



5

Lab

Tìm hiểu về 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.

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: **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 **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 3 – Tìm hiểu cơ bản về Kỹ thuật dịch ngược để phân tích file **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 **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.

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 các package nếu cần thiết

Bước kiểm tra này đảm bảo môi trường Linux có thể thực thi file **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 bomb
$ ./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.

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

```
$ ifconfig
```

```
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ ifconfig
ens33  Link encap:Ethernet  HWaddr 00:0c:29:78:bf:f9
       inet addr: 192.168.229.132 Bcast:192.168.229.255 Mask:255.255.255.0
       inet6 addr: fe80::a585:dc8:b66a:86a0/64 Scope:Link
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:9166 errors:0 dropped:0 overruns:0 frame:0
       TX packets:3691 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:1000
       RX bytes:12306137 (12.3 MB)  TX bytes:236655 (236.6 KB)
```

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

```
$ ipconfig
```

```
C:\Windows\system32\cmd.exe

IPv4 Address. . . . . :
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :

Ethernet adapter VMware Network Adapter VMnet8:

Connection-specific DNS Suffix  . :
Link-local IPv6 Address . . . . . : fe80::54a3:9bf4:3996:a402%7
IPv4 Address. . . . . : 192.168.229.1
Subnet Mask . . . . . : 255.255.255.0
Default Gateway . . . . . :
```

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 **192.168.229.132** và **192.168.229.1** (Cùng mạng Vmnet8 của VMWare). Thực hiện lệnh **ping** để kiểm tra, nếu ping thành công thì 2 máy đã được kết nối.

```
C:\Windows\system32\cmd.exe

Microsoft Windows [Version 10.0.19042.630]
(c) 2020 Microsoft Corporation. All rights reserved.

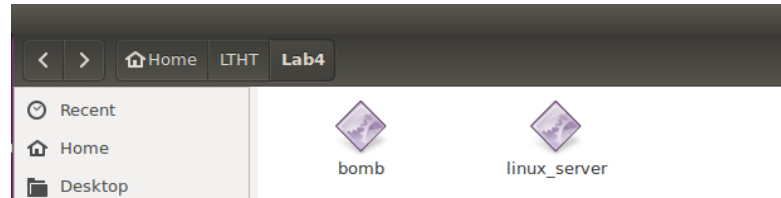
C:\Users\Thu-Hien Do>ping 192.168.229.132

Pinging 192.168.229.132 with 32 bytes of data:
Reply from 192.168.229.132: bytes=32 time<1ms TTL=64
Reply from 192.168.229.132: bytes=32 time<1ms TTL=64
Reply from 192.168.229.132: bytes=32 time<1ms TTL=64
Reply from 192.168.229.132: bytes=32 time<1ms TTL=64
```

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.

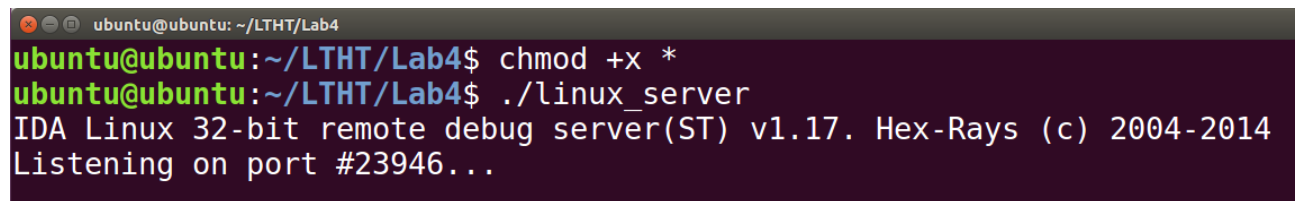
Bước 3. Sao chép file **bomb** vào máy ảo Linux ở một thư mục nhất định.

