



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 :

The screenshot shows the Docker Desktop interface. At the top, there's a 'Stack details' section for a stack named 'multidae'. Below this, there's a 'Stack duplication / migration' section. The main part of the interface is the 'Containers' section, which displays a table of running containers.

Name	State	Quick Actions	Stack	Image	Created	IP Address	Published Ports	Ownership
database	running	[Icons]	multidae	docker.io/webpwnized/mutillidae:database	2025-03-18 13:47:31	172.20.0.2	-	administrators
database_admin	running	[Icons]	multidae	docker.io/webpwnized/mutillidae:database_admin	2025-03-18 13:47:31	172.20.0.3	81:80	administrators
directory	running	[Icons]	multidae	docker.io/webpwnized/mutillidae:ldap	2025-03-18 13:47:31	172.19.0.2	389:389	administrators
directory_admin	running	[Icons]	multidae	docker.io/webpwnized/mutillidae:ldap_admin	2025-03-18 13:47:31	172.19.0.3	82:80	administrators
www	running	[Icons]	multidae	docker.io/webpwnized/mutillidae:www	2025-03-18 13:47:31	172.20.0.4	443:443 80:80	administrators

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

The database server at **database** appears to be offline.

1. [Click here](#) to attempt to setup the database. Sometimes this works.
2. Be sure the username and password to MySQL is the same as configured in `includes/database-config.inc`
3. Be aware that MySQL disables password authentication for root user upon installation or update in some systems. This may happen even for a minor update. Please check the username and password to MySQL is the same as configured in `includes/database-config.inc`
4. A [video is available](#) to help reset MySQL root password
5. Check the error message below for more hints
6. If you think this message is a false-positive, you can opt-out of these warnings below

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

The screenshot shows the OWASP Mutillidae II web application. At the top, there's a header with the title "OWASP Mutillidae II: Keep Calm and Pwn On" and a version number "Version: 2.12.3". Below the header, there's a navigation bar with links: Home, Login/Register, Toggle Hints, Toggle Security, Enforce TLS, Reset DB, View Log, and View Captured Data. The main content area is divided into a left sidebar and a main panel. The sidebar contains a list of links: OWASP 2017, OWASP 2013, OWASP 2010, OWASP 2007, Web Services, Others, Labs, Documentation, Resources, Donate, Want to Help?, Video Tutorials, Announcements, and Getting Started. The main panel has a search bar labeled "Hints and Videos" and a tip: "TIP: Click Hint and Videos on each page". Below the search bar, there are several links with icons: What Should I Do?, Help Me!, Listing of vulnerabilities, Video Tutorials, Release Announcements, Latest Version, Helpful hints and scripts, and Mutillidae LDIF File. At the bottom, there's a footer with browser and PHP version information.

OWASP Mutillidae II: Keep Calm and Pwn On

Version: 2.12.3 Security Level: 0 (Hosed) Hints: Enabled Not Logged In

Home | Login/Register | Toggle Hints | Toggle Security | Enforce TLS | Reset DB | View Log | View Captured Data

OWASP 2017
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
Others
Labs
Documentation
Resources

Donate
Want to Help?
Video Tutorials
Announcements
Getting Started

Hints and Videos

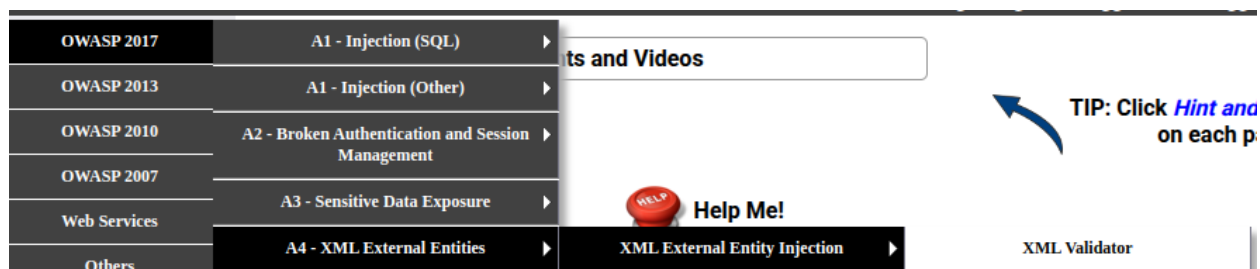
TIP: Click *Hint and Videos* on each page

What Should I Do? Help Me!
Listing of vulnerabilities Video Tutorials
Release Announcements Latest Version
Helpful hints and scripts Mutillidae LDIF File

