3

Lab

Tìm hiểu cơ bản về Kỹ thuật Dịch ngược

Basic Reverse Engineering

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.
- Cài đặt và tìm hiểu trình Disassembler IDA - một trong những phần mềm disassembler được sử dụng bởi rất nhiều chuyên gia nghiên cứu về bảo mật.

A.2 Môi trường

A.2.1 Chuẩn bị môi trường

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

A.2.2 Các tập tin được cung cấp sẵn

- File **basic-reverse** là file thực thi được build trên môi trường Linux 32-bit
- Mỗi nhóm sinh viên sẽ có riêng 1 phiên bản **basic-reverse** riêng.

A.3 Liên quan

- Sinh viên cần vận dụng kiến thức trong Chương 3 (Lý thuyết).
- Tìm hiểu các kiến thức về remote debugger trên desktop khi sử dụng IDA (optional)
- Tham khảo tài liệu (Mục E).

B. THỰC HÀNH

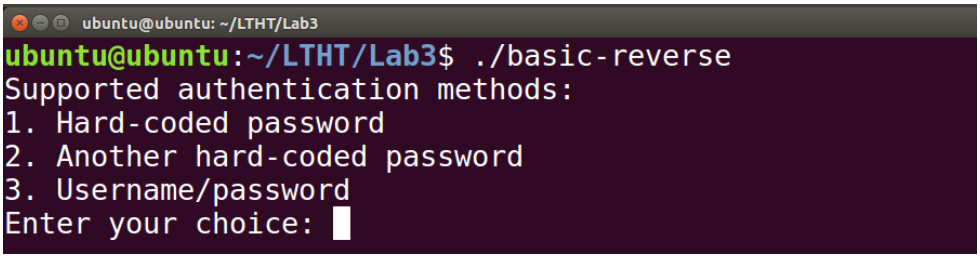
Bài thực hành này sẽ được chia làm 2 giai đoạn tương ứng với 2 yêu cầu:

- **Giai đoạn 1**: ***Thiết lập môi trường***

Trong giai đoạn này, sinh viên chuẩn bị môi trường như trong phần **A.2.** Sinh viên tự tìm hiểu tài liệu và quan sát quá trình Giảng viên hướng dẫn sơ lược.

- **Giai đoạn 2**: ***Thực hành dịch ngược file basic-reverse***

Sinh viên được cung cấp một file thực thi ELF 32 bit chạy dưới dạng command line.



File này thực thi 3 phương pháp chứng thực đơn giản yêu cầu người dùng nhập 1 chuỗi passpharse (1, 2) hoặc một cặp username/password (3) để có thể nhận một thông điệp bí mật nào đó. Sau khi thiết lập môi trường, sinh viên áp dụng kỹ thuật dịch ngược và sử dụng công cụ IDA Pro để phân tích file **basic-reverse** nhằm tìm ra các passphrase hoặc cặp username/password đúng yêu cầu.

B.1 Yêu cầu 1 - Thiết lập môi trường

Yêu cầu thiết lập môi trường bao gồm cài đặt công cụ IDA Pro trên máy Windows và chuẩn bị môi trường Linux để chạy file thực thi 32-bit **basic-reverse**.

B.1.1 Cài đặt IDA Pro trên máy Windows

- **Bước 1: Tải và giải nén IDA Pro**

IDA Pro là một disassembler mạnh mẽ, là lựa chọn của nhiều người phân tích mã độc, dịch ngược và phân tích lỗ hổng. IDA Pro cung cấp nhiều tính năng như phân tích chương trình thành hợp ngữ, phát hiện function, xác định biến cục bộ, phân tích stack,…

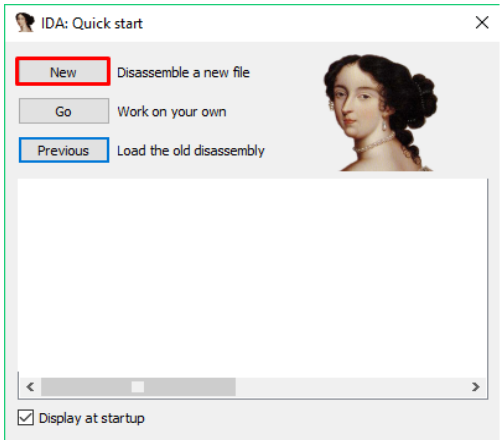
GVTH cung cấp sẵn cho sinh viên file nén chứa source đầy đủ của công cụ IDA Pro phiên bản 6.6. Sau khi tải file nén về, sinh viên tiến hành giải nén vào một thư mục trên máy Windows (*lưu ý: sinh viên nên giải nén trong ổ đĩa C:\ của máy tính).*

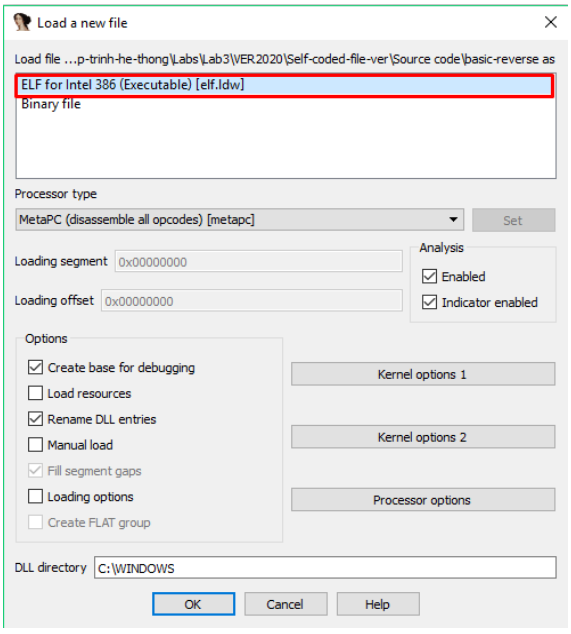
- **Bước 2: Chạy thử công cụ IDA Pro**

Ở thư mục đã giải nén của IDA Pro, chọn chạy IDA **bản 32 bit (idaq.exe)** (do file cần phân tích ở bài thực hành là 32 bit). Chọn **New** để tiến hành phân tích file mới.