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

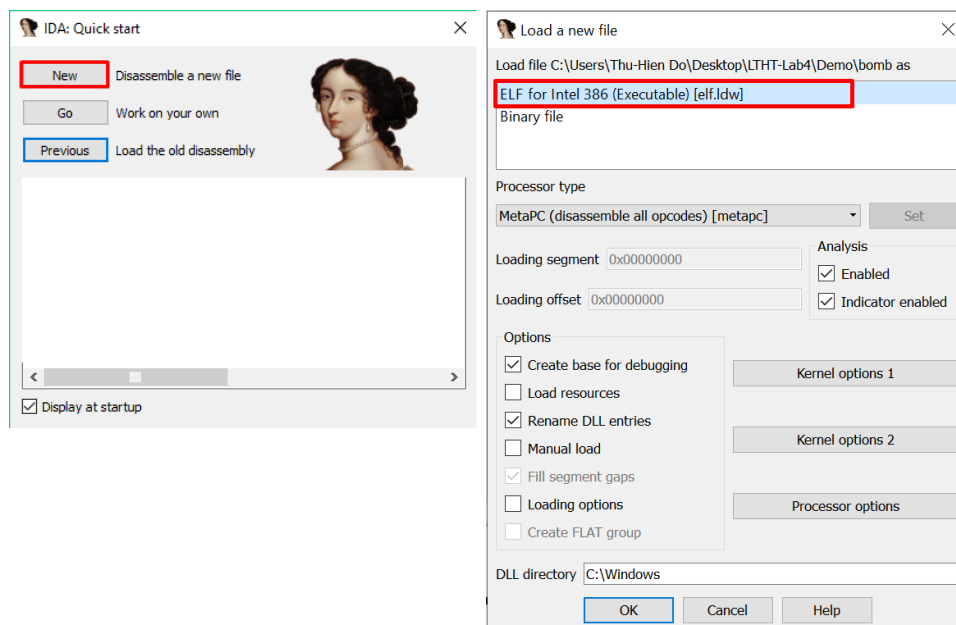


Bước 5. Trên máy Linux, với terminal đang ở thư mục chứa file **bomb** và **linux_server** của IDA, thực hiện các lệnh bên dưới để mở debug server bên máy Linux.

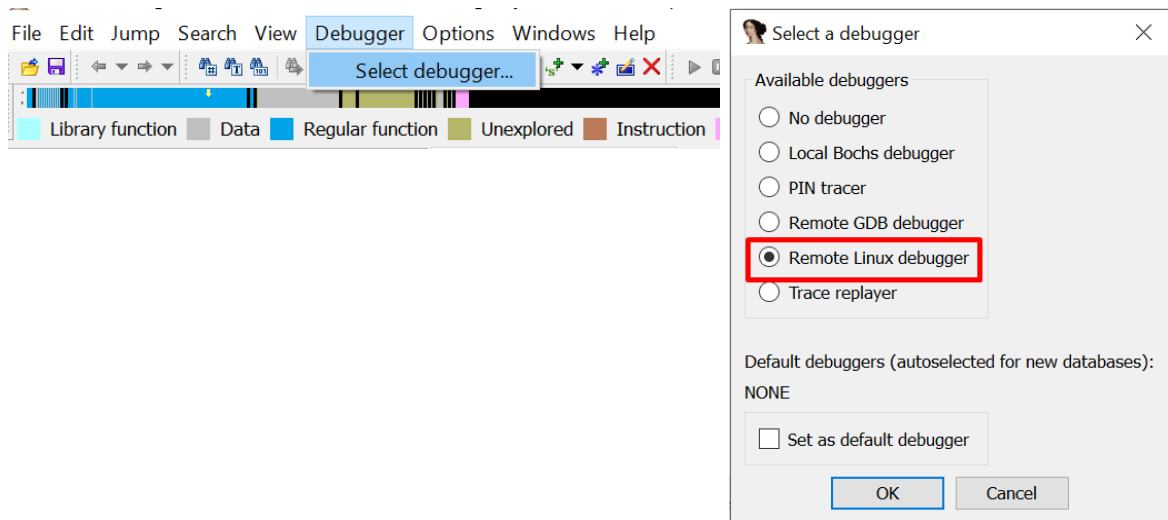
```
$ chmod +x *           # chỉ cần chạy 1 lần
$ ./linux_server
```



