# KiteCMS 的漏洞挖掘之旅（任意文件写入、任意文件读取和反序列化）

## 任意文件写入

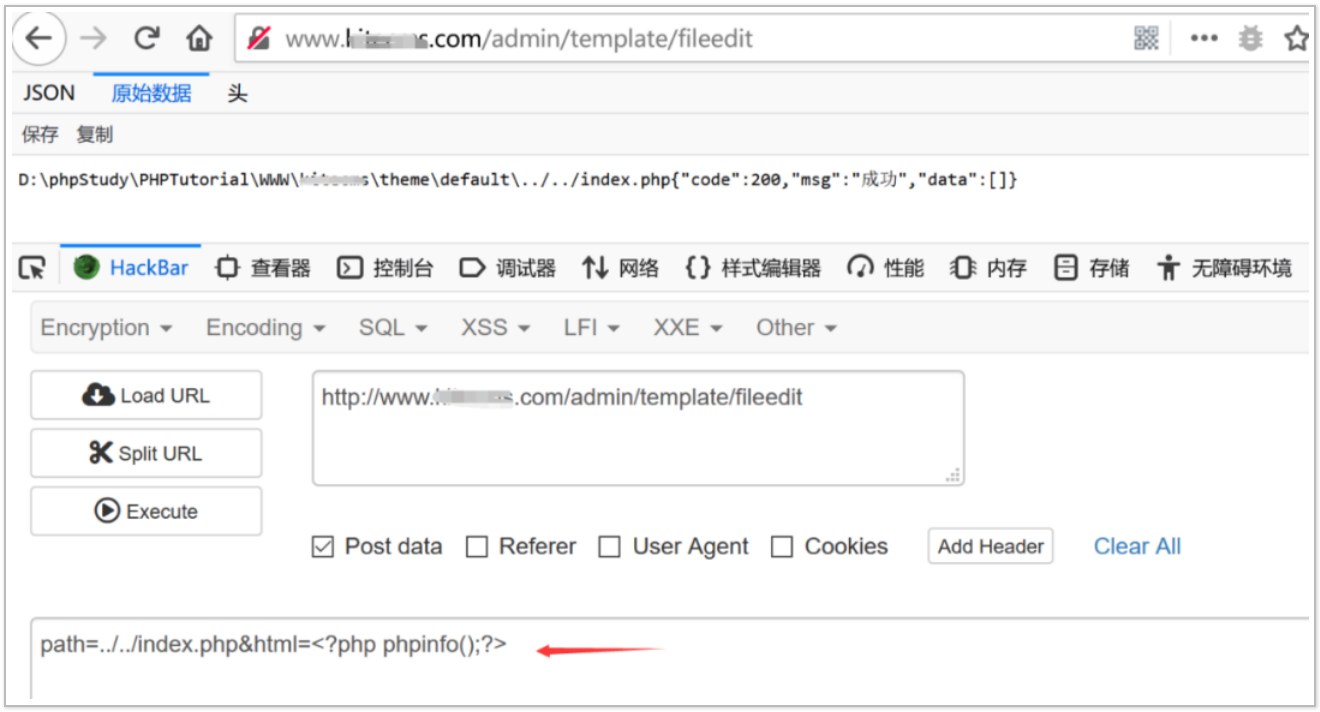
这个 cms 是基于 thinkphp5.1 的基础开发的，一般我们挖 cms 如果想 rce 的话，可以在 application 文件夹直接搜索`file_put_content`等危险函数，如下图，我们直接全局定位到这个`fileedit`方法里面的`file_put_content`

```
25      public function fileedit()
26      {
27          $path = Request::param('path');
28          $siteObj = new Site;
29          $template = $siteObj->where('id', $this->site_id)->value('theme');
30          $rootpath = Env::get('root_path') . 'theme' . DIRECTORY_SEPARATOR . $template . DIRECTORY_SEPARATOR . $path;
31          // 判断文件是否存在
32          if (!file_exists($rootpath) && !preg_match("/theme/", $rootpath)) {
33              throw new HttpException(404, 'This is not file');
34          }
35
36          if (Request::isPost()) {
37              if (is_writable($rootpath)) {
38                  $html = file_put_contents($rootpath, htmlspecialchars_decode(Request::param('html')));
39              } else {
40                  throw new HttpException(404, 'File not readabled');
41              }
42              if ($html) {
43                  return $this->response(200, Lang::get('Success'));
44              } else {
45                  return $this->response(201, Lang::get('Fail'));
46              }
47          } else {
48              if (is_readable($rootpath)) {
49                  $html = file_get_contents($rootpath);
50              } else {
51                  throw new HttpException(404, 'File not readabled');
```