Sau đó, mở file thực thi 32-bit bất kỳ với IDA Pro. Chọn **File → Open** và trỏ đến đường dẫn của file hoặc kéo thả file trực tiếp vào cửa sổ làm việc của IDA Pro. IDA Pro sẽ tự động phát hiện định dạng của file và kiến trúc của bộ xử lý, ví dụ ở đây là **ELF for Intel 386 (Executable)**. Có nhiều options để load file vào IDA Pro:

- PE/ Executable files: file được ánh xạ vào bộ nhớ tương tự như lúc nó được load bởi hệ điều hành.
- Binary File: phân tích file dưới dạng file nhị phân.



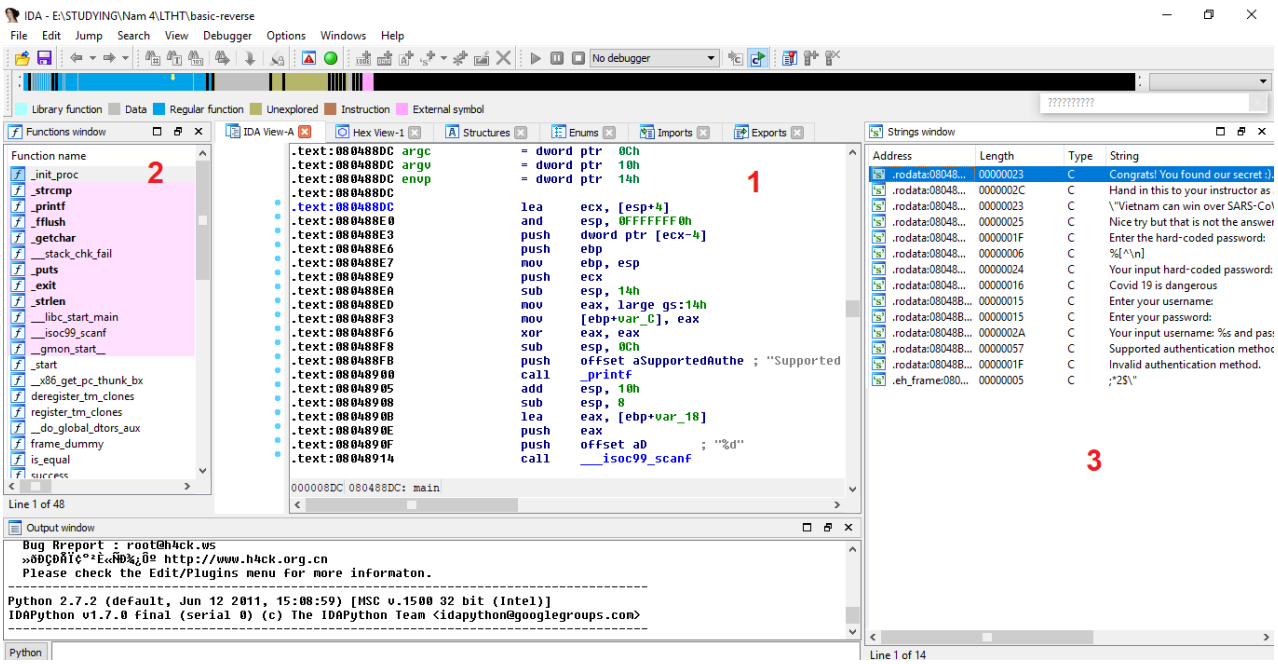


Chọn **OK** để load file, nếu không xảy ra lỗi thì đến đây chúng ta đã có công cụ IDA Pro được cài đặt để sẵn sàng phân tích các file.

- **Bước 3: Quan sát các cửa sổ làm việc của IDA Pro**

Có nhiều cửa sổ làm việc khác nhau được hiển thị sau khi load thành công file thực thi bằng IDA Pro, chúng có tính năng liên kết và tham chiếu chéo (cross-reference):

- **Disassembly window (1):** cửa sổ chính chứa mã hợp ngữ của chương trình cần phân tích. Mã hợp ngữ có thể được xem ở chế độ text hoặc graph.
- **Functions window (2):** liệt kê tất cả các hàm có trong file thực thi, có hỗ trợ sắp xếp và lọc.
- **Names window:** liệt kê tất cả các địa chỉ với tên gọi, bao gồm các hàm, các code, dữ liệu được đặt tên, chuỗi…
- **Strings window (3):** hiển thị tất cả các chuỗi. Mặc định chỉ hiển thị các chuỗi ASCII dài từ 5 ký tự trở lên.
- **Imports/Exports window:** liệt kê các hàm import/export của file.
- **Structure window:** các cấu trúc dữ liệu có trong file.



IDA Pro có hỗ trợ điều hướng, nhiều cửa sổ được liên kết với cửa sổ disassembly chính. Các cửa sổ có thể được mở bằng cách chọn **Views → Open subviews** và chọn cửa sổ tương ứng.

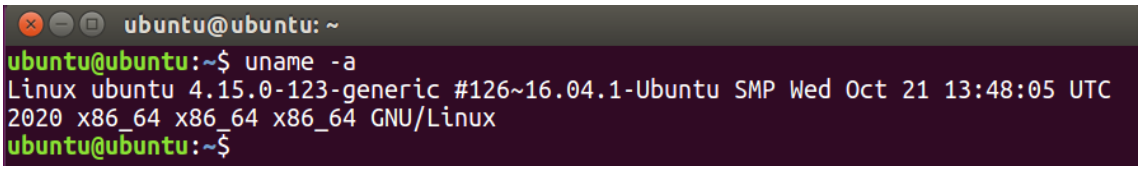
B.1.2 Thiết lập môi trường chạy file thực thi Linux 32-bit

Do file cần phân tích được build trên môi trường Linux 32-bit, để chạy được ta cần thiết lập một môi trường Linux có thể thực thi được các file thực thi 32-bit.

