Chương 4
CÁC KỸ THUẬT PHÂN TÍCH TĨNH

Trong các chương trước giáo trình đã trình bày quy trình phân tích mã độc, trong quy trình phân tích có mô tả các phương pháp sử dụng để phân tích mã. Chương 4 này sẽ giúp sinh viên hiểu rõ hơn phương pháp phân tích tĩnh. Trước khi phân tích thì sinh viên cần cách xây dựng môi trường hỗ trợ công việc phân tích.

4.1 Xây dựng môi trường phân tích tĩnh

Xây dựng môi trường phân tích mã độc là công việc quan trọng. Như đã biết mã độc luôn tiềm ẩn các hành vi nghi ngờ và rất dễ lây lan. Việc xây dựng môi trường phân tích mã phù hợp và an toàn là việc cần thiết.

Để tạo ra được môi trường an toàn trong khi phân tích mã độc hại, trong phương pháp phân tích tĩnh để phân tích mã độc hại người ta dùng 2 cách:

- Xây dựng môi trường ảo để phân tích mã độc.
- Xây dựng môi trường thật để phân tích mã độc.

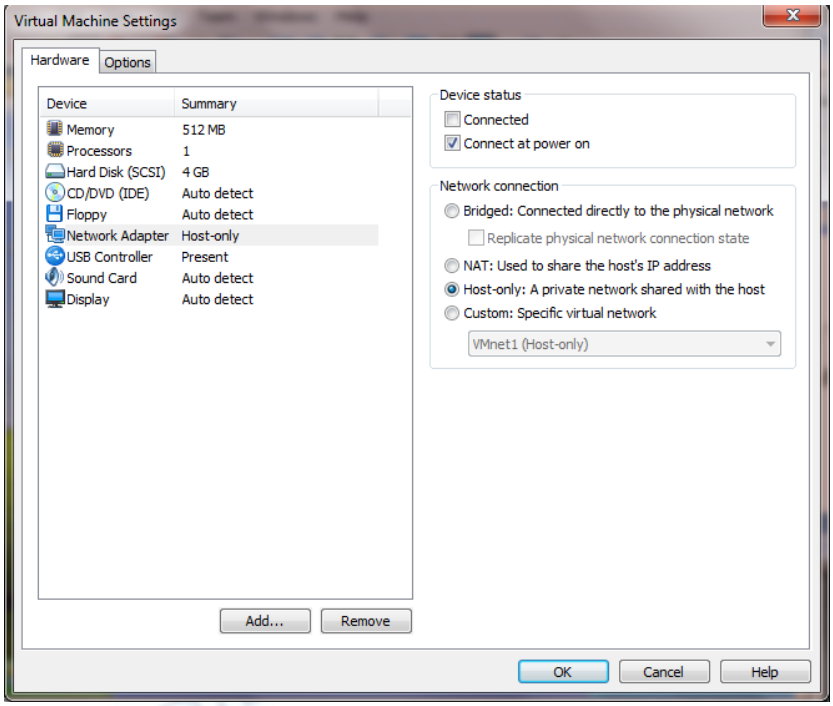
4.1.1 Xây dựng môi trường ảo.

Việc xây dựng môi trường ảo đảm bảo cho các loại mã độc hại không phá hoại được hệ thống máy tính thật. Các phần mềm để xây dựng môi trường ảo như Vmware workstation, virtualbox…. Ở đây sẽ lựa chọn xây dựng môi trường phân tích với VM workstation.

Phương án xây dựng như sau:

- Một máy tính thật cài Windows 7
- Trên Win 7 cài đặt một máy ảo Vmware, trong máy ảo cài đặt Windows XP(môi trường để chạy malware và thực hiện phân tích).
- Cài đặt mạng phù hợp với việc phân tích.
- Cài các tool cần thiết phục vụ việc phân tích

Thực hiện việc cài đặt Vmware sao cho memory (515 mb) và Hard Disk (10Gb), sau khi thực hiện cài đặt xong, tùy thuộc vào từng loại mã độc hại để cài cấu hình mạng.



