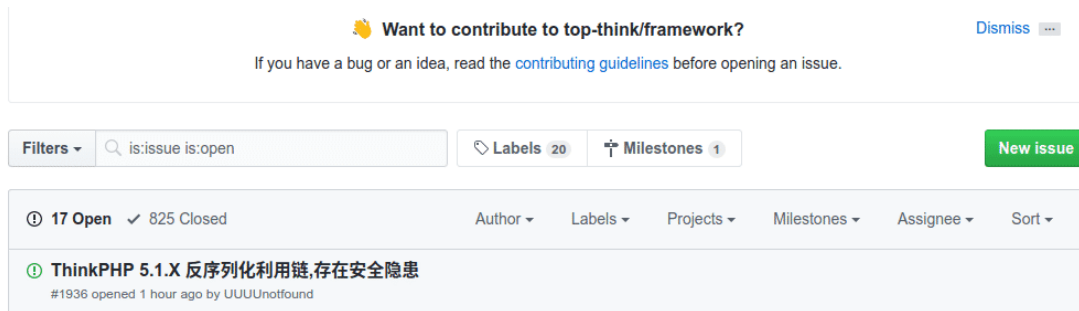


2019年7月25日，在 ThinkPHP 官方 github 上有人提交了这个 issue，遂想一探究竟。(鉴于该POP链已大范围公开，这里便公开之前写的分析文章)



## 环境搭建

```
→ composer create-project --prefer-dist topthink/think tp5137
→ cd tp5137
→ vim composer.json # 将"topthink/framework": "5.1.*"改为"topthink/framework": "5.1.37"
→ composer update
```

将 application/index/controller/Index.php 代码修改成如下：

```
<?php
namespace app\index\controller;

class Index
{
    public function index()
    {
        $u = unserialize($_GET['c']);
        return 'hhh';
    }

    public function hello($name = 'ThinkPHP5')
    {
        return 'hello,' . $name;
    }
}
```

## 利用条件

有一个内容完全可控的反序列化点，例如：`unserialize(■■■■■)`

存在文件上传、文件名完全可控、使用了文件操作函数，例如：`file_exists('phar://■■■■■')`

(满足以上任意一个条件即可)

## 漏洞链

这个漏洞个人认为比较有意思的是：通过 `file_exists` 函数触发类的 `__toString` 方法。下面，我们具体分析一下整个漏洞攻击链。

在 `think\process\pipes\Windows` 类的 `__destruct` 方法中，存在一个删除文件功能，而这里的文件名 `$filename` 变量是可控。如果我们将一个类赋值给 `$filename` 变量，那么在 `file_exists($filename)` 的时候，就会触发这个类的 `__toString` 方法。因为 `file_exists` 函数需要的是一个字符串类型的参数，如果传入一个对象，就会先调用该类 `__toString` 方法，将其转换成字符串，然后再判断。

```

56     public function __destruct()
57     {
58         $this->close();
59         $this->removeFiles();
60     }
137 public function close()
138 {
139     parent::close();
140     foreach ($this->fileHandles as $handle) {
141         fclose($handle);
142     }
143     $this->fileHandles = [];
144 }
160 private function removeFiles()
161 {
162     foreach ($this->files as $filename) {
163         if (file_exists($filename)) {
164             @unlink($filename);
165         }
166     }
167     $this->files = [];
168 }

```

先知社区

接下来，我们就来寻找可利用的 `__toString` 方法。全局搜索到的 `__toString` 方法其实不多，这里有两处都可以利用。它们的区别在于利用 `think\Collection` 构造的链要多构造一步，我们这里只分析链较短的 `think\model\concern\Conversion`。

Find in Path

Match case Words Regex File mask: \*.php

5 matches in 5 files

In Project	Module	Directory	Scope
public function __toString()			Input 448
public function __toString()			Expression 44
public function __toString()			Paginator 402
public function __toString()			think\Collection 532
public function __toString()			Conversion 238

tp5137/thinkphp/library/think/model/concern/Conversion.php

```

238 public function __toString()
239 {
240     return $this->toJson();
241 }

