

BÁO CÁO THỰC HÀNH

Lab 04: Reverse Engineering

Môn học: Lập trình hệ thống

Tên chủ đề: **Kỹ thuật dịch ngược**

GVHD: Đỗ Thị Thu Hiền

- THÔNG TIN CHUNG:**

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT334.M21.ANTN

STT	Họ và tên	MSSV	Email
1	Nguyễn Văn Tài	19520250	19520250@gm.uit.edu.vn
2	Trần Hoàng Khang	19521671	19521671@gm.uit.edu.vn

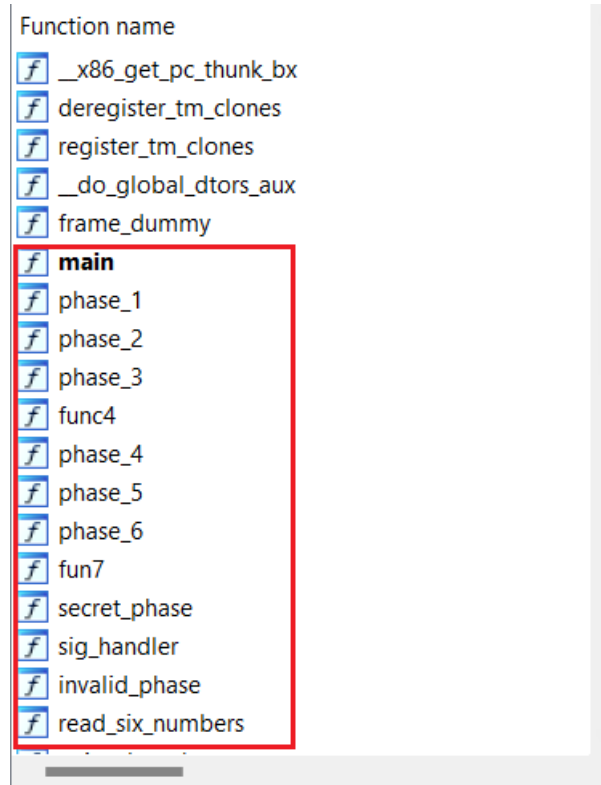
- NỘI DUNG THỰC HIỆN:¹**

STT	Công việc	Kết quả tự đánh giá
1	Phase 1	100%
2	Phase 2	100%
3	Phase 3	100%
4	Phase 4	100%
5	Phase 5	100%
6	Phase 6	100%
7	Secret Phase	

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

BÁO CÁO CHI TIẾT

Sử dụng IDA Pro để phân tích file **bomp**, và chương trình bao gồm các hàm chính sau đây:



Phân tích từng phase và tìm ra các flag cần thiết để vượt qua các yêu cầu

1. Phase_1

Xem mã giả của phase_1:

```
int __cdecl phase_1(int a1) {
    int result; // eax

    result = strings_not_equal(a1, "I was trying to give Tina Fey more material.");
    if (result)
        explode_bomb();
    return result;
}
```

Đầu vào của hàm phase_1 là biến a1 và so sánh với một chuỗi được gán sẵn trong hệ thống:

- Nếu chuỗi nhập vào giống nhau, thì sẽ vượt qua được phase_1
- Ngược lại, nhập chuỗi sai sẽ khiến quả bom kích hoạt và thất bại

Vậy flag phase_1 là: **I was trying to give Tina Fey more material.**

Kết quả:

```
(kali㉿kali)-[~/LTHT/Lab4]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
█
```

2. Phase_2

Xem mã giả phase_2

```
unsigned int __cdecl phase_2(int a1)
{
    int i; // [esp+10h] [ebp-28h]
    int v3[6]; // [esp+14h] [ebp-24h] BYREF
    unsigned int v4; // [esp+2Ch] [ebp-Ch]

    v4 = __readgsdword(0x14u);
    read_six_numbers(a1, v3);
    if ( v3[0] != 1 )
        explode_bomb();
    for ( i = 1; i <= 5; ++i )
    {
        if ( v3[i] != 2 * v3[i - 1] )
            explode_bomb();
    }
    return __readgsdword(0x14u) ^ v4;
}
```

Nhìn trông cũng khá easy, chưa tăng độ được bao nhiêu, hàm thực hiện lấy 6 số từ chuỗi nhập vào **read_six_numbers(a1, v3)**; với a1 là chuỗi input có dạng **num1 num2 num3 num4 num5 num6** và đưa 6 số này vào mảng **v3[6]**.

