Mục lục

[Câu 1: Trình bày khái niệm lỗ hổng dư 1 (off-by-one), chỉ ra ví dụ về lỗ hỏng dư 1 và khai thác lỗ hổng dư 1 2](#_Toc138700751)

[Câu 2: Anh/chị hãy trình bày khái niệm lỗ hổng chuỗi định dạng (fomat string), giải thích cơ chế khai thác lỗ hổng chuỗi định dạng để ghi 1 giá trị tùy ý vào 1 địa chỉ tùy ý 3](#_Toc138700752)

[Câu 3: Trình bày khái niệm lỗ hổng tràn số nguyên (integer overflow), chỉ ra ví dụ về lỗ hổng tràn số nguyên và khai thác lỗ hổng tràn số nguyên 4](#_Toc138700753)

[Câu 4: Trình bày lỗ hổng tràn bộ đệm (buffer overflow), trình bày cơ chế khai thác lỗ hổng tràn bộ đệm để ghi đè địa chỉ trả về nhằm chuyển hướng thực thi tới một hàm tùy ý 7](#_Toc138700754)

[Câu 5: Trình bày khái niệm lỗ hổng trường hợp đua (race condition), chỉ ra ví dụ về lỗ hổng trường hợp đua và khai thác lỗ hổng trường hợp đua 8](#_Toc138700755)

[1. Kiến trúc máy tính 9](#_Toc138700756)

[2. Stack/heap 11](#_Toc138700757)

[3. Hàm và gọi hàm 12](#_Toc138700758)

[4. Lỗ hổng phần mềm: 14](#_Toc138700759)

Lý thuyết Khai Thác Lỗ Hổng phần mềm

# Câu 1: Trình bày khái niệm lỗ hổng dư 1 (off-by-one), chỉ ra ví dụ về lỗ hỏng dư 1 và khai thác lỗ hổng dư 1

Trả lời

* Lỗ hổng off by one là lỗi logic thường gặp khi làm việc với độ dài của mảng, xâu khi người lập trình mắc lỗi xét tới giá trị bắt đầu là 0 hay 1, hoặc khi sử dụng điều kiện dừng lặp sai cách ( <= thay vì < ), hoặc khi làm việc với 1 số hàm xử lý xâu như strncat, strncpy.
* Ví dụ

void copy (const char \*str) {

char buf[64];

strncpy (buf, str, sizeof(buf));

}

* Ở đây, người lập trình nghi bằng cách giới hạn số lượng ký tự được copy (sizeof = 64) thì sẽ tránh được lỗi ràn bộ đệm off by one. Nhưng strncpy sẽ copy nhiều nhất 64 ký tựu rồi thêm 1 byte NULL ở cuối để kết thúc chuỗi. Byte NULL này nằm ngoài phạm vi của buffer và đã ghi đè dữ liệu của vùng nhớ khác
* Bằng cách dùng 1 xâu có độ dài 64 ký tự hoặc lớn hơn, kẻ tấn công có thể làm tràn bộ đẹm 1 byte và có thể gây ra crash
* Sửa bằng cách dùng sizeoff(buf) – 1 để bù cho byte NULL mà strncpy thêm vào

# Câu 2: Anh/chị hãy trình bày khái niệm lỗ hổng chuỗi định dạng (fomat string), giải thích cơ chế khai thác lỗ hổng chuỗi định dạng để ghi 1 giá trị tùy ý vào 1 địa chỉ tùy ý

Trả lời

* Lỗ hổng format string là 1 lỗ hổng phổ biến khi lập trình viên dùng printf và các hàm liên quan sai cách. Lỗ hổng này xảy ra khi chương trình in 1 xâu nhưng xâu đó bị hiểu sai định dạng, từ đó kẻ tấn công có thể thực thi code, đọc stack hay gây ra sập chương trình
* Ví dụ:

void fmt\_atk() {

int flag = 0;

char buf[512];

fgets(buf, 512, stdin);

printf(buf);

}

* Trong ví dụ trên, kẻ tấn công có thể nhập xâu bất kỳ và xâu đó được coi là xâu định dạng (tham số đầu tiên của printf). Hacker có thể nhập %x%x%x và printf sẽ in ra 3 giá trị 32bit tiếp theo trong stack

# Câu 3: Trình bày khái niệm lỗ hổng tràn số nguyên (integer overflow), chỉ ra ví dụ về lỗ hổng tràn số nguyên và khai thác lỗ hổng tràn số nguyên

