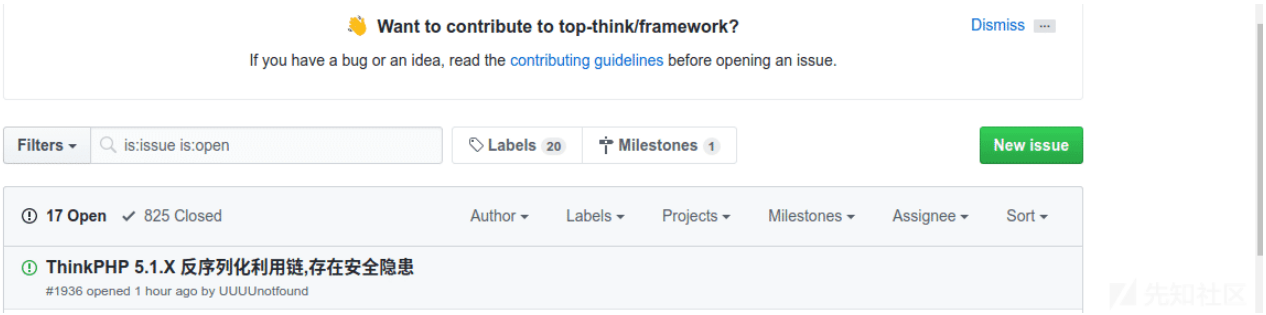
ThinkPHP5.1.X反序列化利用链

mochazz / 2019-10-06 08:10:57 / 浏览数 5503

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 # ■"topthink/framework": "5.1.*"■■"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）