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

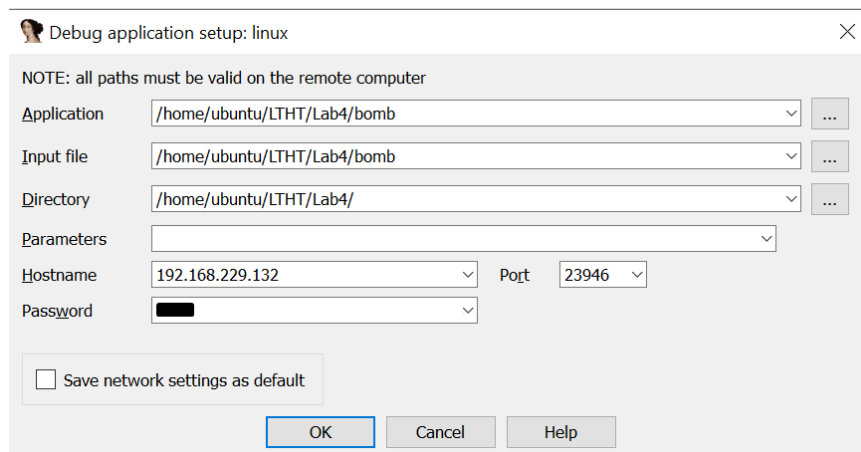


Bước 7. Sau khi load file thành công, chọn tab **Debugger** → **Select Debugger** và click chọn **Remote Linux Debugger** để thiết lập môi trường debug đến máy đích là máy Linux ở xa.

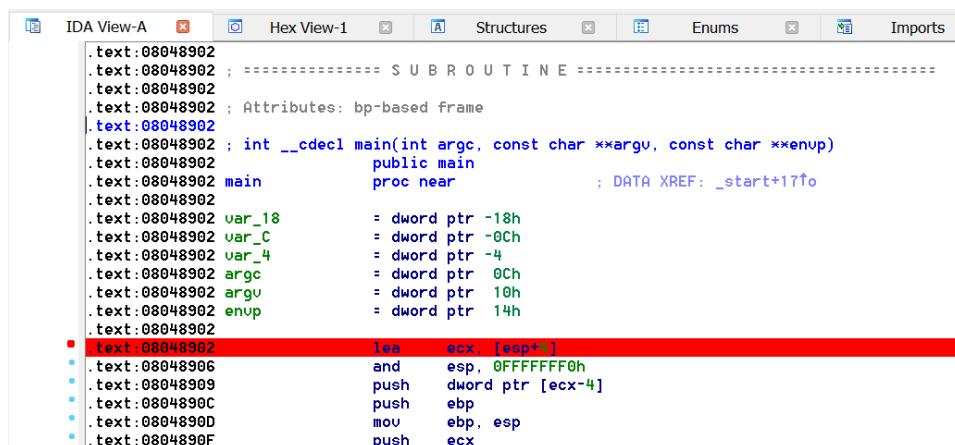


Sau đó, mở cửa sổ cài đặt debugger với **Debugger → Process Options** và điền các thông tin như sau:

- Directory: đường dẫn chứa file **bomb** trên máy Linux
- Application và Input file: đường dẫn đến file **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. Thử đặt breakpoint tại lệnh assembly đầu tiên của hàm **main** để debug.



