

5

Lab

Kỹ thuật Dịch ngược (tt)

Reverse Engineering (cont)

Thực hành Lập trình Hệ thống

Lưu hành nội bộ

A. TỔNG QUAN

A.1 Mục tiêu

- Tìm hiểu và làm quen với Kỹ thuật Dịch ngược (tiếp theo).
- Cài đặt và tìm hiểu trình Disassembler IDA.
- Tìm hiểu cách remote debugger trên desktop (*không bắt buộc*).

A.2 Môi trường

A.2.1 Chuẩn bị môi trường

- 01 máy Linux (máy ảo) dùng để làm môi trường chạy file cần phân tích.
- 01 máy Windows chạy chương trình IDA Pro (6.6 hoặc 7.0) cho quá trình phân tích.

A.2.2 Các tập tin được cung cấp sẵn

- File cần phân tích: **nt209-uit-bomb** – là một file thực thi Linux 32-bit.

A.3 Liên quan

- Sinh viên cần vận dụng kiến thức trong Chương 3 (Lý thuyết).
- Tìm hiểu thêm các kiến thức về remote debugger trên desktop khi sử dụng IDA.

B. THỰC HÀNH

Bài thực hành này yêu cầu sinh viên thực hiện phân tích file thực thi **nt209-uit-bomb**. File thực thi Linux 32-bit này được thiết kế như 1 trò chơi phá bomb. Quả bomb này gồm nhiều pha, mỗi pha yêu cầu nhập vào một input riêng để ngắt pha, nếu sai bomb sẽ nổ và trò chơi kết thúc. Sinh viên cần áp dụng kỹ thuật dịch ngược và sử dụng IDA để tìm ra các input để ngắt pha và ngăn quả bomb phát nổ.

B.1 Yêu cầu 1 - Thiết lập môi trường

B.1.1 Tuỳ chọn 1 – Thiết lập môi trường phân tích tĩnh file

Sinh viên có thể sử dụng môi trường đã được hướng dẫn thiết lập từ bài thực hành 4 – Kỹ thuật dịch ngược cơ bản để phân tích file **nt209-uit-bomb** trong bài thực hành này. Việc phân tích file bomb sẽ thực hiện trên IDA Pro ở máy Windows và thực thi thử nghiệm trên máy Linux.

Sinh viên cần đảm bảo:

- **Với máy Windows:**
 - + Cài đặt được IDA Pro trên máy Windows dùng để phân tích file thực thi.
 - + Trên máy Windows có file **nt209-uit-bomb** và mở được với IDA Pro phiên bản phù hợp.
- **Với máy Linux:**
 - + Thiết lập được môi trường thực thi file 32-bit trên máy ảo Linux.

Lab 5: Kỹ thuật dịch ngược (tt)

B.1.2 Tuỳ chọn 2 - Thiết lập môi trường remote debug

Ở tuỳ chọn này, sinh viên có thể thiết lập môi trường phân tích nâng cao với tính năng debug file của IDA Pro. Tuy nhiên do cần phân tích file thực thi Linux 32-bit (không chạy được trên Windows), cần thiết lập môi trường **Remote debug** kết nối từ IDA Pro trên máy Windows sang tiến trình thực thi file cần phân tích trên máy Linux.

Bước 1. Kiểm tra phiên bản môi trường Linux và cài đặt package nếu cần thiết

Bước kiểm tra này đảm bảo môi trường Linux có thể thực thi file **nt209-uit-bomb** (32-bit) và để xác định các file IDA Pro cần thiết cho quá trình thiết lập môi trường.

Sau bước này, cần đảm bảo thực thi được file bomb với lệnh:

```
$ chmod +x nt209-uit-bomb
$ ./nt209-uit-bomb
```

Bước 2. Kiểm tra kết nối giữa máy ảo Linux và máy Windows

Lưu ý: Máy ảo Linux ví dụ dưới đây được tạo với VMWare Workstation. Các máy ảo có kết mạng dạng NAT, Bridged, Host-only hoặc Custom đều có thể kết nối đến máy thật.