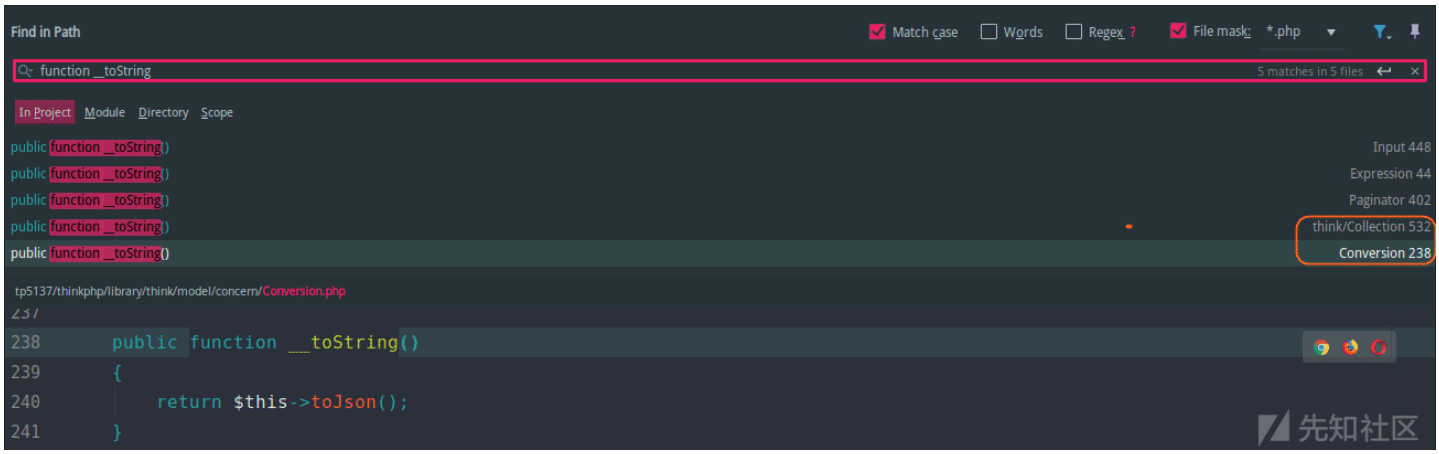
漏洞链

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。

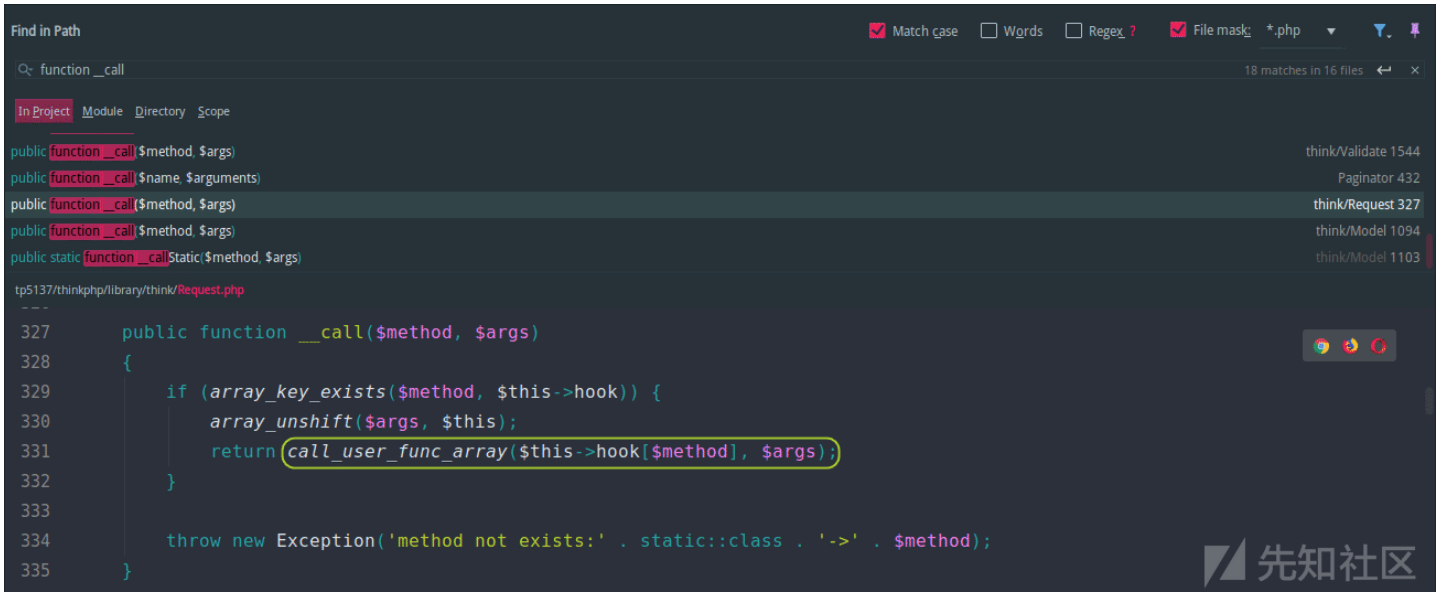
```php
56        public function __destruct()
57        {
58            $this->close();
59            $this->removeFiles();
60        }
137       public function close()
138       {
139           parent::close();
140           foreach ($this->fileHandles as $handle) {
141               fclose($handle);
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       }
```