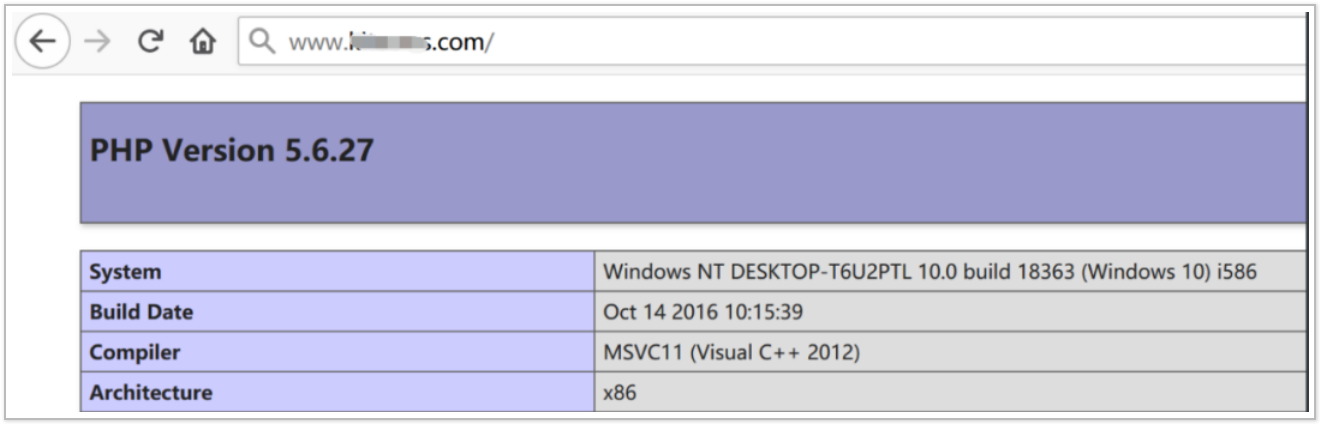
我们看到第一个参数`$rootpath`，他是被拼接了这么一段路径

```
$rootpath = Env::get('root_path') . 'theme' . DIRECTORY_SEPARATOR . $template . DIRECT
ORY_SEPARATOR . $path;
```

其中`$path`是我们可控的，那么一般就可以考虑下是否存在路径穿越的问题
再看到第二个参数`htmlspecialchars_decode(Request::param('html')`也是我们可控的
所以这里就比较清晰了，我们只需要`../`就可以进行路径穿越，`htmlspecialchars_decode`也对我们写入 php 代码没有什么影响，所以我们直接 post 传参 `path=../../index.php&html=<?php phpinfo();?>`即可