Trên terminal của máy ảo, gõ lệnh sau để hiển thị các IP hiện có của máy ảo:

```
$ ifconfig
```

Trên máy Windows, mở terminal và gõ lệnh sau để xem IP:

```
$ ipconfig
```

Quan sát kết quả hiển thị, tìm 2 IP thuộc cùng 1 mạng của máy thật và máy ảo. Ví dụ ở hình trên là 2 IP là **192.168.6.128** và **192.168.6.1** (Cùng mạng Vmnet8 – NAT của VMWare). Thực hiện lệnh **ping** để kiểm tra, nếu ping thành công thì 2 máy đã được kết nối.

Lab 5: Kỹ thuật dịch ngược (tt)

Khi đó IP bên máy ảo có thể sử dụng ở **Bước 7** khi kết nối IDA Pro với môi trường thực thi trên máy ảo.

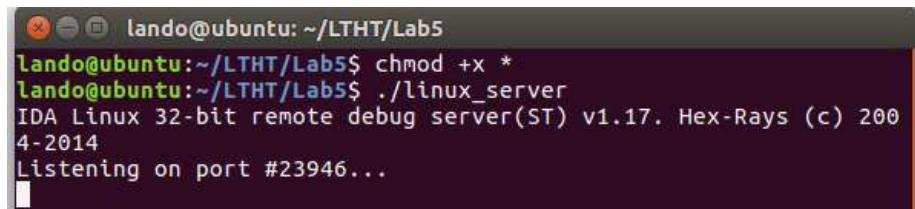
Bước 3. Sao chép file **bomb** vào máy ảo Linux ở một thư mục nhất định.

Bước 4. Vào thư mục cài đặt trên Windows của IDA, thường là **C:\Program Files** hay **C:\Program Files (x86)**. Trong thư mục **dbgsrv** sao chép file **linux_server** vào cùng thư mục trên máy Linux với file **nt209-uit-bomb**.

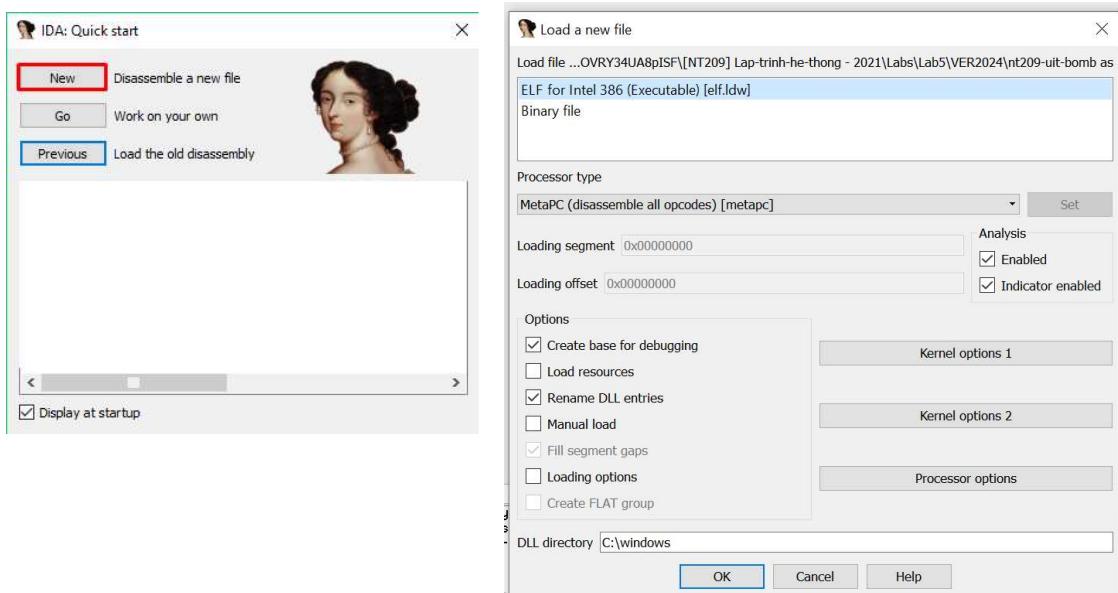


Bước 5. Trên máy Linux, với terminal ở thư mục chứa **nt209-uit-bomb** và **linux_server** của IDA, thực hiện các lệnh bên dưới để mở debug server bên máy Linux.

```
$ chmod +x *           # chỉ cần chạy 1 lần
$ ./linux_server
```

