TAMDA Zohir
BTS SIO 2 option SLAM
Bloc 3 - Cybersécurité

Le 11 mars 2025

Attaques de type XXE

Sommaire

Introduction

Le XML (Extensible Markup Language) est un langage de balisage qui est utilisé pour représenter les données et ainsi les transmettre vers différentes machines sur le réseau.

Cependant, il est possible de récupérer des informations confidentielles en faisant une requête XML. C'est ce qu'on appelle une attaque XXE *(XML External Entity)*.

Mise en place de l'attaque XXE

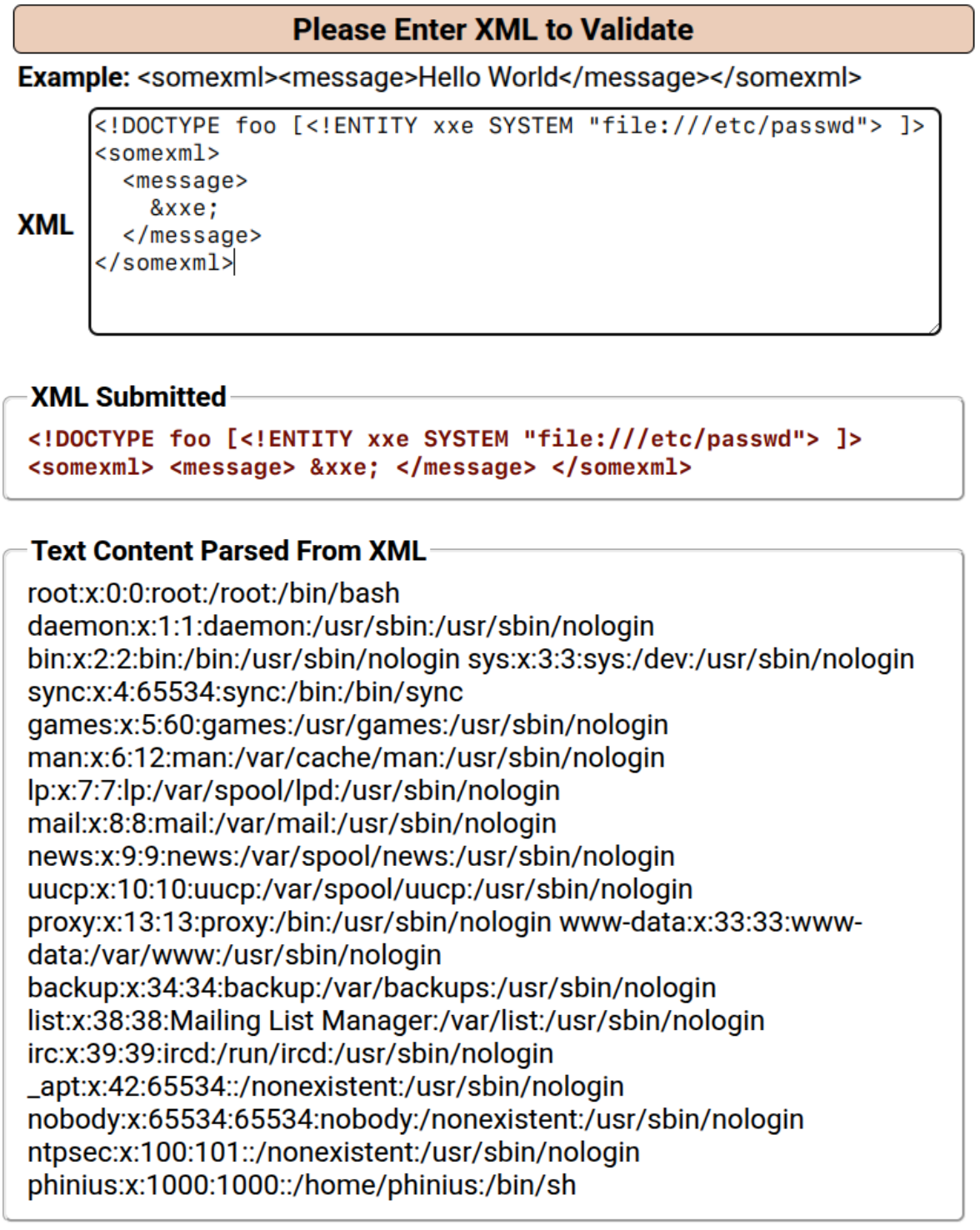
Sans le mode sécurisé