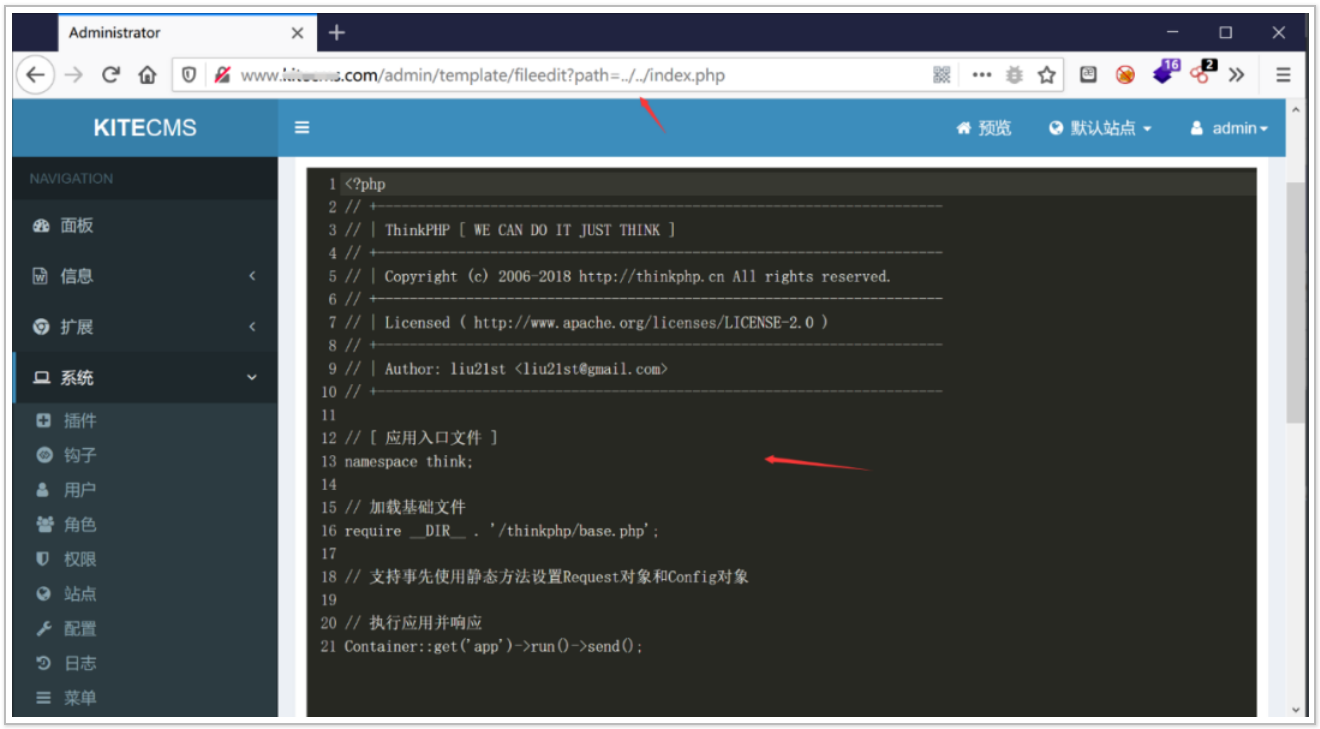

可以看到已经成功 rce




## 任意文件读取

我们再顺着 `fileedit` 这个方法往下瞅瞅，发现还有一个 `file_get_contents` ，他的参数也是 `$rootpath` ，所以这里也是我们可控的，不同的是进入这个 else 分支我们用 get 传参即可

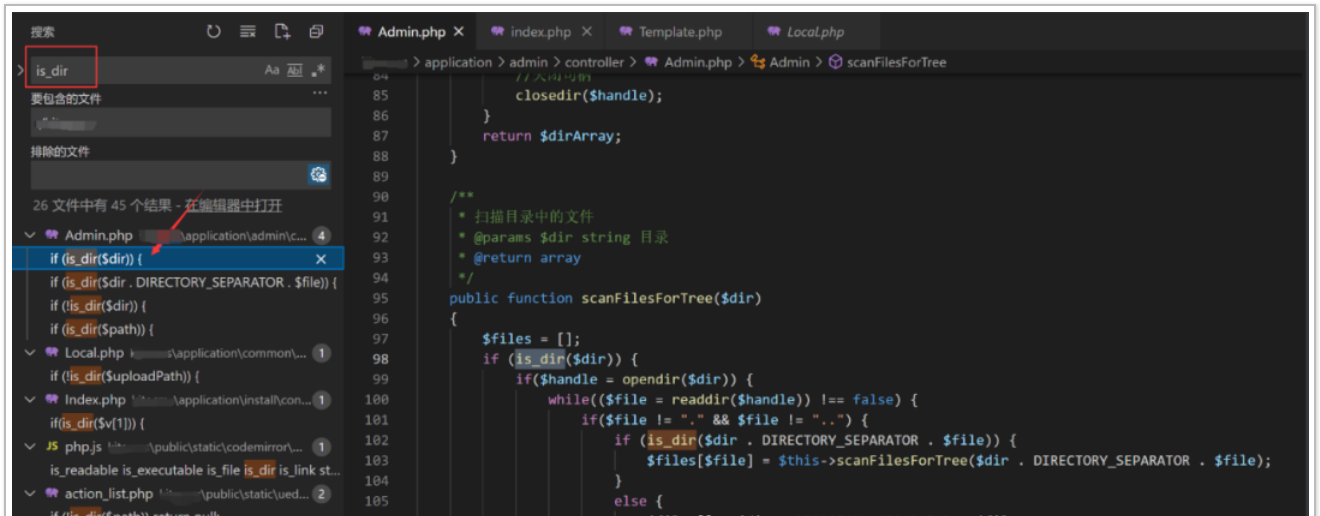
我们直接传入 `../../index.php` ，发现已经成功把 `index.php` 读取出来了