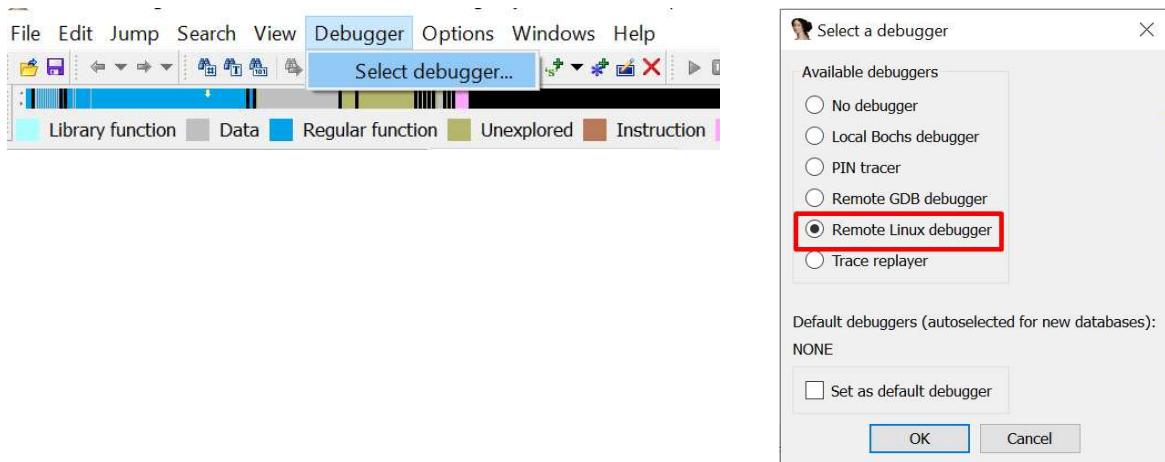


Bước 6. Trên Windows, chạy IDA bản 32 bit (idaq.exe) và mở file **nt209-uit-bomb**.



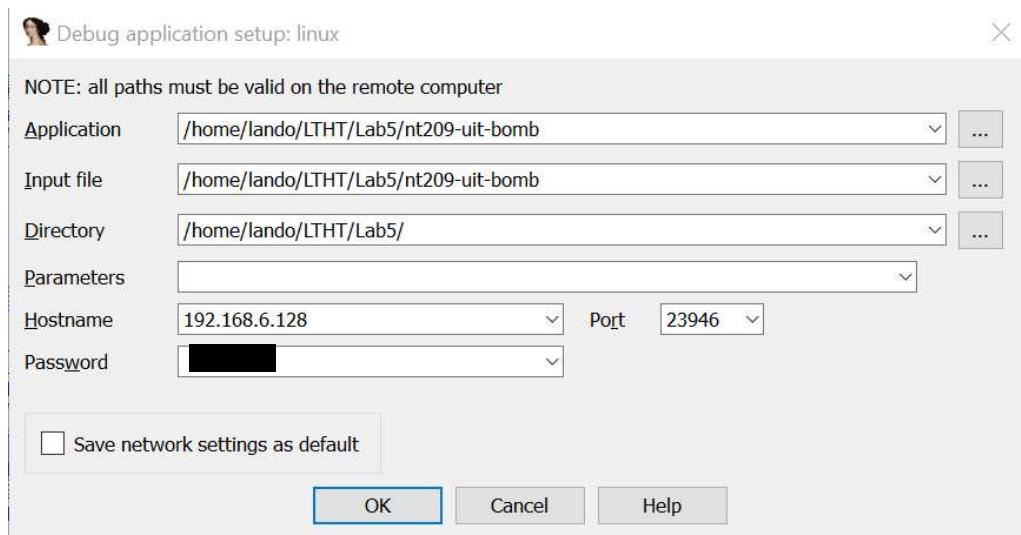
Bước 7. Sau khi load file thành công, chọn tab **Debugger** → **Select Debugger** và click chọn **Remote Linux Debugger** để thiết lập môi trường debug đến máy đích là máy Linux ở xa.