Browser: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/134.0.0.0 Safari/537.36
PHP Version: 8.4.4

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 :

Version: 2.12.3 Security Level: 0 (Hosed) Hints: Enabled Logged In User: yo

Home | Logout | Toggle Hints | Toggle Security | Enforce TLS | Reset DB | View Log | View Captured Data

Deprecated: Function libxml_disable_entity_loader() is deprecated since 8.0, as external entity loading is disabled by default in /var/www/mutillidae/xml-validator.php on line 32

XML Validator

[Back](#) [Help Me!](#)

[Hints and Videos](#)

Please Enter XML to Validate

Example: <somexml><message>Hello World</message></somexml>

XML

Validate XML

XML Submitted

```
<somexml><message>Hello World</message></somexml>
```

Text Content Parsed From XML

Hello World

2 . Injection de code malveillant :

Version: 2.12.3 Security Level: 0 (Hosed) Hints: Enabled Logged In User: yo  
[Home](#) | [Logout](#) | [Toggle Hints](#) | [Toggle Security](#) | [Enforce TLS](#) | [Reset DB](#) | [View Log](#) | [View Captured Data](#)

Deprecated: Function libxml_disable_entity_loader() is deprecated since 8.0, as external entity loading is disabled by default in [/var/www/mutillidae/xml-validator.php](#) on line 32

XML Validator

 [Back](#)  [Help Me!](#)

 [Hints and Videos](#)

Please Enter XML to Validate

Example: <somexml><message>Hello World</message></somexml>

XML

```
<!DOCTYPE foo [
<?ENTITY xx SYSTEM "file:///etc/passwd" ?> ]>
<somexml>
<message>
&xx;
</message>
</somexml>
```

[Validate XML](#)

XML Submitted

```
<!DOCTYPE foo [
<?ENTITY xx SYSTEM "file:///etc/passwd" ?> ]>
<somexml>
<message>
&xx;
</message>
</somexml>
```

Text Content Parsed From XML

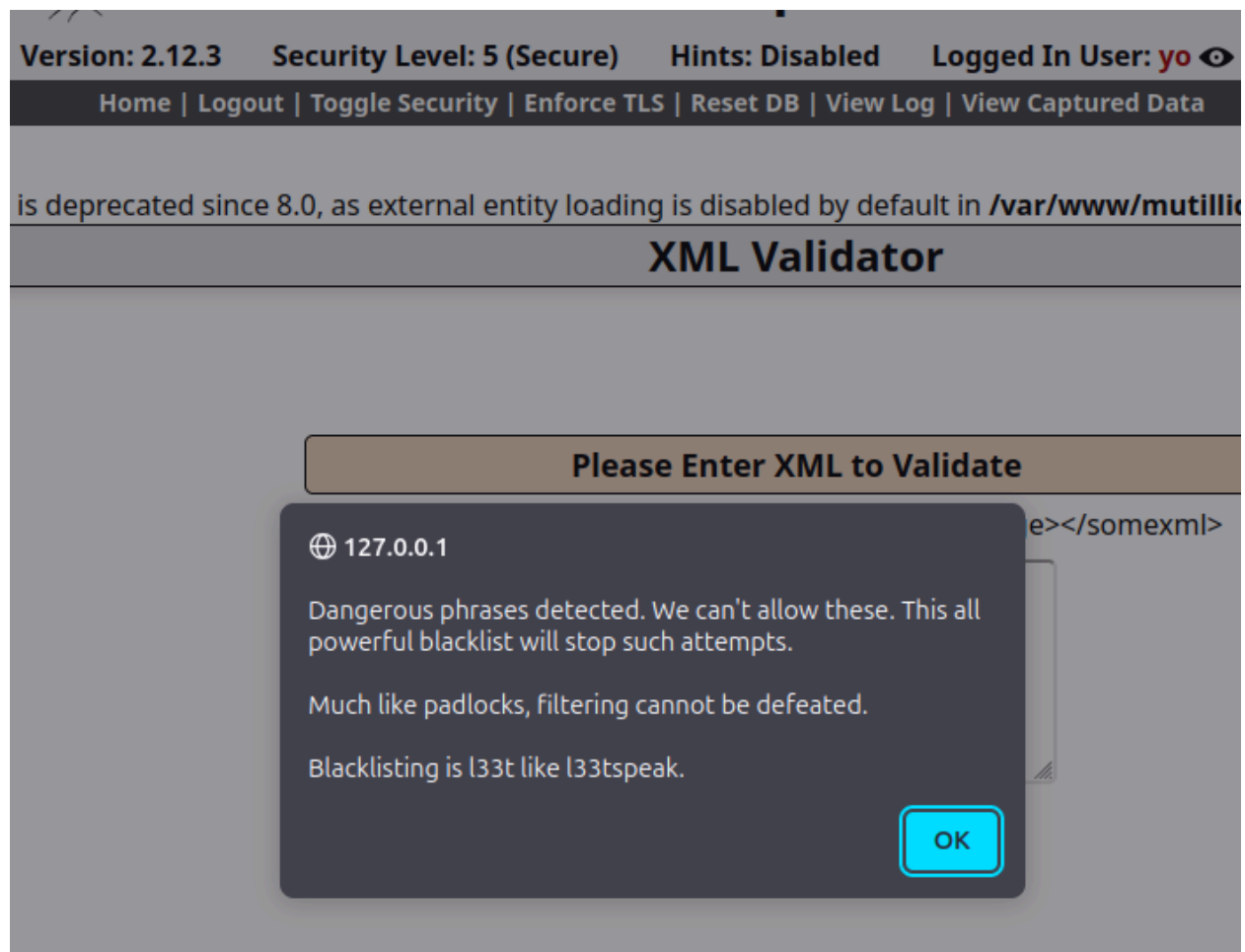
```
root:x:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534:nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
ntpsec:x:100:101:nonexistent:/usr/sbin/nologin
philius:x:1000:1000:/home/philius:/bin/sh
```