Le niveau de sécurité est définie à 0, ce qui permet d'étudier l'attaque XXE.

Exemple d'utilisation simple du code XML non malveillante.

Le code XML contient une balise parente <somexml> et une balise enfant <message> qui contient "Hello world" comme valeur. Ce code permet d'afficher tout simplement "Hello world".

Exemple d'utilisation simple du code XML malveillante.

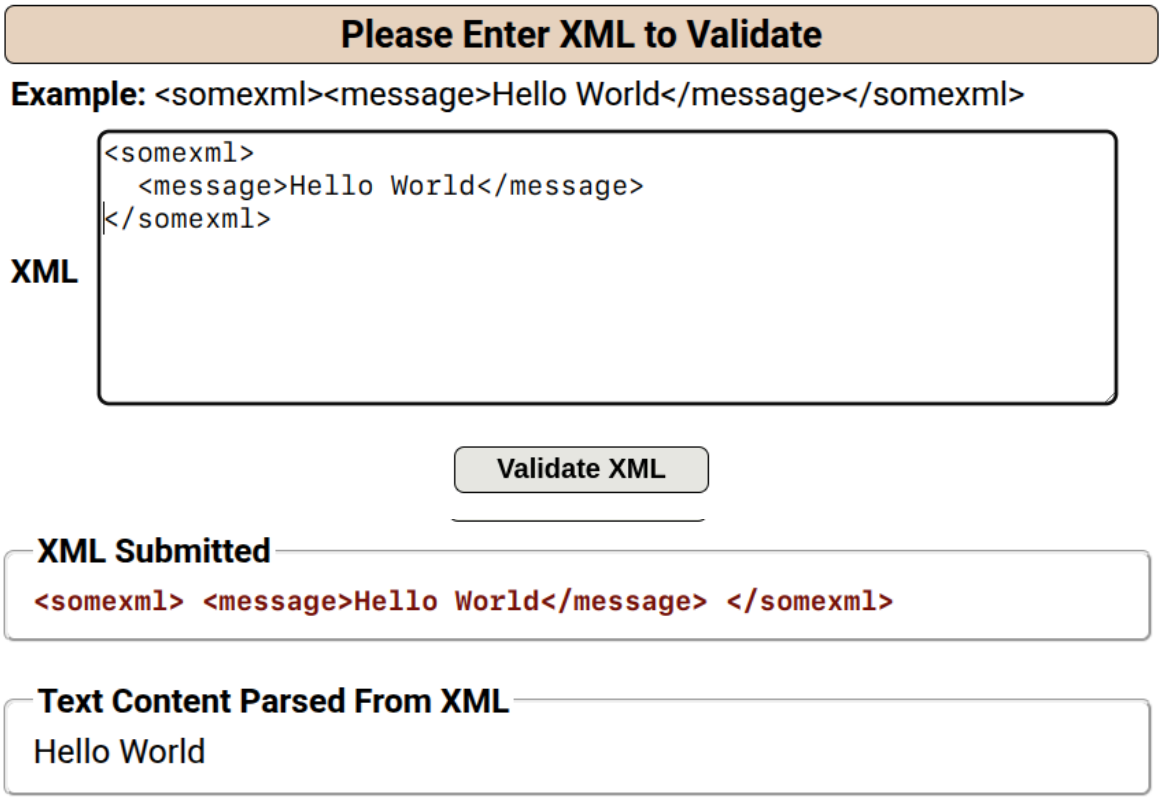


Résultat : on obtient le contenu du fichier /etc/passwd. Ceci constitue une brèche de confidentialité car on a accès à un fichier de configuration système propre à Linux.