Lab 5: Kỹ thuật dịch ngược (tt)

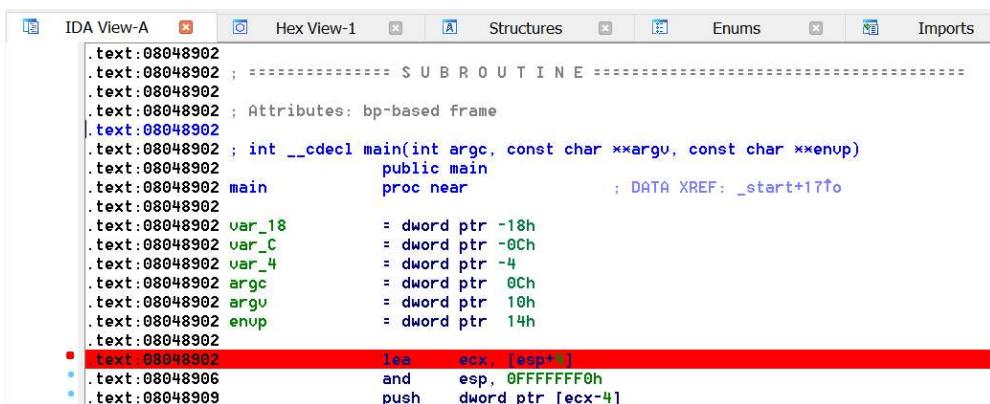


Sau đó, mở cửa sổ cài đặt debugger với **Debugger → Process Options** và điền các thông tin như sau:

- Directory: đường dẫn tuyệt đối chứa file **nt209-uit-bomb** trên máy Linux
- Application và Input file: đường dẫn tuyệt đối đến file **nt209-uit-bomb** trên máy Linux
- Hostname: địa chỉ IP của máy ảo Linux đã tìm thấy ở **Bước 2** (port để mặc định)
- Password: password tài khoản đăng nhập của máy ảo Linux.



Bước 8. Thủ đặt breakpoint tại lệnh assembly đầu tiên của hàm **main** để debug.



Lab 5: Kỹ thuật dịch ngược (tt)

Bước 9. Lưu ý: cần đảm bảo đang thực thi file **linux_server** trên máy ảo Linux như ở **Bước 7** trước khi thực hiện bước này.

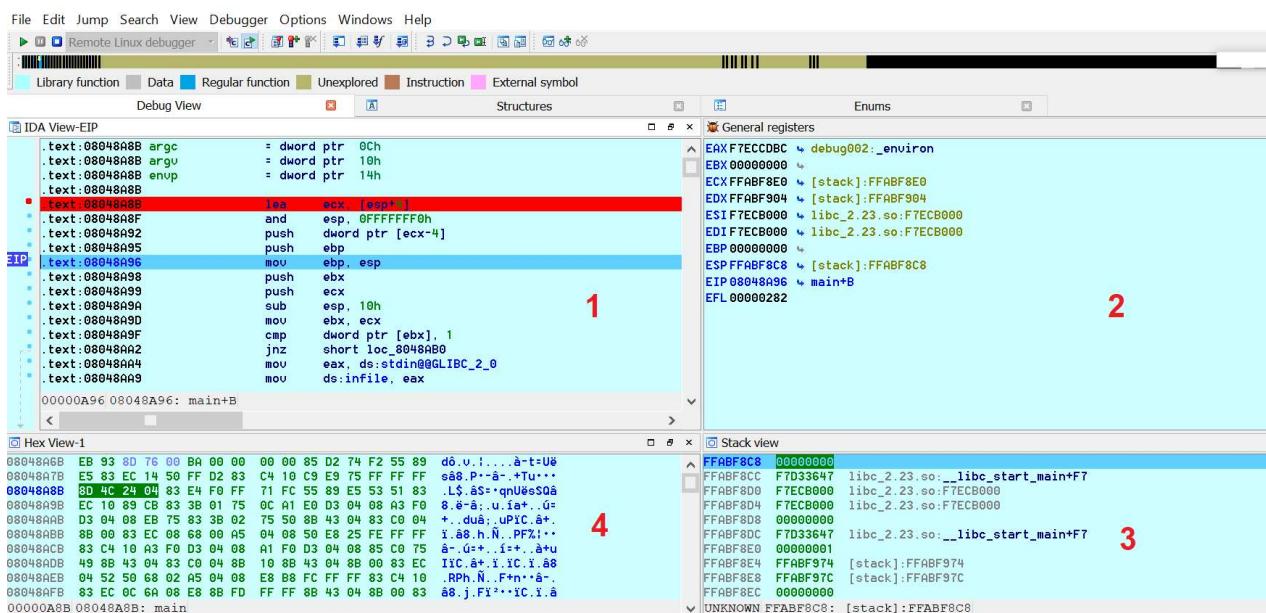
Chạy thử chương trình trên IDA Pro với dấu để kiểm tra remote debug. Nếu quan sát thấy các kết quả dưới đây thì quá trình thiết lập môi trường remote debug đã hoàn tất.

