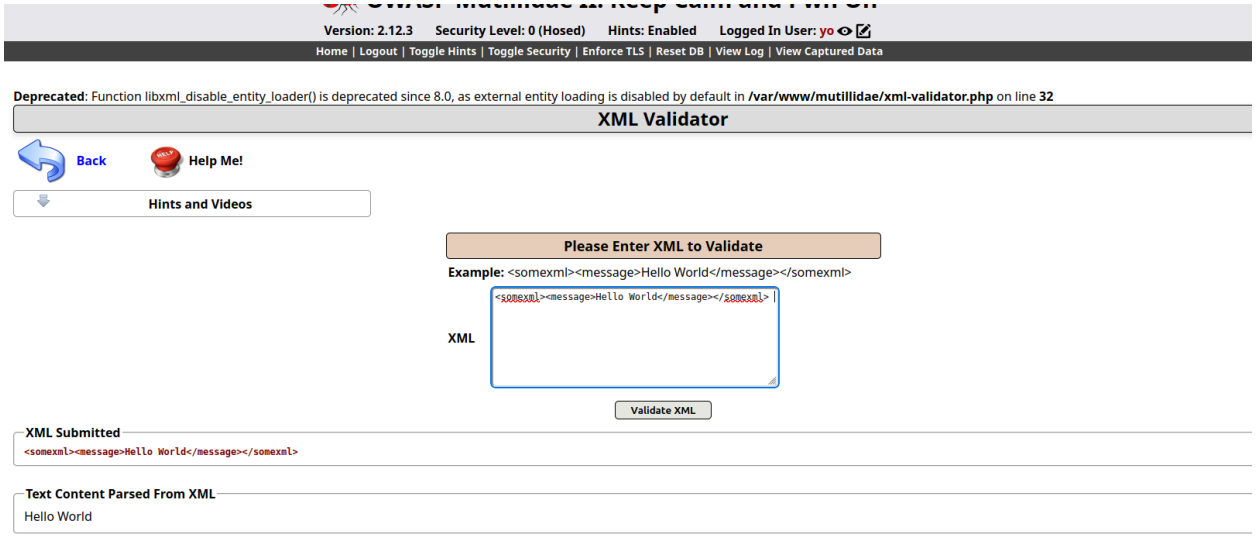
Une fois arrivé sur cette page vous devez cliquer sur CLICK HERE ceci vous renvoie sur le site web OWASP Mutillidae II :



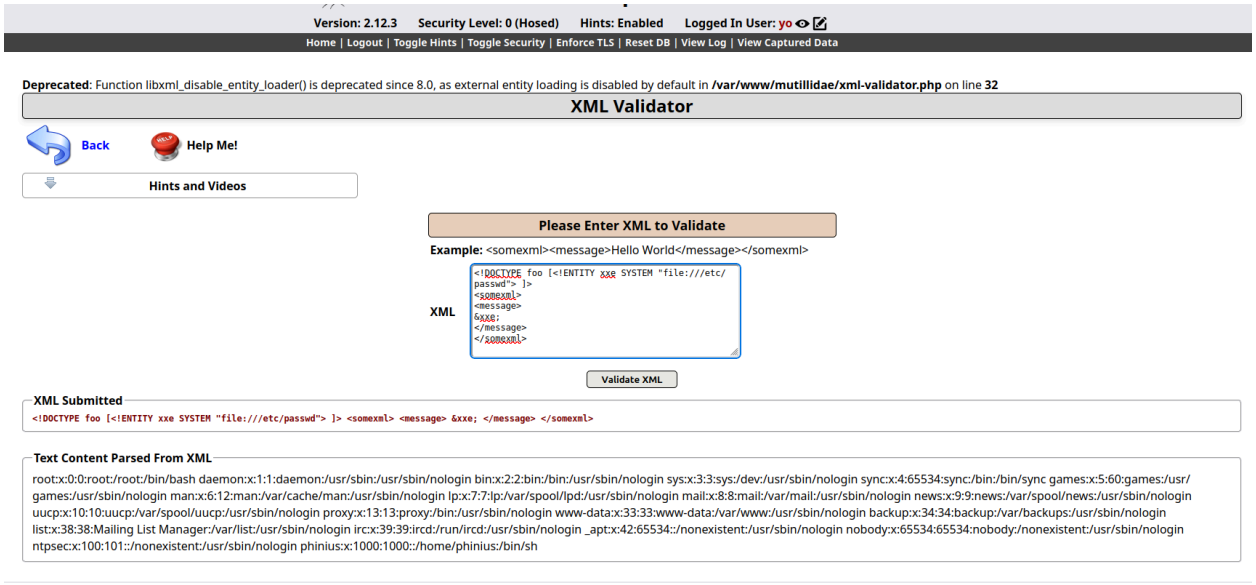
Nouvelle tentative en mode sécurisé et analyse du code source.

