Author:1x2Bytes

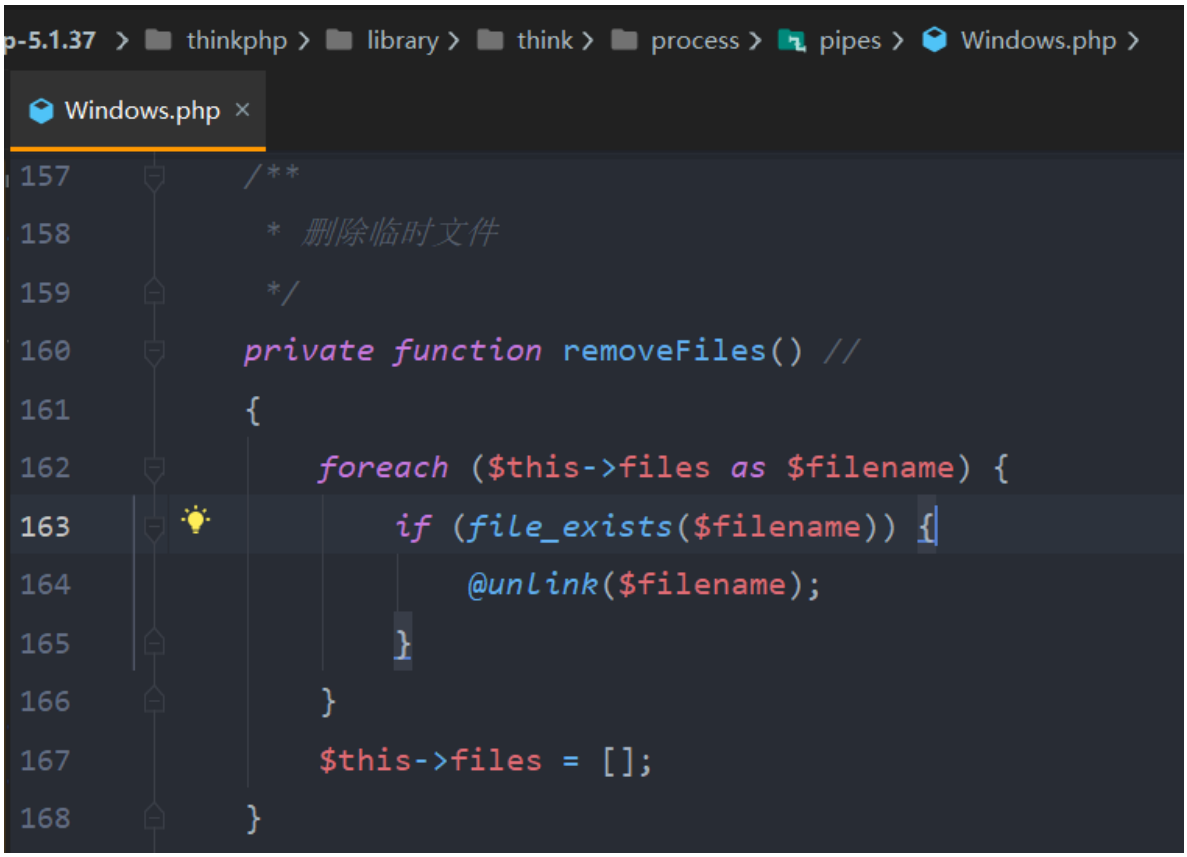
关于`ThinkPHP5.1.37`的反序列化问题已经有带师傅发出来很久了,之前Ashe师傅将一篇详细的文章发在群里,看了看不是很明白其中的gadget如何构造成pop利用链,放了一段时间没有看

随后先知又出了多篇关于Thinkphp与Laravel框架中的反序列问题分析的文章,开始对php中的反序列化问题感兴趣

于是尝试进行理解并分析,以5.1.37版本为例,第一个Gadget为任意文件删除,原功能是用来删除缓存文件的(看注释可以知道),文件名为Windows.php

在`\thinkphp\library\think\process\pipes\Windows.php`文件的第`160`行

具体代码如下:

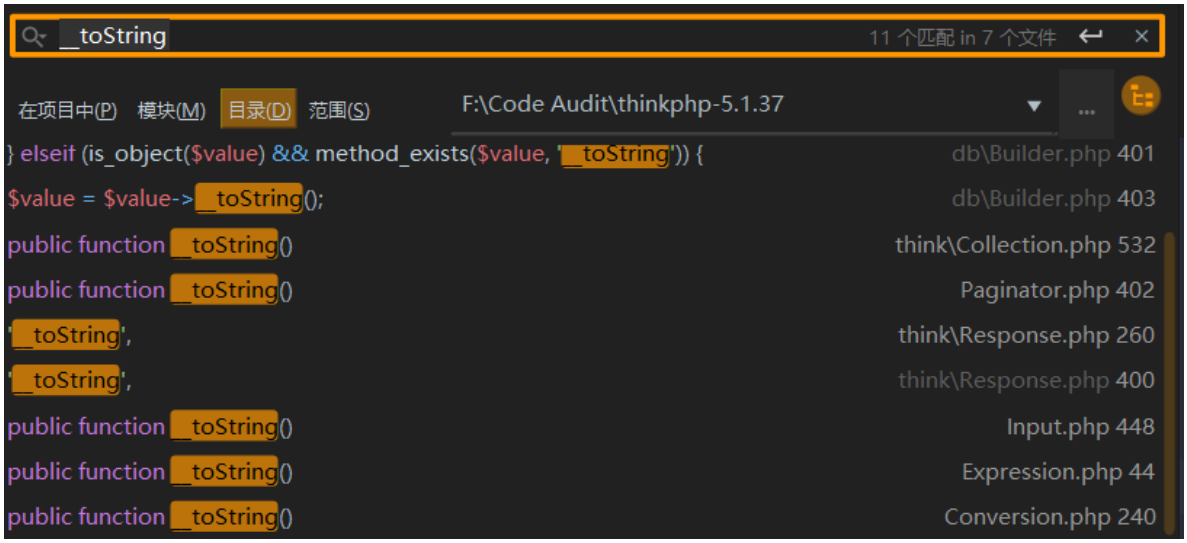


问题在于`removeFiles`方法中的`file_exists`函数,`removeFiles`首先为反序列化中的一个任意文件删除Gadget

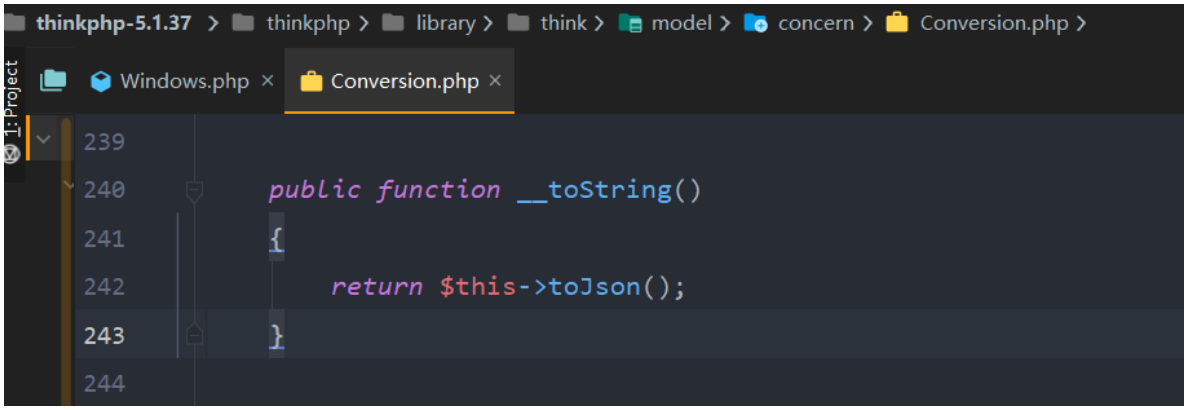
`file_exists`函数传入的`filename`是字符串类型,当`filename`被我们传入一个对象时就会触发`__toString`魔术方法