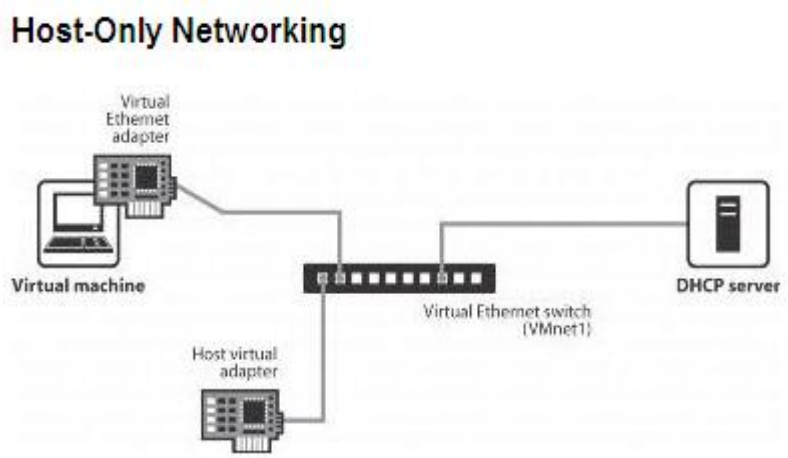
Hình 4-1: Chọn cấu hình mạng

Sau khi cài đặt xong, Vmware sẽ tạo nên 2 card mạng Vmware 1 và Vmware 8 trên máy thật và máy thật có thể sử dụng 2 card mạng này để kết nối với các máy ảo. khi lựa chon cấu hình mạng cho máy ảo ta có thể chọn một trong các chế độ sau:

Bridged networking: Card mạng của máy ảo sẽ được gắn trực tiếp với card mạng của máy thật (sử dụng switch ảo VMnet0). Lúc này, máy ảo sẽ đóng vai trò như một máy trong mạng thật, có thể nhận DHCP từ mạng ngoài, hoặc đặt IP tĩnh cùng dải với mạng ngoài để giao tiếp với các máy ngoài mạng hoặc lên Internet.

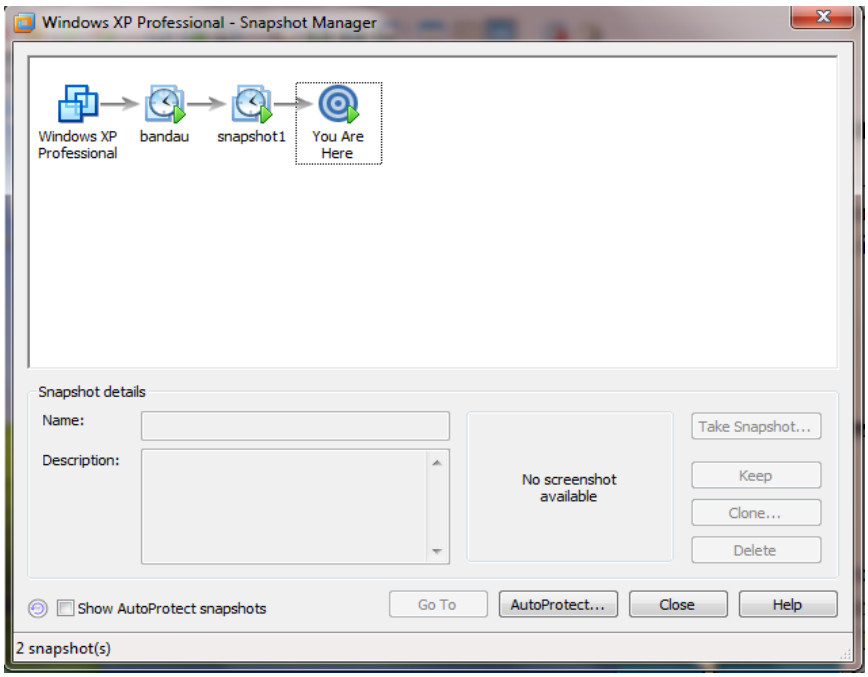
NAT: Máy ảo được cấu hình NAT sẽ sử dụng IP của máy thật để giao tiếp với mạng ngoài. Các máy ảo được cấp địa chỉ IP nhờ một DHCP ảo của VMware. Lúc này, các máy ảo sẽ kết nối với máy thật qua switch ảo VMnet8, và máy thật sẽ đóng vai trò NAT server cho các máy ảo.