Se rendre sur la page de test de l'attaque XXE via le lien suivant : OWASP 2017=> XML External Entities => XML External Entity Injection => XML Validator.

1 . Utilisation non malveillante du XML :



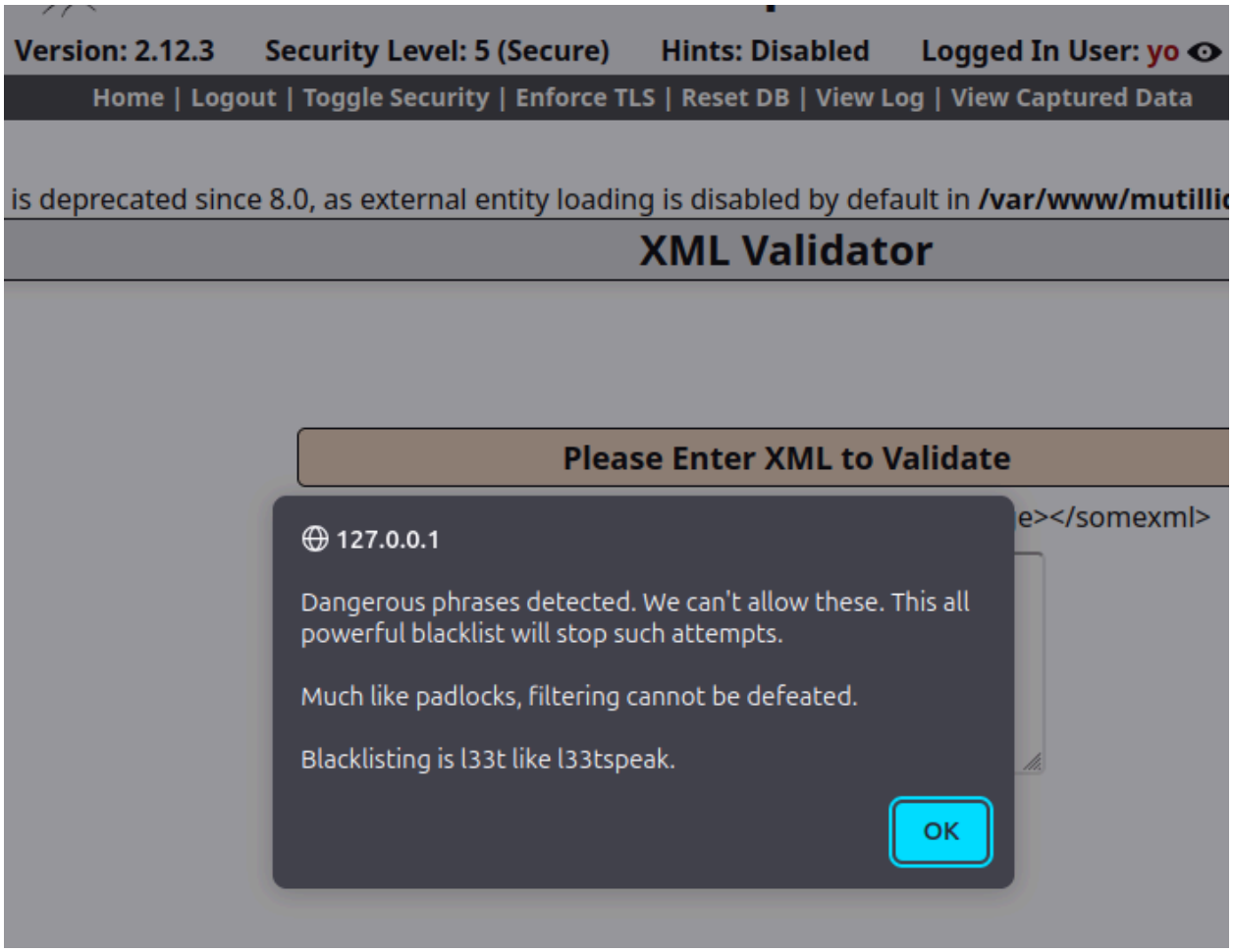
2 . Injection de code malveillant :