> Tips:当对象当作字符串使用时便会触发__toString方法

于是我们可以查找全局的`__toString`方法,看到很多个文件都存在`__toString`方法，但可利用的只有Conversion.php

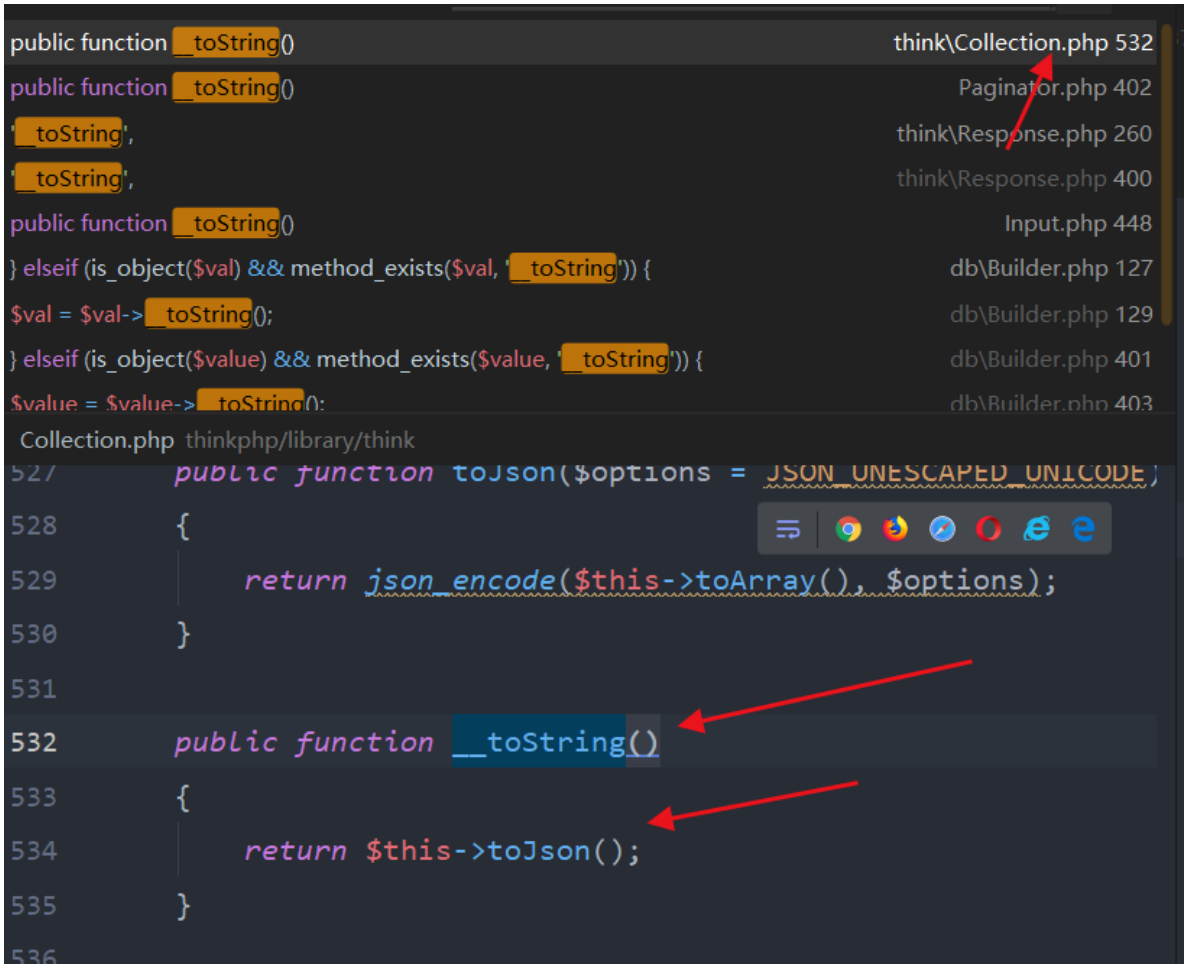


在`thinkphp\library\think\model\concern\Conversion.php`的`240`行



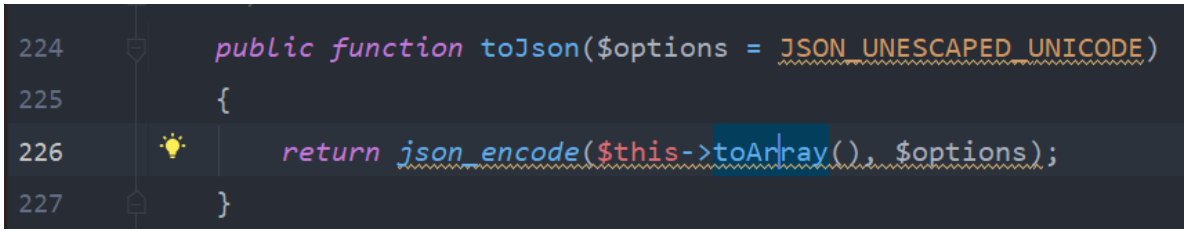
其中调用了`toJson`方法,通过前面搜索`__toString`会发现`\thinkphp\library\think\model\`下的Collection.php `532`行

也存在相同的方法



但是一路跟踪下去发现不能利用(感兴趣的朋友可以看看),于是我们回到刚刚的Conversion.php文件继续跟踪toJson方法

可以看到又调用了toArray方法