Host-only Networking: Khi cấu hình máy ảo sử dụng host-only networking, máy ảo sẽ được kết nối với máy thật trong một mạng riêng thông qua Switch ảo VMnet1. Địa chỉ của máy ảo và máy thật trong mạng host-only có thể được cấp bởi DHCP ảo gắn liền với Switch ảo Vmnet1 hoặc có thể đặt địa chỉ IP tĩnh cùng dải để kết nối với nhau. Ta sẽ chủ yếu dùng loại này để cấu hình khi phân tích



Hình 4-2: Host- only Networking

Sau khi cấu hình mạng xong, ta nên tạo take snapshot lại một bảng sạch. Snapshot giúp chúng ta restore lại trạng thái làm việc mới nhất. Để sau mỗi lần phân tích ta lại có 1 một máy mới mà không cần phải đi cài lại. Ta chỉ cần vào snapshot, chạy lại snapshot sạch ban đầu khi mà chưa cho bất kì vào phân tích.



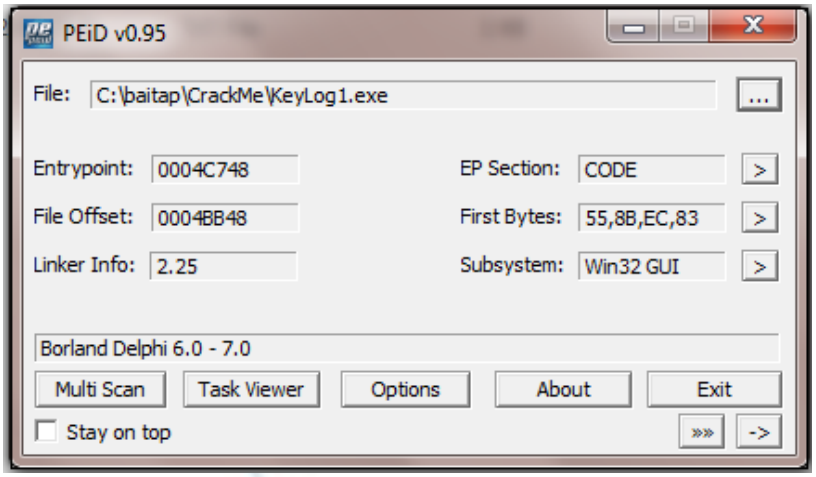
Hình 4-3: Tạo snapshot

Sau khi cài xong Vmware, ta cài hệ điều hành Xp lên đó, cài đặt các công cụ lên để giám sát sự thay đổi và tác động của các chương trình nên hệ thống đang thực thi.

Một số công cụ cần thiết cho môi trường phân tích tĩnh như sau:

4.1.1 Công cụ Peid

Là công cụ phát hiện xem file được viết bằng ngôn ngữ gì. Ngoài ra đây là một công cụ cần thiết xem file có bị nén (pack) hay không? Nếu bị nén (pack) thì người phân tích cần phải tìm cách giải nén (unpack file) phù hợp.