Bước 1. Kiểm tra phiên bản môi trường Linux

Bước kiểm tra này đảm bảo môi trường Linux đã chuẩn bị có thể thực thi file 32-bit hay không. Sinh viên sử dụng câu lệnh sau trên terminal:

```
$ uname -a
```

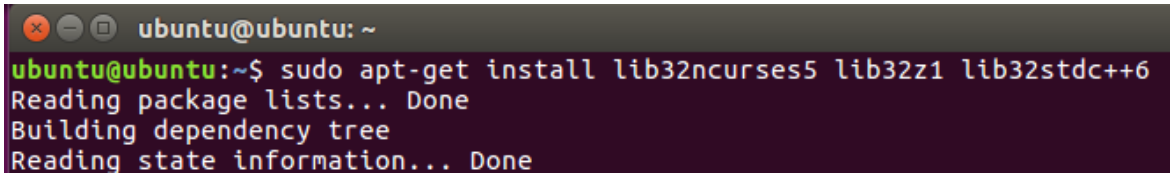


- Nếu kết quả hiển thị có keyword **x86_64/64**: môi trường 64-bit, cần cài đặt thêm một số package ở **Bước 2**.
- Nếu kết quả chỉ hiển thị các keyword **x86, i386,**…: môi trường 32-bit và thực thi được ngay các file 32-bit → Sang **Bước 3**.

Bước 2. ***(Chỉ dành cho trường hợp sử dụng máy Linux bản 64-bit làm môi trường chạy)***

Sử dụng các lệnh sau trên terminal của máy Linux 64-bit để cài đặt thêm một số package hỗ trợ chạy file thực thi 32-bit trên môi trường 64-bit.

```
$ sudo apt-get install lib32ncurses5 lib32z1 lib32stdc++6
```

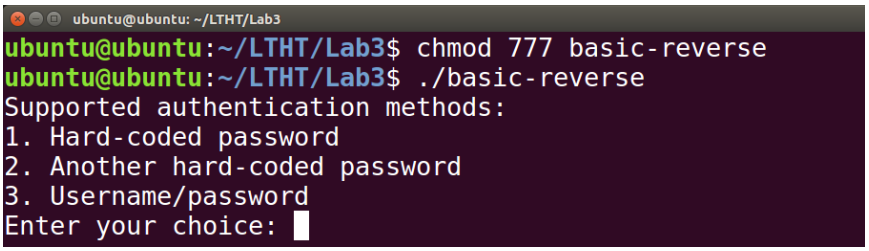


Lưu ý: nếu hệ thống không tìm thấy package ***lib32ncurses5***, sinh viên thử thay thế bằng ***lib32ncurses6***.

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

Bước 4. Trên máy Linux, với terminal đang ở thư mục chứa file **basic-reverse**, thực hiện các lệnh bên dưới để chạy thử file **basic-reverse**.

```
$ chmod 777 basic-reverse
$ ./basic-reverse
```



Nếu hiện được giao diện như ở hình trên, ta đã thiết lập môi trường thành công để thực thi file **basic-reverse.** Ngược lại, nếu gặp lỗi bất kỳ, ví dụ **No such file or directory**, sinh viên cần xem lại các bước thiết lập môi trường thực thi cho file **basic-reverse**.

B.2 Yêu cầu 2 – Thực hành phân tích file basic-reverse

B.2.1 Tổng quan về file basic-reverse

Basic-reverse là một chương trình đã được lập trình sẵn, yêu cầu người dùng nhập vào một số chuỗi để đánh giá và xuất 1 số thông điệp bí mật. Chương trình có 3 option khác nhau tương ứng với 3 yêu cầu đầu vào khác nhau.

- ***1. Hard-coded password*** - *Chuỗi passphrase cố định*: là một chuỗi password được định nghĩa sẵn trong chương trình. Khi chọn option này, người dùng cần nhập vào một chuỗi kết thúc bằng ký tự xuống dòng.
- ***2. Another hard-coded password*** – *tương tự như option 1*, cũng là một chuỗi password được định nghĩa sẵn trong chương trình nhưng khó tìm hơn. Khi chọn option này, người dùng cần nhập vào một chuỗi kết thúc bằng ký tự xuống dòng.
- ***3. Username/password*** - *Cặp username/password*: giữa username và password sẽ có những điều kiện ràng buộc nhất định của chương trình. Đầu vào yêu cầu là 1 cặp username và password tương ứng.

Khi nhập đúng các passpharse hoặc username/password yêu cầu, chương trình sẽ in ra thông báo thành công kèm theo thông tin bí mật.

B.2.2 Yêu cầu thực hành

Yêu cầu 2.1. Phân tích và tìm ***passphrase cố định*** **(option 1)** của **basic-reverse** với phương pháp chứng thực 1. Báo cáo phương pháp phân tích, input tìm được và hình ảnh minh chứng chạy file.

Yêu cầu 2.2. Phân tích và tìm ***passphrase cố định*** **(option 2)** của **basic-reverse** với phương pháp chứng thực 2. Báo cáo phương pháp phân tích, input tìm được và hình ảnh minh chứng chạy file.

Yêu cầu 2.3. Phân tích, tìm ***username/password*** phù hợp của **basic-reverse** với phương pháp chứng thực 3. Báo cáo phương pháp và input tìm được.

Lưu ý bắt buộc: ***username*** được tạo từ MSSV của các thành viên trong nhóm.