继续跟踪toArray方法看看情况,跟踪到`131`行,具体代码就不贴出来了,看师傅们的思路是找到一个可以自己指定的类然后访问里面的visible方法,因为类里面不存在该方法导致触发`__call`魔术方法,我们来看代码

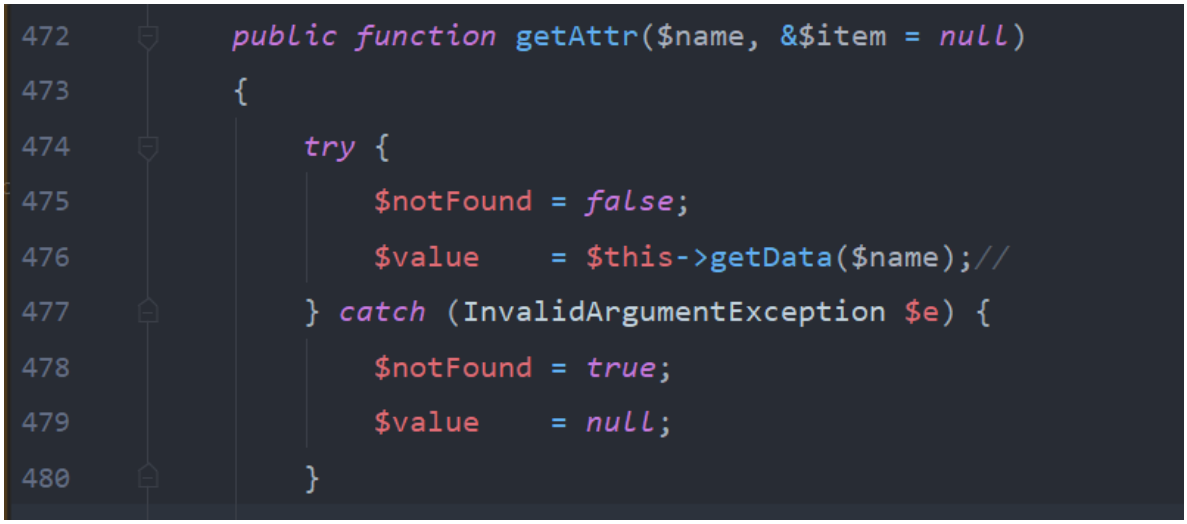
跳到`184`行,此处是存在一个可以控制的类的点,因为该处的append数组我们可控,且`188`行中`$relation`变量的值为空

具体可以分析 文件`thinkphp\library\think\model\concern\RelationShip.php`中的`getRelation`方法,代码如下

```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;
    }
```

满足`190`行的if分支,进入该分支后进行将前面的`$key`变量传入`getAttr`方法并赋值给`$relation`变量,从这里不难看出当`$relation`变量为一个类名会访问里面的visible方法(当类中不存在该方法的时候会调用`__call`方法),跟进getAttr方法看看我们的`$relation`是否可控