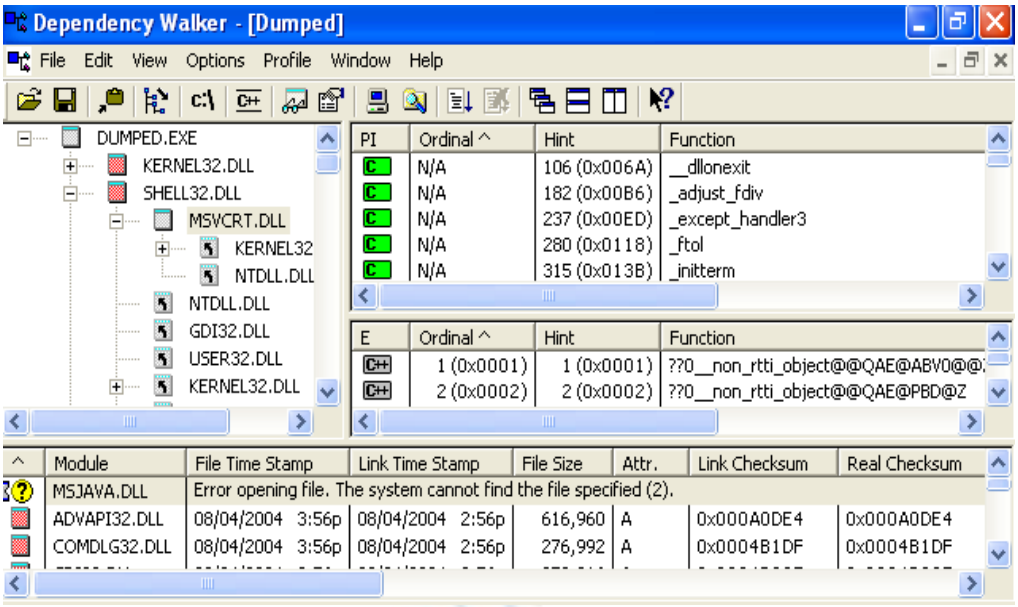
Hình 4-4: Phần mềm Peid

Như trong hình có thể thấy chương trình Peid đã nhận ra mã nguồn chương trình Keylog1.exe viết bằng Borland Delphi 6.0-7.0

4.1.2 Dependency Walker

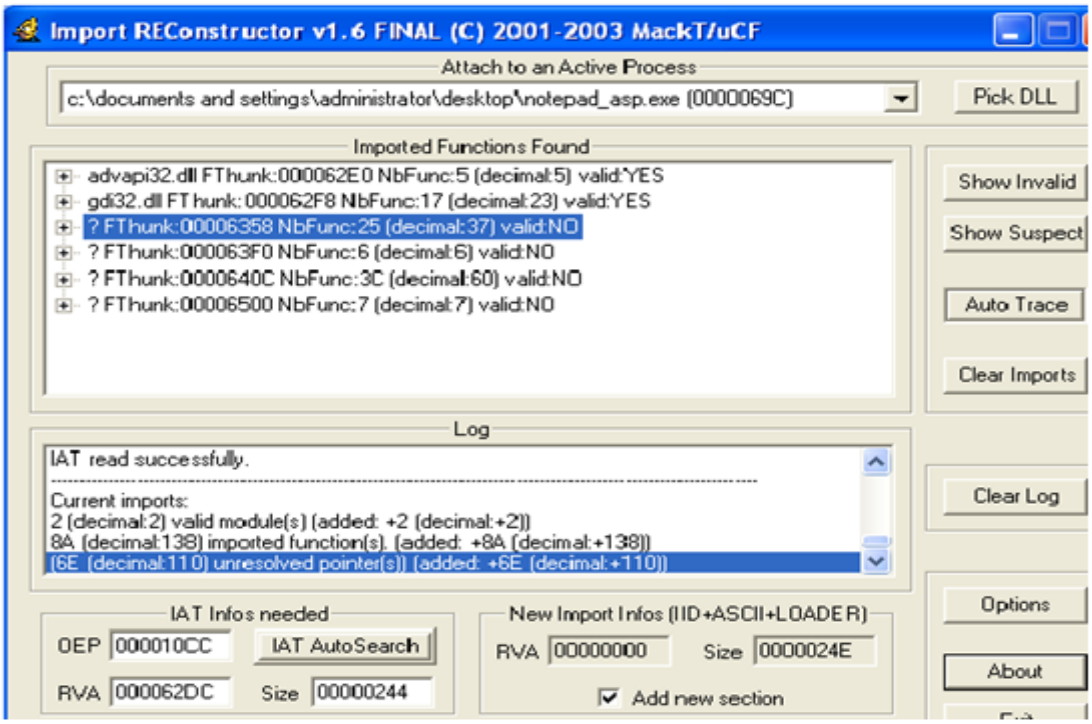
http://www.dependencywalker.com/

Là một công cụ để tìm các file Dll và hàm import của mã độc hại. Chương trình được sử dụng trên cả x86, x64 được xây dựng theo một sơ đồ phân cấp của tất cả các file Dll sẽ được nạp vào bộ nhớ khi phần mềm độc hại được chạy.



Hình 4-5: Dependency Walker

Là công cụ sử dụng thực hiện việc unpack, sau khi dump debugged process ta sẽ có 1 file (.exe),người phân tích cần phải sử dụng công cụ này để fix lại bảng IAT bằng tay hoặc tự động fix.