- Nhóm **3 sinh viên**, lấy 3 số cuối nối nhau. Ví dụ 21520013, 21520123 và 21521021 sẽ có username là **013123021**.
- Nhóm **2 sinh viên**, lấy 4 số cuối nối nhau bằng dấu "-". Ví dụ 21520013, 21520123 sẽ có username là 0013-0123.
- Nhóm có **1 sinh viên** có MSSV là 2152xxxx thì username là **2152-xxxx**.

B.2.3 Hướng dẫn cách thực hiện phân tích file basic-reverse

- **Bước 1: Thực thi thử file basic-reverse trên Linux**

```
./basic-reverse
```

Bước đầu ta cần lựa chọn 1 trong 3 phương pháp chứng thực bằng cách nhập số 1, 2 hoặc 3. Tương ứng với từng lựa chọn chương trình sẽ in ra yêu cầu nhập input tương ứng, giả sử bên dưới lựa chọn phương pháp 1 là dùng passphrase cố định thì sẽ được yêu cầu nhập 1 chuỗi. Vì ban đầu chưa biết input cần nhập là gì nên có thể nhập một chuỗi bất kì. Lúc này chương trình thông báo sai và thoát.

```
ubuntu@ubuntu: ~/LTHT/Lab3
ubuntu@ubuntu:~/LTHT/Lab3$ ./basic-reverse
Supported authentication methods:
1. Hard-coded password
2. Another hard-coded password
3. Username/password
Enter your choice: 1
Enter the hard-coded password (option 1):
Hello world
Your input hard-coded password: Hello world
Nice try but that is not the answer.
ubuntu@ubuntu:~/LTHT/Lab3$
```

- **Bước 2: Phân tích file basic-reverse với IDA Pro**

Mở file **basic-reverse** trong IDA Pro phiên bản 32-bit.

- **Xác định hàm cần quan tâm**

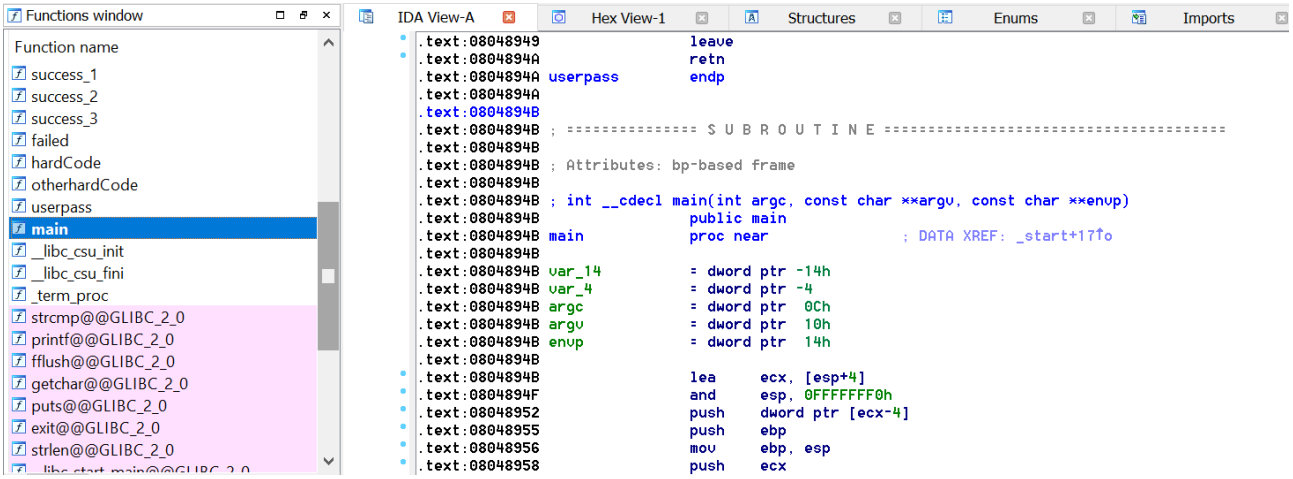
Các chương trình thường định nghĩa nhiều hàm hoặc sử dụng một số hàm thư viện trong hoạt động của nó. Chọn **View → Open Subviews → Functions** để hiện cửa sổ một số *function* có trong file.

Ta cần lưu ý:

+ Chương trình thường bắt đầu với một số hàm có tên là **start** hay **main**. Khi load thành công file thực thi, IDA Pro thường sẽ đứng ở vị trí hàm này. Ở đây ta thử xem xét từ hàm **main** trước.
+ Một số hàm thư viện đã biết chỉ cần nắm được về chức năng và các tham số, không cần phân tích code cụ thể.

- **Phân tích code của hàm main**

Nhấp đúp vào tên hàm **main** trong **Function windows** để xem mã assembly của hàm đó. *Lưu ý: Mã assembly được hiển thị trong IDA Pro dưới định dạng Intel.*



Gợi ý phân tích:

+ Một số dòng code assembly ở đầu hàm **main** dùng để set-up không gian bộ nhớ cho hàm, chưa thực thi chức năng chính của nó.
+ Các dòng assembly được đóng khung như bên dưới thực hiện chức năng truyền tham số và gọi hàm thư viện **printf** để in 1 chuỗi nào đó.