## 反序列化漏洞

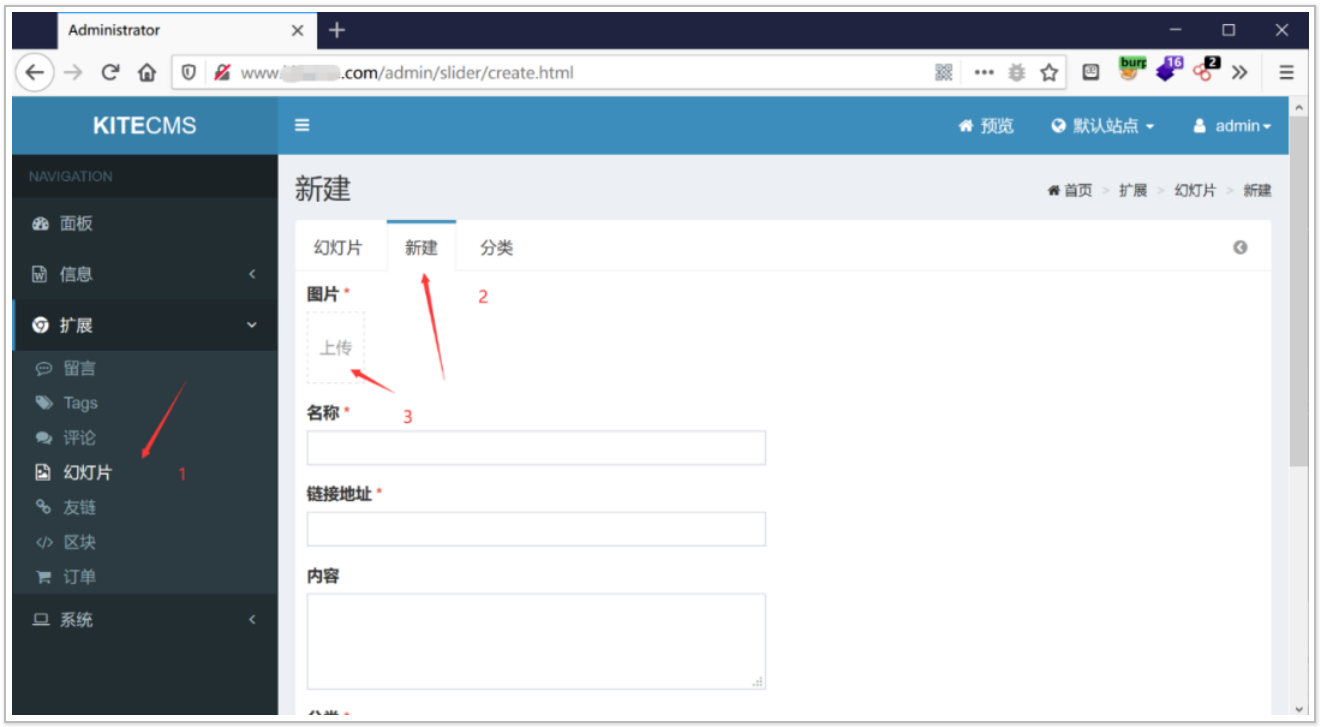
上面两个漏洞是利用了 `file_get_contents` 和 `file_put_content` ，这两个函数都是涉及了 IO 的操作函数，也就是说可以进行操作 phar 反序列化漏洞，但是他们的路径并不是完全可控的，只是后面一小部分可控，所以这条路走不通，所以接下来的思路就是搜索有没有可以操作 `phar` 的函数