3 . Codage sécurisé

Activation du niveau 5 .

L’attaque a échoué :



Avant la modification :

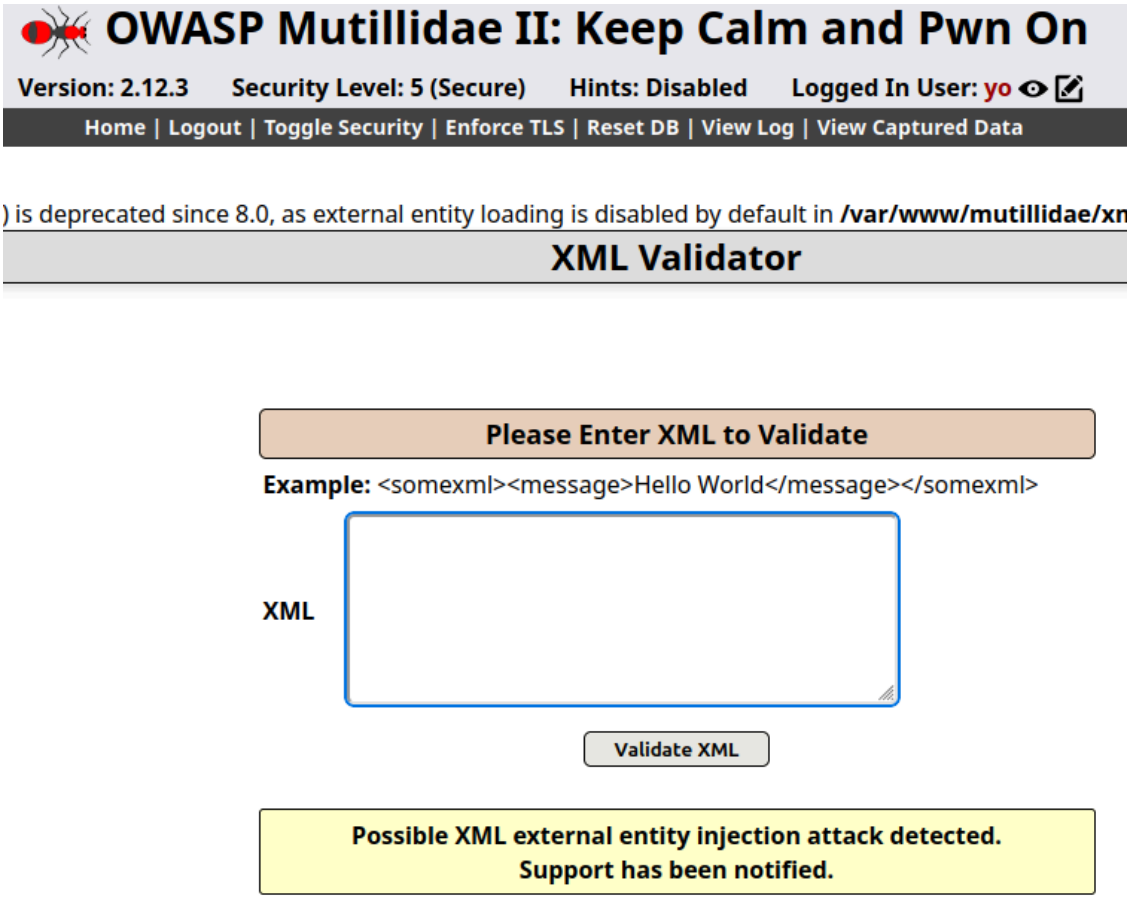
```
                case "2":
                case "3":
                case "4":
        case "5": // This code is fairly secure
                        $lEnableHTMLControls = true;
                //$lFormMethod = "POST";
                        $lEnableJavaScriptValidation = true;
                        $lEnableXMLValidation = true;
                        $lEnableXMLEncoding = true;
                        $lProtectAgainstMethodTampering = true;
                        libxml_disable_entity_loader(true);
                break;
 }//end switch
```

