Họ và tên: Nguyễn Hữu Hoàng Mã SV: AT140523 Số BD: 35
Học phần: Phát hiện lỗi và lỗ hổng phần mềm
Ngày thi: 05/01/2022 Mã đề: 01

Điểm thi	CBChT1	CBChT2

Bài làm

Câu 1: Khái niệm lỗ hổng off by one, ví dụ & khai thác

Câu 2.

```
void sub (int a, int b, int *s) {
    *s = a - b;
}
int main () {
    int a, b, t;
    scanf ("%d", &a, &b);
    sub (a, b, &t)
    printf ("%d, t);
    return 0;
```

Trình bày stack frame của hàm sub, trình bày hđộng.
Trên hình thể hiện tham số hình thức của hàm, chỉ ra các tham số thực sự tương ứng

Câu 3 Lỗ hổng format-string, giải thích cơ chế ghi tùy ý

Câu 4.

```
int vun () {
  int result;
  char buf        sp+8h  bp-20h
  int, v2, v3;    sp+18 bp-10   ;   sp+1C  bp-0C)
```

```
v3 = 6000
puts ("Flag ... 6000$");
printf("your money : %d;\n", 6000);
puts ("You bet");
read ( 0, &buf, 0x Fu)
if ( strchr ( &buf, '-') ) {
    Bye hacker
    exit
}

v2 = strtoul (&buf, 0, 0)
if ( v2 > v3 ) {
    You think I'm stupid ?
    exit
}

v3 -= v2
print f ( your money %d, v3)
if ( v3 <= 6000 )
    result = puts ( bye loser);
else
    result = system( cat flag)
return result.
}
int main(){
    vun();
}
```

a) Trình bày hđ của ctrình

b) chỉ ra lỗ hổng

c) trình bày cách thức để khiến ctrình cat flag.

Ng Hữu Hoàng - AT140523

Câu 1.

Lỗi không off by one là lỗi logic thường gặp. Khi làm việc với độ dài của mảng, xảy khi người lập trình mắc lỗi khi xét từ 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 1 số làm xử lý xâu như strncat, strncpy.

Ví dụ:

```
void copy (const char * str) {
    char buf [64];
    strn.cpy (buf, str, sizeof (buf));
}
```

- Ở đây, người lập trình 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 64 ký tự hoặc lớp hơn, kẻ tấn công có thể làm tràn bộ đệm 1 byte và có thể gây ra sập chương trình.

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

Câu 3:

Lỗ hổng format string là 1 lỗ hổng phổ biến khi lập trình viên dùng các printf và các hàm liên quan sai cách. Lỗ hổng này xảy ra khi chương trình in 1 xâu nhưng xâu đó lại hiểu thành xâu 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); // dòng này có lỗi format str
}
```

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). Kẻ tấn công có thể nhập

%x %x %x

và printf sẽ in ra 3 giá trị 32 bit tiếp theo trong stack

Câu 4

a) Phân tích code:

- Chương trình đưa cho người chơi 60000 đô la.
- Chương trình yêu cầu nhập 1 số đô cược.
- Số này (khi chưa ép kiểu) không được có ký tự - (dấu trừ), ý là không cho nhập số âm
- Sử dụng strtoul (string to unsinged long) để chuyển xâu số này ra kiểu UL và ép về kiểu int (v2)
- Số người dùng nhập không được lớn hơn số tiền hiện có, nếu k

- trừ tiền của người dùng với số được nhập.
- Nếu tiền > 6000 thì in ra file flag

b) Lỗ hổng của chương trình là: tràn số nguyên integer overflow sinh ra khi ép kiểu unsigned long về kiểu int.

· kiểu int có giá trị lớn nhất là $2^{31} - 1$
= 2 147 483 647.
giá trị này được biểu diễn dưới dạng hex:
0x 7FFFFFFF

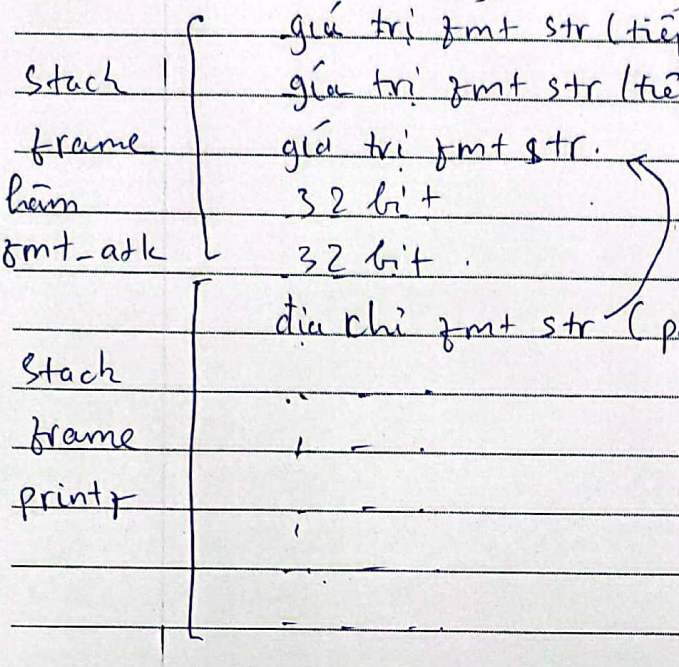
chỉ cần thêm 1 là thành:
0x 80000000 : số nhỏ nhất biểu diễn được bằng int.

Mà kiểu unsigned long có giới hạn lớn hơn rất nhiều => Nếu nhập số nguyên lớn hơn $2^{31} - 1$ là sẽ thành số âm khi ép kiểu sang int.

c)
Ta chọn số ~~âm~~ dương với giá trị lớn hơn $2^{31} - 1 + 6000$ để tránh bị tràn lần nữa
=> chọn số 3000 000 000 cho thuộc âm.
=> số này khi ép thành int có giá trị là
- 1294967295
=> Ta đã nhập được số âm mà không ~~có dấu~~ cần dấu trừ.
=> khi chương trình lấy số tiền trừ cho số trên sẽ được kết quả là:
6000 - (- 1294967295) = 1.294.973.295
vẫn trong phạm vi biểu diễn của int nên không bị tràn nữa
=> số tiền của người dùng > 6000
=> chương trình in ra flag

Câu 3 ý f:

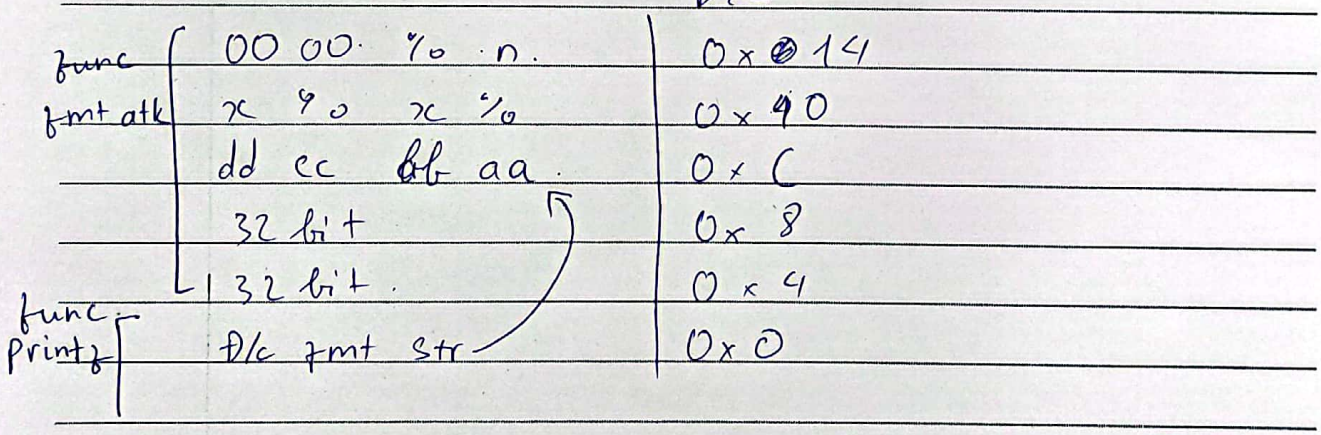
chương trình có dạng stack của app có dạng sau, mỗi dòng là 32 bit.



kẻ tấn công có thể truyền như sau

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

Lúc đó stach sẽ thành



Hàm printf sẽ in ra 4 byte đầu như thường (ascii 0x a a bb cc dd) sau đó xử lý %o x => in 4 byte tại địa chỉ 0x4, sau đó %o x => in 4 byte tại 0x8 và

Sau đó xử lý %o 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 địa chỉ dd cc bb aa

Ng Huu Hoang AT140523

Câu 2:

Stack frame của hàm sub trước khi chạy

đ/c cao		
↓ hướng phát triển của stack	sub.	
	*s	Đ/c của biến s
	b	giá trị biến b
	a	giá trị biến a
	ret addr	
	EBP	← ESP trỏ vào
đ/c, thấp		

Stack frame của hàm Sub sau khi chạy

	sub	
	*s	
	b.	
	a.	
	Đ/c của lệnh tiếp	(printf("%d", t);
	EBP	← ESP

- Trong hàm Sub, nó được truyền giá trị a, giá trị b và con trỏ (địa chỉ) s.

- Ở hàm sub lấy a - b rồi gán giá trị kết quả vào địa chỉ trỏ tới bởi con trỏ s.