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 。



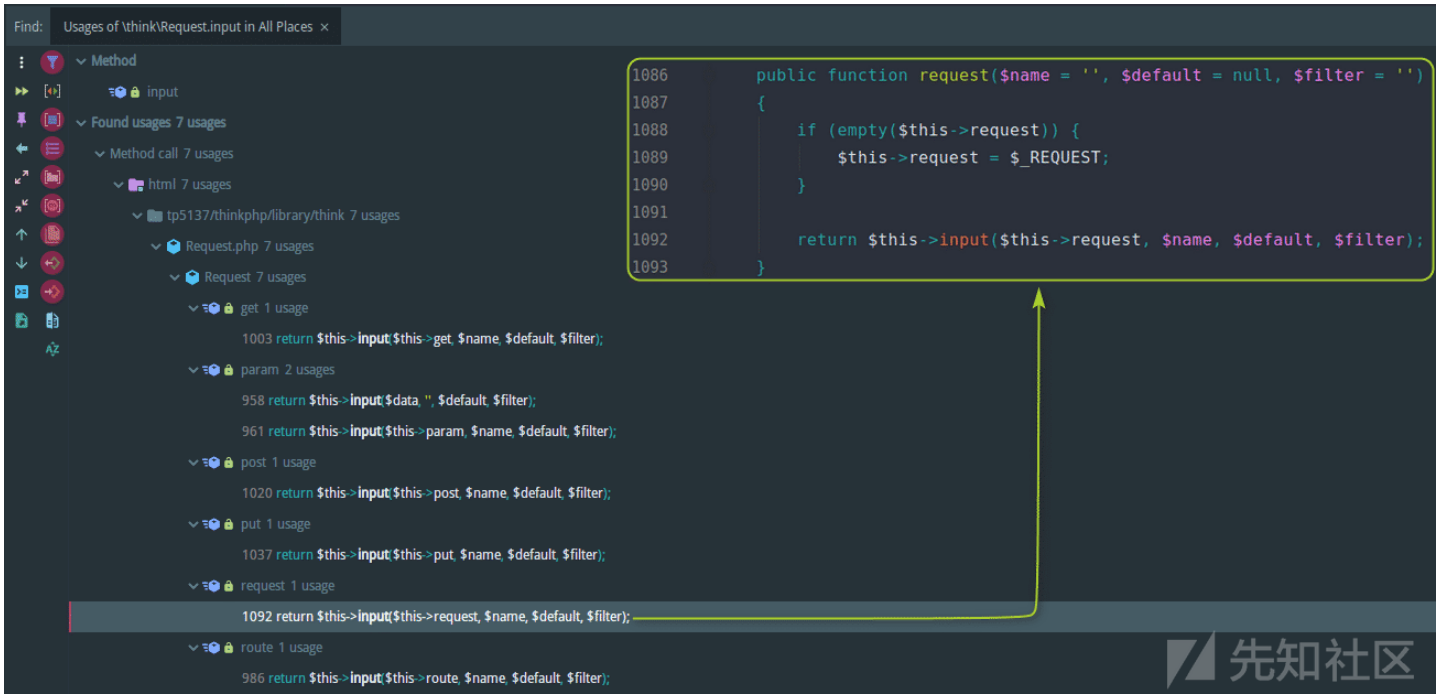
如下图 第191-192行 所示，$relation 变量来自 $this->data[$name] ，而这个变量是可以控制的。第192行 的 $name 变量来自 $this->append ，也是可以控制的。所以 $relation->visible($name) 就变成了：可控类->visible(可控变量) 。那么接下来，就要找可利用的 visible 方法，或者没有 visible 方法，但有可利用的 __call 方法。