Quan sát các lệnh, ta thấy trước khi thực hiện lệnh **call printf**, có 1 giá trị được **push** vào stack, lệnh này có tác dụng truyền tham số cho hàm. Với **printf** cần ít nhất 1 tham số là chuỗi cần in, như vậy giá trị được push vào là địa chỉ chuỗi cần in (trong IDA Pro, địa chỉ có dạng **offset...**), cụ thể là chuỗi "**Supported authentication methods:...**"

```
.text:08048959                sub     esp, 14h
.text:0804895C                sub     esp, 0Ch
.text:0804895F                push    offset aSupportedAuthe ; "Supported authentication methods:\n1. H"...
.text:08048964                call    _printf
.text:08048969                add     esp, 10h
.text:0804896C                sub     esp, 8
.text:0804896F                lea     eax, [ebp+var_14]
```

Như vậy, 2 dòng lệnh assembly này tương ứng với dòng lệnh C:

```
printf("Supported authentication methods:...");
```

+ Các dòng code assembly tiếp theo truyền tham số và gọi hàm thư viện **scanf.**

```
.text:0804895F                push    offset aSupportedAuthe ; "Supported authentication methods:\n1. H"...
.text:08048964                call    _printf
.text:08048969                add     esp, 10h
.text:0804896C                sub     esp, 8
.text:0804896F                lea     eax, [ebp+var_14]
.text:08048972                push    eax
.text:08048973                push    offset aD       ; "%d"
.text:08048978                call    ___isoc99_scanf
.text:0804897D                add     esp, 10h
```

Để ý rằng, hàm **scanf** cần 2 tham số: 1 chuỗi định dạng và vị trí sẽ lưu giá trị nhập vào. Có 2 dòng lệnh **push** các giá trị vào stack trước lệnh **call scanf,** đây cũng là 2 tham số được truyền. Tham số được push vào sau là địa chỉ chuỗi **"%d"** – chuỗi định dạng dữ liệu, tham số còn lại sẽ là địa chỉ ô nhớ sẽ lưu giá trị nhập vào. Theo đoạn mã, địa chỉ ô nhớ này được tính toán bằng **[ebp + var_14]** (được lấy với lệnh **lea**). Như vậy, input được nhập với scanf sẽ được lưu tại **[ebp + var_14]**. Do đó, những xử lý tiếp theo trên vị trí **[ebp + var_14]** này có thể hiểu là xử lý giá trị người dùng nhập vào.

Ở những đoạn mã phía sau, ta nhận thấy giá trị lưu ở **[ebp + var_14]** sẽ được lấy vào **%eax** để thực hiện nhiều phép so sánh với 1, 2 và 3, sau đó có lệnh **call** đến một số hàm nhất định. Như vậy đây là đoạn mã kiểm tra giá trị lựa chọn phương pháp chứng thực của người dùng (1, 2 hoặc 3) để gọi các hàm xử lý tương ứng là hàm **hardCode**, **otherhardCode** và **userpass**.

```
.text:0804898E                add     esp, 10h
.text:08048991                mov     eax, [ebp+var_14]
.text:08048994                cmp     eax, 1
.text:08048997                jnz     short loc_80489A0
.text:08048999                call    hardCode
.text:0804899E                jmp     short loc_80489D8
.text:080489A0 ; ---------------------------------------------------------------------------
.text:080489A0
.text:080489A0 loc_80489A0:                           ; CODE XREF: main+4C↑j
.text:080489A0                mov     eax, [ebp+var_14]
.text:080489A3                cmp     eax, 2
.text:080489A6                jnz     short loc_80489AF
.text:080489A8                call    otherhardCode
.text:080489AD                jmp     short loc_80489D8
.text:080489AF ; ---------------------------------------------------------------------------
.text:080489AF
.text:080489AF loc_80489AF:                           ; CODE XREF: main+5B↑j
.text:080489AF                mov     eax, [ebp+var_14]
.text:080489B2                cmp     eax, 3
.text:080489B5                jnz     short loc_80489BE
.text:080489B7                call    userpass
.text:080489BC                jmp     short loc_80489D8
.text:080489BE ; ---------------------------------------------------------------------------
```

Click đúp chuột vào 1 trong 3 tên hàm để nhảy đến đoạn mã của hàm đó. Giả sử ở đây chọn hàm **hardCode**.

```
.text:08048690 hardCode        proc near               ; CODE XREF: main+4E↓p
.text:08048690
.text:08048690 s1              = byte ptr -3F0h
.text:08048690
.text:08048690                 push    ebp
.text:08048691                 mov     ebp, esp
.text:08048693                 sub     esp, 3F8h
.text:08048699                 call    _getchar
.text:0804869E                 sub     esp, 0Ch
.text:080486A1                 push    offset aEnterTheHardCo ; "Enter the hard-coded password (option 1"...
.text:080486A6                 call    _puts
.text:080486AB                 add     esp, 10h
.text:080486AE                 sub     esp, 8
.text:080486B1                 lea     eax, [ebp+s1]
.text:080486B7                 push    eax
.text:080486B8                 push    offset asc_804923E ; "%[^\n]"
.text:080486BD                 call    ___isoc99_scanf
```

Hàm này tương tự cũng in ra một dòng thông báo yêu cầu nhập chuỗi passphrase với ***puts*** và đọc input của người dùng vào với ***scanf()***. Ở đây ta thấy tham số vị trí lưu của **scanf** là **[ebp + s1]**. Do đó, **[ebp + s1]** sẽ được dùng làm địa chỉ tham chiếu đến chuỗi input mà người dùng đã nhập để kiểm tra xem chuỗi được nhập có đúng yêu cầu không.

Sinh viên thực hiện tiếp quá trình phân tích để xác định chuỗi passphrase cần tìm trong hàm **hardCode** là gì. Quá trình phân tích tương tự cũng có thể áp dụng với hàm **otherhardCode** và **userpass** xử lý phương pháp chứng thực thứ 2 và thứ 3.