- **v3[0] != 1** thì bomb nổ => **v3[0] = 1**
- Đưa vào vòng lặp 5 lần, với mỗi số phía trước gấp 2 lần số sau, nếu không sẽ nổ

```
if ( v3[i] != 2 * v3[i - 1] )
    explode_bomb();
```

Vậy các số mình nhập vào chỉ cần thỏa mã điều kiện này và số đầu tiên phải **bằng 1**.

Ví dụ một chuỗi input hợp lệ cho phase 2: **1 2 4 8 16 32**

Test thực nghiệm:

```
(virus@virus)-[~/Desktop]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
█
```

Ngoài ra hàm không kiểm tra gì thêm mà chỉ kiểm tra với 6 số đầu, vậy nên nếu ta có nhập thêm thì cũng chả bị sao, hihi 😊

Ví dụ như: **1 2 4 8 16 32 64 128** cũng là một chuỗi hợp lệ

Vậy valid input có dạng tổng quát: **1 2 4 8 16 32 (64 128 ... 2^n)**

```
(virus@virus)-[~/Desktop]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32 64 128 256 512 1024
That's number 2. Keep going!
█
```

3. Phase_3

Xem psedocode của phase_3:

```
unsigned int __cdecl phase_3(int a1)
{
    unsigned int result; // eax
    int v2; // [esp+1Ch] [ebp-1Ch] BYREF
    int v3; // [esp+20h] [ebp-18h] BYREF
    int v4; // [esp+24h] [ebp-14h]
    int v5; // [esp+28h] [ebp-10h]
    unsigned int v6; // [esp+2Ch] [ebp-Ch]

    v6 = __readgsdword(0x14u);
    v4 = 0;
    v5 = 0;
    v5 = __isoc99_sscanf(a1, "%d %d", &v2, &v3);
    if (v5 <= 1)
        explode_bomb();
    switch (v2)
    {
    case 0:
```

```
    v4 += 431;
    goto LABEL_5;
case 1:
LABEL_5:
    v4 -= 858;
    goto LABEL_6;
case 2:
LABEL_6:
    v4 += 437;
    goto LABEL_7;
case 3:
LABEL_7:
    v4 -= 578;
    goto LABEL_8;
case 4:
LABEL_8:
    v4 += 578;
    goto LABEL_9;
case 5:
LABEL_9:
    v4 -= 578;
    goto LABEL_10;
case 6:
LABEL_10:
    v4 += 578;
    break;
case 7:
    break;
default:
    explode_bomb();
    return result;
}
v4 -= 578;
if ( v2 > 5 || v4 != v3 )
    explode_bomb();
return __readgsdword(0x14u) ^ v6;
}
```

Ở phase này, chương trình sẽ đọc vào 2 số nguyên (v2 và v3).

- v2 nằm trong khoảng giá trị từ [0, 5], nếu không sẽ kích hoạt quả bom
- v4 sẽ được tính toán tương ứng với từng giá trị của v2 và so sánh với giá trị của v3 (nếu khác nhau thì bom sẽ nổ)

Như vậy thì ứng với mỗi giá trị của v2 thì sẽ có một cặp kết quả {v2, v3} hợp lệ

⇒ Có tất cả 6 input đầu vào hợp lệ để bypass qua phase_3 này

Các input hợp lệ:

v2	v4	{v2, v3}
0	-568	0 -568
1	-999	1 -999
2	-141	2 -141
3	-578	3 -578
4	0	4 0
5	-578	5 -578

Kết quả:

```
(kali㉿kali)-[~/LTHT/Lab4]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
1 -999
Halfway there!
█
```

