



# THE XXE DILEMMA

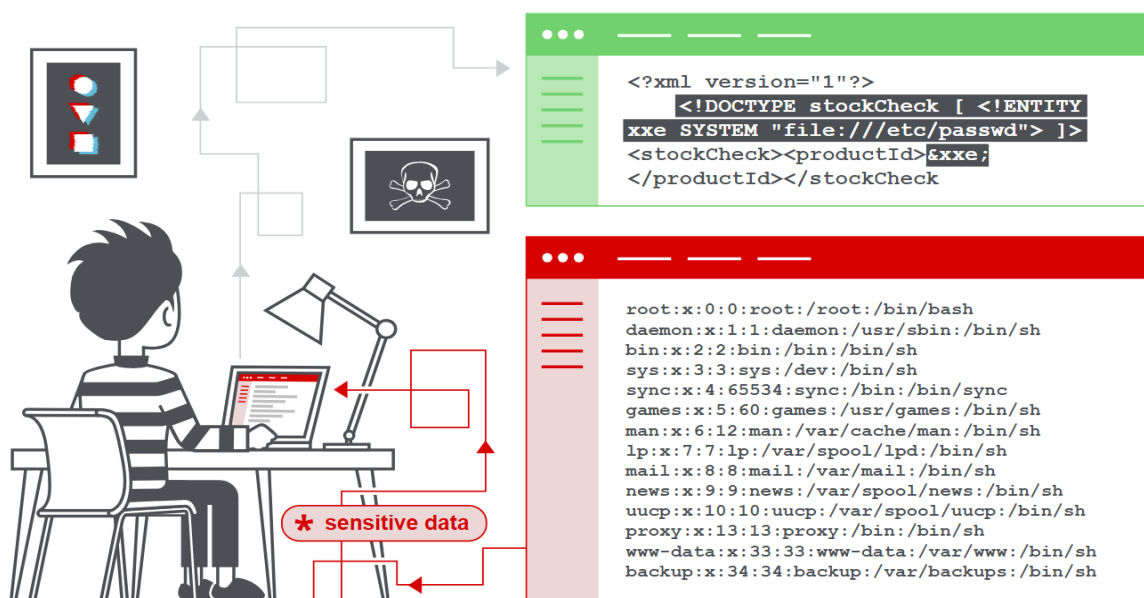
UNDERSTANDING AND PREVENTING XML  
EXTERNAL ENTITY ATTACKS

# XML external entity (XXE) injection

## 1. Introduction

XML External Entity (XXE) injection is a security flaw in web applications that occurs when XML data is processed insecurely. This vulnerability allows attackers to manipulate the application's XML parsing process, which may grant access to files stored on the server's filesystem or allow interaction with back-end or external systems accessible to the application.

Sometimes, it is possible to escalate the issue of XXE to breach the server or even the underlying infrastructure of the server. The vulnerability of XXE is used to send SSRF, which helps in making unauthorized requests on internal and external services in the name of the server.



## 2. What is XML?

XML is an extensible markup language, a text-based format for encoding documents in a machine-readable and human-readable form. It is used widely in data exchange between systems, thus making communication seamless and data sharing easy.

## 3. What is XXE?

External entities are special XML constructs that enable the inclusion of data from other sources within an XML document. The entities can refer to files, URLs, or any other resources. It is thus a flexible means of managing and reusing data.

## 4. Understanding XXE Injection

- **Definition:** XXE Injection is a type of vulnerability that exploits the XML parser's capability to process external entities. By manipulating the XML input, an attacker can inject malicious entities that the XML parser inadvertently executes.
- **How XXE Injection works:** In XXE-based attack, a cracker attaches hostile entities inside XML input files especially using DTDS (Doctype Declaration) so that processing this file via the XML parser accidentally performs illegitimate actions-like fetches locally situated files and data transfer externally-servers.
- **Example:** Consider the following:

```
Xml

<!DOCTYPE example [
<!ELEMENT example ANY >
<!ENTITY file SYSTEM "file:///etc/passwd" >
]>
<example>&file;</example>
```

- In this case, the external entity &file; references the contents of the /etc/passwd file. When processed, the XML parser retrieves and incorporates this file's content into the XML output.

