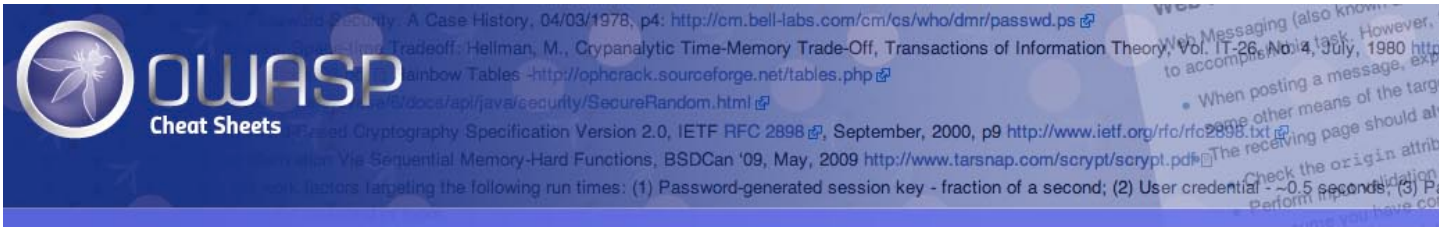
XML External Entity (XXE) Prevention Cheat Sheet

From OWASP



Last revision (mm/dd/yy): **02/6/2017**

Introduction

An *XML External Entity* attack is a type of attack against an application that parses XML input. This attack occurs when **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, 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) Processing.

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 doctypes 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 (http://xmlsoft.org/html/libxml-parser.html#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: https://mail.gnome.org/archives/xml/2012-October/msg00045.html, starting with libxml2 version 2.9, XXE has been disabled by default as committed by the following patch: http://git.gnome.org/browse/libxml2/commit/?id=4629ee02ac649c27f9c0cf98ba017c6b5526070f.

libxerces-c

Use of XercesDOMParser do this to prevent XXE.

```
XercesDOMParser *parser = new XercesDOMParser;
parser->setCreateEntityReferenceNodes(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 (http://docs.oracle.com/javase/7/docs/api/javax/xml/parsers/DocumentBuilderFactory.html#setFeature(java.lang.String,%20boolean)) 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 (http://docs.oracle.com/javase/7/docs/api/org/xml/sax/XMLReader.html#setFeature%28java.lang.String,%20boolean%29) method. Each XML processor implementation has its own features that govern how DTDs and external entities are processed.

For a syntax highlighted code snippet for DocumentBuilderFactory, click here (https://gist.github.com/Prandium/dee14ea650ff7900f2c0).

For a syntax highlighted code snippet for SAXParserFactory, click here (https://gist.github.com/asudhakar02/45e2e6fd8bcdfb4bc3b2).

```
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
      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
      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" (see reference below)
      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());
            ...
        }
```

Xerces 1 (http://xerces.apache.org/xerces-j/) Features (http://xerces.apache.org/xerces-j/features.html):

- Do not include external entities by setting this feature (http://xerces.apache.org/xerces-j/features.html#external-general-entities) to `false`.
- Do not include parameter entities by setting this feature (http://xerces.apache.org/xerces-j/features.html#external-parameter-entities) to `false`.
- Do not include external DTDs by setting this feature (http://xerces.apache.org/xerces-j/features.html#load-external-dtd) to `false`.

Xerces 2 (http://xerces.apache.org/xerces2-j/) Features (http://xerces.apache.org/xerces2-j/features.html):

- Disallow an inline DTD by setting this feature (http://xerces.apache.org/xerces2-j/features.html#disallow-doctype-decl) to `true`.
- Do not include external entities by setting this feature (http://xerces.apache.org/xerces2-j/features.html#external-general-entities) to `false`.
- Do not include parameter entities by setting this feature (http://xerces.apache.org/xerces2-j/features.html#external-parameter-entities) to `false`.
- Do not include external DTDs by setting this feature (http://xerces.apache.org/xerces-j/features.html#load-external-dtd) to `false`.

Note: Please use Java 7 update 67, Java 8 update 20 or above, otherwise the above countermeasures for DocumentBuilderFactory and SAXParserFactory do not work . For details, please refer to CVE-2014-6517[1] (http://www.cvedetails.com/cve/CVE-2014-6517/).

When uses of SAXParser, XMLReader in SAXParserFactory, do the following to avoid XXE

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

StAX and XMLInputFactory

StAX (http://en.wikipedia.org/wiki/StAX) parsers such as XMLInputFactory (http://docs.oracle.com/javase/7/docs/api/javax/xml/stream/XMLInputFactory.html) allow various properties and features to be set.

To protect a Java XMLInputFactory from XXE, do this:

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

TransformerFactory

To protect a Java 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 Java 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 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 Java SAXTransformerFactory from XXE, do this:

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

XMLReader

To protect a Java XMLReader from XXE, do this:

```
XMLReader spf = XMLReaderFactory.createXMLReader();
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  );
```

saxReader

To protect a Java 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);
```

PS. Based on testing, missing one of this can still be vulnerable to XXE attack.

saxBuilder

To protect a Java saxBuilder from XXE, do this:

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

Note: When uses of SAXbuilder, using the followings are not enough to prevent the XXE.

```
.setFeature(" http://xml.org/sax/features/external-general-entities  ", false);
.setFeature(" http://xml.org/sax/features/external-parameter-entities  ", false);
```

Unmarshaller

Since an 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:

```
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);
```

```
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

An XPathExpression is similar to an Unmarshaller where it can't 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, "");
builder = df.newDocumentBuilder();
xPathExpression.evaluate( builder.parse(new ByteArrayInputStream(xml.getBytes())) );
```

java.beans.XMLDecoder

The readObject() (https://docs.oracle.com/javase/8/docs/api/java/beans/XMLDecoder.html#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 (http://stackoverflow.com/questions/14307442/is-it-safe-to-use-xmldecoder-to-read-document-files). 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 3rd 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 3rd 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 (http://pivotal.io/security/cve-2013-4152) and Spring MVC (http://pivotal.io/security/cve-2013-7315). 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:

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

So, per the Spring OXM CVE writeup (http://pivotal.io/security/cve-2013-4152), 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.

