**Mục lục**

[1. Trình bày khái niệm lỗ hổng dư 1 (off-by-one); chỉ ra ví dụ về lỗ hổng dư 1 và khai thác lỗ hổng dư 1 1](#_heading=h.gjdgxs)

[2. Trình bày khái niệm lỗ hổng chuỗi định dạng (format string); giải thích cơ chế khai thác để ghi một giá trị tùy ý vào một địa chỉ tùy ý. 2](#_heading=h.30j0zll)

[3. Trình bày khái niệm lỗ hổng tràn số nguyên (integer overflow), chỉ ra ví dụ 4](#_heading=h.1fob9te)

[4. Trình bày khái niệm lỗ hổng tràn bộ đệm (buffer overflow), chỉ ra ví dụ 5](#_heading=h.3znysh7)

[5. Trình bày khái niệm lỗ hổng đường đua (race condition), chỉ ra ví dụ 5](#_heading=h.2et92p0)

# **1. Trình bày khái niệm lỗ hổng dư 1 (off-by-one); chỉ ra ví dụ về lỗ hổng dư 1 và khai thác lỗ hổng dư 1**

**Khái niệm**: Là lỗi logic thường gặp khi làm việc với độ dài của mảng, xâu khi người lập trình mắc lỗi khi xét tới giá trị bắt đầu là 0 hay 1. Hay khi sử dụng điều kiện dừng lặp sai cách (<= thay vì <), hay khi làm việc với một 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 tình nghĩ bằng cách giới hạn số lượng ký tự được copy (sizeof=64) thì sẽ tránh được lỗi tràn bộ đệm off by one. Nhưng strncpy sẽ copy nhiều nhất 64 ký tự 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 bằng hoặc lớn hơn 64 ký tự, hacker có thể làm tràn bộ đệm 1 byte và có thể gây ra sập chương trình

Sửa bằng cách dùng sizeof(buf) -1 để bù cho byte NULL mà strncpy thêm vào

# **2. Trình bày khái niệm lỗ hổng chuỗi định dạng (format string); giải thích cơ chế khai thác để ghi một giá trị tùy ý vào một địa chỉ tùy ý.**

**Khái niệm**: Là một lỗ hổng phổ biến khi dev dùng printf và các hàm liên quan sai cách. Lỗ hổng này xảy ra khi chương trình in một xâu nhưng xâu đó bị hiểu thành xâu định dạng, từ đó hacker 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); // dòng này có lỗi fmt\_atk

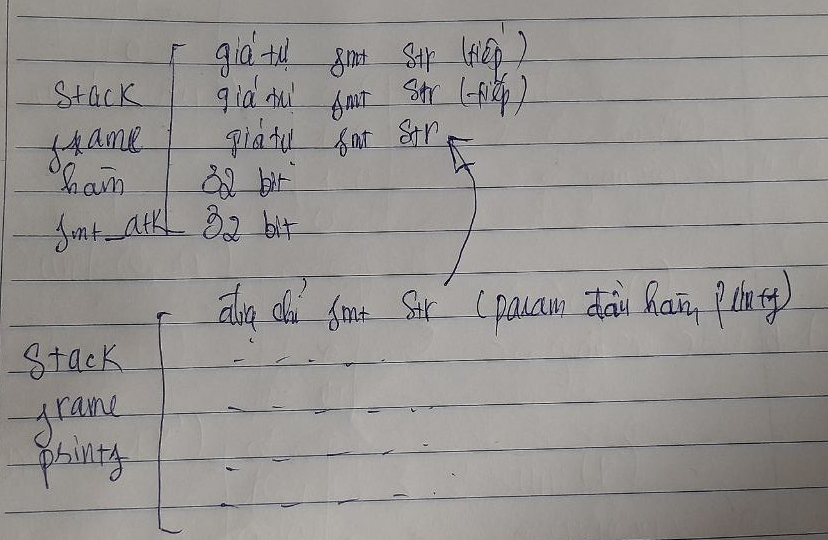
}

In this case, hacker 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 32 bít tiếp theo trong stack

Stack của app có dạng sau, mỗi dòng 32-bit



Hacker có thể truyền như sau:

\x aa \x bb \x cc \x dd %x %x %n

Lúc đó stack sẽ thành

A close-up of a paper

Description automatically generated with low confidence

Hàm “printf” sẽ in ra 4 byte đầu như bth ( ASCII 0xaabbccdd) sau đó xử lý %x => in ra 4 byte tại đ/c 0x4, sau đó %x => in 4 byte tại 0x8.