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

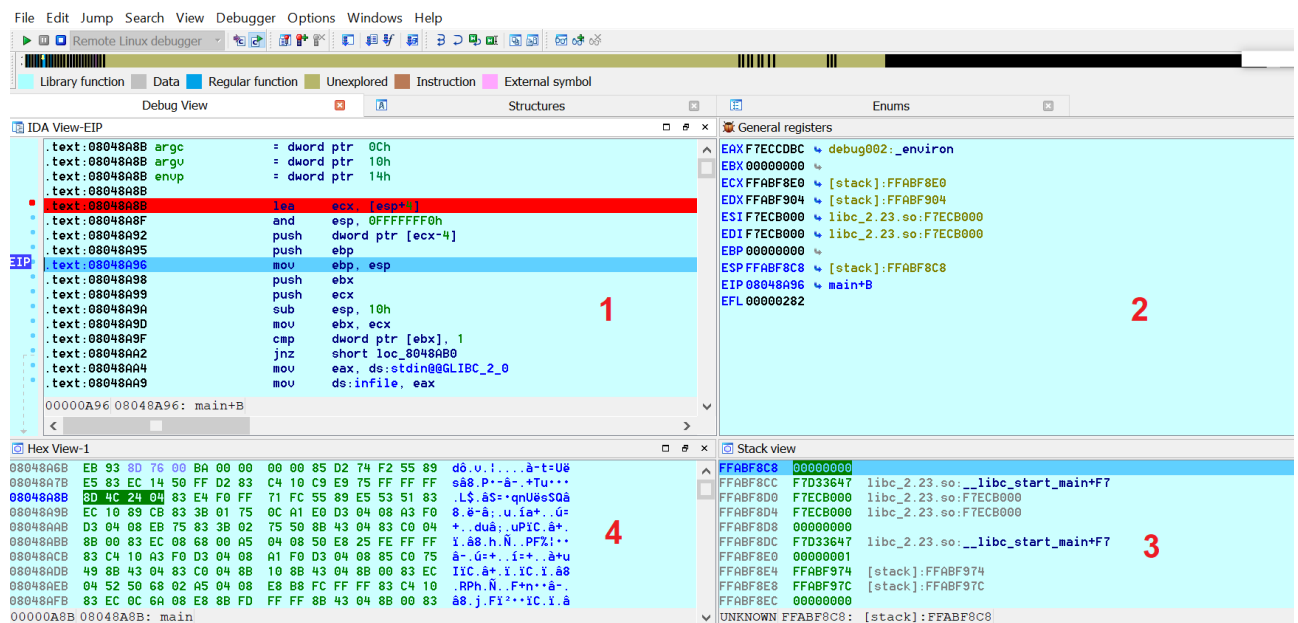
Chạy thử chương trình trên IDA Pro với dấu ▶ để kiểm tra remote debug. Nếu quan sát thấy các kết quả dưới đây thì quá trình thiết lập môi trường remote debug đã hoàn tất.

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

```
ubuntu@ubuntu: ~/LTHT/Lab4
ubuntu@ubuntu:~/LTHT/Lab4$ chmod 777 *
ubuntu@ubuntu:~/LTHT/Lab4$ ./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 192.168.229.1...
```

- **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 hiển thị nhiều cửa sổ mô tả trạng thái của hệ thống khi đang chạy chương trình **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 bomb)

B.2.1 Tổng quan về các pha

File thực thi **bomb** là một chương trình dạng command line, đã được lập trình sẵn và có hoạt động như một quả bomb. Quả bomb này gồm 6 pha, mỗi pha yêu cầu một input riêng để ngắt pha. Nếu input sai, bomb sẽ nổ và trò chơi kết thúc.

```
ubuntu@ubuntu: ~/LTHT/Lab4
ubuntu@ubuntu:~/LTHT/Lab4$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Hello world

BOOM!!!
The bomb has blown up.
ubuntu@ubuntu:~/LTHT/Lab4$
```

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

```
ubuntu@ubuntu: ~/LTHT/Lab4
ubuntu@ubuntu:~/LTHT/Lab4$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
```

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
- Bộ pha rất khó: Pha 6
- Một pha bí mật (🤔)

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

- Pha 1: *
- Pha 2: **
- Pha 3: ***
- Pha 4: *****
- Pha 5: ****
- Pha 6: *****

Trong phần này, sinh viên được yêu cầu giải mỗi pha của Bomb khi thực thi file “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 **bomb** khác nhau, bài hướng dẫn sử dụng 1 phiên bản bất kỳ để phân tích.

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

```
./bomb
```

Sau một số thông báo chào mừng, có thể thấy file tạm dừng để yêu cầu cần nhập một input. Vì ban đầu chưa biết input cần những yêu cầu gì nên có thể nhập một đoạn input bất kì. Lúc này chương trình thông báo sai, quả bomb nổ và thoát.

```
ubuntu@ubuntu: ~/LTHT/Lab4
ubuntu@ubuntu:~/LTHT/Lab4$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Hello world

BOOM!!!
The bomb has blown up.
ubuntu@ubuntu:~/LTHT/Lab4$
```

- **Bước 2: Phân tích file bomb**

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


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

Sinh viên đọc code assembly của file bomb như ở Bài thực hành 3. Các phân tích ở bước này 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:

```
ubuntu@ubuntu: ~/LTHT/Lab4
ubuntu@ubuntu:~/LTHT/Lab4$ ./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 192.168.229.1...
```

