

[Help](#)[Donate](#)[Log in](#)[Register](#)

defusedxml 0.5.0



Newer version
available (0.6.0)

```
pip install defusedxml==0.5.0
```




Last released: Feb 9,
2017

XML bomb protection for Python stdlib modules

Navigation

 [Project
description](#)

 [Release
history](#)

 [Download
files](#)

Project links

 [Homepage](#)

 [Download](#)

Statistics

Project description

“It’s just XML, what could probably go wrong?”

Christian Heimes <christian@python.org 

Synopsis

The results of an attack on a vulnerable XML library can be fairly dramatic. With just a few hundred **Bytes** of XML data an attacker can occupy several **Gigabytes** of memory within **seconds**. An attacker can also keep CPUs busy for a long time with a small to medium size request. Under some circumstances it is even possible to access local files on your server, to circumvent a firewall, or to abuse services to rebound attacks to third parties.

The attacks use and abuse less common features of XML and its parsers. The majority of developers are unacquainted with features

GitHub statistics:

★ **Stars: 126** [↗](#)

🔗 **Forks: 25** [↗](#)

🔔 **Open
issues/PRs: 8** [↗](#)

View statistics for this project via

[Libraries.io](#) [↗](#), or by

using [Google](#)

[BigQuery](#) [↗](#)

Meta

License: Python
Software Foundation
License (PSFL)

Author: [Christian
Heimes](#) [✉](#)

📁 xml, bomb, DoS

Maintainers



tiran

Classifiers

Development Status

5 -

[Production/Stable](#)

Intended Audience

[Developers](#)

License

[OSI Approved ::](#)

[Python Software](#)

such as processing instructions and entity expansions that XML inherited from SGML. At best they know about `<!DOCTYPE>` from experience with HTML but they are not aware that a document type definition (DTD) can generate an HTTP request or load a file from the file system.

None of the issues is new. They have been known for a long time. Billion laughs was first reported in 2003. Nevertheless some XML libraries and applications are still vulnerable and even heavy users of XML are surprised by these features. It's hard to say whom to blame for the situation. It's too short sighted to shift all blame on XML parsers and XML libraries for using insecure default settings. After all they properly implement XML specifications. Application developers must not rely that a library is always configured for security and potential harmful data by default.

Table of Contents

- [Synopsis](#)
- [Attack vectors](#)
 - [billion laughs / exponential entity expansion](#)
 - [quadratic blowup entity expansion](#)
 - [external entity expansion \(remote\)](#)
 - [external entity expansion \(local file\)](#)
 - [DTD retrieval](#)
- [Python XML Libraries](#)
 - [Settings in standard library](#)
- [defusedxml](#)
 - [defusedxml \(package\)](#)
 - [defusedxml.cElementTree](#)
 - [defusedxml.ElementTree](#)
 - [defusedxml.expatreader](#)
 - [defusedxml.sax](#)
 - [defusedxml.expatbuilder](#)
 - [defusedxml.minidom](#)
 - [defusedxml.pulldom](#)
 - [defusedxml.xmlrpc](#)

- Foundation License
 - Natural Language
 - English
 - Programming Language
 - Python
 - Python :: 2
 - Python :: 2.7
 - Python :: 3
 - Python :: 3.4
 - Python :: 3.5
 - Python :: 3.6
 - Topic
 - Text Processing :: Markup :: XML
- [defusedxml.lxml](#)
 - [defusedexpat](#)
 - [Modifications in expat](#)
 - [How to avoid XML vulnerabilities](#)
 - [Other things to consider](#)
 - [attribute blowup / hash collision attack](#)
 - [decompression bomb](#)
 - [Processing Instruction](#)
 - [Other DTD features](#)
 - [XPath](#)
 - [XPath injection attacks](#)
 - [XInclude](#)
 - [XMLSchema location](#)
 - [XSL Transformation](#)
 - [Related CVEs](#)
 - [Other languages / frameworks](#)
 - [Perl](#)
 - [Ruby](#)
 - [PHP](#)
 - [C# / .NET / Mono](#)
 - [Java](#)
 - [TODO](#)
 - [License](#)
 - [Acknowledgements](#)
 - [References](#)
 - [Changelog](#)
 - [defusedxml 0.5.0](#)
 - [defusedxml 0.5.0.rc1](#)
 - [defusedxml 0.4.1](#)
 - [defusedxml 0.4](#)
 - [defusedxml 0.3](#)
 - [defusedxml 0.2](#)
 - [defusedxml 0.1](#)