4. Phase_4

Xem mã giả phase_4

```
unsigned int __cdecl phase_4(int a1)
{
    int v2; // [esp+18h] [ebp-20h] BYREF
    int v3; // [esp+1Ch] [ebp-1Ch] BYREF
    int v4; // [esp+20h] [ebp-18h]
    int v5; // [esp+24h] [ebp-14h]
    int v6; // [esp+28h] [ebp-10h]
    unsigned int v7; // [esp+2Ch] [ebp-Ch]

    v7 = __readgsdword(0x14u);
    v4 = __isoc99_sscanf(a1, "%d %d", &v2, &v3);
    if ( v4 != 2 || v2 < 0 || v2 > 14 )
        explode_bomb();
    v5 = 15;
```



```
v6 = func4(v2, 0, 14);  
if ( v6 != v5 || v3 != v5 )  
    explode_bomb();  
return __readgsdword(0x14u) ^ v7;  
}
```

Tại phase_4, hàm `__isoc99_sscanf(a1, "%d %d", &v2, &v3)` có chức năng tương tự như trong phase_3 -> Đọc 2 số input đầu vào:

```
if ( v4 != 2 || v2 < 0 || v2 > 14 )  
    explode_bomb();
```

- Kiểm tra **số lượng input đầu vào != 2** => Bomb nổ
- **v2 nằm trong khoảng 0 <= v2 <= 14**

```
v5 = 15;  
v6 = func4(v2, 0, 14);
```

- Set giá trị **v5 = 15**
- **v6** là giá trị trả về của hàm `func4(v2, 0, 14)` với các giá trị truyền vào như trên

```
if ( v6 != v5 || v3 != v5 )  
    explode_bomb();
```

Nếu giá trị trả về **v6 != 15 (v5)** và **v3 != 15** => Nổ

Xem hàm ***func4()***

```
int __cdecl func4(int a1, int a2, int a3)  
{  
    int v4; // [esp+Ch] [ebp-Ch]  
  
    v4 = (a3 - a2) / 2 + a2;  
    if ( v4 > a1 )  
        return func4(a1, a2, v4 - 1) + v4;  
    if ( v4 >= a1 )  
        return (a3 - a2) / 2 + a2;  
    return func4(a1, v4 + 1, a3) + v4;  
}
```

Cơ bản thì *func4* là một hàm có liên quan đến giải thuật đệ quy, nhưng ta không quan tâm lắm, mục đích là tìm giá trị **v2** đầu vào ứng với tham số thứ 1 (**int a1**) của hàm *func4* để đầu ra giá trị trả về là 15 và được gán vào v6, còn điều kiện **v3 = 15** mình có thể set bằng tay. Vì giá trị của **v2 hữu hạn, số nguyên và nhỏ** - trong khoảng **[0, 14]**. Vậy ta có thể chạy chương trình vét cạn và xem trường hợp khả thi. Code solve:

```
def func4(a1, a2, a3):  
    v4 = (a3 - a2) / 2 + a2  
    if v4 > a1:  
        return func4(a1, a2, v4 - 1) + v4  
    if v4 >= a1:  
        return (a3 - a2) / 2 + a2  
    return func4(a1, v4 + 1, a3) + v4  
  
for v2 in range(15):  
    print(str(func4(v2, 0, 14)) + " " + str(v2))
```

Giá trị output:

```
16:58:11 Lab4 38ms  
→python .\solve.py  
11.0 0  
11.0 1  
13.0 2  
10.0 3  
19.0 4  
15.0 5  
21.0 6  
7.0 7  
35.0 8  
10.0 3  
19.0 4  
15.0 5  
21.0 6  
7.0 7  
35.0 8  
27.0 9  
37.0 10  
18.0 11  
43.0 12  
31.0 13  
45.0 14
```

Có giá trị $v2 = 5$ thì output bằng 15, tức là $v6 = 15$. Vậy cặp giá trị này thỏa điều kiện, $v3$ thì mình set bằng input luôn là 15. Vậy giá trị input hợp lệ cho phase 4 là: **5 15**

Test thực nghiệm:


```
(virus@virus)-[~/Desktop]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32 64 128 256 512 1024
That's number 2. Keep going!
0 -568
Halfway there!
5 15
So you got that one. Try this one.
```