`\thinkphp\library\think\model\concern\Attribute.php`第`472`行



在该方法的最后会返回`$value`值所以我们跟进去看`$value`变量的获取方式,跟进getData方法,在同文件的`265`行

第一个分支不满足进入下一个分支,这里出现了一个data数组,当在data数组中找到与`$name`相同的键时将会返回该键的值

其中data可控,传入的`$key`变量也为我们可以控制的变量,这样`$relation`变量也可以控制，其中要注意将data数组的键设置为与前面append数组中的键一致,这样我们只需控制data数组中键的值为我们的类名,同时该类中没有visible方法即可触发前面的`__call`方法 查找全局`__call`方法,发现多处调用`__call`方法,但是都用处不大且有些值是固定的无法利用,参考师傅的文章是利用`\thinkphp\library\think\Request.php`中`327`行的`__call`方法

```php
public function __call($method, $args)
{
    if (array_key_exists($method, $this->hook)) {
        array_unshift( &array: $args, $this);
        return call_user_func_array($this->hook[$method], $args);
    }

    throw new Exception( message: 'method not exists:' . static::class . '->' . $method);
}
```

因为 `__call` 方法中传入两个参数 `$method` 与 `$args`，其中 `$method` 参数的值为当前尝试访问的方法即前面的visible方法,所以只有 `$args` 可控, `$args` 的值为我们前面我们尝试访问的visible方法中传入的 `$name` 变量,从该段代码中可以看出如果hook可控,我们可以控制hook数组中的键为当前的方法名(visible)值为我们要调用的任意方法即 `$method=array('visible'=>'任意方法')` 达到任意代码执行,但是因为array_unshift函数的原因导致 `$args` 数组中的第一个成员为当前类对象,导致我们难以利用,但是如果将call_user_func_array中的第一个参数设置为数组,成员分别为类与类中的方法即可调用其他方法达到我们的目的 即 `call_user_func_array(array(类,方法),$args)` ,本来想着直接调用filterValue这个命令执行神器就可以了啊,但是参数不可控,只能想办法找其他婉转调用的地方

```php
private function filterValue(&$value, $key, $filters)
{
    $default = array_pop( &array: $filters);

    foreach ($filters as $filter) {
        if (is_callable($filter)) {
            // 调用函数或者方法过滤
            $value = call_user_func($filter, $value);
        } elseif (is_scalar($value)) {
            if (false !== strpos($filter,  needle: '/')) {
                // 正则过滤
                if (!preg_match($filter, $value)) {
                    // 匹配不成功返回默认值
                    $value = $default;
```

于是搜索调用filterValue的地方发现 `1347` 行中的input方法调用了该方法 同时 cookie方法也调用了该方法,但是参数我们控制不了,同时也找不到其他可以利用的点,于是我们把目光转向input方法

```
1347    public function input($data = [], $name = '', $default = null, $filter = '')
1348    {
1349        if (false === $name) {...}
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( &input: $data, [$this, 'filterValue'], $filter);
1377            if (version_compare( version1: PHP_VERSION,  version2: '7.1.0',  operator: '<')) {
1378                // 恢复PHP版本低于 7.1 时 array_walk_recursive 中消耗的内部指针
1379                $this->arrayReset( &: $data);
```

但是并不能直接使用input方法,因为$data中的变量依然是前面__**call**中传入的数组 其中第一个成员是Request对象所以Input方法我们传入的值不可控,但是我们发现引用input方法的地方很多,可以尝试构造其他的利用点,这里我们先研究一下input中filterValue的逻辑,首先$filter是通过访问getFilter方法获取的值,我们跟进getFilter方法,在文件1428行

```
1428    protected function getFilter($filter, $default)
1429    {
1430        if (is_null($filter)) {
1431            $filter = [];
1432        } else {
1433            $filter = $filter ?: $this->filter;
1434            if (is_string($filter) && false === strpos($filter,  needle: '/')) {
1435                $filter = explode( delimiter: ',', $filter);
1436            } else {
1437                $filter = (array) $filter;
1438            }
1439        }
1440
1441        $filter[] = $default;
1442
1443        return $filter;
1444    }
```