3 . Codage sécurisé

Activation du niveau 5 .

Version: 2.12.3 Security Level: 5 (Secure) Hints: Disabled Logged In User: yo  
[Home](#) | [Logout](#) | [Toggle Security](#) | [Enforce TLS](#) | [Reset DB](#) | [View Log](#) | [View Captured Data](#)

L'attaque a échoué :



Avant la modification :

```

        case "2":
        case "3":
        case "4":
    case "5": // This code is fairly secure
        $!EnableHTMLControls = true;
        //$!FormMethod = "POST";
        $!EnableJavaScriptValidation = true;
        $!EnableXMLValidation = true;
        $!EnableXMLEncoding = true;
        $!ProtectAgainstMethodTampering = true;
        libxml_disable_entity_loader(true);
        break;
} //end switch

```

Après 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** :

 **OWASP Mutillidae II: Keep Calm and Pwn On**

Version: 2.12.3 Security Level: 5 (Secure) Hints: Disabled Logged In User: yo  

[Home](#) | [Logout](#) | [Toggle Security](#) | [Enforce TLS](#) | [Reset DB](#) | [View Log](#) | [View Captured Data](#)

) is deprecated since 8.0, as external entity loading is disabled by default in `/var/www/mutillidae/xn`

XML Validator

Please Enter XML to Validate

Example: <somexml><message>Hello World</message></somexml>

XML

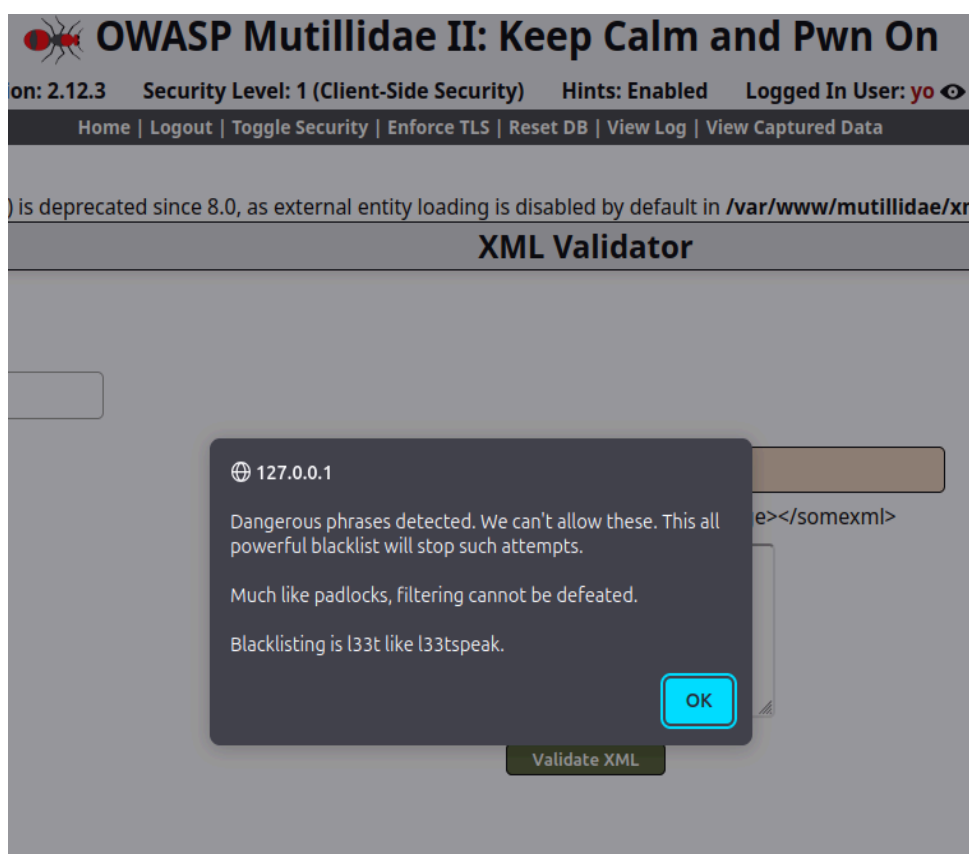
Validate XML

Possible XML external entity injection attack detected.
Support has been notified.

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 :

```
function onSubmitOfForm(/*HTMLElement*/ theForm){
    try{
        var lUnsafePhrases = /ENTITY/i;

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

return false;

```
            }// end if

            if (theForm.xml.value.search(lUnsafePhrases) > -1){
                alert('Dangerous phrases detected. We can\'t allow these. This all powerful blacklist
```

return false;

```
            }// end if
        }// end if(lValidateInput)

        return true;
    }catch(e){
        alert("Error: " + e.message);
    }// end catch
}// end function onSubmitOfForm(/*HTMLElement*/ theForm)
</script>
```

AG Help AO Write Out AW Where Is AK Cut AT Execute AC Location M-U Undo M-A Set Mark
 AX Exit AR Read File AN Replace AU Paste AJ Justify M-G Go To Line M-E Redo M-C Copy

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;

```
    }// end if
```

