1. **rop-1-32**

In this case, we can find that the size of buffer is 0x88 then we put the 0x88 junk bytes into it. After that, find the address of setregid and execve then construct the ROP via the stack.

**rop.py**

**Text

Description automatically generated**

1. **rop-1-64**

In this case, the size of buffer is 0x88 then put 0x88 junk bytes into it. After that we do the similar thing like rop-1-32. However, we have to pop the arguments to the correspond registers rdi, rsi, rdx then trigger the function.

**rop.py**

Text

Description automatically generated

1. **rop-2-32**

In this case, we can get the flag through calling open, read, and write function. First, the open function was used to open the flag file, but not directly. By using the symbolic link, the main was linked to the flag file. Then, use read function to access the content of flag then write it out. To achieve this, we use the ROP here and pass the arguments via stack. Thus, we can get the flag.

**rop.py**

**Text

Description automatically generated**

1. **rop-2-64**

In this case, we did the similar thing like rop-2-32. But, we have to pop the arguments to the correspond registers rdi, rsi, rdx then trigger the function. After open the file, it will be read into a space then be wrote out.

**rop.py**

**Text

Description automatically generated**

1. **rop-3-32**

In this case, we use mprotect to change protection for the calling process’s memory pages. And, 0xfffff000 was used to aligned to a page boundary. We put the shellcode into the string and trigger mprotect then return to the g\_buf to execute shellcode.

**rop.py**

**Text

Description automatically generated**

1. **rop-3-64**

Compared with the rop-3-32, the only different thing is that pop the arguments to the correspond register rdi, rsi, rdx then trigger the function. In addition, the shellcode was also change.

**rop.py**

**Text

Description automatically generated**

1. **rop-4-32**

In this case, we have to use the characters which have existed in the program to generate the absolute path of the flag by using strcpy. Then use the complete path to open the flag file then read it and put the content at the usable space then print it.

**Text

Description automatically generatedText

Description automatically generatedrop.py**

1. **rop-4-64**

In this case, we do the similar thing like rop-4-32. The only different thing is that the arguments have to be popped out to the correspond register rdi, rsi, rdx then trigger the function.

**Text

Description automatically generatedText

Description automatically generatedrop.py**

1. **rop-5-32**

In this case, we use the printf to print the leak address of Libc then find the specific address about printf then get the offset between printf and execve. After that, we can use offset to calculate the address of the execve in Libc. Finally, we can execute the execve then get the flag.

**Text

Description automatically generatedText

Description automatically generatedrop.py**

1. **rop-5-64**

In this case, we do the similar thing like rop-5-32. The only different thing is that pop the arguments to the correspond registers rdi, rsi, rdx then trigger the function.

**Text

Description automatically generatedText

Description automatically generatedrop.py**

1. **rop-6-64**

In this case, we cannot find the “pop rdx” by using the ROPGADGET. Thus, we use the \_\_libc\_csu\_init to achieve the goal.

Text

Description automatically generatedText

Description automatically generated**rop.py**