XML External Entity Prevention Cheat Sheet

Introduction

XML eXternal Entity injection (XXE), which is now part of the OWASP Top 10 via the point **A4**, is a type of attack against an application that parses XML input.

XXE issue is referenced under the ID 611 in the Common Weakness Enumeration referential.

This attack occurs when untrusted XML input containing a **reference to an external entity is processed by a weakly configured XML parser**.

This attack may lead to the disclosure of confidential data, denial of service, Server Side Request Forgery (SSRF), port scanning from the perspective of the machine where the parser is located, and other system impacts. The following guide provides concise information to prevent this vulnerability.

For more information on XXE, please visit XML External Entity (XXE).

General Guidance

The safest way to prevent XXE is always to disable DTDs (External Entities) completely. Depending on the parser, the method should be similar to the following:

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

Disabling DTDs also makes the parser secure against denial of services (DOS) attacks such as Billion Laughs. If it is not possible to disable DTDs completely, then external entities and external document type declarations must be disabled in the way that's specific to each parser.

Detailed XXE Prevention guidance for a number of languages and commonly used XML parsers in those languages is provided below.

C/C++

libxml2

The Enum xmlParserOption should not have the following options defined:

- `XML_PARSE_NOENT`: Expands entities and substitutes them with replacement text

- `XML_PARSE_DTDLOAD`: Load the external DTD

Note:

Per: According to this post, starting with libxml2 version 2.9, XXE has been disabled by default as committed by the following patch.

Search for the usage of the following APIs to ensure there is no `XML_PARSE_NOENT` and `XML_PARSE_DTDLOAD` defined in the parameters:

- `xmlCtxtReadDoc`
- `xmlCtxtReadFd`
- `xmlCtxtReadFile`
- `xmlCtxtReadIO`
- `xmlCtxtReadMemory`
- `xmlCtxtUseOptions`
- `xmlParseInNodeContext`
- `xmlReadDoc`
- `xmlReadFd`
- `xmlReadFile`
- `xmlReadIO`
- `xmlReadMemory`

libxerces-c

Use of `XercesDOMParser` do this to prevent XXE:

```
XercesDOMParser *parser = new XercesDOMParser;
parser->setCreateEntityReferenceNodes(true);
parser->setDisableDefaultEntityResolution(true);
```

Use of SAXParser, do this to prevent XXE:

```
SAXParser* parser = new SAXParser;
parser->setDisableDefaultEntityResolution(true);
```

Use of SAX2XMLReader, do this to prevent XXE:

```
SAX2XMLReader* reader = XMLReaderFactory::createXMLReader();
parser->setFeature(XMLUni::fgXercesDisableDefaultEntityResolution, true);
```

Java

Java applications using XML libraries are particularly vulnerable to XXE because the default settings for most Java XML parsers is to have XXE enabled. To use these parsers safely, you have to explicitly disable XXE in the parser you use. The following describes how to disable XXE in the most commonly used XML parsers for Java.

JAXP DocumentBuilderFactory, SAXParserFactory and DOM4J

`DocumentBuilderFactory,` `SAXParserFactory` and `DOM4J` `XML` Parsers can be configured using the same techniques to protect them against XXE.

Only the `DocumentBuilderFactory` example is presented here. The JAXP `DocumentBuilderFactory` setFeature method allows a developer to control which implementation-specific XML processor features are enabled or disabled.

The features can either be set on the factory or the underlying `XMLReader` setFeature method.

Each XML processor implementation has its own features that govern how DTDs and external entities are processed.

For a syntax highlighted example code snippet using `SAXParserFactory`, look here.

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException; // catching unsupported
features

...

DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
String FEATURE = null;
try {
    // This is the PRIMARY defense. If DTDs (doctypes) are disallowed, almost
all
    // XML entity attacks are prevented
    // Xerces 2 only - http://xerces.apache.org/xerces2-
j/features.html#disallow-doctype-decl
    FEATURE = "http://apache.org/xml/features/disallow-doctype-decl";
    dbf.setFeature(FEATURE, true);

    // If you can't completely disable DTDs, then at least do the following:
    // Xerces 1 - http://xerces.apache.org/xerces-j/features.html#external-
general-entities
    // Xerces 2 - http://xerces.apache.org/xerces2-j/features.html#external-
general-entities
    // JDK7+ - http://xml.org/sax/features/external-general-entities
    //This feature has to be used together with the following one, otherwise
it will not protect you from XXE for sure
    FEATURE = "http://xml.org/sax/features/external-general-entities";
    dbf.setFeature(FEATURE, false);