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 **bomb** với 2 hàm **puts** như hình dưới.

```

.text:08048B25 call initialize_bomb
.text:08048B2A sub esp, 0Ch
.text:08048B2D push offset s ; "Welcome to my fiendish little bomb. You"...
.text:08048B32 call _puts
.text:08048B37 add esp, 10h
.text:08048B3A sub esp, 0Ch
.text:08048B3D push offset aWhichToBlowYou ; "which to blow yourself up. Have a nice "...
.text:08048B42 call _puts
.text:08048B47 add esp, 10h
.text:08048B4A call read_line
.text:08048B4F mov [ebp+input], eax
.text:08048B52 sub esp, 0Ch
.text:08048B55 push [ebp+input]
.text:08048B58 call phase_1
.text:08048B5D add esp, 10h
.text:08048B60 call phase_defused
```


Hàm **main** ở đây gọi một hàm **read_line**, có tác dụng chính là để đọc 1 chuỗi input từ người dùng. Hàm **read_line** luôn đọc vào 1 chuỗi, đồng thời **eax** thường chứa giá trị trả về của 1 hàm, **eax** ngay sau khi thực thi xong **read_line** sẽ là chuỗi đã nhập. Sau đó giá trị trong **eax** được đưa vào **[ebp + input]**, và **[ebp + input]** sẽ được push vào stack để làm tham số trước khi gọi **phase_1** (pha 1). Như vậy hàm **main** đã dùng **read_line** để đọc input từ người dùng và truyền input này như tham số của hàm **phase_1**.

Có thể thấy, sau khi gọi hàm **phase_1**, một hàm **phase_defused** sẽ được gọi. Theo tên gọi có vẻ như nó dùng để vô hiệu hóa bomb, là hàm được gọi khi 1 pha được giải thành công. Như vậy, ở hàm **phase_1** có thể sẽ kiểm tra input người dùng, chỉ cần trả về bình thường là vô hiệu hoá được 1 pha, ngược lại sẽ nổ bom trong chính hàm **phase_1**. Trình tự tương tự cũng áp dụng cho các hàm **phase_2**, **phase_3**,... ứng với các pha còn lại.

Nhiệm vụ tiếp theo là đọc code cụ thể của những hàm **phase_1**, **phase_2**,... này để hiểu hoạt động của từng pha.

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

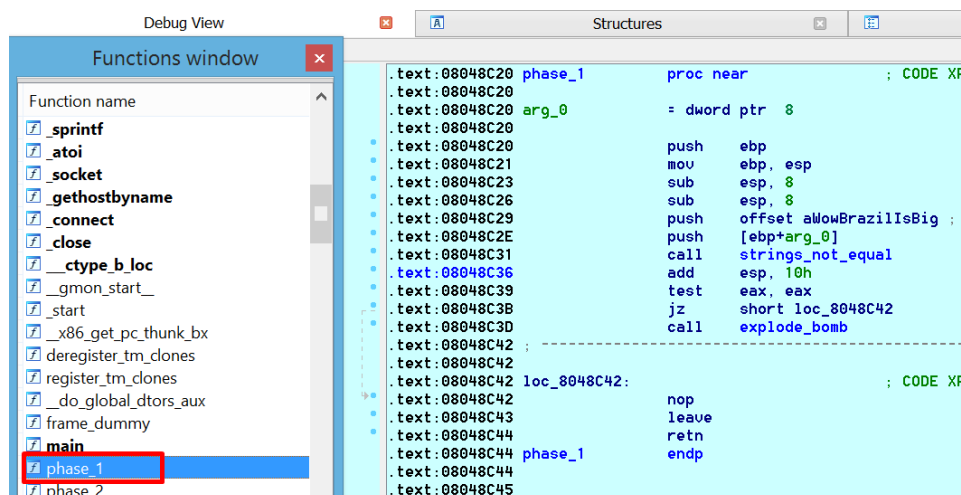
Nhìn chung, ở mỗi pha của bài lab, để phá bom thành công, sinh viên cần xác định được các đặc điểm của input như sau:

- Định dạng của input (kí tự, số 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.

