RAPPORT D'EXÉCUTION ET D'ANALYSE

Interactions XML / Vulnérabilité XXE

PARTIE 1 : Exemples d'interactions XML / XXE

1. Usage standard XML

Code exécuté :

```xml
<?xml version="1.0" ?>
<foo>
Hello Ali
</foo>
```

Résultat après exécution :
L'application affiche simplement :

Hello Ali

Interprétation :

- Le document XML est bien formé.
- Aucun DTD (Document Type Definition) n'est défini.
- Le parseur XML affiche uniquement le contenu textuel de la balise <foo>.
- **Aucun risque de sécurité** à ce stade.

---

2. Usage avec une entité XML

Code exécuté :

```xml
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY nom "Acharf" >
]>
<foo>
Hello &nom;
</foo>
```

Résultat après exécution :

Hello Acharf

Réalisé par : Romain AKPO | **Tél** : +229 019765335 | **Email** : romainakpo86@gmail.com

Interprétation :

- Une entité interne `nom` est définie dans la DTD.
- Le parseur XML remplace `&nom;` par sa valeur `"Acharf"`.
- Cela démontre que **le parseur autorise l'expansion des entités**.
- **Première indication d'un risque XXE**, si les entités externes sont aussi autorisées.

---

3. Déni de service (Billion Laughs Attack)

Code exécuté :

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY nom "lol ">
<!ENTITY t1 "&nom;&nom;">
<!ENTITY t2 "&t1;&t1;&t1;&t1;">
<!ENTITY t3 "&t2;&t2;&t2;&t2;&t2;">
]>
<foo>
Hello &t3;
</foo>
```

Résultat après exécution :

- L'application devient très lente **et** plante.
- Forte consommation de mémoire et CPU.
- Dans mon cas : **crash du serveur ou erreur interne**.

Interprétation :

- Il s'agit d'une **attaque de type déni de service XML**.
- Les entités sont expansées récursivement.
- Le volume de données explose de manière exponentielle.
- Cela prouve que le parseur XML est **dangereusement mal configuré**.

PARTIE 2 : Vulnérabilité XXE d'une application Web

Étape 1 : Identification de la vulnérabilité

Injection XML simple dans le champ de saisie

Observation :

- Le contenu XML est interprété.
- Les entités internes sont correctement résolues.

Conclusion :
➡️ L'application **parse le XML côté serveur** sans restriction.

Injection avec entité

Observation après clic sur « Envoyer » :

- Les entités sont interprétées et affichées.
- Cela confirme que **les DTD sont autorisées**.

Conclusion :
➡️ L'application est **vulnérable à XXE**.

Étape 2 : Exploitation de la vulnérabilité XXE

Création du fichier passwd.txt
Exemple de contenu :
`admin : admin123`
`user :test456`
Injection XML malveillante

Principe de l'attaque :

- Une entité externe est utilisée pour lire un fichier local du serveur.
- Exemple conceptuel :

```
<!DOCTYPE root [
<!ENTITY xxe SYSTEM "C:\xampp\htdocs\a4\a4.2\passwd.txt">
]>
<foo>&xxe;</foo>
```

Résultat après exécution :

- Le contenu du fichier `passwd.txt` est affiché à l'écran.

Interprétation :

- Le serveur lit un fichier local sans autorisation.
- Il s'agit d'une **faille critique de type XXE – Disclosure de fichiers**.
- Des fichiers sensibles peuvent être exposés.

PARTIE 3 : Protection contre l'attaque XXE

Solutions recommandées

1. Désactiver les DTD

- Interdire totalement les déclarations `<!DOCTYPE>`.

2. Désactiver les entités externes

Exemples :

- PHP (libxml) :

```
libxml_disable_entity_loader(true);
```

- Java :

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

3. Utiliser un parseur sécurisé

- Mode **secure processing**
- Limitation mémoire et temps d'exécution

4. Validation stricte des entrées utilisateur

- Refuser tout contenu XML non attendu
- Utiliser un schéma XSD strict

CONCLUSION GÉNÉRALE

- L'application analysée est **clairement vulnérable à XXE**.
- Les risques identifiés :
  - Lecture de fichiers sensibles
  - Déni de service
  - Compromission du serveur
- La **désactivation des DTD et entités externes est obligatoire** en production.