Trả lời

* Tràn số nguyên là việc kết quả của phép tính trên số nguyên vượt quá phạm vi biểu diễn của kiểu dữ liệu nguyên. Khi một tràn số nguyên xảy ra, giá trị tính toán vượt qua giới hạn của kiểu dữ liệu số nguyên, và kết quả không thể đại diện chính xác. Thay vào đó, giá trị bị ghi đè lên và trở về giá trị nhỏ hơn giới hạn tối đa hoặc nhỏ nhất của kiểu dữ liệu.
* Ví dụ 1:

#include <stdio.h>

int main() {

    int a = 2147483647;  // Giá trị tối đa cho kiểu dữ liệu int

    int b = a + 1;      // Tràn số nguyên xảy ra

    printf("%d", b);

    return 0;

}

* Trong ví dụ trên, biến a được gán giá trị là 2147483647, giới hạn tối đa cho kiểu dữ liệu int trên một số kiến trúc hệ thống. Tuy nhiên, khi thực hiện phép tính a + 1, sẽ xảy ra tràn số nguyên vì giá trị kết quả vượt quá giới hạn tối đa. Kết quả được gán cho biến b không thể đại diện chính xác và thường là một giá trị nhỏ hơn giới hạn tối đa của kiểu dữ liệu int, ví dụ như -2147483648.
* Lỗ hổng tràn số nguyên có thể được khai thác trong các tình huống nhạy cảm để thực hiện các hành động độc hại hoặc kiểm soát luồng chương trình. Ví dụ, một hacker có thể tận dụng lỗ hổng tràn số nguyên để ghi đè lên các biến quan trọng, thay đổi luồng chạy của chương trình hoặc thực hiện các hành động không mong muốn.
* Để ngăn chặn lỗ hổng tràn số nguyên, cần kiểm tra và xử lý kỹ lưỡng giới hạn của kiểu dữ liệu và giá trị đầu vào để đảm bảo rằng không có tràn số xảy ra.
* Ví dụ 2:
* Tràn số nguyên là việc kết quả của phép tính trên số nguyên vượt quá phạm vi biểu diễn của kiểu dữ liệu nguyên. Khi một tràn số nguyên xảy ra, giá trị tính toán vượt qua giới hạn của kiểu dữ liệu số nguyên, và kết quả không thể đại diện chính xác. Thay vào đó, giá trị bị ghi đè lên và trở về giá trị nhỏ hơn giới hạn tối đa hoặc nhỏ nhất của kiểu dữ liệu.

#include <stdio.h>

#include < stdlib.h>

#include <unistd.h>

#define SIZE 256

int main ( int argc , char ∗∗ argv )

{

char buf [ SIZE ] ;

short len ;

if ( argc < 2 )

{

return 0 ;

}

len = atoi ( argv [ 1 ] ) ;

if ( len > SIZE )

{

return 0 ;

}

puts ( " Input\_a\_string" ) ;

fgets ( buf , ( len & 0xFFFF ) , stdin ) ;

return 0 ;

}

* Theo đoạn code trên, chương trình sử dụng hàm atoi để chuyển đổi tham số dòng lệnh thứ nhất (argv[1]) thành một giá trị ngắn và gán cho biến len, kiểu dữ liệu của len là short. Tuy nhiên, ở dòng tiếp theo, chương trình không kiểm tra xem giá trị của len có vượt quá giới hạn SIZE (256) hay không. Nếu giá trị của len lớn hơn SIZE, lỗi tràn số nguyên sẽ xảy ra.
* Vì biến len có kiểu dữ liệu là short, chỉ có thể chứa các giá trị từ -32,768 đến 32,767. Nếu tham số dòng lệnh được nhập vào có giá trị lớn hơn 32,767, len sẽ không thể chứa giá trị đó và sẽ xảy ra tràn số nguyên.
* Khi giá trị của len vượt quá giới hạn của kiểu short, lỗi xảy ra. Khi chương trình tiếp tục chạy và đến dòng 21, tham số thứ hai của hàm fgets là (len & 0xFFFF). Khi giá trị âm của len được sử dụng như một giá trị dương, phép AND len & 0xFFFF không thể giới hạn giá trị của len, và fgets sẽ đọc vào nhiều ký tự hơn mảng buf có thể chứa. Điều này dẫn đến lỗi tràn bộ đệm.
* Để khắc phục lỗi tràn số nguyên này, chương trình cần kiểm tra giới hạn của len trước khi sử dụng nó. Nếu giá trị của len vượt quá giới hạn cho phép, chương trình nên thực hiện xử lý phù hợp như thông báo lỗi hoặc cắt giá trị len xuống giới hạn cho phép.