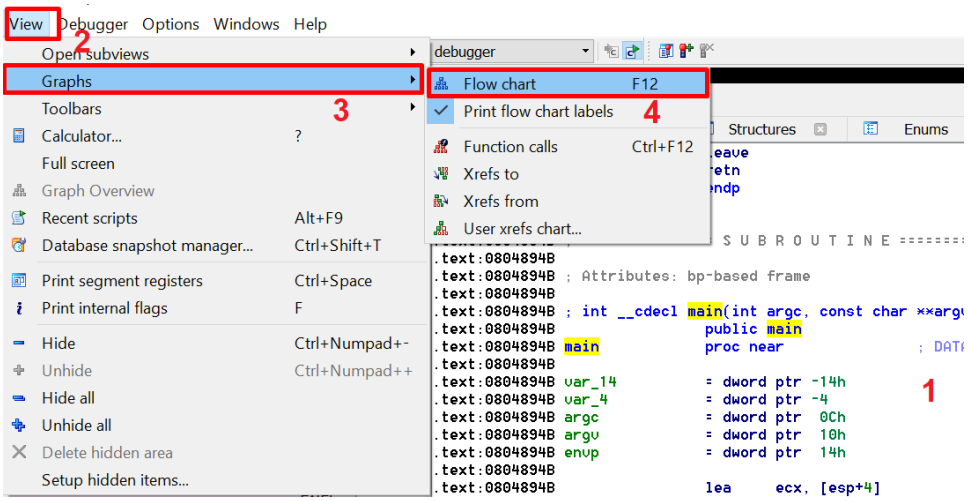
C. MỘT SỐ GỢI Ý PHÂN TÍCH FILE VỚI IDA PRO

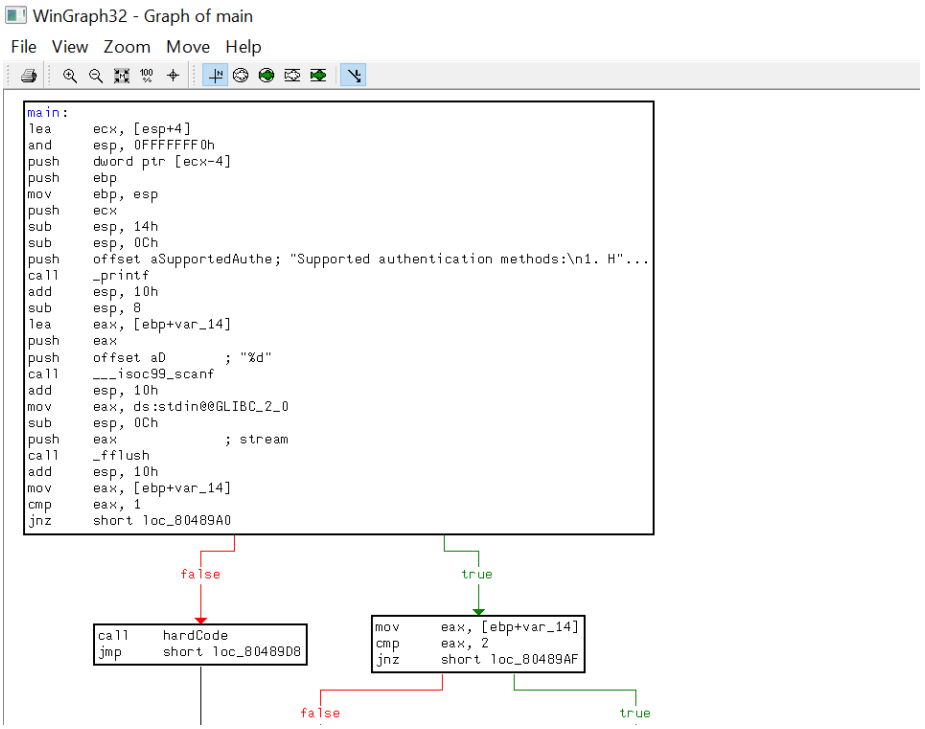
C.1 Sử dụng chế độ view phù hợp

IDA Pro hỗ trợ việc phân tích file thực thi với nhiều chế độ view khác nhau nhằm hiển thị mã assembly hoặc mã tương đương ở dạng dễ hiểu hơn.

- **Flow chart**

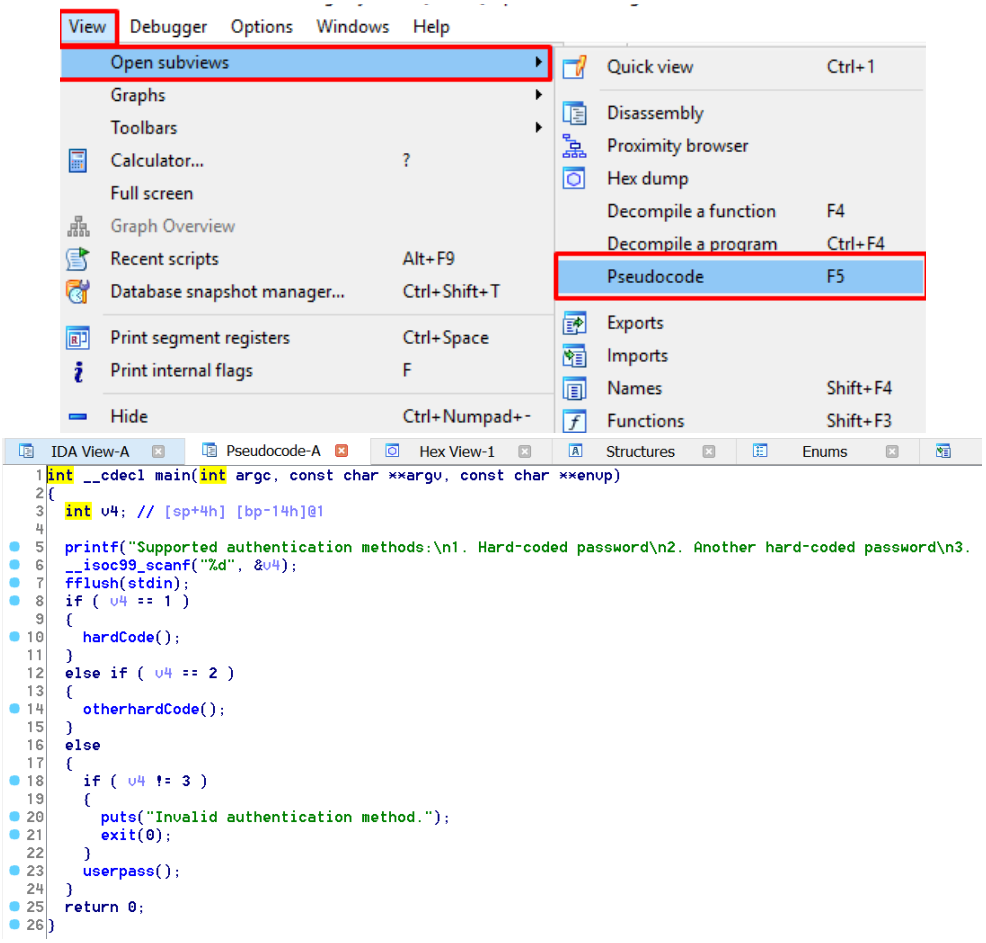
Ở chế độ này, mối liên hệ của nhiều đoạn mã, hàm sẽ được hiển thị rõ hơn để nắm được flow thực thi của chương trình. Sau khi click chuột vào vị trí của một đoạn mã, chọn **View → Graphs → Flow chart** hoặc nhấn phím **F12** để hiển thị dạng flow chart.