    // Xerces 1 - http://xerces.apache.org/xerces-j/features.html#external-
parameter-entities
    // Xerces 2 - http://xerces.apache.org/xerces2-j/features.html#external-
parameter-entities
```

```
    // JDK7+ - http://xml.org/sax/features/external-parameter-entities
    //This feature has to be used together with the previous one, otherwise it
will not protect you from XXE for sure
    FEATURE = "http://xml.org/sax/features/external-parameter-entities";
    dbf.setFeature(FEATURE, false);

    // Disable external DTDs as well
    FEATURE = "http://apache.org/xml/features/nonvalidating/load-external-
dtd";
    dbf.setFeature(FEATURE, false);

    // and these as well, per Timothy Morgan's 2014 paper: "XML Schema, DTD,
and Entity Attacks"
    dbf.setXIncludeAware(false);
    dbf.setExpandEntityReferences(false);

    // And, per Timothy Morgan: "If for some reason support for inline
DOCTYPEs are a requirement, then
    // ensure the entity settings are disabled (as shown above) and beware
that SSRF attacks
    // (http://cwe.mitre.org/data/definitions/918.html) and denial
    // of service attacks (such as billion laughs or decompression bombs via
"jar:") are a risk."

    // remaining parser logic
    ...
} catch (ParserConfigurationException e) {
    // This should catch a failed setFeature feature
    logger.info("ParserConfigurationException was thrown. The feature '" +
FEATURE
    + "' is probably not supported by your XML processor.");
    ...
} catch (SAXException e) {
    // On Apache, this should be thrown when disallowing DOCTYPE
    logger.warning("A DOCTYPE was passed into the XML document");
    ...
} catch (IOException e) {
    // XXE that points to a file that doesn't exist
    logger.error("IOException occurred, XXE may still possible: " +
e.getMessage());
    ...
}

// Load XML file or stream using a XXE agnostic configured parser...
DocumentBuilder safebuilder = dbf.newDocumentBuilder();
```

Xerces 1 Features:

- Do not include external entities by setting this feature to `false`.
- Do not include parameter entities by setting this feature to `false`.
- Do not include external DTDs by setting this feature to `false`.

Xerces 2 Features:

- Disallow an inline DTD by setting this feature to `true`.

- Do not include external entities by setting this feature to `false`.
- Do not include parameter entities by setting this feature to `false`.
- Do not include external DTDs by setting this feature to `false`.

Note: The above defenses require Java 7 update 67, Java 8 update 20, or above, because the above countermeasures for `DocumentBuilderFactory` and SAXParserFactory are broken in earlier Java versions, per: CVE-2014-6517.

XMLInputFactory (a StAX parser)

StAX parsers such as `XMLInputFactory` allow various properties and features to be set.

To protect a Java `XMLInputFactory` from XXE, do this:

```
// This disables DTDs entirely for that factory
xmlInputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, false);
// disable external entities
xmlInputFactory.setProperty("javax.xml.stream.isSupportingExternalEntities",
false);
```

Oracle DOM Parser

Follow Oracle reomendation e.g.:

```
    // Extend oracle.xml.parser.v2.XMLParser
    DOMParser domParser = new DOMParser();

    // Do not expand entity references
    domParser.setAttribute(DOMParser.EXPAND_ENTITYREF, false);

    // dtdObj is an instance of oracle.xml.parser.v2.DTD
    domParser.setAttribute(DOMParser.DTD_OBJECT, dtdObj);

    // Do not allow more than 11 levels of entity expansion
    domParser.setAttribute(DOMParser.ENTITY_EXPANSION_DEPTH, 12);
```

TransformerFactory

To protect a `javax.xml.transform.TransformerFactory` from XXE, do this:

```
TransformerFactory tf = TransformerFactory.newInstance();
tf.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
tf.setAttribute(XMLConstants.ACCESS_EXTERNAL_STYLESHEET, "");
```

Validator

To protect a `javax.xml.validation.Validator` from XXE, do this:

```
SchemaFactory factory =
SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
Schema schema = factory.newSchema();
Validator validator = schema.newValidator();
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
validator.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
```

SchemaFactory

To protect a `javax.xml.validation.SchemaFactory` from XXE, do this:

```
SchemaFactory factory =
SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
factory.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
factory.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
Schema schema = factory.newSchema(Source);
```

SAXTransformerFactory

To protect a `javax.xml.transform.sax.SAXTransformerFactory` from XXE, do this:

```
SAXTransformerFactory sf = SAXTransformerFactory.newInstance();
sf.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
sf.setAttribute(XMLConstants.ACCESS_EXTERNAL_STYLESHEET, "");
sf.newXMLFilter(Source);
```

Note: Use of the following `XMLConstants` requires JAXP 1.5, which was added to Java in 7u40 and Java 8:

- `javax.xml.XMLConstants.ACCESS_EXTERNAL_DTD`
- `javax.xml.XMLConstants.ACCESS_EXTERNAL_SCHEMA`
- `javax.xml.XMLConstants.ACCESS_EXTERNAL_STYLESHEET`

XMLReader

To protect a Java `org.xml.sax.XMLReader` from XXE, do this:

```
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setFeature("http://apache.org/xml/features/disallow-doctype-decl",
true);
// This may not be strictly required as DTDs shouldn't be allowed at all, per
previous line.
reader.setFeature("http://apache.org/xml/features/nonvalidating/load-external-
dtd", false);
reader.setFeature("http://xml.org/sax/features/external-general-entities",
false);
reader.setFeature("http://xml.org/sax/features/external-parameter-entities",
false);
```

SAXReader

To protect a Java `org.dom4j.io.SAXReader` from XXE, do this:

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

Based on testing, if you are missing one of these, you can still be vulnerable to an XXE attack.

SAXBuilder

To protect a Java `org.jdom2.input.SAXBuilder` from XXE, do this:

```
SAXBuilder builder = new SAXBuilder();
builder.setFeature("http://apache.org/xml/features/disallow-doctype-
decl",true);
builder.setFeature("http://xml.org/sax/features/external-general-entities",
false);
builder.setFeature("http://xml.org/sax/features/external-parameter-entities",
false);
builder.setExpandEntities(false);
Document doc = builder.build(new File(fileName));
```

No-op EntityResolver

For APIs that take an `EntityResolver`, you can neutralize an XML parser's ability to resolve entities by supplying a no-op implementation:

```
public final class NoOpEntityResolver implements EntityResolver {
    public InputSource resolveEntity(String publicId, String systemId) {
        return new InputSource(new StringReader(""));
    }
}

// ...

xmlReader.setEntityResolver(new NoOpEntityResolver());
documentBuilder.setEntityResolver(new NoOpEntityResolver());
```

or more simply:

```
EntityResolver noop = (publicId, systemId) -> new InputSource(new
StringReader(""));
xmlReader.setEntityResolver(noop);
documentBuilder.setEntityResolver(noop);
```

JAXB Unmarshaller

Since a `javax.xml.bind.Unmarshaller` parses XML and does not support any flags for disabling XXE, it's imperative to parse the untrusted XML through a configurable secure parser first, generate a source object as a result, and pass the source object to the Unmarshaller. For example:

```
//Disable XXE
SAXParserFactory spf = SAXParserFactory.newInstance();
spf.setFeature("http://xml.org/sax/features/external-general-entities",
false);
spf.setFeature("http://xml.org/sax/features/external-parameter-entities",
false);
spf.setFeature("http://apache.org/xml/features/nonvalidating/load-external-
dtd", false);

//Do unmarshall operation
Source xmlSource = new SAXSource(spf.newSAXParser().getXMLReader(),
                                 new InputSource(new StringReader(xml)));
JAXBContext jc = JAXBContext.newInstance(Object.class);
Unmarshaller um = jc.createUnmarshaller();
um.unmarshal(xmlSource);
```

XPathExpression

A `javax.xml.xpath.XPathExpression` can not be configured securely by itself, so the untrusted data must be parsed through another securable XML parser first.

For example:

```
DocumentBuilderFactory df = DocumentBuilderFactory.newInstance();
df.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
df.setAttribute(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
DocumentBuilder builder = df.newDocumentBuilder();
String result = new XPathExpression().evaluate( builder.parse(
                            new ByteArrayInputStream(xml.getBytes())) );
```

java.beans.XMLDecoder

The readObject() method in this class is fundamentally unsafe.

Not only is the XML it parses subject to XXE, but the method can be used to construct any Java object, and execute arbitrary code as described here.

And there is no way to make use of this class safe except to trust or properly validate the input being passed into it.

As such, we'd strongly recommend completely avoiding the use of this class and replacing it with a safe or properly configured XML parser as described elsewhere in this cheat sheet.

Other XML Parsers

There are many third-party libraries that parse XML either directly or through their use of other libraries. Please test and verify their XML parser is secure against XXE by default. If the parser is not secure by default, look for flags supported by the parser to disable all possible external resource inclusions like the examples given above. If there's no control exposed to the outside, make sure the untrusted content is passed through a secure parser first and then passed to insecure third-party parser similar to how the Unmarshaller is secured.

Spring Framework MVC/OXM XXE Vulnerabilities

For example, some XXE vulnerabilities were found in Spring OXM and Spring MVC. The following versions of the Spring Framework are vulnerable to XXE:

- **3.0.0** to **3.2.3** (Spring OXM & Spring MVC)
- **4.0.0.M1** (Spring OXM)
- **4.0.0.M1-4.0.0.M2** (Spring MVC)

There were other issues as well that were fixed later, so to fully address these issues, Spring recommends you upgrade to Spring Framework 3.2.8+ or 4.0.2+.

For Spring OXM, this is referring to the use of org.springframework.oxm.jaxb.Jaxb2Marshaller. Note that the CVE for Spring OXM specifically indicates that 2 XML parsing situations are up to the developer to get right, and 2 are the responsibility of Spring and were fixed to address this CVE.

Here's what they say:

Two situations developers must handle:

- For a `DOMSource`, the XML has already been parsed by user code and that code is responsible for protecting against XXE.
- For a `StAXSource`, the XMLStreamReader has already been created by user code and that code is responsible for protecting against XXE.

The issue Spring fixed:

For SAXSource and StreamSource instances, Spring processed external entities by default thereby creating this vulnerability.

Here's an example of using a StreamSource that was vulnerable, but is now safe, if you are using a fixed version of Spring OXM or Spring MVC:

```
import org.springframework.oxm.Jaxb2Marshaller;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;

Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
// Must cast return Object to whatever type you are unmarshalling
marshaller.unmarshal(new StreamSource(new
StringReader(some_string_containing_XML));
```

So, per the Spring OXM CVE writeup, the above is now safe. But if you were to use a DOMSource or StAXSource instead, it would be up to you to configure those sources to be safe from XXE.

Castor

Castor is a data binding framework for Java. It allows conversion between Java objects, XML, and relational tables. The XML features in Castor **prior to version 1.3.3** are vulnerable to XXE, and should be upgraded to the latest version. For additional information, check the official XML configuration file

.NET

The following information for XXE injection in .NET is directly from this web application of unit tests by Dean Fleming.

This web application covers all currently supported .NET XML parsers, and has test cases for each demonstrating when they are safe from XXE injection and when they are not.

Previously, this information was based on James Jardine's excellent .NET XXE article.

It originally provided more recent and more detailed information than the older article from Microsoft on how to prevent XXE and XML Denial of Service in .NET, however, it has some inaccuracies that the web application covers.

The following table lists all supported .NET XML parsers and their default safety levels:

XML Parser	Safe by default?
LINQ to XML	Yes
XmlDictionaryReader	Yes
XmlDocument	
...prior to 4.5.2	No

XML Parser	Safe by default?
...in versions 4.5.2+	Yes
XmlNodeReader	Yes
XmlReader	Yes
XmlTextReader	
...prior to 4.5.2	No
...in versions 4.5.2+	Yes
XPathNavigator	
...prior to 4.5.2	No
...in versions 4.5.2+	Yes
XslCompiledTransform	Yes

LINQ to XML

Both the `XElement` and `XDocument` objects in the `System.Xml.Linq` library are safe from XXE injection by default. `XElement` parses only the elements within the XML file, so DTDs are ignored altogether. `XDocument` has DTDs disabled by default, and is only unsafe if constructed with a different unsafe XML parser.

XmlDictionaryReader

`System.Xml.XmlDictionaryReader` is safe by default, as when it attempts to parse the DTD, the compiler throws an exception saying that "CData elements not valid at top level of an XML document". It becomes unsafe if constructed with a different unsafe XML parser.

XmlDocument

Prior to .NET Framework version 4.5.2, `System.Xml.XmlDocument` is **unsafe** by default. The `XmlDocument` object has an `XmlResolver` object within it that needs to be set to null in versions prior to 4.5.2. In versions 4.5.2 and up, this `XmlResolver` is set to null by default.

The following example shows how it is made safe:

```
 static void LoadXML()
 {
   string xxePayload = "<!DOCTYPE doc [<!ENTITY win SYSTEM
'file:///C:/Users/testdata2.txt'>]>"
                     + "<doc>&win;</doc>";
   string xml = "<?xml version='1.0' ?>" + xxePayload;

   XmlDocument xmlDoc = new XmlDocument();
   // Setting this to NULL disables DTDs - Its NOT null by default.
   xmlDoc.XmlResolver = null;
   xmlDoc.LoadXml(xml);
   Console.WriteLine(xmlDoc.InnerText);
   Console.ReadLine();
 }
```

`XmlDocument` can become unsafe if you create your own nonnull `XmlResolver` with default or unsafe settings. If you need to enable DTD processing, instructions on how to do so safely are described in detail in the referenced MSDN article.

XmlNodeReader

`System.Xml.XmlNodeReader` objects are safe by default and will ignore DTDs even when constructed with an unsafe parser or wrapped in another unsafe parser.

XmlReader

`System.Xml.XmlReader` objects are safe by default.

They are set by default to have their ProhibitDtd property set to false in .NET Framework versions 4.0 and earlier, or their `DtdProcessing` property set to Prohibit in .NET versions 4.0 and later.

Additionally, in .NET versions 4.5.2 and later, the `XmlReaderSettings` belonging to the `XmlReader` has its `XmlResolver` set to null by default, which provides an additional layer of safety.

Therefore, `XmlReader` objects will only become unsafe in version 4.5.2 and up if both the `DtdProcessing` property is set to Parse and the `XmlReaderSetting`'s `XmlResolver` is set to a nonnull XmlResolver with default or unsafe settings. If you need to enable DTD processing, instructions on how to do so safely are described in detail in the referenced MSDN article.

XmlTextReader

`System.Xml.XmlTextReader` is **unsafe** by default in .NET Framework versions prior to 4.5.2. Here is how to make it safe in various .NET versions:

Prior to .NET 4.0

In .NET Framework versions prior to 4.0, DTD parsing behavior for `XmlReader` objects like `XmlTextReader` are controlled by the Boolean `ProhibitDtd` property found in the `System.Xml.XmlReaderSettings` and `System.Xml.XmlTextReader` classes.

Set these values to true to disable inline DTDs completely.

```
XmlTextReader reader = new XmlTextReader(stream);
// NEEDED because the default is FALSE!!
reader.ProhibitDtd = true;
```

.NET 4.0 - .NET 4.5.2

In .NET Framework version 4.0, DTD parsing behavior has been changed. The `ProhibitDtd` property has been deprecated in favor of the new `DtdProcessing` property.

However, they didn't change the default settings so `XmlTextReader` is still vulnerable to XXE by default.

Setting `DtdProcessing` to `Prohibit` causes the runtime to throw an exception if a `<!DOCTYPE>` element is present in the XML.

To set this value yourself, it looks like this:

```
XmlTextReader reader = new XmlTextReader(stream);
// NEEDED because the default is Parse!!
reader.DtdProcessing = DtdProcessing.Prohibit;
```

Alternatively, you can set the `DtdProcessing` property to `Ignore`, which will not throw an exception on encountering a `<!DOCTYPE>` element but will simply skip over it and not process it. Finally, you can set `DtdProcessing` to `Parse` if you do want to allow and process inline DTDs.

.NET 4.5.2 and later

In .NET Framework versions 4.5.2 and up, `XmlTextReader`'s internal `XmlResolver` is set to null by default, making the `XmlTextReader` ignore DTDs by default. The `XmlTextReader` can become unsafe if if you create your own nonnull `XmlResolver` with default or unsafe settings.

XPathNavigator

`System.Xml.XPath.XPathNavigator` is **unsafe** by default in .NET Framework versions prior to 4.5.2.

This is due to the fact that it implements `IXPathNavigable` objects like `XmlDocument`, which are also unsafe by default in versions prior to 4.5.2.

You can make `XPathNavigator` safe by giving it a safe parser like `XmlReader` (which is safe by default) in the `XPathDocument`'s constructor.

Here is an example:

```
XmlReader reader = XmlReader.Create("example.xml");
XPathDocument doc = new XPathDocument(reader);
XPathNavigator nav = doc.CreateNavigator();
string xml = nav.InnerXml.ToString();
```

XslCompiledTransform

`System.Xml.Xsl.XslCompiledTransform` (an XML transformer) is safe by default as long as the parser it's given is safe.

It is safe by default because the default parser of the `Transform()` methods is an `XmlReader`, which is safe by default (per above).

The source code for this method is here.

Some of the `Transform()` methods accept an `XmlReader` or `IXPathNavigable` (e.g., `XmlDocument`) as an input, and if you pass in an unsafe XML Parser then the `Transform` will also be unsafe.

iOS

libxml2

iOS includes the C/C++ libxml2 library described above, so that guidance applies if you are using libxml2 directly.

However, the version of libxml2 provided up through iOS6 is prior to version 2.9 of libxml2 (which protects against XXE by default).

NSXMLDocument

iOS also provides an `NSXMLDocument` type, which is built on top of libxml2.

However, `NSXMLDocument` provides some additional protections against XXE that aren't available in libxml2 directly.

Per the 'NSXMLDocument External Entity Restriction API' section of this page:

- iOS4 and earlier: All external entities are loaded by default.
- iOS5 and later: Only entities that don't require network access are loaded. (which is safer)

However, to completely disable XXE in an `NSXMLDocument` in any version of iOS you simply specify `NSXMLNodeLoadExternalEntitiesNever` when creating the `NSXMLDocument`.

PHP

Per the PHP documentation, the following should be set when using the default PHP XML parser in order to prevent XXE:

```
libxml_disable_entity_loader(true);
```

A description of how to abuse this in PHP is presented in a good SensePost article describing a cool PHP based XXE vulnerability that was fixed in Facebook.

Python

The Python 3 official documentation contains a section on xml vulnerabilities. As of the 1st January 2020 Python 2 is no longer supported, however the Python website still contains some legacy documentation.

The following table gives an overview of various modules in Python 3 used for XML parsing and whether or not they are vulnerable.

Attack Type	sax	etree	minidom	pulldom	xmlrpc
Billion Laughs	Vulnerable	Vulnerable	Vulnerable	Vulnerable	Vulnerab
Quadratic Blowup	Vulnerable	Vulnerable	Vulnerable	Vulnerable	Vulnerab
External Entity Expansion	Safe	Safe	Safe	Safe	Safe
DTD Retrieval	Safe	Safe	Safe	Safe	Safe
Decompression Bomb	Safe	Safe	Safe	Safe	Vulnerab

To protect your application from the applicable attacks, two packages exist to help you sanitize your input and protect your application against DDoS and remote attacks.

Semgrep Rules

Semgrep is a command-line tool for offline static analysis. Use pre-built or custom rules to enforce code and security standards in your codebase.

Java

Below are the rules for different XML parsers in Java

Digester

Identifying XXE vulnerability in the `org.apache.commons.digester3.Digester` library Rule can be played here https://semgrep.dev/s/salecharohit:xxe-Digester

DocumentBuilderFactory

Identifying XXE vulnerability in the `javax.xml.parsers.DocumentBuilderFactory` library Rule can be played here https://semgrep.dev/s/salecharohit:xxe-dbf

SAXBuilder

Identifying XXE vulnerability in the `org.jdom2.input.SAXBuilder` library Rule can be played here https://semgrep.dev/s/salecharohit:xxe-saxbuilder

SAXParserFactory

Identifying XXE vulnerability in the `javax.xml.parsers.SAXParserFactory` library Rule can be played here https://semgrep.dev/s/salecharohit:xxe-SAXParserFactory

SAXReader

Identifying XXE vulnerability in the `org.dom4j.io.SAXReader` library Rule can be played here https://semgrep.dev/s/salecharohit:xxe-SAXReader

XMLInputFactory

Identifying XXE vulnerability in the `javax.xml.stream.XMLInputFactory` library Rule can be played here https://semgrep.dev/s/salecharohit:xxe-XMLInputFactory

XMLReader

Identifying XXE vulnerability in the `org.xml.sax.XMLReader` library Rule can be played here https://semgrep.dev/s/salecharohit:xxe-XMLReader

References

- XXE by InfoSecInstitute
- OWASP Top 10-2017 A4: XML External Entities (XXE)

- Timothy Morgan's 2014 paper: "XML Schema, DTD, and Entity Attacks"
- FindSecBugs XXE Detection
- XXEbugFind Tool
- Testing for XML Injection