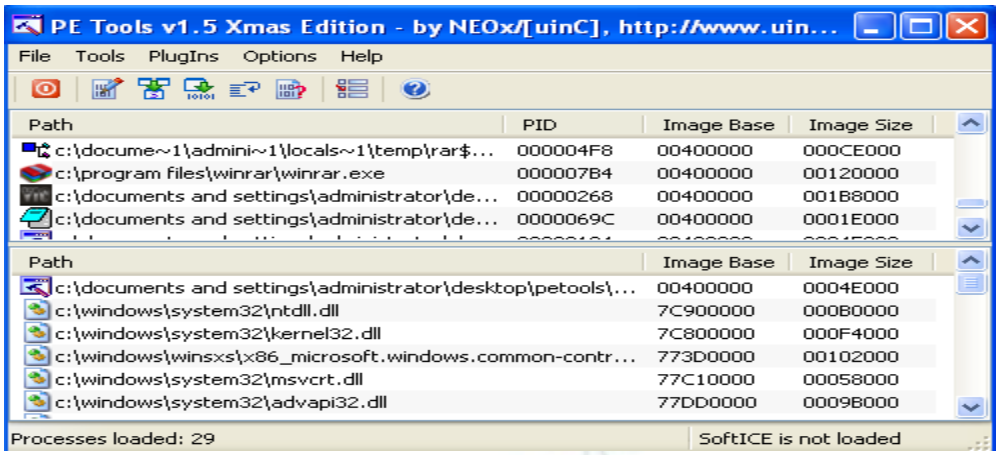


Hình 4-6 ImportREC

4.1.3 Công cụ PE

Đây là công cụ có đầy đủ chức năng để làm việc với file PE/PE+(64 bit). Bao gồm Task Viewer(dump full, process dump…), PE Sniffer(tìm trình biên dịch, packer..) , PE Editor(xem và chỉnh sửa các bảng import

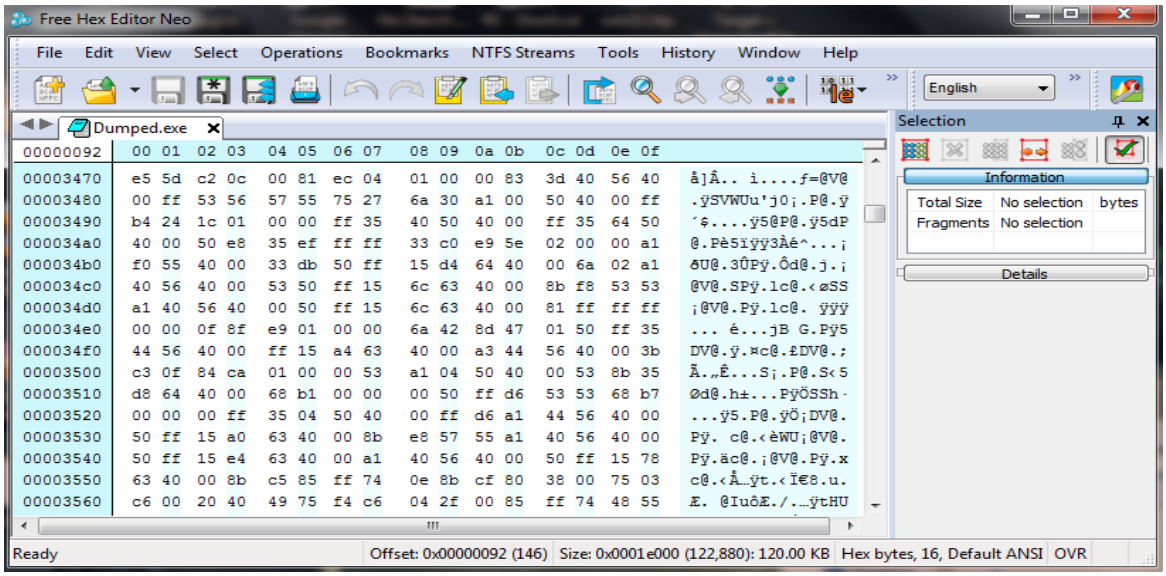
export, sửa CRC...), PE Rebuilder(Thay đổi địa chỉ cơ sở PE của một tập tin..)



Hình 4-7 PE Tools

4.1.4 Công cụ HexEditor

Hex Editor là bộ hoàn chỉnh các công cụ phát triển hệ thập lục phân cho Microsoft windows. Nó tích hợp tiên tiến chỉnh sửa và giải thích dữ liệu nhị phân, với cách nhìn trực quan, xử lí linh hoạt một tiến trình. Với hex editor người phân tích có thể chỉnh sửa, cắt, sao chép, dán, chèn, xóa dữ liệu nhị phân.