Attack vectors

billion laughs / exponential entity expansion

The [Billion Laughs](#) attack – also known as exponential entity expansion – uses multiple levels of nested entities. The original example uses 9 levels of 10 expansions in each level to expand the string `lol` to a string of $3 * 10^9$ bytes, hence the name “billion laughs”. The resulting string occupies 3 GB (2.79 GiB) of memory; intermediate strings require additional memory. Because most parsers don’t cache the intermediate step for every expansion it is repeated over and over again. It increases the CPU load even more.

An XML document of just a few hundred bytes can disrupt all services on a machine within seconds.

Example XML:

```
<!DOCTYPE xmlbomb [  
<!ENTITY a "1234567890" >  
<!ENTITY b "&a;&a;&a;&a;&a;&a;&a;&a;">  
<!ENTITY c "&b;&b;&b;&b;&b;&b;&b;&b;">  
<!ENTITY d "&c;&c;&c;&c;&c;&c;&c;&c;">  
>  
<bomb>&d;</bomb>
```

quadratic blowup entity expansion

A quadratic blowup attack is similar to a [Billion Laughs](#) attack; it abuses entity expansion, too. Instead of nested entities it repeats one large entity with a couple of thousand chars over and over again. The attack isn’t as efficient as the exponential case but it avoids triggering countermeasures of parsers against heavily nested entities. Some parsers limit the depth and breadth of a single entity but not the total amount of expanded text throughout an entire XML document.

A medium-sized XML document with a couple of hundred kilobytes can require a couple of hundred MB to several GB of memory. When the attack is combined with some level of nested expansion an attacker is able to achieve a higher ratio of success.

```
<!DOCTYPE bomb [  
<!ENTITY a "xxxxxxx... a couple of ten thousand chars  
>  
<bomb>&a;&a;&a;... repeat</bomb>
```

external entity expansion (remote)

Entity declarations can contain more than just text for replacement. They can also point to external resources by public identifiers or system identifiers. System identifiers are standard URIs. When the URI is a URL (e.g. a `http://` locator) some parsers download the resource from the remote location and embed them into the XML document verbatim.

Simple example of a parsed external entity:

```
<!DOCTYPE external [  
<!ENTITY ee SYSTEM "http://www.python.org/some.xml">  
>  
<root>&ee;</root>
```

The case of parsed external entities works only for valid XML content. The XML standard also supports unparsed external entities with a `NData declaration`.

External entity expansion opens the door to plenty of exploits. An attacker can abuse a vulnerable XML library and application to rebound and forward network requests with the IP address of the server. It highly depends on the parser and the application what kind of exploit is possible. For example:

- An attacker can circumvent firewalls and gain access to restricted resources as all the requests are made from an

internal and trustworthy IP address, not from the outside.

- An attacker can abuse a service to attack, spy on or DoS your servers but also third party services. The attack is disguised with the IP address of the server and the attacker is able to utilize the high bandwidth of a big machine.
- An attacker can exhaust additional resources on the machine, e.g. with requests to a service that doesn't respond or responds with very large files.
- An attacker may gain knowledge, when, how often and from which IP address a XML document is accessed.
- An attacker could send mail from inside your network if the URL handler supports `smtp://` URIs.

external entity expansion (local file)

External entities with references to local files are a sub-case of external entity expansion. It's listed as an extra attack because it deserves extra attention. Some XML libraries such as lxml disable network access by default but still allow entity expansion with local file access by default. Local files are either referenced with a `file://` URL or by a file path (either relative or absolute).