Apres modification :

```
                break;

                case "2":
                case "3":
                case "4":
        case "5": // This code is fairly secure
                        $lEnableHTMLControls = true;
                //$lFormMethod = "POST";
                        $lEnableJavaScriptValidation = false;
                        $lEnableXMLValidation = true;
                        $lEnableXMLEncoding = true;
                        $lProtectAgainstMethodTampering = true;
                        libxml_disable_entity_loader(true);
                break;
```

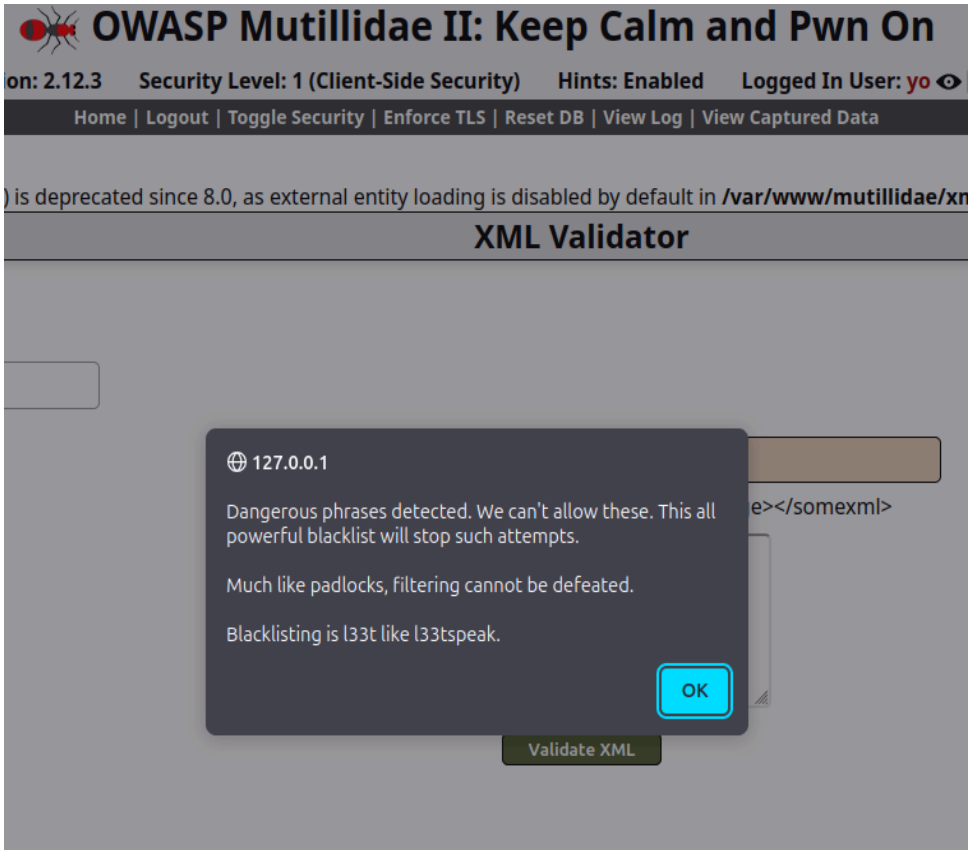
Le message qui s'affiche après avoir changer true par false :



Travail à faire 3

Nouvelle tentative en mode sécurisé et analyse du code source :

Test du niveau de sécurité 1 (Client-Side Security) :