5. Phase_5

Xem pseudocode của phase_5:

```
int __cdecl phase_5(int a1)
{
    int result; // eax
    int i; // [esp+4h] [ebp-14h]
    int v3; // [esp+8h] [ebp-10h]

    result = string_length(a1);
    if ( result != 6 )
        explode_bomb();
    v3 = 0;
    for ( i = 0; i <= 5; ++i )
    {
        result = array_2705[(int)(i + a1) & 0xF];
        v3 += result;
    }
    if ( v3 != 48 )
        explode_bomb();
    return result;
}
```

Ở phase này, input đầu vào là một chuỗi gồm 6 ký tự để thỏa mãn câu lệnh **if** đầu tiên.

Ở vòng lặp **for**, ta có luồng hoạt động như sau

- Từng ký tự của chuỗi input đầu vào sẽ được **&** với **0xF**
- Kết quả nhận được sẽ là **index** để truy xuất tới phần tử của mảng **array_2705**

Mảng **array_2705** chứa các giá trị:

```
.data:0804C1C0 ; int array_2705[16]
.data:0804C1C0 array_2705      dd 2, 0Ah, 6, 1, 0Ch, 10h, 9, 3, 4, 7, 0Eh, 5, 0Bh, 8
.data:0804C1C0                                     ; DATA XREF: phase_5+43↑r
.data:0804C1C0      dd 0Fh, 0Dh
.data:0804C200      public host_table
```

- Cộng giá trị của phần tử này vào **v3**
- Nếu tổng của các phần tử tìm được bằng 48 ($v3 = 48$) thì sẽ vượt qua phase_5 này, ngược lại sẽ nổ bom.

Ta viết một đoạn code để tìm chuỗi cần nhập:

```
a = '000000'
while(len(a) == 6):
    v3 = 0
    for i in range(0, 6):
        res = arr[int(a[i]) & 0xF]
        v3 += res
    if v3 == 48:
        print(a)

    a = str(int(a) + 1)
    if(len(a) != 6):
        a = str(0)*(6 - len(a)) + a
```

Kết quả: có rất nhiều kết quả phù hợp với điều kiện của phase_5 này (từ 000000 -> 999999)

```
001254
001414
001425
001441
001452
001505
001524
001542
001550
001566
001656
```

Từ kết quả xuất hiện khi chạy đoạn code trên, ta chỉ cần lấy 1 giá trị là qua được phase này

Vậy flag là: **001414**

6. Phase_6

Xem psedocode của phase_6, khá phức tạp:

```
unsigned int __cdecl phase_6(int a1)
{
    _DWORD *v2; // [esp+1Ch] [ebp-4Ch]
    int v3; // [esp+1Ch] [ebp-4Ch]
    int v4; // [esp+1Ch] [ebp-4Ch]
    int i; // [esp+20h] [ebp-48h]
    int k; // [esp+20h] [ebp-48h]
```

```
int m; // [esp+20h] [ebp-48h]
int n; // [esp+20h] [ebp-48h]
int j; // [esp+24h] [ebp-44h]
int l; // [esp+24h] [ebp-44h]
int v11; // [esp+28h] [ebp-40h]
int v12[6]; // [esp+2Ch] [ebp-3Ch] BYREF
int v13[6]; // [esp+44h] [ebp-24h]
unsigned int v14; // [esp+5Ch] [ebp-Ch]

v14 = __readgsdword(0x14u);
read_six_numbers(a1, (int)v12);
for ( i = 0; i <= 5; ++i )
{
    if ( v12[i] <= 0 || v12[i] > 6 )
        explode_bomb();
    for ( j = i + 1; j <= 5; ++j )
    {
        if ( v12[i] == v12[j] )
            explode_bomb();
    }
}
for ( k = 0; k <= 5; ++k )
{
    v2 = &node1;
    for ( l = 1; v12[k] > l; ++l )
        v2 = (_DWORD *)v2[2];
    v13[k] = (int)v2;
}
v11 = v13[0];
v3 = v13[0];
for ( m = 1; m <= 5; ++m )
{
    *(_DWORD *)(v3 + 8) = v13[m];
    v3 = *(_DWORD *)(v3 + 8);
}
*(_DWORD *)(v3 + 8) = 0;
v4 = v11;
for ( n = 0; n <= 4; ++n )
{
    if ( *(_DWORD *)v4 > **(_DWORD **)(v4 + 8) )
        explode_bomb();
    v4 = *(_DWORD *)(v4 + 8);
}
return __readgsdword(0x14u) ^ v14;
}
```