- **Trên máy ảo Linux:** có thông báo nhận được kết nối từ máy thật

IDA Linux 32-bit remote debug server(ST) v1.17. Hex-Rays (c) 2004-2014
Listening on port #23946...
=====

[1] Accepting connection from 192.168.6.1...

- **Giao diện IDA Pro:** chuyển sang màn hình nền xanh dương nhạt



Quan sát giao diện trên IDA Pro khi chuyển sang màu xanh nhạt của chế độ debug sẽ có nhiều cửa sổ mô tả trạng thái của hệ thống khi chạy chương trình **nt209-uit-bomb**:

- Cửa sổ **(1) – Debug view** hiển thị mã assembly, cùng vị trí đặt breakpoint (dòng tô màu đỏ) và vị trí đang thực hiện debug (dòng tô màu xanh dương).
- Cửa sổ **(2) – General registers** hiển thị giá trị hiện tại của các thanh ghi.
- Cửa sổ **(3) – Stack view** hiển thị trạng thái của stack khi thực thi chương trình.
- Cửa sổ **(4) – Hex view** hiển thị dạng mã hex nội dung file thực thi.

Trên IDA, dùng icon trên thanh công cụ để xem hoạt động qua từng dòng code của các hàm. Lưu ý: với các hàm phổ biến ta đã biết chức năng của chúng, ở lệnh call tương ứng với thể dùng ký hiệu để thực thi mà không cần vào đoạn code cụ thể của từng hàm.

B.2 Yêu cầu 2 – Thực hành giải các pha (phá bom)

B.2.1 Tổng quan về các pha

File thực thi **nt209-uit-bomb** là một chương trình dạng command line, đã được lập trình sẵn và có hoạt động như một quả bom. Quả bom này gồm 5 pha, mỗi pha yêu cầu một input riêng để ngắt pha và có 1 số gợi ý về input. Nếu nhập input sai, bomb sẽ nổ và trò chơi kết thúc.

```
lando@ubuntu:~/LTHT/Lab5$ ./nt209-uit-bomb
Welcome to UIT's bomb lab.
You have to deactivate our bomb by solving 5 phases with the correct inputs consecutively, and otherwise the bomb will be blown up!

[*] Phase 1
- Hint: Numbers are always magical!
1
BOMB!!!!
The bomb has blown up. Try again.
lando@ubuntu:~/LTHT/Lab5$
```

Trong trường hợp nhập đúng input, chương trình sẽ in ra thông báo chúc mừng tương ứng và chờ nhập input cho pha kế tiếp.

```
Welcome to UIT's bomb lab.
You have to deactivate our bomb by solving 5 phases with the correct inputs consecutively, and otherwise the bomb will be blown up!

[*] Phase 1
- Hint: Numbers are always magical!
Good job! You've cleared the first phase!

[*] Phase 2
- Hint: You must answer your secret question!
```

Các pha có thể được cấu trúc lại thành các nhóm dựa trên độ khó như sau:

- Bộ pha cơ bản: Pha 1, 2, 3
- Bộ pha trung bình - khó: Pha 4 và 5

Độ khó của mỗi pha được đánh giá bằng dấu (*) như sau:

- Pha 1: **
- Pha 2: **
- Pha 3: ***
- Pha 4: *****
- Pha 5: *****

Trong phần này, sinh viên được yêu cầu giải mỗi pha của quả bom khi thực thi file “nt209-uit-bomb” và nộp kèm báo cáo quá trình phân tích, thực hiện.

B.2.2 Hướng dẫn cách giải các pha

Lưu ý: Trong bài thực hành sẽ có **nhiều phiên bản** file nt209-uit-bomb khác nhau, bài hướng dẫn sử dụng 1 phiên bản bất kì để phân tích.