```

先知社区

如下图 第191-192行 所示，`$relation` 变量来自 `$this->data[$name]`，而这个变量是可以控制的。第192行的 `$name` 变量来自 `$this->append`，也是可以控制的。所以 `$relation->visible($name)` 就变成了：可控类->visible(可控变量)。那么接下来，就要找可利用的 `visible` 方法，或者没有 `visible` 方法，但有可利用的 `__call` 方法。

```

thinkphp/library/think/model/concern/Conversion.php
238 public function __toString()
239 {
240     return $this->toJson();
241 }
222 public function toJson($options = JSON_UNESCAPED_UNICODE)
223 {
224     return json_encode($this->toArray(), $options);
225 }
131 public function toArray()
132 {
133     $item      = [];
134     $hasVisible = false;
135
136     foreach ($this->visible as $key => $val) {...}
137
138     foreach ($this->hidden as $key => $val) {...}
139
140     // 合并关联数据
141     $data = array_merge($this->data, $this->relation);
142
143     foreach ($data as $key => $val) {...}
144
145     // 追加属性（必须定义获取器）
146     if (!empty($this->append)) {
147         foreach ($this->append as $key => $name) {
148             if (is_array($name)) {
149                 // 追加关联对象属性
150                 $relation = $this->getRelation($key);
151
152                 if (!$relation) {
153                     $relation = $this->getAttr($key);
154                     $relation->visible($name);
155                 }
156             }
157         }
158     }
159 }

thinkphp/library/think/model/concern/Attribute.php
472 public function getAttr($name, &$item = null)
473 {
474     try {
475         $notFound = false;
476         $value     = $this->getData($name);
477     } catch (InvalidArgumentException $e) {
478         $notFound = true;
479         $value     = null;
480     }
481
482     public function getData($name = null)
483     {
484         if (is_null($name)) {
485             return $this->data;
486         } elseif (array_key_exists($name, $this->data)) {
487             return $this->data[$name];
488         } elseif (array_key_exists($name, $this->relation)) {
489             return $this->relation[$name];
490         }
491     }
492     throw new InvalidArgumentException('property not exists');
493 }

```

全局搜了一下 visible 方法大概有3处，但是都不能利用，所以我们考虑寻找可利用的 \_\_call 方法。在搜 \_\_call 方法的时候，会发现有一处 think\Request 类比较好利用，因为这里 call\_user\_func\_array 函数的第一个参数完全可控。构造 EXP 的时候可以传入数组，变成 call\_user\_func\_array(array(任意类,任意方法),\$args)，这样我们就可以调用任意类的任意方法了。虽然第330行用 array\_unshift 函数把本类对象 \$this 放在数组变量 \$args 的第一个，但是我们可以寻找不受这个参数影响的方法。

```

Find in Path
Q: function __call
18 matches in 16 files

In Project Module Directory Scope
public function __call($method, $args) think/Validate 1544
public function __call($name, $arguments) Paginator 432
public function __call($method, $args) think/Request 327
public function __call($method, $args) think/Model 1094
public static function __callStatic($method, $args) think/Model 1103

tp5137/thinkphp/library/think/Request.php
327 public function __call($method, $args)
328 {
329     if (array_key_exists($method, $this->hook)) {
330         array_unshift($args, $this);
331         return call_user_func_array($this->hook[$method], $args);
332     }
333
334     throw new Exception('method not exists: ' . static::class . ' -> ' . $method);
335 }

```

分析过 ThinkPHP 历史 RCE 漏洞的人可能知道，think\Request 类的 input 方法经常是链中一个非常棒的 Gadget，相当于 call\_user\_func(\$filter,\$data)。但是前面我们说过，\$args 数组变量的第一个元素，是一个固定死的类对象，所以这里我们不能直接调用 input 方法，而应该寻找调用 input 的方法。

```

1347 public function input($data = [], $name = '', $default = null, $filter = '')
1348 {
1349     if (false === $name) {
1350         // 获取原始数据
1351         return $data;
1352     }
1353
1354     $name = (string) $name;
1355     if ('' !== $name) {...}
1371
1372     // 解析过滤器
1373     $filter = $this->getFilter($filter, $default);
1374
1375     if (is_array($data)) {
1376         array_walk_recursive($data, [$this, 'filterValue'], $filter);
1377         if (version_compare(PHP_VERSION, '7.1.0', '<')) {
1378             // 恢复PHP版本低于 7.1 时 array_walk_recursive 中消耗
1379             $this->arrayReset($data);
1380         }
1381     } else {
1382         $this->filterValue($data, $name, $filter);
1383     }
1428     protected function getFilter($filter, $default)
1429     {
1430         if (is_null($filter)) {
1431             $filter = [];
1432         } else {
1433             $filter = $filter ? $this->filter : $filter;
1434             if (is_string($filter) && false === strpos($filter, '/')) {
1435                 $filter = explode(',', $filter);
1436             } else {
1437                 $filter = (array) $filter;
1438             }
1439         }
1440
1441         $filter[] = $default;
1442         return $filter;
1443     }
1444
1454     private function filterValue(&$value, $key, $filters)
1455     {
1456         $default = array_pop($filters);
1457
1458         foreach ($filters as $filter) {
1459             if (is_callable($filter)) {
1460                 // 调用函数或者方法过滤
1461                 $value = call_user_func($filter, $value);
1462             }
1463         }
1464         $value = $default;
1465     }