Avec le mode sécurisé

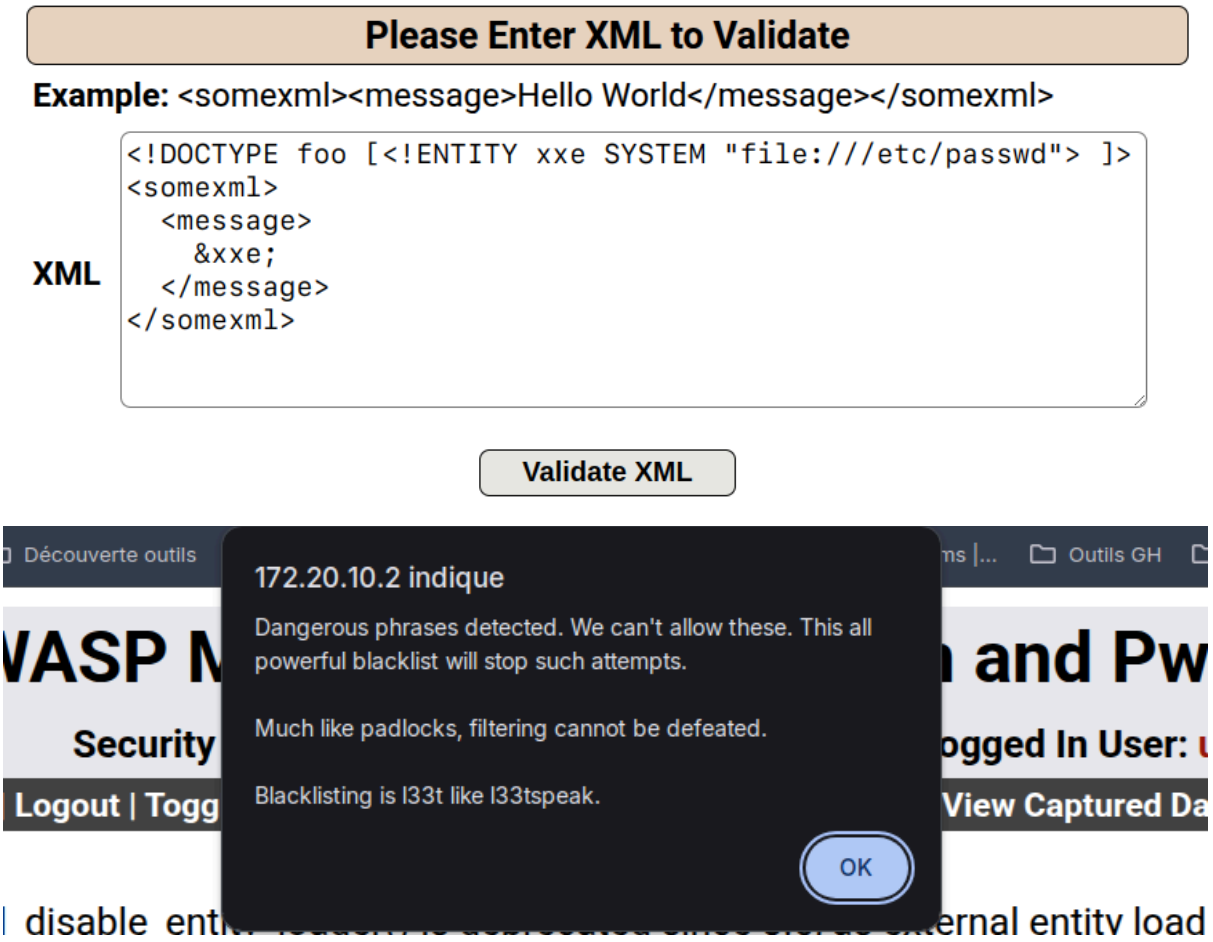
Le niveau de sécurité est défini à 5. Ce paramètre permet de vérifier que l'attaque XXE soit évitée.