An attacker may be able to access and download all files that can be read by the application process. This may include critical configuration files, too.

```
<!DOCTYPE external [
<!ENTITY ee SYSTEM "file:///PATH/TO/simple.xml">
]>
<root>&ee;</root>
```

DTD retrieval

This case is similar to external entity expansion, too. Some XML libraries like Python's xml.dom.pulldom retrieve document type definitions from remote or local locations. Several attack scenarios from the external entity case apply to this issue as well.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional
<html>
  <head/>
  <body>text</body>
</html>
```

Python XML Libraries

vulnerabilities and features

kind	sax	etree	minidom	pulldom	xmllrpc	lxml	genshi
billion laughs	True	True	True	True	True	False (1)	False (5)
quadratic blowup	True	True	True	True	True	True	False (5)
external entity expansion (remote)	True	False (3)	False (4)	True	false	False (1)	False (5)
external entity expansion (local file)	True	False (3)	False (4)	True	false	True	False (5)
DTD retrieval	True	False	False	True	false	False (1)	False
gzip bomb	False	False	False	False	True	partly (2)	False
xpath support (7)	False	False	False	False	False	True	False
xsl(t) support (7)	False	False	False	False	False	True	False

kind	sax	etree	minidom	pulldom	xmlrpc	lxml	genshi
xinclude support (7)	False	True (6)	False	False	False	True (6)	True
C library	expat	expat	expat	expat	expat	libxml2	expat

1. Lxml is protected against billion laughs attacks and doesn't do network lookups by default.
2. libxml2 and lxml are not directly vulnerable to gzip decompression bombs but they don't protect you against them either.
3. xml.etree doesn't expand entities and raises a ParserError when an entity occurs.
4. minidom doesn't expand entities and simply returns the unexpanded entity verbatim.
5. genshi.input of genshi 0.6 doesn't support entity expansion and raises a ParserError when an entity occurs.
6. Library has (limited) XInclude support but requires an additional step to process inclusion.
7. These are features but they may introduce exploitable holes, see [Other things to consider](#)

Settings in standard library

xml.sax.handler Features

`feature_external_ges` (<http://xml.org/sax/features/external-general-entities>)

disables external entity expansion

`feature_external_pes` (<http://xml.org/sax/features/external-parameter-entities>)

the option is ignored and doesn't modify any functionality

DOM `xml.dom.xmlbuilder.Options`

`external_parameter_entities`

ignored

external_general_entities

ignored

external_dtd_subset

ignored

entities

unsure

defusedxml

The [defusedxml package](#) ([defusedxml on PyPI](#)) contains several Python-only workarounds and fixes for denial of service and other vulnerabilities in Python's XML libraries. In order to benefit from the protection you just have to import and use the listed functions / classes from the right defusedxml module instead of the original module. Merely [defusedxml.xmlrpc](#) is implemented as monkey patch.

Instead of:

```
>>> from xml.etree.ElementTree import parse
>>> et = parse(xmlfile)
```

alter code to:

```
>>> from defusedxml.ElementTree import parse
>>> et = parse(xmlfile)
```

Additionally the package has an **untested** function to monkey patch all stdlib modules with `defusedxml.defuse_stdlib()`.

All functions and parser classes accept three additional keyword arguments. They return either the same objects as the original functions or compatible subclasses.

forbid_dtd (default: False)

disallow XML with a `<!DOCTYPE>` processing instruction and raise a *DTDForbidden* exception when a DTD processing instruction is found.

forbid_entities (default: True)

disallow XML with `<!ENTITY>` declarations inside the DTD and raise an *EntitiesForbidden* exception when an entity is declared.

forbid_external (default: True)

disallow any access to remote or local resources in external entities or DTD and raising an *ExternalReferenceForbidden* exception when a DTD or entity references an external resource.

defusedxml (package)

DefusedXmlException, DTDForbidden, EntitiesForbidden, ExternalReferenceForbidden, NotSupportedError

defuse_stdlib() (*experimental*)

defusedxml.cElementTree

parse(), iterparse(), fromstring(), XMLParser

defusedxml.ElementTree

parse(), iterparse(), fromstring(), XMLParser

defusedxml.expatreader

create_parser(), DefusedExpatParser

defusedxml.sax

parse(), parseString(), create_parser()

defusedxml.expatbuilder

parse(), parseString(), DefusedExpatBuilder,
DefusedExpatBuilderNS

defusedxml.minidom

parse(), parseString()

defusedxml.pulldom

parse(), parseString()

defusedxml.xmlrpc

The fix is implemented as monkey patch for the stdlib's xmlrpc package (3.x) or xmlrpclib module (2.x). The function *monkey_patch()* enables the fixes, *unmonkey_patch()* removes the patch and puts the code in its former state.

The monkey patch protects against XML related attacks as well as decompression bombs and excessively large requests or responses. The default setting is 30 MB for requests, responses and gzip decompression. You can modify the default by changing the module variable *MAX_DATA*. A value of *-1* disables the limit.

defusedxml.lxml

The module acts as an *example* how you could protect code that uses lxml.etree. It implements a custom Element class that filters out Entity instances, a custom parser factory and a thread local storage for parser instances. It also has a *check_docinfo()* function which inspects a tree for internal or external DTDs and entity declarations. In order to check for entities lxml > 3.0 is required.

parse(), fromstring() RestrictedElement, GlobalParserTLS,
getDefaultParser(), check_docinfo()

defusedexpat

The `defusedexpat` package ([defusedexpat on PyPI](#)) comes with binary extensions and a `modified expat` library instead of the standard `expat parser`. It's basically a stand-alone version of the patches for Python's standard library C extensions.

Modifications in expat

new definitions:

```
XML_BOMB_PROTECTION
XML_DEFAULT_MAX_ENTITY_INDIRECTIONS
XML_DEFAULT_MAX_ENTITY_EXPANSIONS
XML_DEFAULT_RESET_DTD
```

new XML_FeatureEnum members:

```
XML_FEATURE_MAX_ENTITY_INDIRECTIONS
XML_FEATURE_MAX_ENTITY_EXPANSIONS
XML_FEATURE_IGNORE_DTD
```

new XML_Error members:

```
XML_ERROR_ENTITY_INDIRECTIONS
XML_ERROR_ENTITY_EXPANSION
```

new API functions:

```
int XML_GetFeature(XML_Parser parser,
                  enum XML_FeatureEnum feature,
                  long *value);
int XML_SetFeature(XML_Parser parser,
                  enum XML_FeatureEnum feature,
                  long value);
int XML_GetFeatureDefault(enum XML_FeatureEnum featur
```