Sau đó xử lý %n => Đọc địa chỉ tại 0xC và ghi số lượng ký tự đã in ra (4+4+4=12) vào đó

=> “printf” đã ghi giá trị 12 vào đ/c ddccbbaa

# **3. Trình bày khái niệm lỗ hổng tràn số nguyên (integer overflow), chỉ ra ví dụ**

**Khái niệm**: Là lỗ hổng trong C khi một phép toán số học cố gắng tạo ra một giá trị số nằm ngoài phạm vi có thể biểu diễn với một số bit nhất định (có thể lớn hơn giá trị MAX, nhỏ hơn giá trị MIN) được thể hiện. Kết quả là các bit được thể hiện ít quan trọng nhất được **bao lại** quanh số lớn nhất.

**Ví dụ:**

void int\_over() {

int max\_int = INT\_MAX; // Maximum value for an 'int'

int addend = 1;

int result = max\_int + addend; // expected result: integer overflow

printf("Result: %d\n", result);

return 0;

}

Trong ví dụ này, chúng ta có biến `max\_int`, với giá trị được thiết lập bằng giá trị tối đa có thể được đại diện cho kiểu dữ liệu `int` trong C bằng hằng số `INT\_MAX`. Chúng ta cố gắng thêm `1` vào giá trị này và gán kết quả cho biến `result`.

Phép tính này có thể tràn - overflow khi giá trị tính toán vượt quá giới hạn cho phép của kiểu dữ liệu `int`. Trong trường hợp này, kết quả mong đợi sẽ là một giá trị không xác định, vì theo chuẩn C, khi một biến int vượt quá giới hạn của kiểu dữ liệu, kết quả không được định nghĩa và thường bị sai.

# **4. Trình bày khái niệm lỗ hổng tràn bộ đệm (buffer overflow), chỉ ra ví dụ**

Khái niệm: là một đk bất thường khi một tiến trình lưu trữ data vượt ra ngoài biên của bộ nhớ đệm có chiều dài cố định. Kết quả data (bộ nhớ đệm khác, biến, …) có thể ghi đè lên các bộ nhớ liền kề. Thường xảy ra trên C/C++ phổ biến và nguy hiểm nhất.

Ví dụ:

int main(int argc, char \*\*argv) {

char buf[500];

if(argc>1) {

strcpy(buf, argv[1]);

printf("%s", buf);