Exemple d'utilisation simple du code XML non malveillante.



Le résultat présenté ci-dessus nous donne bien "Hello world".

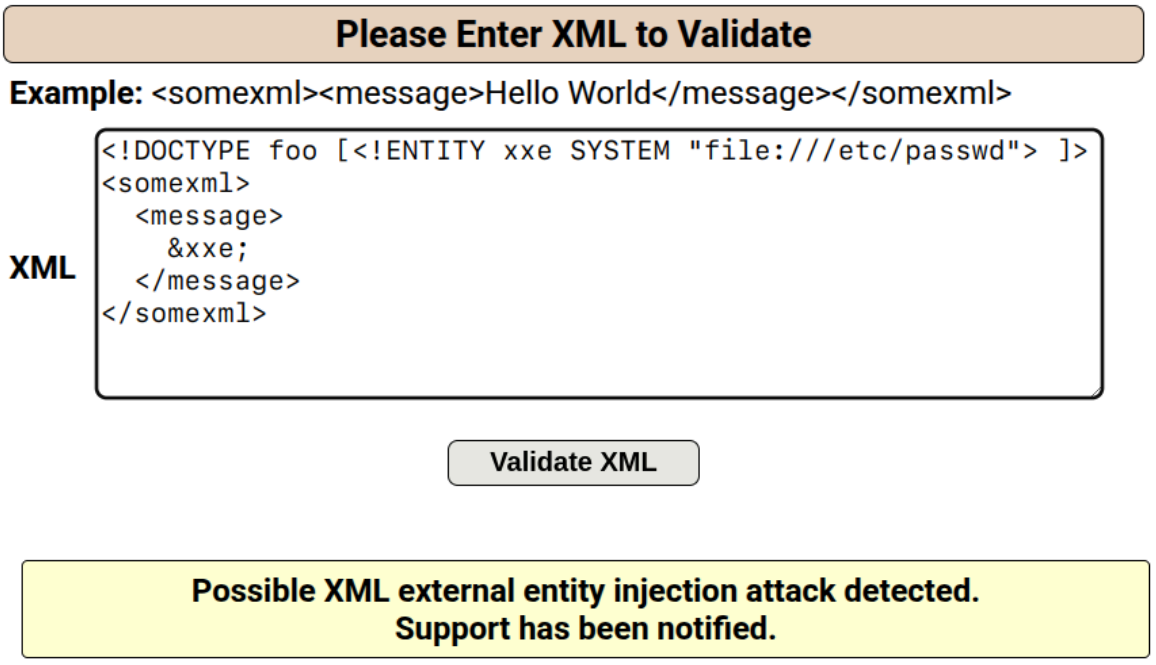
Exemple d'utilisation simple du code XML malveillante.



Cette fois-ci, l'attaque XXE a bien été rejetée et le message de blocage s'affiche.

Pour désactiver les contrôles JavaScript, il faut définir modifier la valeur de la variable $EnableJavaScriptValidation à false;

```
case "5":
-    $EnableJavaScriptValidation = true;
+    $EnableJavaScriptValidation = false;
```



En suivant la procédure, on obtient le message en jaune indiquant qu'une attaque XXE est détecté.

Du coup, le fait de désactiver les validations JavaScript ne permet pas de se protéger de cette attaque car aucune validation n'a été faite côté serveur.

Analyse du code source JavaScript

```javascript
// Fichier: xml-validator.php

var lUnsafePhrases = /ENTITY/i;

if (theForm.xml.value.search(lUnsafePhrases) > -1){
    alert('Dangerous phrases detected. We can\'t allow these.');
    return false;
}
```

Le but de ce code JavaScript est de vérifier si le code XML qui est présent dans ce formulaire contient "ENTITY" (que ce soit des lettres majuscules ou minuscules, il n'y a pas de différence). Si c'est le cas, alors le navigateur va afficher une alerte, indiquant qu'un mot-clé interdit est utilisé.