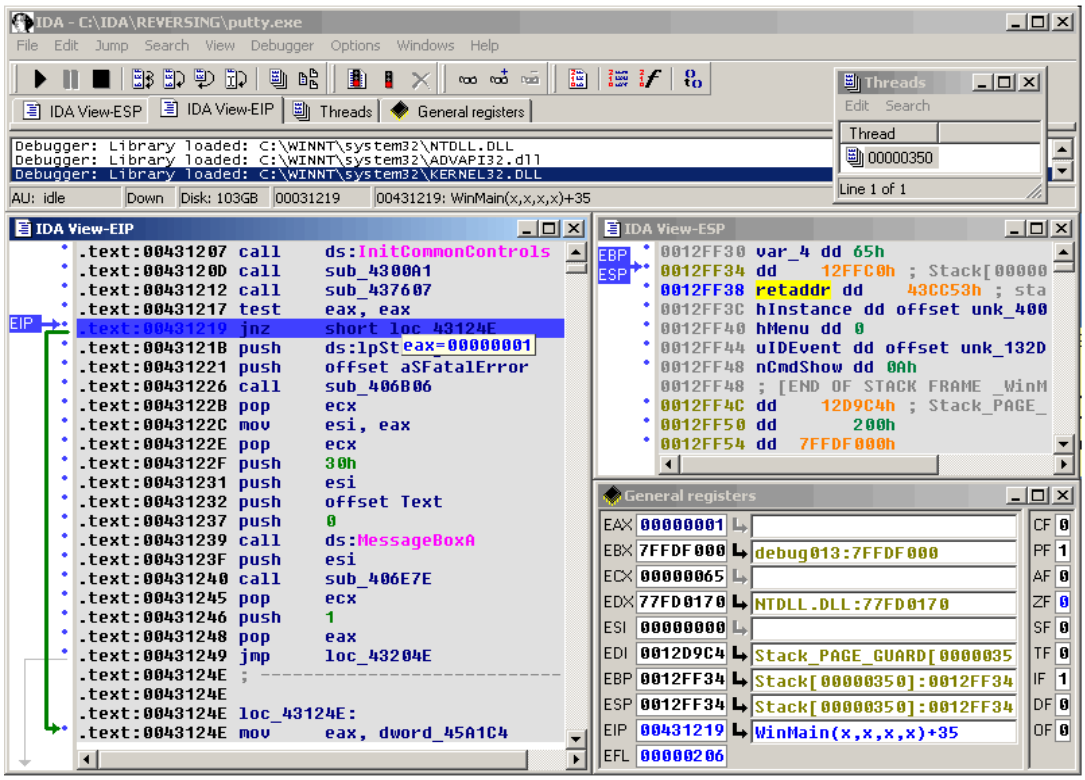
Hình 4-8 HexEditor

4.1.5 IDA pro

4.1.5.1 Giới thiệu chung:

IDA Pro là công cụ được sử dụng rộng rãi nhất để phân tích phần mềm độc hại. IDA pro giúp dịch ngược các mã độc hại về các đoạn mã

assembly. Nó là một công cụ dễ đọc, trực quan và tiện dụng với nhiều chức năng nhưng cũng chính vì vậy mà việc sử dụng IDA Pro cũng khá phức tạp. đi kèm với IDA ta có thể cài đặt thêm nhiều plugin như Bindiff, Hex-Rays Decompiler. Đây là hai plugin hỗ trợ đắc lực cho IDA giúp so sánh các biến thể của phần mềm độc hại, xác định các chức năng mới trong biến thể đó và cho biết nếu có bất kỳ chức năng tương tự bị mất đi.