- **Pseudo code – mã giả:** là dạng hiển thị gần giống với dạng mã của các ngôn ngữ lập trình cấp cao với các biến, khối lệnh,…

Sau khi chọn đoạn mã, chọn **View → Open subviews → Pseudocode** hoặc nhấn phím **F5** để xem mã giả.



C.2 Gợi ý cho các Yêu cầu

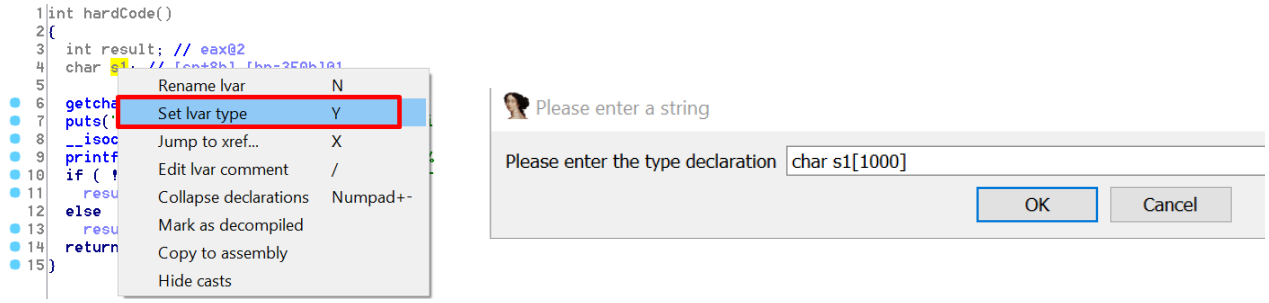
Khi xem mã nguồn của file đang phân tích ở dạng mã giả, IDA Pro sẽ đặt tên một số biến theo thứ tự xuất hiện và tự phát hiện về kiểu dữ liệu (gạch chân màu đỏ), đồng thời cung cấp thông tin vị trí của nó ở phần comment (sau dấu // gạch chân màu xanh).

```
int result; // eax@4
int v4; // edx@4
char s1; // [sp+8h] [bp-20h]@1
int v6; // [sp+1Ch] [bp-Ch]@1
```

Có một số trường hợp IDA Pro hiển thị mã giả nhưng ẩn một số thông tin hữu ích cho việc phân tích file.

C.2.1 Yêu cầu 2.1 – Chuyển kiểu dữ liệu của chuỗi s1

Trong phân tích mã giả của hàm hardCode, có thể thấy **s1** là 1 chuỗi sẽ chứa input người dùng nhập vào. Tuy nhiên ban đầu IDA Pro đặt biến này có kiểu **char** (ký tự). Có thể chuyển **s1** sang kiểu chuỗi bằng cách nhấp chuột phải vào **s1**, chọn **Set lvar type** và đổi thành kiểu dữ liệu mong muốn.



C.2.2 Gợi ý cho yêu cầu 2.2 – Chuyển kiểu dữ liệu của WHAT_THAT

Trong phân tích mã giả của hàm otherhardCode, có thể thấy **WHAT_THAT** là 1 dữ liệu nào đó sẽ được tham chiếu để so sánh với chuỗi nhập vào. Nhấp đúp chuột vào tên WHAT_THAT sẽ đến được phần khai báo của nó trong file.

WHAT_THAT là 1 mảng nhiều chuỗi, mỗi phần tử của mảng này có kích thước 4 bytes và là 1 địa chỉ của 1 chuỗi nào đó, ví dụ a) bên dưới 0x8048A70 hay 0x8048A9B là các địa chỉ của các chuỗi. IDA Pro có hỗ trợ hiển thị những giá trị này bằng những cái tên gợi nhớ dựa trên giá trị của chuỗi nó trỏ đến.

```
.data:0804B060                 public WHAT_THAT
.data:0804B060 WHAT_THAT       dd 8048A70h
.data:0804B064                 dd 8048A9Bh
.data:0804B068                 dd 8048AB4h
.data:0804B06C                 dd 8048AEDh
.data:0804B070                 dd 8048B04h
.data:0804B074                 dd 8048B3Bh
```

a) Giá trị thực tế trong mảng WHAT_THAT

```
WHAT_THAT       dd offset aYouScratchMyBa ; DATA XREF:
                                          ; "You scratch
                dd offset aNewOneInOldOne ; "New one i
                dd offset aItTooLateToLoc ; "It' too l
                dd offset aWithAgeComesWi ; "With age
                dd offset aNothingIsMoreP ; "Nothing i
```

b) Hiển thị của IDA Pro

WHAT_THAT có kiểu dữ liệu là **char * WHAT_THAT[x]**, với x không vượt quá 100. Có thể chuyển WHAT_THAT thành kiểu dữ liệu phù hợp để dễ phân tích hơn.