Pour être plus précis, la variable lUnsafePhrases contient une expression qu'on appelle le regex (expression régulière) qui contient un mot ENTITY. La lettre i signifie que l'expression est insensible à la casse (que ce soit des lettres majuscules ou minuscules).

Puis, une condition est présente. À l'intérieur de cette condition, la fonction search est utilisée pour récupérer la position d'un mot dans une chaîne de caractères. Si la fonction search renvoie un nombre entier positif et qu'il est strictement supérieur à -1, alors cela signifie que le mot "ENTITY" est présent. Et dans ce cas, cette condition sera exécutée et l'alerte sera affichée.

Pour récapitulatif : le niveau de sécurité à 5 ne permet pas d'éviter l'attaque XXE car aucune validation n'a été faite côté serveur.

```php
<?php

function handleXmlError($errno, $errstr, $errfile, $errline)
{
   if ($errno == E_WARNING && (substr_count($errstr, "DOMDocument::loadXML()") > 0)) {
       throw new DOMException($errstr);
   } else {
       return false;
   } //end if
} // end function HandleXmlError
```

```php
try {
   switch ($_SESSION["security-level"]) {
       default: // Default case: This code is insecure
       case "0": // This code is insecure
           $lEnableHTMLControls = false;
           //$lFormMethod = "GET";
           $lEnableJavaScriptValidation = false;
           $lEnableXMLValidation = false;
           $lEnableXMLEncoding = false;
           $lProtectAgainstMethodTampering = false;
           libxml_disable_entity_loader(false);
           break;

       case "1": // This code is insecure
           $lEnableHTMLControls = true;
           //$lFormMethod = "GET";
           $lEnableJavaScriptValidation = true;
           $lEnableXMLValidation = false;
           $lEnableXMLEncoding = false;
           $lProtectAgainstMethodTampering = false;
           libxml_disable_entity_loader(false);
           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;
   } //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 ($lFormSubmitted) {
        if ($lProtectAgainstMethodTampering) {
            $lXMLValidatorSubmitButton =
$_POST["xml-validator-php-submit-button"];
            $lXML = $_POST["xml"];
        } else {
            $lXMLValidatorSubmitButton =
$_REQUEST["xml-validator-php-submit-button"];
            $lXML = $_REQUEST["xml"];
        } // end if $lProtectAgainstMethodTampering

        try {
            if ($lEnableXMLEncoding) {
                $lXMLToLog = $Encoder->encodeForXML($lXML);
            } else {
                $lXMLToLog = $lXML;
            };
            $LogHandler->writeToLog("Recieved request to validate XML for: " .
$lXMLToLog);
        } catch (Exception $e) {
            //do nothing
        } // end try
    } // end if $lFormSubmitted

} catch (Exception $e) {
    echo $CustomErrorHandler->FormatError($e, $lQueryString);
} // end try;
?>

<script type="text/javascript">
    <?php
    if ($lEnableJavaScriptValidation) {
        echo "var lValidateInput = \"true\"" . PHP_EOL;
    } else {
        echo "var lValidateInput = \"false\"" . PHP_EOL;
    } // end if
    ?>

    function onSubmitOfForm( /*HTMLFormElement*/ 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 will stop such attempts.\n\nMuch like padlocks,
filtering cannot be defeated.\n\nBlacklisting is l33t like l33tspeak.');
                    return false;
                } // end if
            } // end if(lValidateInput)

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

<div class="page-title">XML Validator</div>

<?php include_once __SITE_ROOT__ . '/includes/back-button.inc'; ?>
<?php include_once __SITE_ROOT__ . '/includes/hints/hints-menu-wrapper.inc'; ?>

<form action="./index.php?page=xml-validator.php"
    method="POST"
    enctype="application/x-www-form-urlencoded"
    onsubmit="return onSubmitOfForm(this);">
    <input type="hidden" name="page" value="xml-validator.php" />
    <table>
        <tr id="id-bad-cred-tr" style="display: none;">
            <td colspan="2" class="error-message">
                Authentication Error: Bad XML Input
            </td>
        </tr>
        <tr>
            <td></td>
        </tr>
        <tr>
            <td colspan="2" class="form-header">Please Enter XML to Validate</td>
        </tr>
        <tr>
            <td colspan="2">
                <span class="label">Example:</span>
                &lt;somexml&gt;&lt;message&gt;Hello
World&lt;/message&gt;&lt;/somexml&gt;
            </td>
        </tr>
        <tr>
            <td class="label">XML</td>
            <td>
                <textarea name="xml" rows="8" cols="50" id="idXMLTextArea"
title="Please enter XML to validate" autofocus="autofocus" <?php echo
$lHTMLControlAttributes ?>></textarea>
            </td>
```

```
        </tr>
        <tr>
            <td></td>
        </tr>
        <tr>
            <td colspan="2" style="text-align:center;">
                <input name="xml-validator-php-submit-button" class="button"
type="submit" value="Validate XML" />
            </td>
        </tr>
    </table>
</form>

<?php
if (isset($lXMLValidatorSubmitButton) && !empty($lXMLValidatorSubmitButton) &&
strlen($lXML) > 0) {

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

            echo "<fieldset>";
            echo "<legend>XML Submitted</legend>";
            echo "<div width='600px' class=\"important-code\">" .
$Encoder->encodeForXML($lXML) . "</div>";
            echo "</fieldset>";
            echo "<div> </div>";

            try {
                set_error_handler('handleXmlError');

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

                echo "<fieldset>";
                echo "<legend>Text Content Parsed From XML</legend>";
                echo "<div width='600px'>" . $lDOMDocument->textContent .
"</div>";
                echo "</fieldset>";
                echo "<div> </div>";

                restore_error_handler();
            } catch (Exception $e) {
                echo $CustomErrorHandler->FormatError($e, "Could not parse XML
because the input is mal-formed or could not be interpreted.");
            } //end try
```

```
      } else {
          echo "<div> </div>";
          echo "<div style=\"width:500px;margin-right:auto;margin-left:auto;\"
class=\"warning-message\">
              Possible XML external entity injection attack detected.<br/>
              Support has been notified.
          </div>";
      } //end if

   } catch (Exception $e) {
       echo $CustomErrorHandler->FormatError($e, $lQueryString);
   } // end try;

} // end if (isset($_POST))
?>
```