.NET

The following information for XXE in .NET is directly from James Jardine's excellent .NET XXE article: https://www.jardinesoftware.net/2016/05/26/xxe-and-net/. This newer article provides more recent and more detailed information than the older article from Microsoft on how to prevent XXE and XML Denial of Service in .NET: http://msdn.microsoft.com/en-us/magazine/ee335713.aspx.

XML Object	Safe by Default?
XMLReader	
Prior to 4.0	Yes
4.0 +	Yes
XMLTextReader	
Prior to 4.0	No
4.0 +	No
XMLDocument	
Prior to 4.6	No
4.6 +	Yes

Prior to .NET 4.0

In .NET Framework versions prior to 4.0, DTD parsing behavior for XmlReader and XmlTextReader is 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:

XmlReader:

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.ProhibitDtd = true; // Not explicitly needed because the default is 'true'
XmlReader reader = XmlReader.Create(stream, settings);
```

XmlTextReader:

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

XmlDocumentReader:

XmlDocumentReader doesn't use a ProhibitDtd property. Instead you have to set its XmlResolver to null.

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

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

.NET 4.0 and later

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:

```
XmlReaderSettings settings = new XmlReaderSettings();
settings.DtdProcessing = DtdProcessing.Prohibit;
XmlReader reader = XmlReader.Create(stream, settings);
```

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 4.6 and later

Starting with .NET 4.6, Microsoft finally changed the default behavior of XmlDocument to be safe from XXE by default, by setting the XmlResolver to null.

For more details on all of this, please read James Jardine's article (https://www.jardinesoftware.net/2016/05/26/xxe-and-net/).

If you need to enable DTD processing, instructions on how to do so safely are described in detail in the referenced MSDN article (http://msdn.microsoft.com/en-us/magazine/ee335713.aspx).

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: http://developer.apple.com/library/ios/#releasenotes/Foundation/RN-Foundation-iOS/Foundation_iOS5.html:

- 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 (http://php.net/manual/en/function.libxml-disable-entity-loader.php), 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 (https://www.sensepost.com/blog/2014/revisting-xxe-and-abusing-protocols/) describing a cool PHP based XXE vulnerability that was fixed in Facebook.

Reference

- FindBugs [2] (https://find-sec-bugs.github.io/bugs.htm#XXE_SAXPARSER)
- XXEbugFind Tool [3] (https://github.com/ssexxe/XXEBugFind)
- Testing for XML injection [4] (https://www.owasp.org/index.php/Testing_for_XML_Injection_(OTG-INPVAL-008))

Authors and Primary Editors

Dave Wichers - dave.wichers[at]owasp.org
Xiaoran Wang - xiaoran[at]attacker-domain.com
James Jardine - james[at]jardinesoftware.com
Tony Hsu (Hsiang-Chih)

Other Cheatsheets

V - T - E (https://www.owasp.org/index.php?title=XML_External_Entity_(XXE)_Prevention_Cheat_Sheet&action=edit)	**Cheat Sheets** [Collapse]
Developer / Builder	3rd Party Javascript Management · Access Control · AJAX Security Cheat Sheet · Authentication (ES) · Bean Validation Cheat Sheet · Choosing and Using Security Questions · Clickjacking Defense · C-Based Toolchain Hardening · Credential Stuffing Prevention Cheat Sheet · Cross-Site Request Forgery (CSRF) Prevention · Cryptographic Storage · Deserialization · DOM based XSS Prevention · Forgot Password · HTML5 Security · HTTP Strict Transport Security · Injection Prevention Cheat Sheet · Injection Prevention Cheat Sheet in Java · JSON Web Token (JWT) Cheat Sheet for Java · Input Validation · JAAS · LDAP Injection Prevention · Logging · Mass Assignment Cheat Sheet · .NET Security · OWASP Top Ten · Password Storage · Pinning · Query Parameterization · Ruby on Rails · REST Security · Session Management · SAML Security · SQL Injection Prevention · Transaction Authorization · Transport Layer Protection · Unvalidated Redirects and Forwards · User Privacy Protection · Web Service Security · XSS (Cross Site Scripting) Prevention · **XML External Entity (XXE) Prevention Cheat Sheet** · XML Security Cheat Sheet
Assessment / Breaker	Attack Surface Analysis · XSS Filter Evasion · REST Assessment · Web Application Security Testing
Mobile	Android Testing · IOS Developer · Mobile Jailbreaking
OpSec / Defender	Virtual Patching
Draft and Beta	Application Security Architecture · Business Logic Security · Command Injection Defense Cheat Sheet · Denial of Service Cheat Sheet · PHP Security · Regular Expression Security Cheatsheet · Secure Coding · Secure SDLC · Threat Modeling · Grails Secure Code Review · IOS Application Security Testing · Key Management · Insecure Direct Object Reference Prevention · Content Security Policy
All Pages In This Category	

Retrieved from "https://www.owasp.org/index.php?title=XML_External_Entity_(XXE)_Prevention_Cheat_Sheet&oldid=226036"

Category: Cheatsheets