C.2.3 Gợi ý cho yêu cầu 2.3 – Mảng ẩn tiềm năng trong mã giả

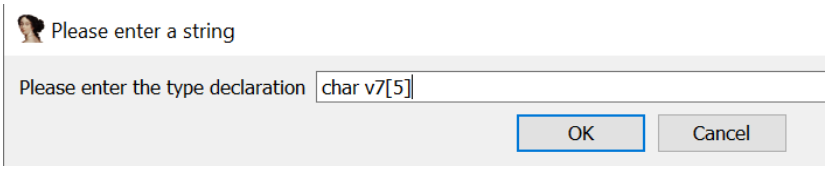
Ví dụ bên dưới ứng với hàm xử lý username/password của file **basic-reverse**, ta thấy có 5 biến kiểu char nằm ở các vị trí liên tục từ **(sp + 0x27)** đến **(sp + 0x2B)**, hay nói cách khác là có 5 ký tự nằm liên tục nhau (cách nhau 1 byte) từ địa chỉ của biến **v7**.

```
10  char v7; // [sp+27h] [bp-11h]@1
11  char v8; // [sp+28h] [bp-10h]@1
12  char v9; // [sp+29h] [bp-Fh]@1
13  char v10; // [sp+2Ah] [bp-Eh]@1
14  char v11; // [sp+2Bh] [bp-Dh]@1
```

Sau đó, trong đoạn mã giả có đoạn truy xuất các giá trị dựa trên địa chỉ của **v7** (sử dụng * và địa chỉ **&v7**) mà không xử lý các biến v8, v9,…

```
if ( (signed int)i > 3 )
  v4[i] = *(&v7 + i - 4);
else
  v4[i] = s[i + 5];
```

Từ đó ta có thể đặt giả thiết đây thực chất là 1 mảng ký tự bắt đầu từ ký tự **v7**. Ta thử chuyển kiểu biến của **v7** sang một mảng gồm 5 ký tự (từ **v7** đến **v11**) bằng cách nhấp chọn vào biến này, sau đó gõ ký tự **y** hoặc chuột phải và chọn **Set lvar type**. Chuyển từ **char v7** thành **char v7[5]** như hình dưới.



Khi đó đoạn code xử lý có thể trở thành dạng quen hơn với truy xuất các phần tử của mảng. Tuy nhiên, cần phân tích thêm để xem việc chuyển đổi có chính xác hay không.

```
if ( (signed int)i > 3 )
  v4[i] = v7[i - 4];
else
  v4[i] = s[i + 5];
```

C.3 Tham khảo thêm

C.3.1 Truyền tham số cho hàm trong assembly IA32

Một số lưu ý về hàm trong mã assembly IA32 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 thường 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 đoạn mã:

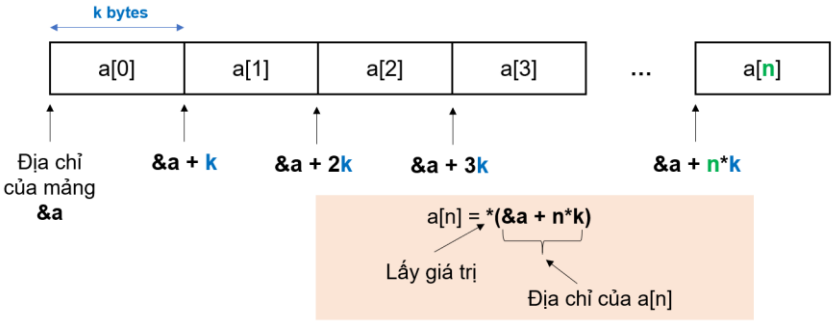
```
push b
push a
call add
```

- Sau khi được gọi với lệnh call, nếu hàm có trả về giá trị thì thường lưu trong thanh ghi %eax.

C.3.2 Mảng

Trong hệ thống, các phần tử của 1 Mảng (array) được đặt ở các vị trí liền kề nhau trong bộ nhớ. Trong 1 mảng, ta luôn có:

- Phần tử đầu tiên (index = 0) luôn nằm ở ô nhớ có địa chỉ thấp nhất so với các phần tử còn lại, sau đó đến phần tử thứ 2, thứ 3,… lần lượt đến phần tử cuối của mảng.
- Vị trí của các phần tử nằm cách nhau 1 khoảng bằng kích thước của mỗi phần tử. Ví dụ, một mảng ký tự có các phần tử nằm cách nhau 1 byte, các phần tử của mảng số nguyên (integer) cách nhau 4 byte.



D. YÊU CẦU & ĐÁNH GIÁ

Sinh viên thực hành và nộp bài theo **nhóm tối đa 3 sinh viên**. Các nhóm nộp file báo cáo kết quả **(.pdf)** trình bày cách phân tích file và tìm kiếm các passphrase cũng như cặp username/password bao gồm:

- Mô tả cách thức kiểm tra của mỗi hàm xử lý (phép so sánh, thuật toán) để tìm ra input tương ứng với 3 phương pháp chứng thực.
  - *(Khuyến khích)* Phân tích bằng mã assembly.
  - Sinh viên có thể dựa trên mã giả để hiểu hoạt động, sau đó tìm và chỉ ra đoạn mã assembly nào tương ứng với đoạn mã giả đó.
  - Sinh viên có thể tự code các đoạn code (C/Python...) để hỗ trợ tìm input nhanh/tự động (cần kèm theo báo cáo).
- Hình ảnh chụp màn hình kết quả thực thi file **basic-reverse** với input đã tìm được và chuỗi bí mật được tìm thấy cho 3 phương pháp chứng thực.
- Định dạng file **Lab3_NhomX_MSSV1-MSSV2-MSSV3.pdf**

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