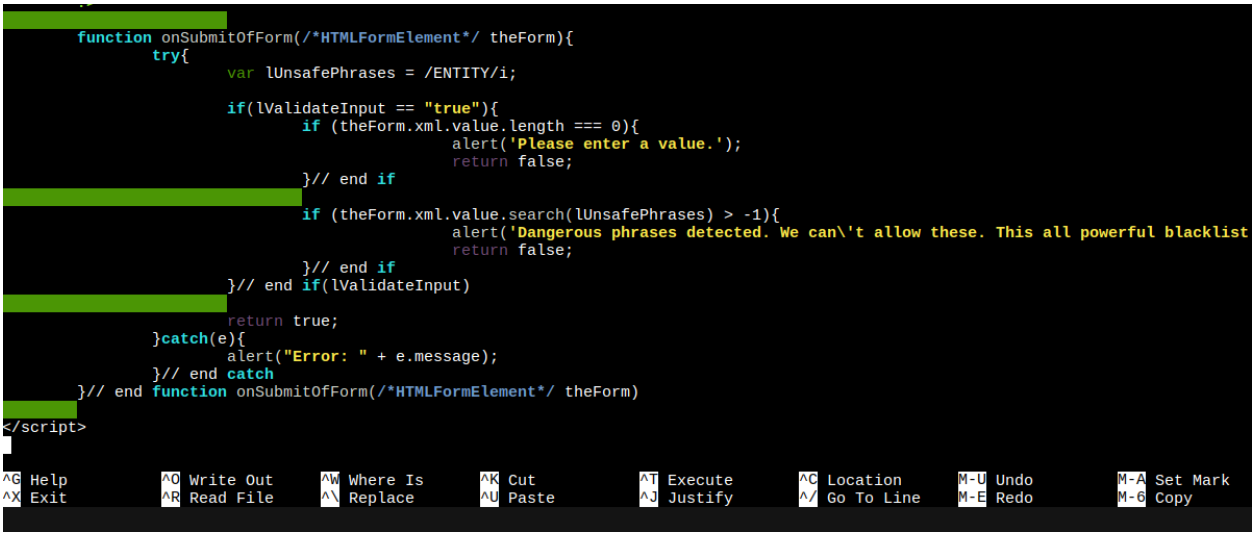


Q1. Est-ce que le niveau de sécurité 1 permet d'éviter l'attaque XXE ?

Si oui, est-ce dû à une Protection spécifique contre le XXE ?

Non, car le niveau 1 ne permet pas nécessairement d'éviter une attaque XXE (XML External Entity). Cependant, d'après le message d'erreur affiché sur l'écran, il semble que l'application utilise une approche de **blacklist** pour filtrer certains contenus jugés dangereux.

Q2. Dans la page xml-validator.php, expliquer le rôle des blocs de code suivants :



Le code suivant dans la page **xml-validator.php** a un rôle de filtrage côté client pour empêcher l'injection de certaines chaînes de caractères potentiellement dangereuses :

```
var lUnsafePhrases = /ENTITY/i;

if(lValidateInput == "true"){
        if (theForm.xml.value.length === 0){
                alert('Please enter a value.');
                return false;
```

Cette ligne déclare une variable `lUnsafePhrases` contenant une **expression régulière** (`/ENTITY/i`).

- `theForm.xml.value` récupère la valeur entrée dans le champ du formulaire.

- `search(lUnsafePhrases)` recherche la présence de **"ENTITY"** dans cette valeur.

- Si la chaîne est trouvée (`search()` renvoie un index ≥ 0), la condition est vraie.

Un **message d'alerte** est affiché pour informer l'utilisateur que l'entrée est interdite.