- **Bước 1: Thực thi thử file nt209-uit-bomb trên Linux** với lệnh:

```
./nt209-uit-bomb
```

Sau một số thông báo chào mừng, có thể thấy file in ra gợi ý liên quan đến số của pha 1 và tạm dừng để yêu cầu nhập một input. Vì ban đầu chưa biết input cần những yêu cầu gì nên có thể nhập một con số bất kỳ. Lúc này chương trình thông báo sai, quả bom nổ và thoát.

```
lando@ubuntu:~/LTHT/Lab5$ ./nt209-uit-bomb
Welcome to UIT's bomb lab.
You have to deactivate our bomb by solving 5 phases with the correct inputs consecutively, and otherwise the bomb will be blown up!

[*] Phase 1
- Hint: Numbers are always magical!
1
BOMB!!!!
The bomb has blown up. Try again.
lando@ubuntu:~/LTHT/Lab5$
```

- **Bước 2: Phân tích file nt209-uit-bomb**

Có 2 cách để thực hiện phân tích file bomb với 2 tùy chọn môi trường ở phần **B.1**

Tùy chọn 1: Phân tích tĩnh file nt209-uit-bomb với IDA Pro

Sinh viên đọc code assembly/mã giả của file bomb như ở Bài thực hành 4. Các phân tích ở bên dưới với tùy chọn 2 vẫn có thể áp dụng với tùy chọn 1.

Tùy chọn 2: Debug file bomb với IDA Pro

Sau khi thiết lập môi trường debug như phần B.1.2, với việc thực thi file **linux_server** trên máy Linux, đặt breakpoint trong mã assembly ở vị trí hàm main trong IDA Pro, và nhấp chọn biểu tượng màu xanh ▶ trên thanh công cụ để bắt đầu debug, ta được kết quả sau:

```
lando@ubuntu:~/LTHT/Labs$ chmod +x *
lando@ubuntu:~/LTHT/Lab5$ ./linux_server
IDA Linux 32-bit remote debug server(ST) v1.17. Hex-Rays (c) 2004-2014
Listening on port #23946...
=====
[1] Accepting connection from 192.168.6.1...
```

Lab 5: Kỹ thuật dịch ngược (tt)

Dùng icon để chạy từng dòng code của hàm **main** để xem hoạt động của chương trình. Ta có thể thấy đoạn mã in ra các dòng chữ trong lần chạy thử file **nt209-uit-bomb** với hàm **puts** như hình dưới.

```

EIP : .text:08048C6B      lea     ecx, [esp+4]
.EIP:08048C6F      and    esp, 0FFFFFFF0h
.EIP:08048C72      push   dword ptr [ecx-4]
.EIP:08048C75      push   ebp
.EIP:08048C76      mov    ebp, esp
.EIP:08048C78      push   ebx
.EIP:08048C79      push   ecx
.EIP:08048C7A      sub    esp, 10h
.EIP:08048C7D      mov    ebx, ecx
.EIP:08048C7F      sub    esp, 0Ch
.EIP:08048C82      push   offset aWelcomeToUitsB ; "Welcome to UIT's bomb lab.\nYou have to"...
.EIP:08048C87      call   _puts
.EIP:08048C8C      add    esp, 10h
.EIP:08048C8F      cmp    dword ptr [ebx], 1
.EIP:08048C92      jnz    short loc_8048CA0
.EIP:08048C94      mov    eax, ds:stdin@@GLIBC_2_0
.EIP:08048C99      mou    ds:infile, eax
.EIP:08048C9E      jmp    short loc_8048D15