```

调用 input 的方法共有7处，我这里直接选择比较简单的 request 方法来分析，因为这7处关键代码都类似。如果这里通过调用 request 方法间接调用 input 方法，实际上框架会报错退出的。因为这里传给 input 方法的 \$name（下图右边第1092行），实际上是先前 call\_user\_func\_array(array(任意类/任意方法), \$args) 中 \$args 数组的第一个变量，即我们前面说的一个固定死的类对象。然而如果把一个类对象作为 \$data 传给 input 方法，那么在强转成字符串的时候（上图左边1354行），框架就会报错退出。

Find: Usages of \think\Request::input in All Places x

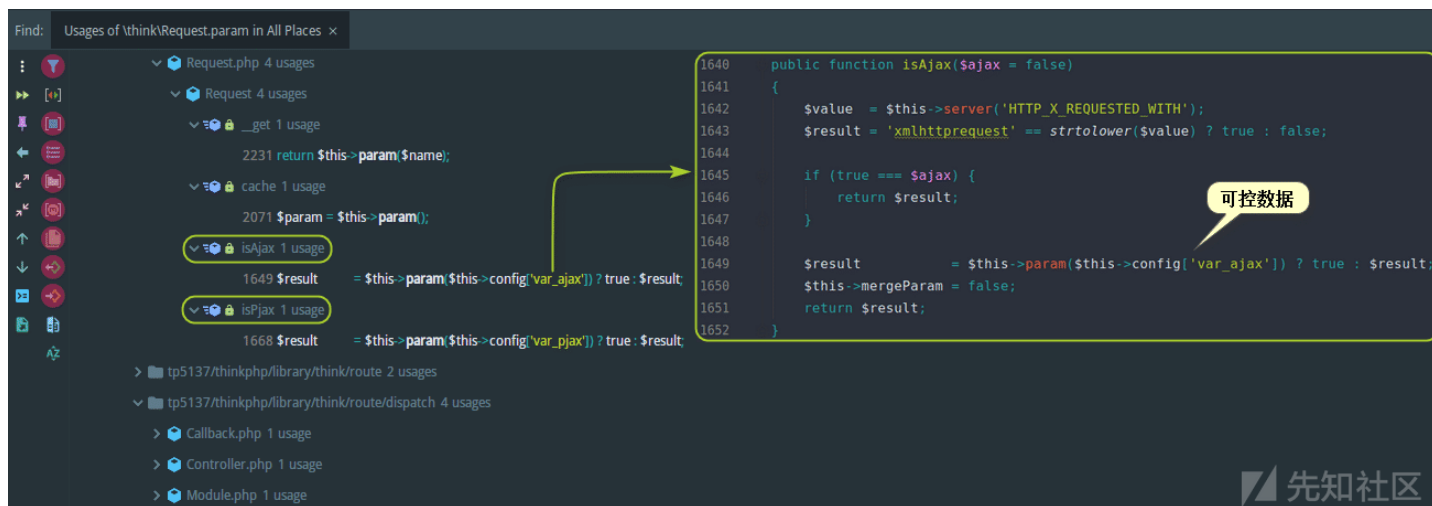
- Method
  - input
- Found usages 7 usages
  - Method call 7 usages
    - html 7 usages
      - tp5137/thinkphp/library/think 7 usages
        - Request.php 7 usages
          - Request 7 usages
            - get 1 usage
              - 1003 return \$this->input(\$this->get(\$name, \$default, \$filter));
            - param 2 usages
              - 958 return \$this->input(\$data, \$default, \$filter);
              - 961 return \$this->input(\$this->param(\$name, \$default, \$filter));
            - post 1 usage
              - 1020 return \$this->input(\$this->post(\$name, \$default, \$filter));
            - put 1 usage
              - 1037 return \$this->input(\$this->put(\$name, \$default, \$filter));
            - request 1 usage
              - 1092 return \$this->input(\$this->request(\$name, \$default, \$filter);
            - route 1 usage
              - 986 return \$this->input(\$this->route(\$name, \$default, \$filter));

```

1086 public function request($name = '', $default = null, $filter = '')
1087 {
1088     if (empty($this->request)) {
1089         $this->request = $_REQUEST;
1090     }
1091
1092     return $this->input($this->request, $name, $default, $filter);
1093 }