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。

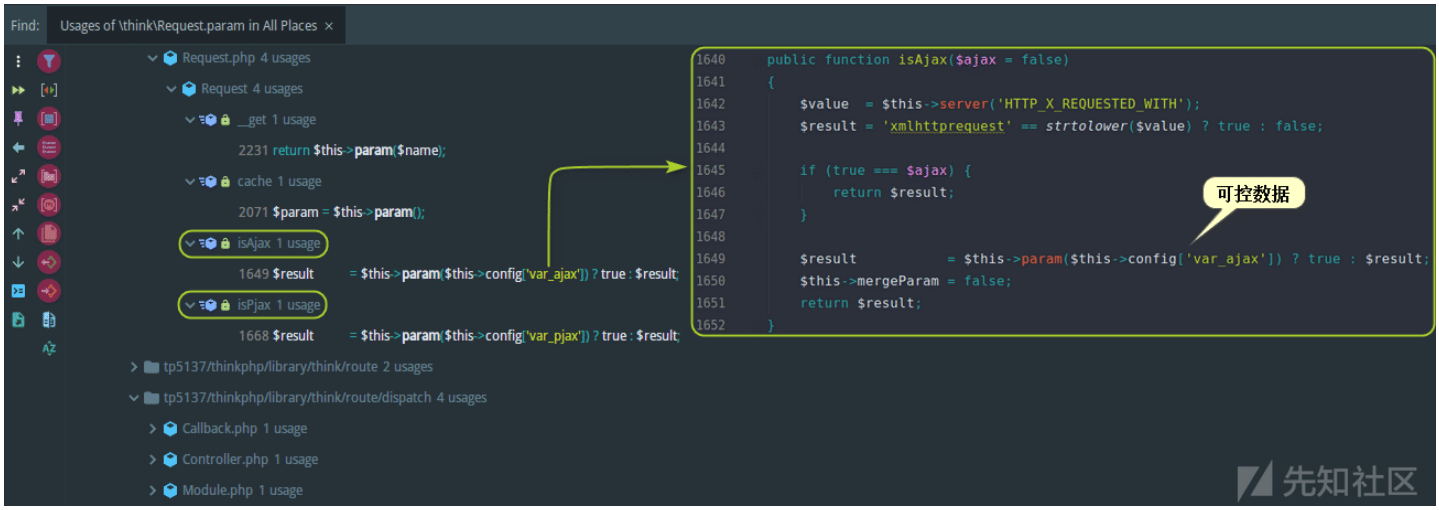


分析过 ThinkPHP 历史 RCE 漏洞的人可能知道， think\Request 类的 input 方法经常是链中一个非常棒的 Gadget ，相当于 call_user_func($filter,$data) 。但是前面我们说过， $args 数组变量的第一个元素，是一个固定死的类对象，所以这里我们不能直接调用 input 方法，而应该寻找调用 input 的方法。

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。



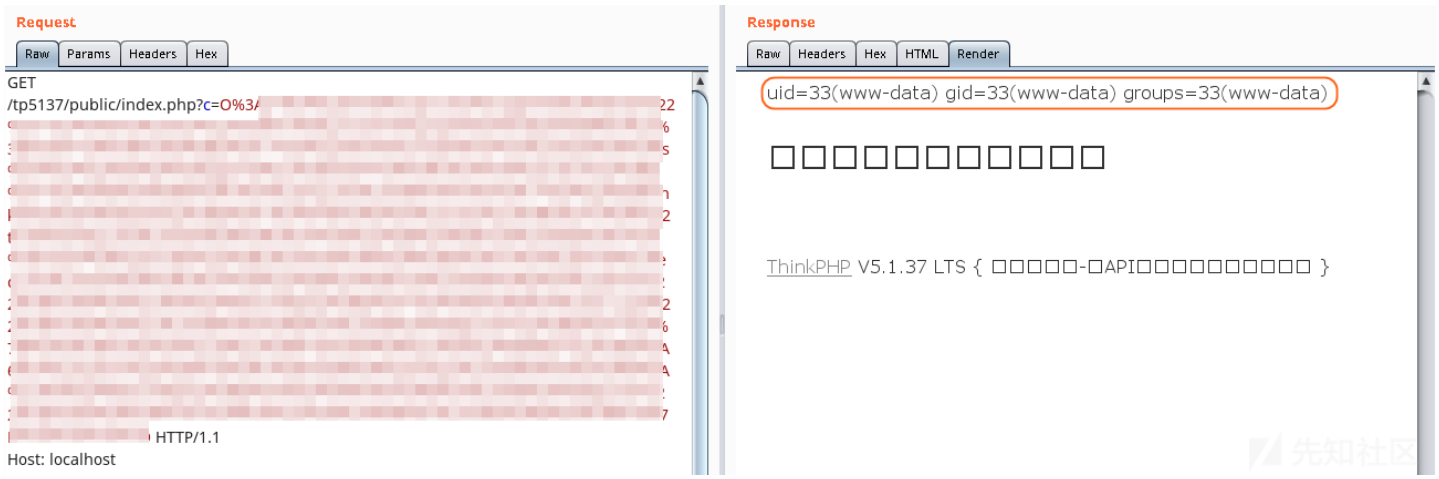
所以我们这里还要继续找有哪些地方调用了这7处。这里搜了调用 param 方法的地方，发现 isAjax 和 isPjax 都可以利用，因为他们传入 param 方法的第一个参数均可控。



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：

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

鸡助，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