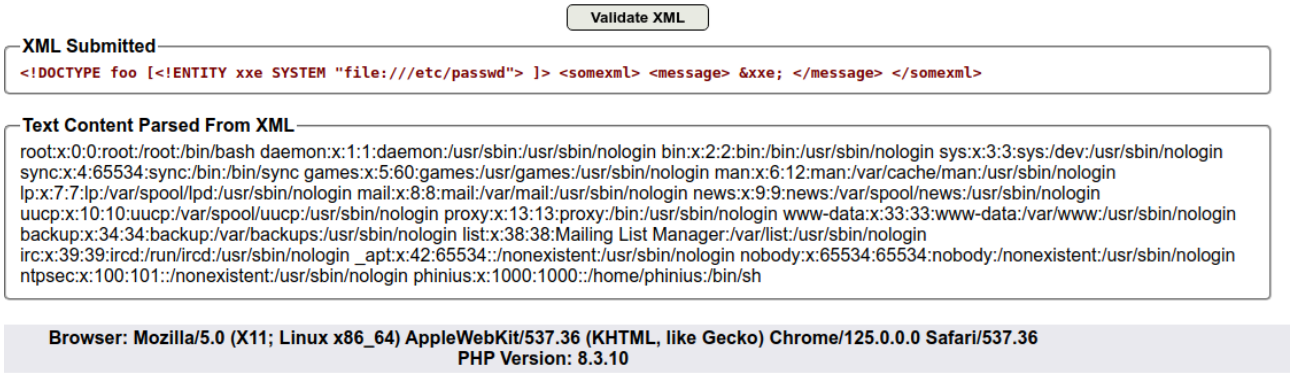
Activité 5 : Attaques de type XXE (XML External Entities)

Hassani Ayoub

1. Mise en place de l environnement de travail en démarrant le serveur Mutillidae ainsi que la machine cliente.

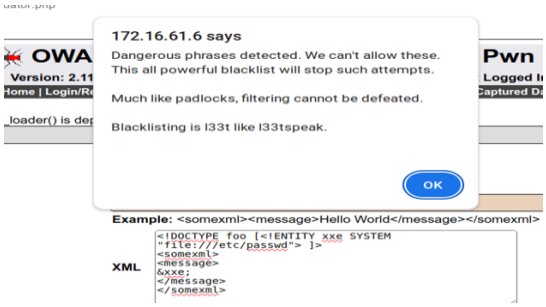
2. Nous avons injecté ce code malveillant après avoir suivi le chemin suivant **OWASP 2017 => XML External Entities => XML External Entity Injection => XML Validator**:

<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<somexml>
<message>
&xxe;
</message>
</somexml>
Ceci nous a permis de recuperer le contenu du fichier système /etc/passwd.

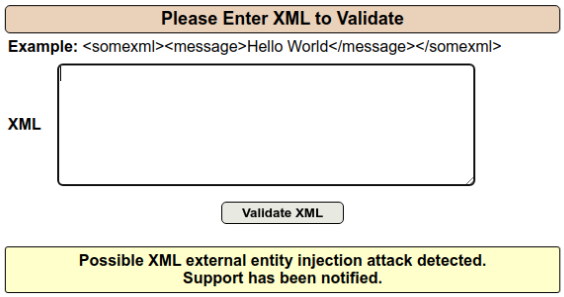


Nous allons par la suite mettre le niveau de sécurité au niveaux 5 (maximum) puis nous allons reproduire les mêmes étapes :

Nous voyons ici que l'on nous préviens d une « dangerous phrases detected », cependant ceci est hérité du niveau 1 de sécurité et non du niveau 5. Nous allons donc modifié le code source pour desactivé les securitées de niveau 1.



voila ce que nous obtenons a la suite des modification du fichier xml-validator.php que nous avons directement modifier sur portainer.
Apres avoir tapé cette commande Nano xml-validator.php
Nous avons modifié cette ligne ci :
case “5”:
$EnableJavaScriptValidation = TRUE to case “5”:
$EnableJavaScriptValidation = FALSE