B.3 Hướng dẫn giải chi tiết pha 1

• Bước 1: Tìm kiếm định dạng input

Ở trên ta đã tìm thấy function có tên là **phase_1**, đây là phần mã nguồn của pha 1. Nhấp chuột vào tên hàm để xuất hiện mã assembly của pha.

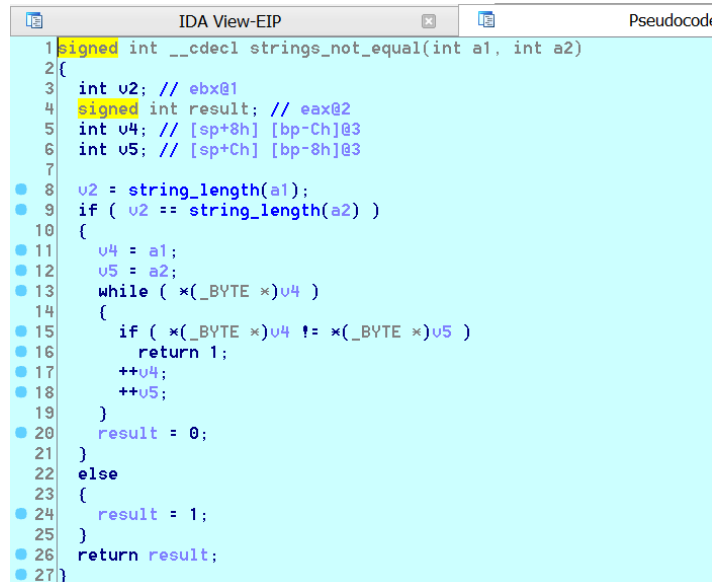


Phân tích từ hàm **main** trước đó, tham số truyền cho **phase_1** là một chuỗi. Debug theo từng dòng assembly, ta thấy ở vị trí **[ebp + arg_0]** (ebp + 8) chính là vị trí tham số này, hay chính là địa chỉ của chuỗi ta đã nhập. Chuỗi này được sử dụng để so sánh luôn nên ở pha 1 yêu cầu nhập vào 1 chuỗi.

- **Bước 2: Phân tích mã nguồn để tìm điều kiện thỏa mãn của input**

Từ mã assembly, ta có những hoạt động sau:

- Các địa chỉ của chuỗi input ở **[ebp + arg_0]** và một biến **aWowBrazillsBig** được **push** vào stack trước khi gọi **strings_not_equal**. Các giá trị này là tham số của hàm.
- Có thể coi sơ hoạt động hàm **strings_not_equal** bằng mã giả bằng cách chọn hàm và nhấn **F5**: hàm này thực hiện so sánh từng phần tử của 2 mảng. Hàm trả về 1 nếu 2 mảng không bằng, 0 nếu ngược lại.



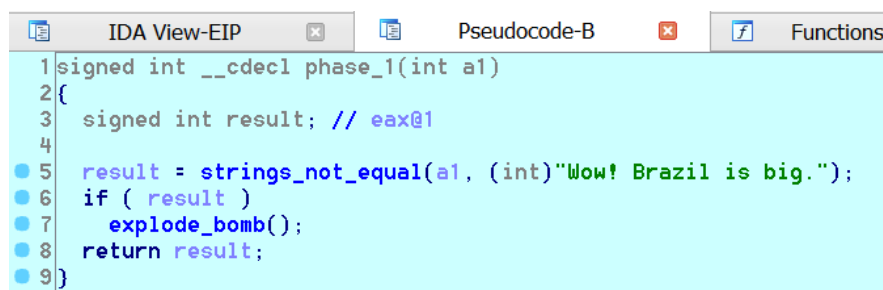
```

1 signed int __cdecl strings_not_equal(int a1, int a2)
2 {
3     int v2; // ebx@1
4     signed int result; // eax@2
5     int v4; // [sp+8h] [bp-Ch]@3
6     int v5; // [sp+Ch] [bp-8h]@3
7
8     v2 = string_length(a1);
9     if ( v2 == string_length(a2) )
10    {
11        v4 = a1;
12        v5 = a2;
13        while ( *(_BYTE *)v4 )
14        {
15            if ( *(_BYTE *)v4 != *(_BYTE *)v5 )
16                return 1;
17            ++v4;
18            ++v5;
19        }
20        result = 0;
21    }
22    else
23    {
24        result = 1;
25    }
26    return result;
27 }