## 5. Impact

- Potential Consequences of XXE Injection:
  - **Data Exposure:** Unauthorized access to sensitive files and data.
  - **Denial of Service (DoS) Attacks:** Overloading the system by consuming resources.
  - **Remote Code Execution (RCE):** Executing arbitrary code on the server.
  - **SSRF** (Server-Side Request Forgery): Abusing Server functionality to access or modify resources
- Case study:
  - A real-world example of the attack from XXE Injection is one that happened in 2013 against a US-based firm. The online service of the company was vulnerable to XXE Injection so that attackers retrieved sensitive data from the server. The attacker had complete access to files containing personal information through the exploitation of the XXE vulnerability, causing a massive data breach.
  - This incident is highlighted for securing XML parsers as well as input validation to prevent similar vulnerability. It also brings forth the potential consequences of having XXE Injection, such as data exposure and unauthorized access to sensitive information.

## 6. Detection

### 1) Automated Tools

- Static Analysis Tools: These tools scan codebases for potential XXE vulnerabilities. Examples include SonarQube and Checkmarx.
- Dynamic Analysis Tools: These tools test applications during runtime to identify vulnerabilities. Examples include Burp Suite and OWASP ZAP.

### 2) Manual Techniques

- Code Reviews: Conduct thorough code reviews to identify instances where XML input is processed without proper validation or configuration.
- Security Audits: Regular security audits can help identify XXE vulnerabilities in existing systems.
- Input Validation: Ensure that all XML inputs are validated and sanitized to prevent malicious entities from being processed.

### 3) Configuration Checks

- XML Parser Configuration: Ensure that XML parsers are configured to disable external entity processing. For example, in Java, you can disable external entities by setting the **disallow-doctype-decl** feature to true:

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

### 4) Monitoring and Logging:

- Log Analysis: Monitor application logs for unusual activity that may indicate an XXE attack, such as unexpected file access or network requests.
- Intrusion Detection Systems (IDS): Use IDS to detect potential XXE attacks by monitoring network traffic for suspicious patterns.

## 7. Prevention

### 1) Secure Coding Practices

- **Avoid External Entities:** Unless absolutely necessary, do not use external entities in your XML. If they must be used, ensure strict validation and sanitization.
- **Input Validation:** Always validate and sanitize XML input to ensure it meets the expected format and does not contain malicious content.
- **Use Least Privilege:** Run applications with the least privilege necessary to limit the impact of potential vulnerabilities.

### 2) Configuring XML Parsers Securely

- **Disable External Entity Processing:** Configure XML parsers to disable external entity processing. Here are examples for different programming languages:

#### **JAVA:**

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();  
dbf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);  
dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);  
dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
```

**Python:**

```
from lxml import etree  
parser = etree.XMLParser(resolve_entities=False)
```

**CSharp:**

```
XmlReaderSettings settings = new XmlReaderSettings();  
settings.DtdProcessing = DtdProcessing.Prohibit;
```

**PHP:**

```
libxml_disable_entity_loader(true);
```

## 8. Mitigation

- **Regular Security Audits:** Conduct regular security audits and code reviews to identify and address XXE vulnerabilities.
- **Static and Dynamic Analysis:** Use static and dynamic analysis tools to detect potential XXE vulnerabilities during development and testing phases.
- **Intrusion Detection Systems (IDS):** Implement IDS to monitor for and alert on potential XXE attacks based on abnormal behavior and access patterns.

## 9. Tools

List of Tools for Detecting and Mitigating XXE Vulnerabilities:








- **Static Analysis Tools:** SonarQube, Checkmarx.
- **Dynamic Analysis Tools:** Burp Suite, OWASP ZAP

## 10. Sources

- <https://www.w3.org/XML/>
- [https://cheatsheetseries.owasp.org/cheatsheets/XML\\_External\\_Entity\\_Prevention\\_Cheat\\_Sheet.html?form=MG0AV3](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html?form=MG0AV3)
- <https://portswigger.net/web-security/xxe>

## Labs Practiced

### XML external entity (XXE) injection

 LAB	<b>APPRENTICE</b> Exploiting XXE using external entities to retrieve files →	✓ Solved
 LAB	<b>APPRENTICE</b> Exploiting XXE to perform SSRF attacks →	✓ Solved
 LAB	<b>PRACTITIONER</b> Blind XXE with out-of-band interaction →	✓ Solved
 LAB	<b>PRACTITIONER</b> Blind XXE with out-of-band interaction via XML parameter entities →	✓ Solved
 LAB	<b>PRACTITIONER</b> Exploiting blind XXE to exfiltrate data using a malicious external DTD →	✓ Solved
 LAB	<b>PRACTITIONER</b> Exploiting blind XXE to retrieve data via error messages →	✓ Solved
 LAB	<b>PRACTITIONER</b> Exploiting XInclude to retrieve files →	✓ Solved
 LAB	<b>PRACTITIONER</b> Exploiting XXE via image file upload →	✓ Solved