我们直接全局搜索 `is_dir` ，一个一个分析是否可以利用

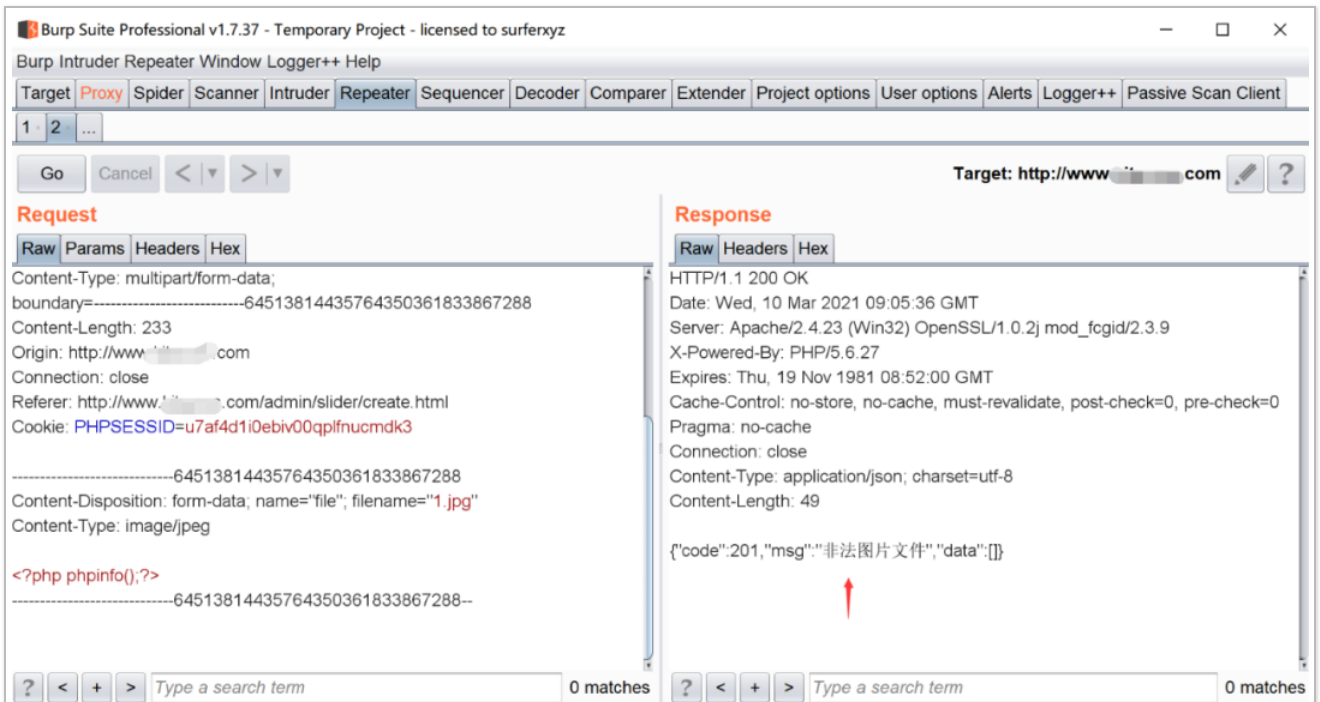



这里我的运气比较好，映入眼帘的是 `scanFilesForTree` 这个方法，他的 `$dir` 是直接可控的，文章的开头说了这个 cms 是基于 thinkphp5.1 二次开发的，所以我们可以直接利用这个漏洞生成 phar 文件来进行 rce

我们首先看看能不能上传 phar 文件，在后台一处发现可以上传文件




我们先抓个包试试水，发现提示非法图片文件，应该是写了什么过滤




我们找到 `upload` 这个函数发现对图片的类型和大小进行了一些验证

```
public function upload($file, $fileType = 'image')
    {
        // 验证文件类型及大小
        switch ($fileType)
        {
            case 'image':
                $result = $file->check(['ext' => $this->config['upload_image_ext'], 's
ize' => $this->config['upload_image_size']*1024]);
                if(empty($result)){
                    // 上传失败获取错误信息
                    $this->error = $file->getError();
                    return false;
                }
                break;
        $result =  $this->uploadHandler->upload($file);
        $data   =  array_merge($result, ['site_id' => $this->site_id]);
        SiteFile::create($data);
        return $data;
    }
```

然后尝试加了 GIF89a 头就可以上传了，看来多打 CTF 还是有用的，于是直接上传我们的 phar 文件就好了

这里要记得生成 phar 文件的时候要要加入 GIF89a 头来绕过，如下

```
$phar->setStub('GIF89a'.'<?php __HALT_COMPILER();?>');//设置stub
```

可以看到已经成功上传了，同时记住下面那个路径


最后我们在 `scanFilesForTree` 这里触发我们的 `phar` 文件就可以了


## 总结

本篇的漏洞已经全部上交 cnvd，这个 cms 总的来说比较适合练手，主要的切入点还是通过白盒通过寻找一些危险的函数，再想方设法的去控制它的参数变量