```

- Dòng lệnh **test eax, eax** kiểm tra giá trị trong thanh ghi eax ngay sau khi gọi **string_not_equal**, tức là kiểm tra kết quả trả về của hàm này. Nếu bằng 0 sẽ nhảy đến **loc_8048C42** để kết thúc hàm, còn khác 0 sẽ gọi **explode_bomb** (nổ bom!). Như vậy, kết quả so sánh 2 mảng input nhập vào và biến **aWowBrazillsBig** nếu khác nhau sẽ khiến bom nổ, còn lại thì trả về bình thường.
- ⇒ Từ đó có thể thấy chuỗi ký tự được nhập cần giống với giá trị biến **aWowBrazillsBig**, nhấp đúp vào biến để thấy nó có giá trị **"Wow! Brazil is big."**

Để dễ hơn, có thể nhấn **F5** để xem mã giả của **phase_1**:



```

1 signed int __cdecl phase_1(int a1)
2 {
3     signed int result; // eax@1
4
5     result = strings_not_equal(a1, (int)"Wow! Brazil is big.");
6     if ( result )
7         explode_bomb();
8     return result;
9 }

```

Vậy input cần nhập của pha 1 là: **Wow! Brazil is big.**

- **Bước 3: Kiểm tra kết quả**

```
ubuntu@ubuntu: ~/LTHT/Lab4
ubuntu@ubuntu:~/LTHT/Lab4$ ./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 192.168.229.1...
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
```

Như vậy, với tham số là chuỗi input người dùng nhập vào, trong các hàm xử lý từng pha tương tự như **phase_1**, nếu input không thỏa điều kiện kiểm tra tương ứng, chương trình lập tức chạy hàm **explode_bomb** để nổ bom và thoát chương trình. Ngược lại, hàm trả về bình thường và chương trình gọi tiếp **phase_defused()**. Vậy yêu cầu cần thực hiện là làm sao để các pha được gọi với input được nhập trả về bình thường.

C. MỘT SỐ GỢI Ý CÁC PHA CÒN LẠI

C.1 Định dạng input của mỗi pha

Tất cả các pha của chương trình đều nhận input là chuỗi của người dùng từ hàm **read_line()**, do dữ liệu nhập từ bàn phím luôn là chuỗi. Tuy vậy, trong mỗi pha sẽ có những đoạn gọi hàm **scanf()** khác nhau để từ một chuỗi input 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 như bên dưới, hàm **scanf()** trong pha 2 thực hiện đọc chuỗi input lấy từ **read_line()** với định dạng **"%d %d %d %d %d %d"**, tức là lấy 6 số nguyên.

```
IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | In
1 int __cdecl read_six_numbers(int a1, int a2)
2 {
3     int result; // eax@1
4
5     result = __isoc99_sscanf(a1, "%d %d %d %d %d %d", a2, a2 + 4, a2 + 8, a2 + 12, a2 + 16, a2 + 20);
6     if ( result <= 5 )
7         explode_bomb();
8     return result;
9 }
```

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 **bomb** với các yêu cầu input ở các pha khác nhau, mỗi file gồm 6 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 2 sinh viên**. Ở mỗi pha, sinh viên viết báo cáo ngắn gọn 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.
- (3) Kết luận về input của pha.
- (4) Hình ảnh chụp kết quả thực thi input đã tìm thấy với file bomb

Mỗi phần cần có hình ảnh (đoạn mã assembly/mã giả/ảnh chụp màn hình) minh chứng. Tên file ghi rõ Lab5-MSSV1-MSSV2-<số pha giải được | Full>.pdf

Đặt tên thư mục nộp bài theo quy tắc sau:

LabX-MSSV1-MSSV2-HọTênSV1-HọTênSV2

Ví dụ: Lab2-16520901-17520078-NguyenVanA-TranVanC

E. THAM KHẢO

- [1] Randal E. Bryant, David R. O'Hallaron (2011). *Computer System: A Programmer's Perspective (CSAPP)*
- [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-cho-cac-chuyen-gia-reverse-engineering/>

HẾT