```

Sau khi in các chuỗi với **puts**, hàm **main** vào các đoạn các liên quan đến các pha. Trước mỗi pha đều có hàm **puts** để in ra gợi ý. Sau đó, nó gọi một hàm **read_line**, có tác dụng đọc 1 chuỗi input từ người dùng. Hàm **read_line** luôn đọc vào 1 chuỗi, và trả về địa chỉ đang lưu chuỗi đó. Do vậy, giá trị thanh ghi **eax** ngay sau khi thực thi **read_line** sẽ là địa chỉ ô nhớ lưu chuỗi đã nhập. Có thể thấy giá trị trong **eax** được đưa vào **[ebp + s]**, và **[ebp + s]** sẽ được push vào stack để làm tham số trước khi gọi **phase1** (pha 1). Như vậy hàm **main** đã dùng **read_line** để đọc input từ người dùng và truyền địa chỉ lưu input này như tham số của hàm **phase1**.

```

.text:08048D15 ; 
.text:08048D15 ; 
.text:08048D15 loc_8048D15:                                ; CODE XREF: main+33↑j
; main+5F↑j
.text:08048D15 sub    esp, 0Ch
.text:08048D18 push   offset aPhase1HintNumb ; "\n[*] Phase 1\n- Hint: Numbers are alwa"...
.text:08048D1D call   _puts
.text:08048D22 add    esp, 10h
.text:08048D25 call   read_line
.text:08048D2A mov    [ebp+s], eax
.text:08048D2D sub    esp, 0Ch
.text:08048D30 push   [ebp+s]
.text:08048D33 call   phase1
.text:08048D38 add    esp, 10h
.text:08048D3B call   defuse_bomb
.text:08048D40 sub    esp, 0Ch
.text:08048D43 push   offset aPhase2HintYouM ; "\n[*] Phase 2\n- Hint: You must answer "...
.text:08048D48 call   _puts
.text:08048D4D add    esp, 10h
.text:08048D50 call   read_line
.text:08048D55 mov    [ebp+s], eax
.text:08048D58 sub    esp, 0Ch
.text:08048D5B push   [ebp+s]
.text:08048D5E call   phase2
.text:08048D63 add    esp, 10h
.text:08048D66 call   defuse_bomb

```

Có thể thấy, sau khi gọi hàm **phase1**, một hàm **defuse_bomb** sẽ được gọi. Theo tên gọi có vẻ như nó dùng để vô hiệu hóa bom, được gọi khi 1 pha được giải thành công. Hàm vô hiệu hóa này sẽ được gọi nếu như hàm **phase1** trước đó trả về bình thường. Như vậy, ở hàm **phase1** có thể sẽ kiểm tra input người dùng, và dựa trên kết quả có thể sẽ nổ bom trong chính hàm **phase1**. Thật vậy, nhấp đúp chuột để xem code của hàm **phase1**, ta thấy

Lab 5: Kỹ thuật dịch ngược (tt)

xuất hiện hàm **explode_bomb** có thể dùng để nổ bom. Để giải được pha 1, dễ thấy cần tránh các trường hợp gọi đến hàm **explode_bomb** này và hàm **phase1** cần trả về bình thường để chạy tiếp hàm **phase_defused** vô hiệu hóa bom sau đó.

```

text:080489D2      call    __isoc99_sscanf
text:080489D7      add     esp, 20h
text:080489DA      mov     [ebp+var_10], eax
text:080489DD      cmp     [ebp+var_10], 6
text:080489E1      jz      short loc 80489E8
text:080489E3      call    explode_bomb

```

Hoạt động tương tự cũng áp dụng cho các hàm **phase2**, **phase3**,... ứng với các pha còn lại.

Nhiệm vụ tiếp theo là đọc code cụ thể của những hàm **phase1**, **phase2**,... này để hiểu hoạt động của từng pha.

- **Bước 3: Phân tích các hàm để tìm input cho mỗi pha**

Nhìn chung, ở mỗi pha của bài lab, để phá bom thành công, sinh viên cần xác định được các đặc điểm của input như sau:

- Định dạng của input (cần nhập số, kí tự, hay kết hợp giữa kí tự và số?).
- Số lượng phần tử trong input (bao nhiêu số, ký tự hay chuỗi?)
- Nếu có nhiều phần tử, phân tích mã assembly để tìm quan hệ ràng buộc giữa chúng.

C. MỘT SỐ GỢI Ý CHO CÁC PHA

C.1 Định dạng input của mỗi pha

Tất cả các hàm xử lý của các pha đều nhận tham số là địa chỉ lưu chuỗi input của người dùng, có được từ hàm **read_line()**. Tuy vậy, trong mỗi pha sẽ có những đoạn xử lý khác nhau, ví dụ gọi hàm **sscanf()**, để từ một chuỗi đó có thể lấy ra được những giá trị cần thiết, như một số, một ký tự, hoặc nhóm các số và ký tự.