首先判断了传入的`$filter`变量值是否为null,这时候会将当前类的`$filter`变量赋值给该方法的`$filter`,所以`$filter`可控,这样filterValue的filters也可控,这时候我们就要看我们还缺少哪些RCE的条件

```
1375        if (is_array($data)) {
1376            array_walk_recursive( &input: $data, [$this, 'filterValue'], $filter);
1377            if (version_compare( version1: PHP_VERSION,  version2: '7.1.0',  operator: '<')) {
1378                // 恢复PHP版本低于 7.1 时 array_walk_recursive 中消耗的内部指针
1379                $this->arrayReset( &: $data);
1380            }
1381        } else {
1382            $this->filterValue( &value: $data, $name, $filter);
1383        }
```

可以看到需要$data传入作为参数,但是前面说到当我们调用input方法时第一个data参数不可控,这时候我们需要寻找可以控制的点,这时候我们发现`928`行有一处param方法调用了input方法

```
928     public function param($name = '', $default = null, $filter = '')
929     {
930         if (!$this->mergeParam) {...}
952
953         if (true === $name) {
954             // 获取包含文件上传信息的数组
955             $file = $this->file();
956             $data = is_array($file) ? array_merge($this->param, $file) : $this->param;
957
958             return $this->input($data,  name: '', $default, $filter);
959         }
960
961         return $this->input($this->param, $name, $default, $filter);
962     }
```

这里我们可以看到input方法的第一个参数我们已经可控,但是传入的$name为第一个参数,也就是说传入的是一个对象,此时回到input方法查看会发现一个问题,第1354行中进行了强制类型转换,等于说将对象强制转为string类型,这样会导致框架报错,从而利用失败,如图:

```
1347     public function input($data = [], $name = '', $default = null, $filter = ''
1348     {
1349         if (false === $name) {...}
1353
1354         $name = (string) $name;
1355         if ('' != $name) {...}
1371
1372         // 解析过滤器
1373         $filter = $this->getFilter($filter, $default);
```

于是我们还要继续查找可以控制这个参数值的方法,于是查找调用param方法且可以传参的地方,发现两处

分别为`1640`行的`isAjax`方法

```
1640     public function isAjax($ajax = false)
1641     {
1642         $value  = $this->server( name: 'HTTP_X_REQUESTED_WITH');
1643         $result = 'xmlhttprequest' == strtolower($value) ? true : false;
1644
1645         if (true === $ajax) {
1646             return $result;
1647         }
1648
1649         $result          = $this->param($this->config['var_ajax']) ? true : $result;
1650         $this->mergeParam = false;
1651         return $result;
1652     }
```

与`1660`行的`isPjax`方法

```php
public function isPjax($pjax = false)
{
    $result = !is_null($this->server( name: 'HTTP_X_PJAX')) ? true : false;

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

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

以isAjax方法来说,其中的config['var_ajax']为我们所能控制的值,所以我们可以构造该POP链进行利用,构造poc的时候要注意Conversion类与Attribute类是使用trait复用的继承于Model类,而Model类是抽象类我们不能直接实例化,所以我们需要找他的子类,也就是很多POC里使用的Pivot类,最终完成我们的pop链

先通过removeFile方法传入一个对象触发`__toString`方法然后通过查找相关方法使得我们可以通过我们指定的类访问该类不存在的visible方法从而触发`__call`方法再利用__call方法中的call_user_func_array函数访问指定类中指定的方法达到我们RCE的目的

刚开始分析的时候发现看的许多文章都留了大大小小的坑没有写明白,只有自己亲身去分析才会发现问题所在，感觉php的反序列化还是挺好玩的,不得不感叹师傅们的脑回路,文笔较水,但搞清楚了个人不明白的地方,如有笔误,还望斧正。

参考链接:

https://xz.aliyun.com/t/6619 Lin师傅的文章让我学到很多

https://xz.aliyun.com/t/6467 还有七月火师傅的几篇关于反序列化的文章让我受益匪浅