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(1UnsafePhrases)` 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 ?

Version: 2.12.3 Security Level: 5 (Secure) Hints: Disabled Logged In User: yo  

[Home](#) | [Logout](#) | [Toggle Security](#) | [Enforce TLS](#) | [Reset DB](#) | [View Log](#) | [View Captured Data](#)

is deprecated since 8.0, as external entity loading is disabled by default in `/var/www/mutillidae/xml-valid`

XML Validator

Please Enter XML to Validate

Example: `<somexml><message>Hello World</message></somexml>`

XML

Possible XML external entity injection attack detected.
Support has been notified.

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).

```
GNU nano /./2                                xml-validator.php
        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;
    //end switch

    if ($lEnableHTMLControls) {
        $lHTMLControlAttributes='required="required"';
    }else{
        $lHTMLControlAttributes="";
    }// end if

    $lFormSubmitted = false;
    if (isset($_POST["xml-validator-php-submit-button"]) || isset($_REQUEST["xml-validator-php-submit-button"])) {
        $lFormSubmitted = true;
    }// end if

    if ($lFormSubmitted){
```

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 :

```
if(!$enableXMLValidation && (preg_match(XML_EXTERNAL_ENTITY_REGEX_PATTERNS, $XML) || !preg_match(VAILD_XML_CHARACTERS, $XML))){
    echo "<fieldset>";
    echo "<legend>XML Submitted</legend>";
    echo "<div width='600px' class='important-code'>" . $Encoder->encodeForXML($XML) . "</div>";
    echo "</fieldset>";
    echo "<div>&nbsp;</div>";

    try {
        set_error_handler('handleXmlError!');

        $DOMDocument = new DOMDocument();
        $DOMDocument->resolveExternals = true;
```

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 (!$EnableXMLValidation && (preg_match(XML_EXTERNAL_ENTITY_REGEX_PATTERNS, $XML) || !preg_match(VALID_XML_CHARACTERS, $XML))){
```

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

Encodage des sorties XML (`encodeForXML`) :

```
if ($EnableXMLEncoding){
    $XMLToLog = $Encoder->encodeForXML($XML);
}else{
    $XMLToLog = $XML;
};

$LogHandler->writeToLog("Recieved request to validate XML for: " . $XMLToLog);
} catch (Exception $e) {
    //do nothing
}
```

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