# Câu 4: Trình bày lỗ hổng tràn bộ đệm (buffer overflow), trình bày cơ chế khai thác lỗ hổng tràn bộ đệm để ghi đè địa chỉ trả về nhằm chuyển hướng thực thi tới một hàm tùy ý

Trả lời

* Lỗ hổng tràn bộ đệm (Buffer Overflow) là lỗ hổng trong lập trình, cho phép dữ liệu được ghi vào một buffer có thể tràn ra ngoài buffer đó, ghi đè lên dữ liệu khác và dẫn tới hoạt động bất thường của chương trình.
* Nguyên tắc khai thác:
* Dữ liệu quan trọng phải nằm phía sau (ở địa chỉ cao hơn) so với bộ đệm
* Phần dữ liệu tràn phải đủ lớn để đè lên được dữ liệu quan trọng
* Những dữ liệu khác nằm giữa vùng đệm và dữ liệu mục tiêu cũng bị ghi đè. Việc ghi đè đó có thể ảnh hưởng đến logic làm việc của chương trình, đến khả năng thành công của việc khai thác.
* Ví dụ:

#include <stdio.h>

int main()

{

    int cookie=0;

    char buf[16];

    printf("Your name: ");

    gets(buf);

    if(cookie == 0x41424344)

puts("You win!");

    else

puts("Try again!");

    return 0;

}

* Giải thích: Hàm gets(buf) để nhận đầu vào từ người dùng và lưu trữ nó trong mảng buf. Lỗi xảy ra người dùng nhập vào một chuỗi ký tự dài hơn 16 ký tự, vượt quá kích thước đã được cấp phát cho mảng buf. Trong trường hợp này, đầu vào sẽ ghi đè lên vùng nhớ xung quanh buf, bao gồm cả biến cookie. Nếu người dùng nhập một chuỗi đủ dài, nó có thể ghi đè lên biến cookie và thay đổi giá trị của nó. Điều này có thể dẫn đến việc so sánh cookie với 0x41424344 (ABCD) trả về true, và thông báo "You win!" được hiển thị, mặc dù người dùng không đáng nhận giải thưởng.
* Để khắc phục lỗ hổng này, cần sử dụng các hàm an toàn như fgets() thay vì gets() để đảm bảo rằng đầu vào không vượt quá kích thước được cấp phát cho buffer.

# Câu 5: Trình bày khái niệm lỗ hổng trường hợp đua (race condition), chỉ ra ví dụ về lỗ hổng trường hợp đua và khai thác lỗ hổng trường hợp đua

Trả lời

* RC xảy ra khi nhiều tiến trình truy cập và sửa đổi cùng một dữ liệu vào cùng một lúc, và kết quả của việc thực thi phụ thuộc vào thứ tự của việc truy cập. Nếu một chương trình vướng phải lỗi này, người tận dụng lỗi có thể chạy nhiều tiến trình song song để “đua” với chương trình có lỗi, với mục đích là thay đổi hoạt động của chương trình ấy. Đôi khi, trường hợp đua còn được biết đến với tên gọi thời điểm kiểm tra/thời điểm sử dụng (Time Of Check/Time Of Use, TOC/TOU)
* Ví dụ:

#include < s t d l i b . h>

#include <s t d i o . h>

#include <u ni s t d . h>

int main ( int argc , char ∗∗ argv )

{

FILE ∗ f i l e ;

char b u f f e r [ 2 5 6 ] ;

if ( access ( argv [ 1 ] , R\_OK) == 0 )

{

usleep ( 1 ) ;

file = fopen ( argv [ 1 ] , " r " ) ;

if (file == NULL)

{

goto cleanup ;

}

fgets ( buffer , sizeof(buffer) , file ) ;

fclose (file);

puts (buffer);

return 0 ;

}

cleanup :

perror (“Cannot\_open\_file”)

return 0 ;

}

Kiến thức nền tảng:

1. Kiến trúc máy tính

Central Proccesing Unit – CPU

Control Block

ALU I/O devices

Registers Main memory Disk Printer => Bus

CPU => Main (khớp socket)

Thanh ghi:

Thanh ghi đa dụng: EAX, EBX, ECX, EDX

Thanh ghi xử lý chuỗi: EDI, ESI

Thanh ghi con trỏ lệnh: EIP

Thanh ghi ngăn xếp: ESP, EBP

Thanh ghi cờ: EFLAGS

Thanh ghi phân vùng: không còn được sử dụng ở kiến trúc 32 bít

RAM: chia thành các ô nhớ, mỗi ô 1 byte

Chứa 1 phần hệ điều hành, các lệnh chương trình, các dữ liệu

Mỗi ô nhớ có địa chỉ duy nhất bắt đầu từ 0

Mô hình bộ nhớ tuyến tính: Flat memory model

Mô hình bộ nhớ mà các ô nhớ được đánh địa chỉ liên tiếp từ 0 đến MAXBYTE - 1

32 bit ở chế độ Proteced luôn sử dụng mô hình Flat

Mỗi chương trình có thể coi là nó có riêng 4 GB RAM.