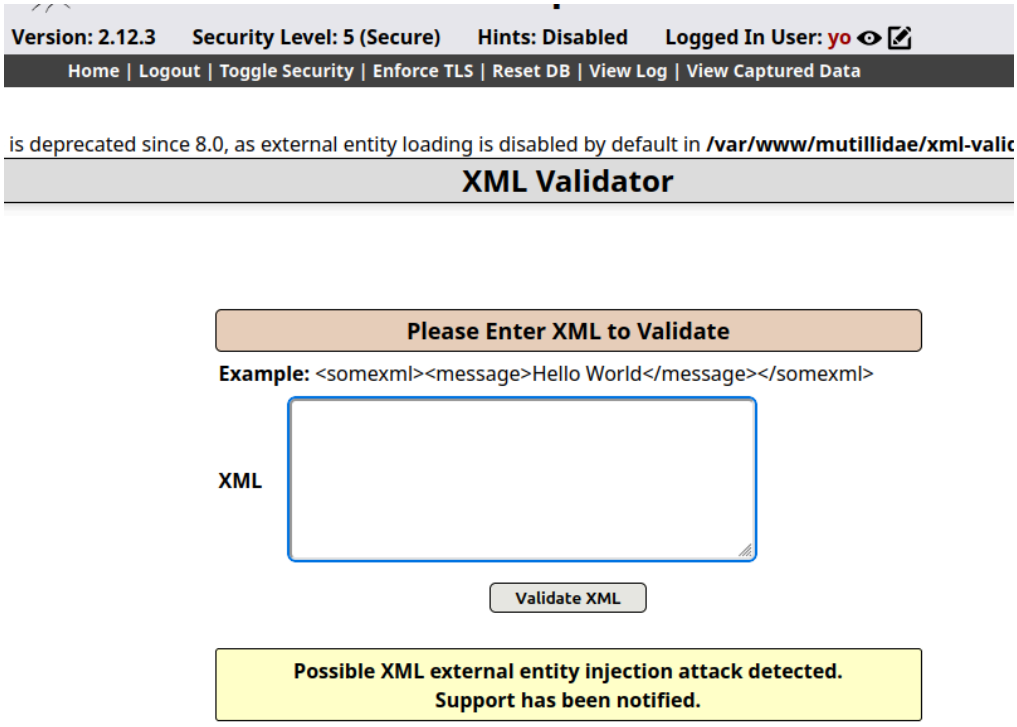
Donc le bloc de code a pour but de **bloquer des attaques XXE** en empêchant l'insertion du mot `ENTITY` dans les entrées utilisateur.

Cela ne suffit pas, car cette protection **est insuffisante** :

- Elle repose sur **une blacklist contournable**.
- Elle s'exécute **uniquement côté client**, donc facilement désactivable.

Test du niveau de sécurité 5 (Secure) :

Q3. Est-ce que le niveau 5 permet d'éviter l'attaque XXE ?



Le niveau 5 dans Mutillidae détecte et bloque les tentatives d'attaque **XXE (XML External Entity Injection)**. Donc Oui, le niveau de sécurité 5 permet d'éviter l'attaque XXE (XML External Entity).

Q4. Afficher le code source de la page xml-validator.php sur le serveur à l'aide de la commande :

La commande suivante permet d'afficher le contenu du fichier `xml-validator.php` sur un serveur Linux :

more /var/www/html/mutillidae/xml-validator.php

Q5. Expliquer le rôle d'une expression régulière (motif).

Une **expression régulière** (ou **regex**) est une suite de caractères permettant de définir un modèle (motif) utilisé pour rechercher, valider, extraire ou remplacer des chaînes de caractères.

Elle est particulièrement utile pour :

- Vérifier si une entrée utilisateur suit un format précis (ex. adresse e-mail, numéro de téléphone).
- Filtrer et nettoyer des données en supprimant les caractères non désirés.
- Rechercher et extraire des informations spécifiques dans un texte.

Par exemple :

$emailPattern="/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/";

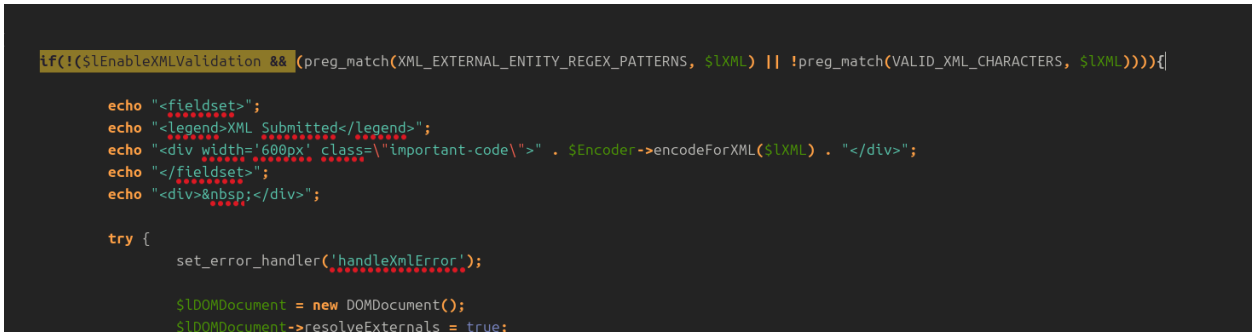
Q6. Toujours dans la même page, expliquer le rôle de la fonction `preg_match`.