```
        long *value);  
int XML_SetFeatureDefault(enum XML_FeatureEnum featur  
        long value);
```

XML_FEATURE_MAX_ENTITY_INDIRECTIONS

Limit the amount of indirections that are allowed to occur during the expansion of a nested entity. A counter starts when an entity reference is encountered. It resets after the entity is fully expanded. The limit protects the parser against exponential entity expansion attacks (aka billion laughs attack). When the limit is exceeded the parser stops and fails with *XML_ERROR_ENTITY_INDIRECTIONS*. A value of 0 disables the protection.

Supported range

0 .. UINT_MAX

Default

40

XML_FEATURE_MAX_ENTITY_EXPANSIONS

Limit the total length of all entity expansions throughout the entire document. The lengths of all entities are accumulated in a parser variable. The setting protects against quadratic blowup attacks (lots of expansions of a large entity declaration). When the sum of all entities exceeds the limit, the parser stops and fails with *XML_ERROR_ENTITY_EXPANSION*. A value of 0 disables the protection.

Supported range

0 .. UINT_MAX

Default

8 MiB

XML_FEATURE_RESET_DTD

Reset all DTD information after the `<!DOCTYPE>` block has been parsed. When the flag is set (default: false) all DTD information after the `endDoctypeDeclHandler` has been called. The flag can be set inside the `endDoctypeDeclHandler`. Without DTD

information any entity reference in the document body leads to *XML_ERROR_UNDEFINED_ENTITY*.

Supported range

0, 1

Default

0

How to avoid XML vulnerabilities

Best practices

- Don't allow DTDs
- Don't expand entities
- Don't resolve externals
- Limit parse depth
- Limit total input size
- Limit parse time
- Favor a SAX or iterparse-like parser for potential large data
- Validate and properly quote arguments to XSL transformations and XPath queries
- Don't use XPath expression from untrusted sources
- Don't apply XSL transformations that come untrusted sources

(based on Brad Hill's [Attacking XML Security](#))

Other things to consider

XML, XML parsers and processing libraries have more features and possible issue that could lead to DoS vulnerabilities or security exploits in applications. I have compiled an incomplete list of theoretical issues that need further research and more attention. The list is deliberately pessimistic and a bit paranoid, too. It contains things that might go wrong under daffy circumstances.

[attribute blowup / hash collision attack](#)

XML parsers may use an algorithm with quadratic runtime $O(n^2)$ to handle attributes and namespaces. If it uses hash tables (dictionaries) to store attributes and namespaces the implementation may be vulnerable to hash collision attacks, thus reducing the performance to $O(n^2)$ again. In either case an attacker is able to forge a denial of service attack with an XML document that contains thousands upon thousands of attributes in a single node.

I haven't researched yet if expat, pyexpat or libxml2 are vulnerable.

decompression bomb

The issue of decompression bombs (aka [ZIP bomb](#)) apply to all XML libraries that can parse compressed XML stream like gzipped HTTP streams or LZMA-ed files. For an attacker it can reduce the amount of transmitted data by three magnitudes or more. Gzip is able to compress 1 GiB zeros to roughly 1 MB, lzma is even better:

```
$ dd if=/dev/zero bs=1M count=1024 | gzip > zeros.gz
$ dd if=/dev/zero bs=1M count=1024 | lzma -z > zeros.lzma
$ ls -sh zeros.*
1020K zeros.gz
148K zeros.lzma
```

None of Python's standard XML libraries decompress streams except for `xmlrpclib`. The module is vulnerable [to decompression bombs](http://bugs.python.org/issue16043).

lxml can load and process compressed data through libxml2 transparently. libxml2 can handle even very large blobs of compressed data efficiently without using too much memory. But it doesn't protect applications from decompression bombs. A carefully written SAX or iterparse-like approach can be safe.

Processing Instruction

PI's like:

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

may impose more threats for XML processing. It depends if and how a processor handles processing instructions. The issue of URL retrieval with network or local file access apply to processing instructions, too.

Other DTD features

DTD has more features like `<!NOTATION>`. I haven't researched how these features may be a security threat.

XPath

XPath statements may introduce DoS vulnerabilities. Code should never execute queries from untrusted sources. An attacker may also be able to create a XML document that makes certain XPath queries costly or resource hungry.

XPath injection attacks

XPath injection attacks pretty much work like SQL injection attacks. Arguments to XPath queries must be quoted and validated properly, especially when they are taken from the user. The page [Avoid the dangers of XPath injection](#) list some ramifications of XPath injections.

Python's standard library doesn't have XPath support. Lxml supports parameterized XPath queries which does proper quoting. You just have to use its `xpath()` method correctly:

```
# DON'T
>>> tree.xpath("/tag[@id='%s']" % value)

# instead do
>>> tree.xpath("/tag[@id=$tagid]", tagid=name)
```


XInclude

[XML Inclusion](#) is another way to load and include external files:

```
<root xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="filename.txt" parse="text" />
</root>
```

This feature should be disabled when XML files from an untrusted source are processed. Some Python XML libraries and libxml2 support XInclude but don't have an option to sandbox inclusion and limit it to allowed directories.

XMLSchema location

A validating XML parser may download schema files from the information in a `xsi:schemaLocation` attribute.