Ví dụ bên dưới, hàm **sscanf()** trong pha 1 thực hiện đọc từ chuỗi input (có địa chỉ **a1**) với định dạng “**%d %d %d %d %d %d**”, tức lấy 6 số nguyên và lưu vào các vị trí của **v2**. Để ý phần chú thích phía sau của **v2**, **v3**... **v7**, thấy rằng **v2** là 1 mảng integer 6 phần tử, các vị trí **v3**, **v4**... **v7** nằm cách vị trí bắt đầu của **v2** lần lượt 4, 8, 12, ... 20 bytes, nên đây thực chất là vị trí các phần tử của mảng **v2**.

```

1 int __cdecl phase1(int a1)
2 {
3     int result; // eax@3
4     int v2[6]; // [sp+Ch] [bp-2Ch]@1
5     int v3[6]; // [sp+10h] [bp-28h]@1
6     int v4[6]; // [sp+14h] [bp-24h]@1
7     int v5[6]; // [sp+18h] [bp-20h]@1
8     int v6[6]; // [sp+1Ch] [bp-1Ch]@1
9     int v7[6]; // [sp+20h] [bp-18h]@1
10    int v8; // [sp+24h] [bp-14h]@3
11    int v9; // [sp+28h] [bp-10h]@1
12    int i; // [sp+2Ch] [bp-Ch]@5
13
14    v9 = __isoc99_sscanf(a1, "%d %d %d %d %d", v2, v3, v4, v5, v6, v7);
15    if ( v9 != 6 )
16 }
```

C.2 Nhắc lại một số kiến thức cơ bản về procedure (hàm)

Một số lưu ý về procedure (hàm) trong mã assembly như sau:

- Một hàm được gọi với lệnh **call <tên hàm>**
- Một hàm có thể có các tham số truyền vào, các tham số này được truyền bằng những câu lệnh **push** giá trị trước lệnh **call** theo thứ tự ngược với khai báo trong C.

Ví dụ một hàm C có tên **add(int a, int b)** thì trong assembly sẽ tìm thấy những đoạn mã như sau để gọi hàm này:

```

push b
push a
call add

```

- **Giá trị trả về** của một hàm thường được lưu trong **thanh ghi eax**. Những xử lý trên giá trị thanh ghi eax ngay sau khi gọi 1 hàm với lệnh **call** có thể hiểu là xử lý giá trị được trả về từ hàm đó.

D. YÊU CẦU & ĐÁNH GIÁ

Bài thực hành sẽ có **nhiều phiên bản khác nhau** của file **nt209-uit-bomb** với các yêu cầu input ở các pha khác nhau, mỗi file gồm 5 pha. Mỗi nhóm sinh viên sẽ được GVTH giao nhiệm vụ phân tích 1 trong các phiên bản.

Sinh viên thực hành và nộp bài **theo nhóm tối đa 3 sinh viên**. Ở mỗi pha, sinh viên viết báo cáo về cách thức tìm thấy input cần tìm với các nội dung tối thiểu:

- (1) Định dạng của input (số/ký tự/tổ hợp số ký tự) và số lượng (nếu có).
- (2) Điều kiện ràng buộc của input (giới hạn giá trị/ràng buộc giữa các phần tử).
- (3) Kết luận về input của pha (liệt kê hoặc mô tả tất cả các input thỏa mãn).
- (4) Hình ảnh chụp kết quả thực thi với 1 hoặc nhiều input đã tìm thấy với file nt209-uit-bomb.

Mỗi nội dung cần có hình ảnh (đoạn mã assembly/mã giả/ảnh chụp màn hình) minh chứng.

Một file .pdf chứa các nội dung trên và đặt tên theo quy tắc:

[NT209.O22.ANTT.2] Lab5_NhomX-MSSV1-MSSV2-MSSV3.pdf

Ví dụ: [NT209.O22.ANTT.2] Lab5_Nhom2-22520901-22520078-22521118.pdf

E. THAM KHẢO

[1] Randal E. Bryant, David R. O'Hallaron (2011). *Computer System: A Programmer's Perspective*

[2] Hướng dẫn sử dụng công cụ dịch ngược IDA Debugger – phần 1 [Online]

<https://securitydaily.net/huong-dan-su-dung-cong-cu-dich-nguoc-ma-may-ida-debugger-phan-1/>

[3] IDA công cụ hoàn hảo cho các chuyên gia Reverse Engineering [Online]

<https://securitydaily.net/ida-cong-cu-hoan-hao-chocac-chuyen-gia-reverse-engineering/>

HẾT