Mã lệnh và dữ liệu cùng nằm trong một không gian địa chỉ.

Hướng ghi dữ liệu

Các hàm nhập dữ liệu trong các ngôn ngữ lập trình luôn ghi dữ liệu vào RAM theo chiều tăng dần của địa chỉ

Trật tự byte: little-endian

Sử dụng little-dian để biểu diễn số dương

ROM:

1. Stack/heap

Stack: vùng nhớ được hệ điều hành phát cho chương trình khi nạp (k chỉ định mặc định là 1Mb)

Kích thước: được xác định khi biên dịch chương trình (1 ô 4byte)

Chức năng:

Chứa các biến cục bộ

Lưu địa chỉ trả về khi gọi hàm

Truyền tham số khi gọi hàm

Lưu trữ con trỏ “this” trong lập trình hướng đối tượng (this.a)

Quản lý qua ESP

Thao tác cơ bản: PUSH giảm giá trị con trỏ, POP tăng giá trị con trỏ (4byte)

Heap: lưu biến cấp phát động, liên kết động

1. Hàm và gọi hàm

Hàm (Procedure) là một đoạn chương trình con mà có thể được gọi bởi một chương trình khác để thực thi một nhiệm vụ nhất định

Procedure\_name:

;some instructions

RET

Thông thường, nếu hàm có trả về một kết quả thì kết quả đó được đặt trong EAX trước khi hàm kết thúc

VD: MySimpleProc:

Add eax,ebx

Sub eax,edx

Ret

Start:

Mov eax,11

Call MySimpleProc ;EAX = 11+22+33=0

Gọi hàm

Tham số có thể được nạp vào thanh ghi

Ưu: nhanh

Nhược: không đủ thanh ghi

Cần kết hợp nạp tham số vào stack

Người xây dựng hàm có toàn quyền lựa chọn cách thức nạp tham số. Nhưng cần có quy ước chung:

Mọi người hiểu mã của nhau

Mọi người có thể sử dụng hàm của nhau

Calling Convention

Phổ biến: stdcall (Windows API) và cdecl (standard C library)

Giống:

Truyền tham số qua stack; tham số được truyền từ phải sang trái

caller phải bảo quản EAX, ECX và EDX nếu cần (callee phải bảo quản các thanh ghi khác)

Khác:

cdecl: caller phải cân bằng stack

stdcall: callee phải cân bằng stack

Có thể gặp: fastcall

1. Lỗ hổng phần mềm:

Lỗ hổng phần mềm (software vulnerability) là khiếm khuyết trong thiết kế, lập trình phần mềm mà kẻ tấn công có thể lợi dụng để làm thay đổi hoạt động bình thường của phần mềm

Lỗ hổng do lập trình: Chương 2

* Buffer Overflow: Lỗ hổng khiến dữ liệu có kích thước lớn có thể tràn ra khỏi vùng đệm để chứa nó
  + VD: char st[10];

gets(st);

* Format String: Lỗ hổng khiến dữ liệu chuỗi bị diễn giải như một chuỗi định dạng
  + VD: char st[10];

gets(st);

printf(st);

* Interger Overflow: Lỗ hổng khiến kết quả phép toán trên số nguyên bị diễn giải sai khi vượt quá phạm vi giá trị
  + VD: int a, b;

scanf("%d", &b);

if(a+b < a)

printf("b < 0");

* Off-by-One: Lỗ hổng khiến ký tự kết thúc chuỗi bị ghi đè
  + VD: char \*st1, \*st2;

...

for(int i=0; i<=strlen(st1); i++)

st1[i] = st2[i];

* Race Condition: Lỗ hổng trong vấn đề đồng bộ dữ liệu khiến một tiến trình vẫn xử lý dữ liệu cũ, trong khi dữ liệu đã được cập nhật bởi một tiến trình khác (thường xảy ra trong xử lý song song)

Lỗ hổng sử dụng mật mã:

* Use of Weak Password-Bassed Systems

Lỗ hổng: 3 loại

-C – Dos

- B – PE – leo thang đặc quyền

- A - RCE

Quy tắc được định hình trong Compiler