```
<ead xmlns="urn:isbn:1-931666-22-9"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst"
      xsi:schemaLocation="urn:isbn:1-931666-22-9 http:
</ead>
```

XSL Transformation

You should keep in mind that XSLT is a Turing complete language. Never process XSLT code from unknown or untrusted source! XSLT processors may allow you to interact with external resources in ways you can't even imagine. Some processors even support extensions that allow read/write access to file system, access to JRE objects or scripting with Jython.

Example from [Attacking XML Security](#) for Xalan-J:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
xmlns:rt="http://xml.apache.org/xalan/java/java.lang
xmlns:ob="http://xml.apache.org/xalan/java/java.lang
exclude-result-prefixes= "rt ob">
<xsl:template match="/">
  <xsl:variable name="runtimeObject" select="rt:getR
  <xsl:variable name="command"
    select="rt:exec($runtimeObject, &apos;c:\Windows
  <xsl:variable name="commandAsString" select="ob:to
  <xsl:value-of select="$commandAsString"/>
</xsl:template>
</xsl:stylesheet>
```

Related CVEs

CVE-2013-1664

Unrestricted entity expansion induces DoS vulnerabilities in Python XML libraries (XML bomb)

CVE-2013-1665

External entity expansion in Python XML libraries inflicts potential security flaws and DoS vulnerabilities

Other languages / frameworks

Several other programming languages and frameworks are vulnerable as well. A couple of them are affected by the fact that libxml2 up to 2.9.0 has no protection against quadratic blowup attacks. Most of them have potential dangerous default settings for entity expansion and external entities, too.

Perl

Perl's XML::Simple is vulnerable to quadratic entity expansion and external entity expansion (both local and remote).

Ruby

Ruby's REXML document parser is vulnerable to entity expansion attacks (both quadratic and exponential) but it doesn't do external

entity expansion by default. In order to counteract entity expansion you have to disable the feature:

```
REXML::Document.entity_expansion_limit = 0
```

libxml-ruby and hpricot don't expand entities in their default configuration.

PHP

PHP's SimpleXML API is vulnerable to quadratic entity expansion and loads entities from local and remote resources. The option `LIBXML_NONET` disables network access but still allows local file access. `LIBXML_NOENT` seems to have no effect on entity expansion in PHP 5.4.6.

C# / .NET / Mono

Information in [XML DoS and Defenses \(MSDN\)](#) suggest that .NET is vulnerable with its default settings. The article contains code snippets how to create a secure XML reader:

```
XmlReaderSettings settings = new XmlReaderSettings();  
settings.ProhibitDtd = false;  
settings.MaxCharactersFromEntities = 1024;  
settings.XmlResolver = null;  
XmlReader reader = XmlReader.Create(stream, settings)
```

Java

Untested. The documentation of Xerces and its [Xerces SecurityManager](#) sounds like Xerces is also vulnerable to billion laugh attacks with its default settings. It also does entity resolving when an `org.xml.sax.EntityResolver` is configured. I'm not yet sure about the default setting here.

Java specialists suggest to have a custom builder factory:

```
DocumentBuilderFactory builderFactory = DocumentBuild
builderFactory.setXIncludeAware(False);
builderFactory.setExpandEntityReferences(False);
builderFactory.setFeature(XMLConstants.FEATURE_SECURE
# either
builderFactory.setFeature("http://apache.org/xml/feat
# or if you need DTDs
builderFactory.setFeature("http://xml.org/sax/feature
builderFactory.setFeature("http://xml.org/sax/feature
builderFactory.setFeature("http://apache.org/xml/feat
builderFactory.setFeature("http://apache.org/xml/feat
```

TODO

- DOM: Use xml.dom.xmlbuilder options for entity handling
- SAX: take feature_external_ges and feature_external_pes (?) into account
- test experimental monkey patching of stdlib modules
- improve documentation

License

Copyright (c) 2013-2017 by Christian Heimes

<christian@python.org  >

Licensed to PSF under a Contributor Agreement.