Travail à faire 3 Nouvelle tentative en mode sécurisé et analyse du code source

1. Il y a un blocage mais à cause des sécurités Javascript, sans ce contrôle sur Javascript, le niveau 1 de sécurité ne bloque pas l’attaque car il ne comporte pas de contrôles sur XXE. Le niveau de sécurité 1 ne permet pas d’éviter l’attaque XXE.
Ce niveau de sécurité repose sur des vérifications côté client en JavaScript, qui bloquent certaines phrases suspectes comme `ENTITY`. Cependant, ces vérifications sont faciles à contourner car elles ne sont pas appliquées côté serveur. Ainsi, si un attaquant désactive JavaScript ou modifie la requête directement, l'attaque XXE peut toujours être exécutée.

2. var lunsafephrases = /ENTITY/i;
if (theForm.xml.value.search(lunsafephrases) > -1){
    alert('Dangerous phrases detected. we can\'t allow these.');
    return false;
}
Ce bloc de code JavaScript vérifie si le texte saisi dans le formulaire XML contient le mot-clé ENTITY . Si le mot-clé est détecté, une alerte est affichée à l'utilisateur, et la soumission du formulaire est bloquée. «lunsafephrases» permet de rechercher le mot-clé ENTITY, theForm.xml.value.search(lunsafephrases) recherche la présence du mot-clé dans le texte XML,alert affiche un message d'avertissement à l'utilisateur et return false empêche la soumission du formulaire si une phrase dangereuse est détectée.

Test du niveau de sécurité 5 (Secure) :

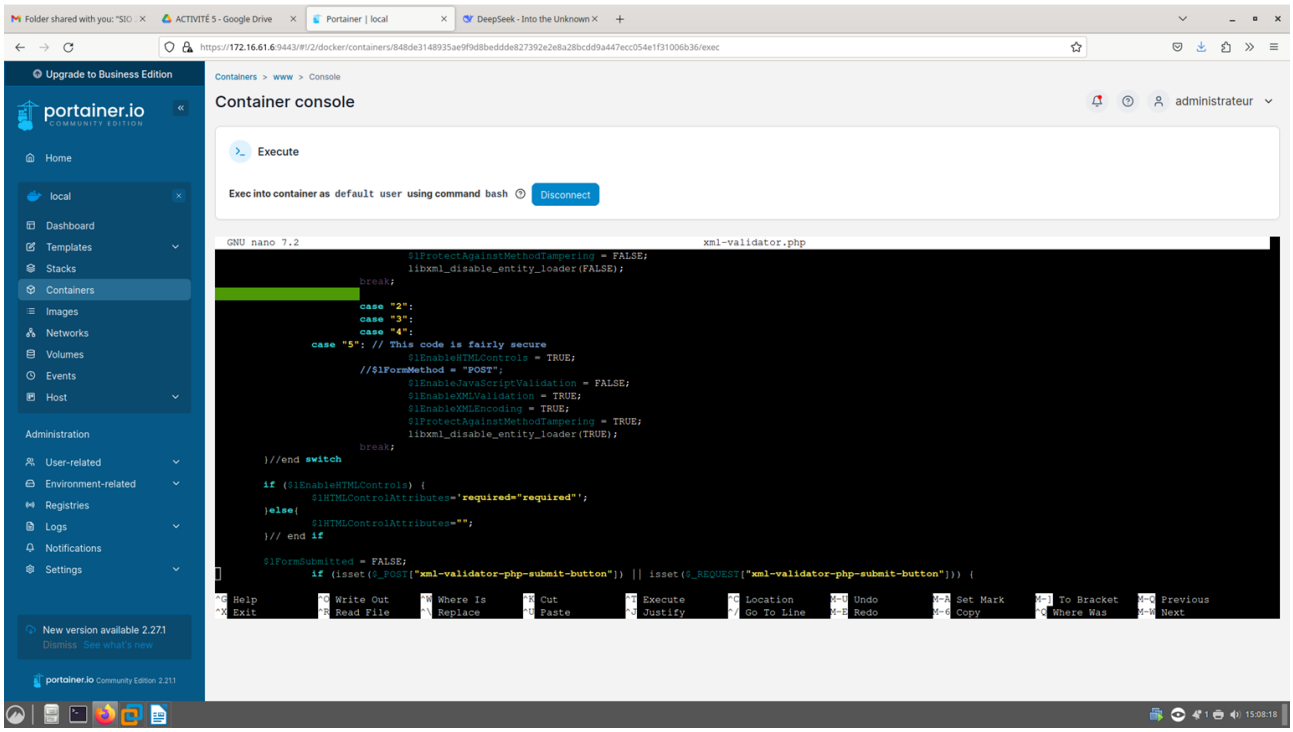
3. Oui, le niveau de sécurité 5 (Server-side Security) permet d’éviter l’attaque XXE. Ce niveau implémente des vérifications côté serveur, notamment en désactivant le chargement des entités