Phân tích code:

❖ Phần xử lý input:

```
v14 = __readgsdword(0x14u);
read_six_numbers(a1, (int)v12);
for ( i = 0; i <= 5; ++i )
{
    if ( v12[i] <= 0 || v12[i] > 6 )
        explode_bomb();
    for ( j = i + 1; j <= 5; ++j )
    {
        if ( v12[i] == v12[j] )
            explode_bomb();
    }
}
```

- Chương trình gọi hàm **read_six_numbers ()** như ở phase_2 và lưu vào biến **a1** => Input đầu vào là 6 số
 - Ở vòng **for**, chương trình thực hiện các điều kiện input:
 - Giới hạn giá trị input nằm trong khoảng từ [0, 6]
 - Không có biến nào có giá trị giống nhau trong một lần nhập
- ⇒ Nếu không phù hợp các điều kiện trên thì bom sẽ được kích hoạt

❖ Các phần tiếp theo khá khó hiểu, nên tiến hành phân tích thông qua code assembly

Sử dụng **gdb-peda** để debug.

Theo như đoạn mã pseudocode trên, thì gần cuối chương trình mới kiểm tra điều kiện cụ thể.

```
v4 = v11;
for ( n = 0; n <= 4; ++n )
{
    if ( *(_DWORD *)v4 > **(_DWORD **)(v4 + 8) )
        explode_bomb();
    v4 = *(_DWORD *)(v4 + 8);
}
```

Tiến hành debug:

Xem code assembly của phase_6 bằng lệnh **disass phase_6**

```

0x08049036 <+284>: jmp 0x804905b <phase_6+321>
0x08049038 <+286>: mov eax,DWORD PTR [ebp-0x4c]
0x0804903b <+289>: mov edx,DWORD PTR [eax]
0x0804903d <+291>: mov eax,DWORD PTR [ebp-0x4c]
0x08049040 <+294>: mov eax,DWORD PTR [eax+0x8]
0x08049043 <+297>: mov eax,DWORD PTR [eax]
0x08049045 <+299>: cmp edx,eax
0x08049047 <+301>: jle 0x804904e <phase_6+308>
0x08049049 <+303>: call 0x80494ba <explode_bomb>
0x0804904e <+308>: mov eax,DWORD PTR [ebp-0x4c]
0x08049051 <+311>: mov eax,DWORD PTR [eax+0x8]
0x08049054 <+314>: mov DWORD PTR [ebp-0x4c],eax
0x08049057 <+317>: add DWORD PTR [ebp-0x48],0x1
0x0804905b <+321>: cmp DWORD PTR [ebp-0x48],0x4
0x0804905f <+325>: jle 0x8049038 <phase_6+286>
0x08049061 <+327>: nop
0x08049062 <+328>: mov eax,DWORD PTR [ebp-0xc]
0x08049065 <+331>: xor eax,DWORD PTR gs:0x14
0x0804906c <+338>: je 0x8049073 <phase_6+345>
0x0804906e <+340>: call 0x8048830 <__stack_chk_fail@plt>
0x08049073 <+345>: leave

```

Đây là đoạn chương trình kiểm tra, ta có thể phác họa chương trình hoạt động như sau:

- Các giá trị được đưa vào hai thanh ghi **eax** và **edx**
- So sánh **edx** và **eax**, nếu giá trị **edx** nhỏ hơn **eax** thì sẽ thực hiện tiếp vòng lặp, nếu không thì sẽ kích hoạt quả bom

Đặt breakpoint và debug đoạn chương trình này:

- Truyền vào phase_6 một đoạn input là "1 2 3 4 5 6"

```

[-----stack-----]
0000| 0xffffd070 → 0xf7fa6d20 → 0xfbad2a84
0004| 0xffffd074 → 0x804d1a0 ("Good work! On to the next... \n one.\none?\ny!\n 6 phases wi
th\n")
0008| 0xffffd078 → 0x1e
0012| 0xffffd07c → 0x804c5d0 ("1 2 3 4 5 6")
0016| 0xffffd080 → 0x1
0020| 0xffffd084 → 0x8048990 (<_start>: xor ebp,ebp)
0024| 0xffffd088 → 0xf7de7e65 (<__ctype_b_loc+5>: add eax,0x1be19b)
0028| 0xffffd08c → 0x804c100 → 0x124
[-----]
Legend: code, data, rodata, value

```

- Thanh ghi **eax** lưu địa chỉ tại **[ebp-0x4c]** sau đó gán giá trị lưu tại **eax** vào **edx**, rồi cho **eax** chứa giá trị được lưu tại địa chỉ **[eax+0x8]** và so sánh hai giá trị lưu trong **eax** và **edx**

```

[-----registers-----]
EAX: 0x804c100 → 0x124
EBX: 0xffffd120 → 0x1
ECX: 0xffffcaf1 → 0x0
EDX: 0x124
ESI: 0x1
EDI: 0x8048990 (<_start>: xor    ebp,ebp)
EBP: 0xffffd0d8 → 0xffffd108 → 0x0
ESP: 0xffffd070 → 0xf7fa6d20 → 0xfbad2a84
EIP: 0x8049040 (<phase_6+294>: mov    eax,DWORD PTR [eax+0x8])
EFLAGS: 0x297 (CARRY PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x8049038 <phase_6+286>: mov    eax,DWORD PTR [ebp-0x4c]
0x804903b <phase_6+289>: mov    edx,DWORD PTR [eax]
0x804903d <phase_6+291>: mov    eax,DWORD PTR [ebp-0x4c]
⇒ 0x8049040 <phase_6+294>: mov    eax,DWORD PTR [eax+0x8]
0x8049043 <phase_6+297>: mov    eax,DWORD PTR [eax]
0x8049045 <phase_6+299>: cmp    edx,eax
0x8049047 <phase_6+301>: jle    0x804904e <phase_6+308>
0x8049049 <phase_6+303>: call   0x80494ba <explode_bomb>
[-----stack-----]

```

- Ta thấy địa chỉ 0x804c100 (**eax** đang giữ) là trỏ đến **node1**

```

gdb-peda$ x/w 0x804c100
0x804c100 <node1>:      0x00000124

```

- Tiếp tục chương trình, lúc này giá trị **eax** đã thay đổi

```

[-----registers-----]
EAX: 0x804c0f4 → 0x399
EBX: 0xffffd120 → 0x1
ECX: 0xffffcaf1 → 0x0
EDX: 0x124
ESI: 0x1
EDI: 0x8048990 (<_start>: xor    ebp,ebp)
EBP: 0xffffd0d8 → 0xffffd108 → 0x0
ESP: 0xffffd070 → 0xf7fa6d20 → 0xfbad2a84
EIP: 0x8049043 (<phase_6+297>: mov    eax,DWORD PTR [eax])
EFLAGS: 0x297 (CARRY PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-----code-----]
0x804903b <phase_6+289>: mov    edx,DWORD PTR [eax]
0x804903d <phase_6+291>: mov    eax,DWORD PTR [ebp-0x4c]
0x8049040 <phase_6+294>: mov    eax,DWORD PTR [eax+0x8]
⇒ 0x8049043 <phase_6+297>: mov    eax,DWORD PTR [eax]
0x8049045 <phase_6+299>: cmp    edx,eax
0x8049047 <phase_6+301>: jle    0x804904e <phase_6+308>
0x8049049 <phase_6+303>: call   0x80494ba <explode_bomb>
0x804904e <phase_6+308>: mov    eax,DWORD PTR [ebp-0x4c]

```

- Và **eax** đang trỏ đến **node2**

```

gdb-peda$ x/w 0x804c0f4
0x804c0f4 <node2>:      0x00000399