Expression régulière (regex)

Une expression régulière (ou regex en anglais) permet de correspondre à certaines combinaisons de caractères dans une chaîne de caractères ou dans un texte.

La fonction native de PHP `preg_match()` consiste à vérifier si une chaîne de caractères correspond au pattern.

La fonction native de PHP `libxml_disable_entity_loader()` consiste à désactiver le chargement des entités externes dans le code XML. Actuellement, cette fonction est dépréciée depuis la version 8.0 de PHP.

Avertissement Cette fonction est *OBSOLÈTE* à partir de PHP 8.0.0. Dépendre de cette fonction est fortement déconseillé.

```
#[\Deprecated]
libxml_disable_entity_loader(bool $disable = true):
bool
```

Méthodes pour lutter contre l’attaque XXE

Pour faire face à cette attaque, plusieurs contre-mesures peuvent être mises en place :

1) Effectuer également des validations côté serveur pour vérifier si le code XML contient le mot-clé ENTITY (c’est le même principe que d’effectuer les validations côté client).

2) Utiliser un autre format de données que le XML : le JSON par exemple.

3) Favoriser l’utilisation des expressions régulières pour filtrer facilement les mots-clés (ENTITY) qui mettraient en danger le serveur.

4) Mettre à jour la version de la bibliothèque libxml à 2.9.0.

5) Mettre en place un pare-feu d’applications / des tests unitaires pour réduire la vulnérabilité d’une application.