See <http://www.python.org/psf/license> for licensing details.

Acknowledgements

Brett Cannon (Python Core developer)

review and code cleanup

Antoine Pitrou (Python Core developer)

code review

Aaron Patterson, Ben Murphy and Michael Koziarski (Ruby community)

Many thanks to Aaron, Ben and Michael from the Ruby community for their report and assistance.

Thierry Carrez (OpenStack)

Many thanks to Thierry for his report to the Python Security Response Team on behalf of the OpenStack security team.

Carl Meyer (Django)

Many thanks to Carl for his report to PSRT on behalf of the Django security team.

Daniel Veillard (libxml2)

Many thanks to Daniel for his insight and assistance with libxml2.

semantics GmbH (<http://www.semantics.de/>)

Many thanks to my employer semantics for letting me work on the issue during working hours as part of semantics's open source initiative.

References

- [XML DoS and Defenses \(MSDN\)](#)
- [Billion Laughs on Wikipedia](#)
- [ZIP bomb on Wikipedia](#)
- [Configure SAX parsers for secure processing](#)
- [Testing for XML Injection](#)

Changelog

defusedxml 0.5.0

Release date: 07-Feb-2017

- No changes

defusedxml 0.5.0.rc1

Release date: 28-Jan-2017

- Add compatibility with Python 3.6
- Drop support for Python 2.6, 3.1, 3.2, 3.3
- Fix lxml tests (XMLSyntaxError: Detected an entity reference loop)

defusedxml 0.4.1

Release date: 28-Mar-2013

- Add more demo exploits, e.g. python_external.py and Xalan XSLT demos.
- Improved documentation.

defusedxml 0.4

Release date: 25-Feb-2013

- As per <http://seclists.org/oss-sec/2013/q1/340> please REJECT CVE-2013-0278, CVE-2013-0279 and CVE-2013-0280 and use CVE-2013-1664, CVE-2013-1665 for OpenStack/etc.
- Add missing parser_list argument to sax.make_parser(). The argument is ignored, though. (thanks to Florian Apolloner)
- Add demo exploit for external entity attack on Python's SAX parser, XML-RPC and WebDAV.

defusedxml 0.3

Release date: 19-Feb-2013

- Improve documentation

defusedxml 0.2

Release date: 15-Feb-2013

- Rename ExternalEntitiesForbidden to ExternalReferenceForbidden
- Rename defusedxml.lxml.check_dtd() to check_docinfo()
- Unify argument names in callbacks
- Add arguments and formatted representation to exceptions
- Add forbid_external argument to all functions and classs
- More tests
- LOTS of documentation
- Add example code for other languages (Ruby, Perl, PHP) and parsers (Genshi)
- Add protection against XML and gzip attacks to xmlrpclib

defusedxml 0.1

Release date: 08-Feb-2013

- Initial and internal release for PSRT review



Help

- [Installing packages](#)
- [Uploading packages](#)
- [User guide](#)
- [FAQs](#)

About PyPI

- [PyPI on Twitter](#)
- [Infrastructure dashboard](#)
- [Package index name retention](#)
- [Our sponsors](#)

Contributing to PyPI

- [Bugs and feedback](#)
- [Contribute on GitHub](#)

Using PyPI

- [Code of conduct](#)
- [Report security issue](#)

[Development credits](#) 

[Privacy policy](#) 

[Terms of use](#)

Status: All Systems Operational 

Developed and maintained by the Python community, for the Python community.
Donate today!

© 2019 Python Software Foundation 

Desktop version

Elastic
Search

Pingdom
Monitoring

Google
BigQuery

Sentry
Error logging

AWS
Cloud computing

DataDog
Monitoring

Fastly
CDN

SignalFx
Supporter

DigiCert
EV certificate

StatusPage
Status page