externes via `libxml_disable_entity_loader` et en validant les entrées XML pour bloquer les données malveillantes.

4. Grace à la

```
//$lFormMethod = "POST";
        $lEnableJavaScriptValidation = FALSE;
        $lEnableXMLValidation = TRUE;
        $lEnableXMLEncoding = TRUE;
        $lProtectAgainstMethodTampering = TRUE;
        libxml_disable_entity_loader(TRUE);
break;
```

commande « more /var/www/html/mutillidae/xml-validator.php » nous voyons directement le code source de notre page.



5. Une expression régulière (ou motif) est une séquence de caractères qui définit un modèle de recherche. Elle est utilisée pour valider, rechercher ou remplacer des chaînes de caractères. Dans le contexte de la sécurité, les expressions régulières permettent de détecter des motifs suspects dans les entrées utilisateur, comme la présence de balises ou d'entités malveillantes dans un document. XML.VALID_XML_CHARACTERS est un motif qui définit les caractères autorisés dans un document XML si une entrée contient des caractères non autorisés, elle est rejetée.

6. La fonction `preg_match` est utilisée dans le fichier `xml-validator.php` pour valider les entrées XML. Elle vérifie si le XML soumis par l'utilisateur correspond à un motif défini (par exemple, des caractères autorisés). Si des caractères non autorisés ou des motifs suspects sont détectés, la fonction retourne `0`, et l'entrée est rejetée. Cela permet de bloquer les attaques XXE en empêchant l'injection de données malveillantes.

7. La fonction `libxml_disable_entity_loader(true)` désactive le chargement des entités externes en PHP, ce qui empêche les attaques XXE. En désactivant cette fonctionnalité, les entités externes ne peuvent plus être interprétées, ce qui bloque les tentatives d'injection XXE.
Depuis PHP 8.0 : La fonction `libxml_disable_entity_loader` est obsolète car le chargement des entités externes est désactivé par défaut. Cela renforce la sécurité sans nécessiter de configuration supplémentaire.

8. Grace au tp effectué nous concluons que la meilleure méthode de codage sécurisée à utiliser pour empêcher les attaques XXE repose sur differents mécanismes clés, premierement la fonction `libxml_disable_entity_loader(true)` peut être utilisée pour désactiver le chargement des entités externes en PHP, ce qui empêche les attaques XXE en bloquant l'interprétation des entités malveillantes. Depuis PHP 8.0, cette fonctionnalité est désactivée par défaut, renforçant ainsi la sécurité sans nécessiter de configuration supplémentaire. La validation des entrées XML des expressions régulières (comme `VALID_XML_CHARACTERS`) sont utilisées pour vérifier que les entrées XML ne contiennent que des caractères autorisés. La fonction `preg_match` est employée pour détecter les motifs suspects et rejeter les données malveillantes avant qu'elles ne soient traitées, les vérifications sont effectuées côté serveur, ce qui rend les protections plus robustes que celles côté client (JavaScript), car elles ne peuvent pas être contournées par un attaquant. Lorsque possible, il est recommandé d'utiliser des formats de données moins complexes comme JSON, qui ne supportent pas les entités externes et réduisent ainsi les risques d'attaques XXE.

En résumé, la méthode de codage sécurisée combine la désactivation des entités externes, la validation des entrées XML, et le filtrage côté serveur pour empêcher les attaques XXE. Ces mesures garantissent que les données XML sont traitées de manière sûre et que les applications web sont protégées contre les injections malveillantes.