}

}

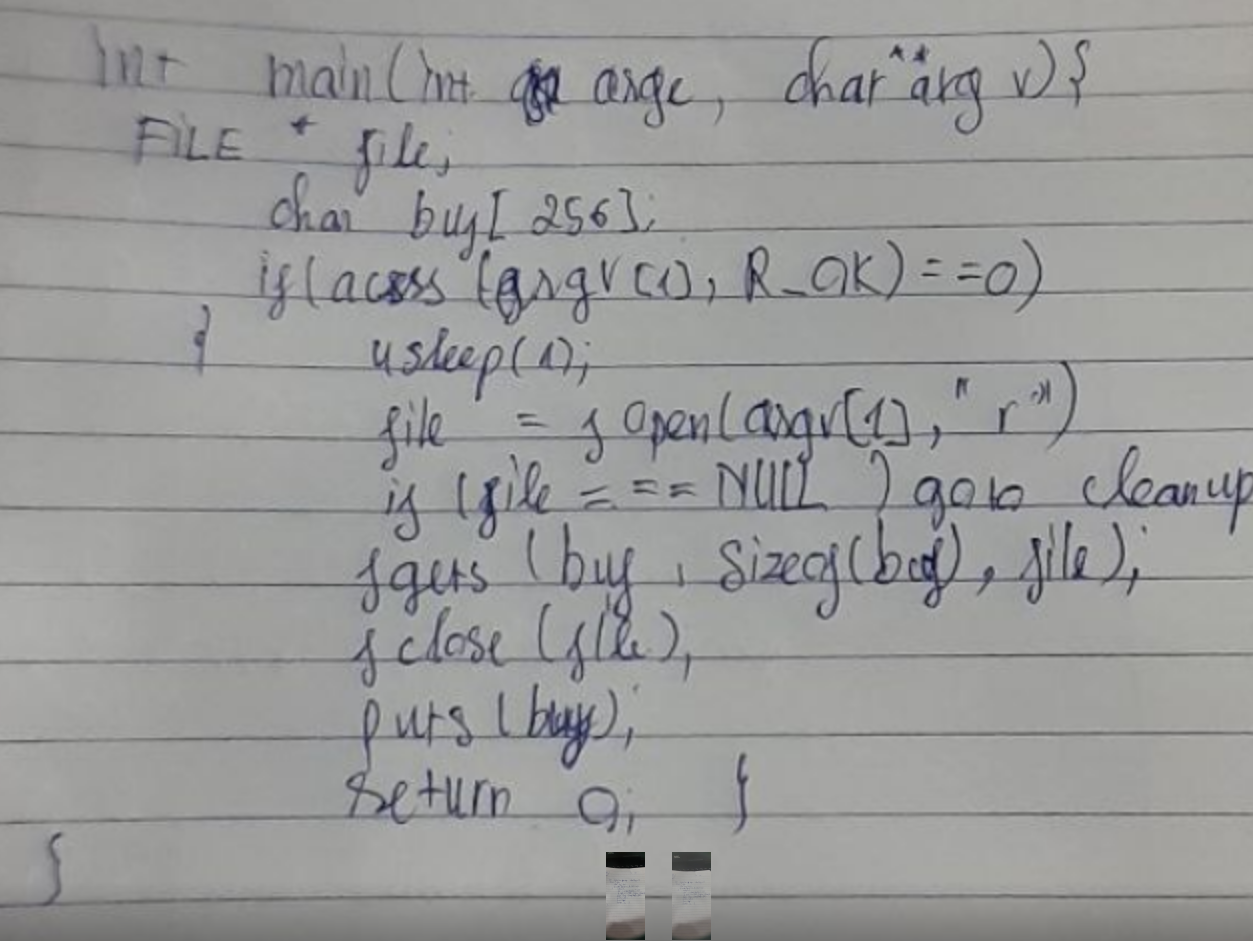
Nếu argv[1] có độ dài lớn hơn giới hạn 500 ký tự được định sẵn cho biến buf, thì việc sao chép nó vào buffer sẽ vượt quá giới hạn cho phép của nó, và ghi đè dữ liệu trong vùng nhớ lân cận. Bằng cách ghi đè “con trỏ lệnh bảo lưu” hacker có thể thực thi code hoặc gây sập hệ thống. Hacker cần phải sắp xếp shellcode ở đâu đó trên bộ nhớ stack và xác định địa chỉ bắt đầu của nó.

Cách khai thác:

Trong VD, shellcode sẽ được tổ chức và truyền qua bộ đệm buf do kích thước bộ đệm bị tràn quá nhỏ (16 byte) không đủ để đặt vừa shellcode. Khi đó địa chỉ trả về sẽ bị ghi đè bởi các mã lệnh thay vì giá trị địa chỉ cần nhảy đến. Ta truyền shellcode vào biến env là hoàn thành khai thác buffer overflow.

# **5. Trình bày khái niệm lỗ hổng đường đua (race condition), chỉ ra ví** dụ

Khái niệm: là một tình huống xảy ra khi nhiều threads cùng truy cập và cùng lúc muốn thay đổi data (có thể là một biến, một row trong CSDL, etc). Vì thuật toán chuyển đổi việc thực thi giữa các threads có thể xảy ra bất kì lúc nào, nên không thể biết được thứ tự của các threads truy cập và thay đổi data đó sẽ dẫn đến giá trị của các data sẽ không như mong muốn



Ta thấy hàm “access” và hàm “fopen” không thực hiện 2 tác vụ kiểm tra quyền và mở tập tin 1 cách không thể tách rời (atomic). Nói một cách khác, có 1 khoảng t/g ngắn giữa 2 hàm này mà OS có thể chuyển qua thực thi một tiến trình khác, rồi quay lại

B1. Tạo 1 liên kết raceexp chỉ đến 1 tập tin chúng ta có thể đọc VD như race.c

B2. Thực thi CT bị lỗi với tham số raceexp để CT này kiểm tra khả năng đọc tập tin raceexp nhưng thực chất là race.c

B3. Nếu may mắn, OS chuyển quyền thực thi lại cho tiến trình được tạo ở bước 1 ngay sau khi bước 2 hoàn thành việc kiểm tra thì ta sẽ chuyển liên kết raceexp chỉ đến tập tin race.txt

B4. OS chuyển lại tiến trình bị lỗi, hàm “fopen” sẽ mở tập tin race.txt thay vì raceexp