La fonction `preg_match` est une fonction PHP utilisée pour effectuer une correspondance entre une chaîne de caractères et un motif d'expression régulière.

Syntaxe : preg_match(pattern, string, matches).

pattern	l'expression régulièrement utilisée pour rechercher une correspondance.
string	la chaîne de caractères à analyser.
matches	(optionnel) : un tableau qui stocke les résultats correspondants.

Dans le code de `xml-validator.php`, `preg_match` est utilisé pour détecter des attaques XXE potentielles :



Cette utilisation de `preg_match` contribue à sécuriser l'application contre certaines attaques basées sur du XML malveillant.

Q7. Expliquer le rôle de la fonction `libxml_disable_entity_loader`. Qu'en est-il de cette fonction depuis PHP 8.0 ?

Rôle de `libxml_disable_entity_loader(true)`

La fonction `libxml_disable_entity_loader(true)` désactive le chargement des **entités externes** dans les fichiers XML. Cela permet d'empêcher les attaques XXE (**XML External Entity**) qui exploitent ces entités pour accéder à des fichiers sensibles du système ou exécuter des requêtes malveillantes.

```
libxml_disable_entity_loader(true);  // Désactive le chargement des entités externes
```

À partir de PHP 8.0, `libxml_disable_entity_loader` est devenue obsolète et n'a plus d'effet. En effet, la protection contre les entités XML externes est maintenant activée par défaut dans la bibliothèque libxml utilisée par PHP.

Cela signifie que même sans appeler `libxml_disable_entity_loader(true)`, les entités externes ne seront plus chargées par défaut.

Dans les versions **PHP 8.0 et supérieures**, il n'est plus nécessaire d'utiliser cette fonction pour protéger contre XXE. Cependant, dans les versions antérieures, elle reste une **bonne pratique** pour sécuriser l'analyse de XML.

Q8. Conclusion sur la méthode de codage sécurisée utilisée pour empêcher l'attaque XXE:

Dans le fichier `xml-validator.php`, plusieurs **mesures de sécurité** ont été mises en place pour prévenir une attaque **XXE** :

Désactivation du chargement des entités externes :

```
        $lEnableXMLEncoding = true;
        $lProtectAgainstMethodTampering = true;
        libxml_disable_entity_loader(true);
break;
```

Cela empêche l'accès aux fichiers système via des entités XML (utile avant PHP 8.0).

Utilisation de `DOMDocument` avec des paramètres sécurisés :

```
$lDOMDocument = new DOMDocument();
$lDOMDocument->resolveExternals = true;
$lDOMDocument->substituteEntities = true;
$lDOMDocument->preserveWhiteSpace=true;
$lDOMDocument->loadXML($lXML);
```

Ces options peuvent être dangereuses si `libxml_disable_entity_loader(true);` n'est pas activé, mais elles sont gérées ici en fonction du niveau de sécurité.

Vérification et filtrage de l'entrée utilisateur avec `preg_match` :

```
if(!($lEnableXMLValidation && (preg_match(XML_EXTERNAL_ENTITY_REGEX_PATTERNS, $lXML) || !preg_match(VALID_XML_CHARACTERS, $lXML)))){
```

- Cette étape détecte et bloque toute tentative d'injection XXE.

Encodage des sorties XML (`encodeForXML`) :

```
if ($lEnableXMLEncoding){
        $lXMLToLog = $Encoder->encodeForXML($lXML);
}else{
        $lXMLToLog = $lXML;
};
        $LogHandler->writeToLog("Recieved request to validate XML for: " . $lXMLToLog);
} catch (Exception $e) {
```

Empêche les injections XML et assure une gestion sécurisée des données.

La méthode utilisée ici est **plutôt sécurisée** lorsqu'elle est bien configurée. Cependant, avec **PHP 8.0 et versions ultérieures**, il faut s'assurer que `libxml_disable_entity_loader(true);` ne soit plus utilisé et que les entités externes restent **désactivées par défaut**.

FIN