```

- Lúc này giá trị **eax** là 0x399 và **edx** là 0x124


```

[----- registers -----]
EAX: 0x399
EBX: 0xffffd120 → 0x1
ECX: 0xffffcaf1 → 0x0
EDX: 0x124
ESI: 0x1
EDI: 0x8048990 (<_start>:      xor    ebp,ebp)
EBP: 0xffffd0d8 → 0xffffd108 → 0x0
ESP: 0xffffd070 → 0xf7fa6d20 → 0xfbad2a84
EIP: 0x8049047 (<phase_6+301>:  jle    0x804904e <phase_6+308>)
EFLAGS: 0x297 (CARRY PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[----- code -----]

```

Vì **edx** < **eax** nên chương trình tiếp tục chạy

- Tiếp theo, chương trình sẽ gán cho **eax** giá trị lưu tại **node3** và **edx** giá trị tại **node2**

```

EAX: 0x804c0e8 → 0x22c
EBX: 0xffffd120 → 0x1
ECX: 0xffffcaf1 → 0x0
EDX: 0x399
ESI: 0x1
EDI: 0x8048990 (<_start>:      xor    ebp,ebp)
EBP: 0xffffd0d8 → 0xffffd108 → 0x0
ESP: 0xffffd070 → 0xf7fa6d20 → 0xfbad2a84
EIP: 0x8049043 (<phase_6+297>:  mov    eax,DWORD PTR [eax])
EFLAGS: 0x293 (CARRY parity ADJUST zero SIGN trap INTERRUPT direction overflow)
[----- code -----]
0x804903b <phase_6+289>:  mov    edx,DWORD PTR [eax]
0x804903d <phase_6+291>:  mov    eax,DWORD PTR [ebp-0x4c]
0x8049040 <phase_6+294>:  mov    eax,DWORD PTR [eax+0x8]
⇒ 0x8049043 <phase_6+297>:  mov    eax,DWORD PTR [eax]
0x8049045 <phase_6+299>:  cmp    edx,eax
0x8049047 <phase_6+301>:  jle    0x804904e <phase_6+308>
0x8049049 <phase_6+303>:  call   0x80494ba <explode_bomb>
0x804904e <phase_6+308>:  mov    eax,DWORD PTR [ebp-0x4c]
[----- stack -----]
0000| 0xffffd070 → 0xf7fa6d20 → 0xfbad2a84
0004| 0xffffd074 → 0x804d1a0 ("Good work!  On to the next ... \n one.\none?\ny!\n 6 phases wi
th\n")
0008| 0xffffd078 → 0x1e
0012| 0xffffd07c → 0x804c5d0 ("1 2 3 4 5 6")
0016| 0xffffd080 → 0x1
0020| 0xffffd084 → 0x8048990 (<_start>:      xor    ebp,ebp)
0024| 0xffffd088 → 0xf7de7e65 (<__ctype_b_loc+5>:  add    eax,0x1be19b)
0028| 0xffffd08c → 0x804c0f4 → 0x399
[-----]
Legend: code, data, rodata, value
0x8049043 in phase_6 ()
gdb-peda$ x/w $eax
0x804c0e8 <node3>:      0x0000022c
gdb-peda$

```

- Vì **eax=0x22c** < **edx=0x399** nên quả bom đã được kích hoạt

Sau khi phân tích trên, ta rút được các kết luận:

- Giá trị nhập vào sẽ liên quan đến **node** được sử dụng
- Để không kích hoạt quả bom, ta cần sắp xếp giá trị các **node** theo giá trị tăng dần

Sau một hồi debug, ta tìm được giá trị của các **node** lần lượt là:

node1 có giá trị **124**

node2 có giá trị **399**

node3 có giá trị **22c**

node4 có giá trị **0ed**

node5 có giá trị **1c1**

node6 có giá trị **15d**

Như vậy thứ tự sắp xếp lần lượt là: **node4 -> node1 -> node6-> node5-> node3 -> node2**

Tương tự, flag cần nhập là: **4 1 6 5 3 2**

Kết quả:

```
(kali㉿kali)-[~/LTHT/Lab4]
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2. Keep going!
1 -999
Halfway there!
5 15
So you got that one. Try this one.
021815
Good work! On to the next ...
4 1 6 5 3 2
Congratulations! You've defused the bomb!
```