```

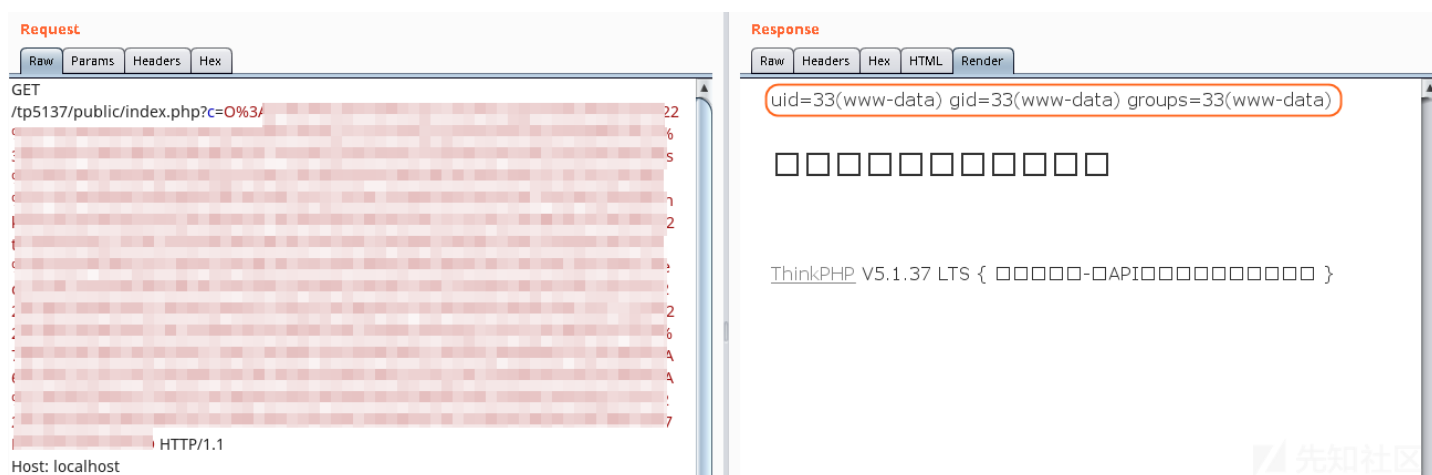
所以我们这里还要继续找有哪些地方调用了这7处。这里搜了调用 param 方法的地方，发现 isAjax 和 isPjax 都可以利用，因为他们传入 param 方法的第一个参数均可控。



这样，整个漏洞链就构造完了。下面举个例子，比如我们想执行 `system('id')` 代码，那么我们只要让传入的 Request 对象的 `$this->filter='system'` 且 `$this->param=array('id')` 即可。最终 EXP 如下（不同版本EXP不一样）：

- 5.1.16<=ThinkPHP版本<=5.1.37
- 5.1.14<=ThinkPHP版本<=5.1.15
- 5.1.3<=ThinkPHP版本<=5.1.13

PS：为了避免不必要的麻烦，已将原文中的 EXP 部分删除。



最后整理一下攻击链的流程图：

```
thinkphp/library/think/process/pipes/Windows.php
56 public function __destruct()
57 {
58     $this->close();
59     $this->removeFiles();
60 }
160 private function removeFiles()
161 {
162     foreach ($this->files as $filename) {
163         if (file_exists($filename)) {
164             @unlink($filename);
165         }
166     }
167     $this->files = [];
168 }

thinkphp/library/think/model/concern/Conversion.php
238 public function __toString()
239 {
240     return $this->toJson();
241 }
222 public function toJson($options = JSON_UNESCAPED_UNICODE)
223 {
224     return json_encode($this->toArray(), $options);
225 }
131 public function toArray()
132 {
133     if (!$relation) {
134         $relation = $this->getAttr($key);
135         $relation->visible($name);
136     }
137 }

thinkphp/library/think/Request.php
327 public function call($method, $args)
328 {
329     if (array_key_exists($method, $this->hook)) {
330         array_unshift($args, $this);
331         return call_user_func_array($this->hook[$method], $args);
332     }
333 }
1640 public function isAjax($ajax = false)
1641 {
1642     $result = $this->param($this->config['var_ajax']) ? true : $result;
1643 }
928 public function param($name = '', $default = null, $filter = '')
929 {
930     return $this->input($this->param, $name, $default, $filter);
931 }
1347 public function input($data = [], $name = '', $default = null, $filter = '')
1348 {
1349     $filter = $this->getFilter($filter, $default);
1350     array_walk_recursive($data, [$this, 'filterValue'], $filter);
1351 }
1454 private function filterValue(&$value, $key, $filters)
1455 {
1456     $default = array_pop($filters);
1457     foreach ($filters as $filter) {
1458         if (is_callable($filter)) {
1459             // 调用函数或者方法过滤
1460             $value = call_user_func($filter, $value);
1461         } elseif (is_scalar($value)) {
1462             // 调用函数或者方法过滤
1463         }
1464     }
1465 }
```

参考

[挖掘暗藏thinkphp中的反序列利用链](#)

点击收藏 | 2 关注 | 1

[上一篇：堆进阶学习之第4大利器——IO File](#) [下一篇：ret2dl\\_resolve从原理到实践](#)

1. 1 条回复



[postma\\*\\*\\*\\*@lanme](#) 2019-10-06 20:27:05

鸡肋，有什么用

0 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)