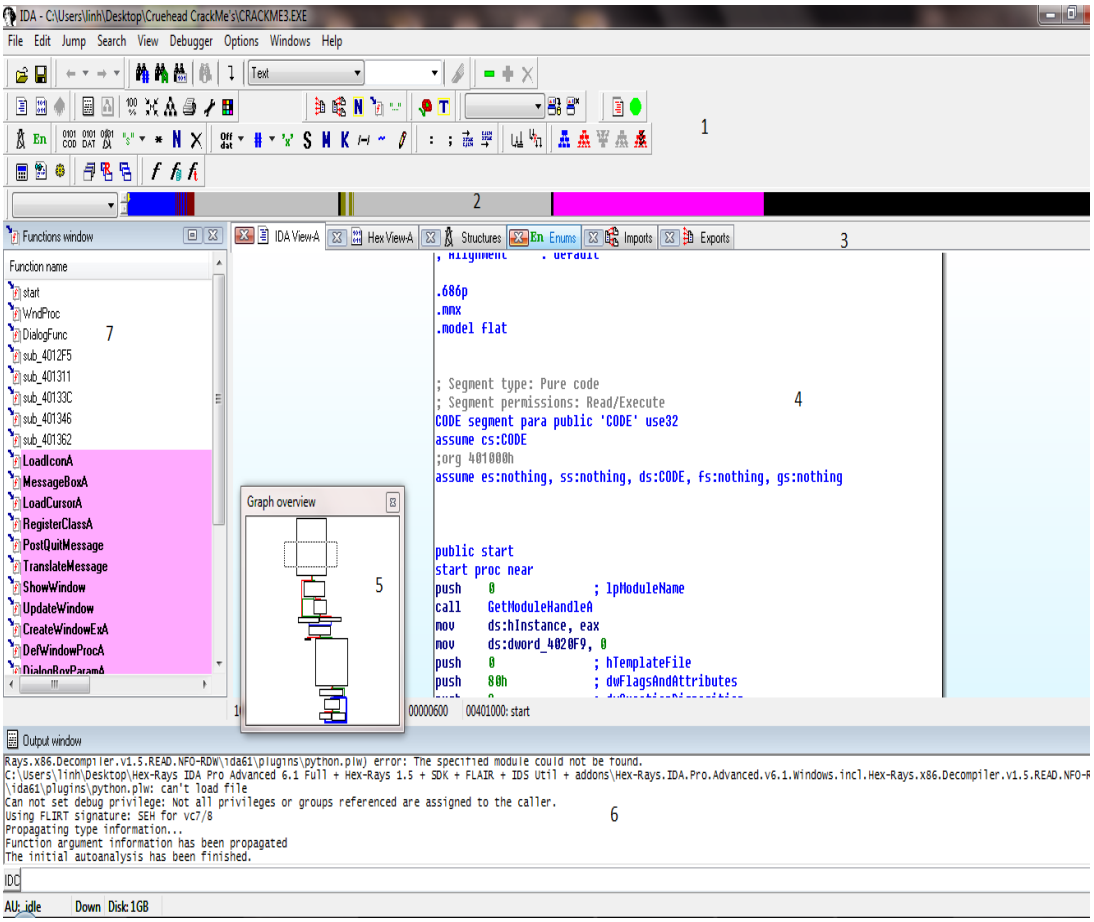
Hình 4-9: IDA pro

Có những loại mã độc hại không debug hay phân tích được bình thường mà phải sử dụng các chương trình Decomplier dành riên cho chúng. Vì vậy ta sẽ cài đặt thêm một số công cụ phục vụ cho việc phân tích như sau.

4.1.5.2 Các chức năng cơ bản

Sau khi người phân tích đưa một file cần phân tích vào, chương trình sẽ hiện lên một hộp thoại bên trong đó có chứa định dạng file và chọn PE(pe.ldw) để làm việc. Trong Option là các tùy chọn để cho IDA tự phân tích file vào cho ra hiển thị ở cửa sổ chính. Sau đó nhấn OK để vào giao diện chính để làm việc.

Giao diện là việc của IDA gồm có 7 cửa sổ:



Hình 4-10 Giao diện làm việc IDA pro

(1) Toobar chứa các thanh công cụ sử dụng trong hoạt động của IDA, để thực hiện các chức năng ta nhấp vào các biểu tượng trên đó.

(2) Navigation band nơi có thể vào và ra khỏi địa chỉ bằng cách nhấp chuột di chuyển bằng cách điều chỉnh theo hướng mũi tên màu vàng. Mỗi màu sắc của cửa sổ thể hiện vùng dữ liệu mà ta làm việc.

(3) Tabs có chứa cửa sổ nhỏ những thông tin chi tiết về file đối tượng, việc phân tích phụ thuộc vào những tabs này. Gồm có IDA View-A, Hex View-A, Struct, Enums, Imports,Exports, String.

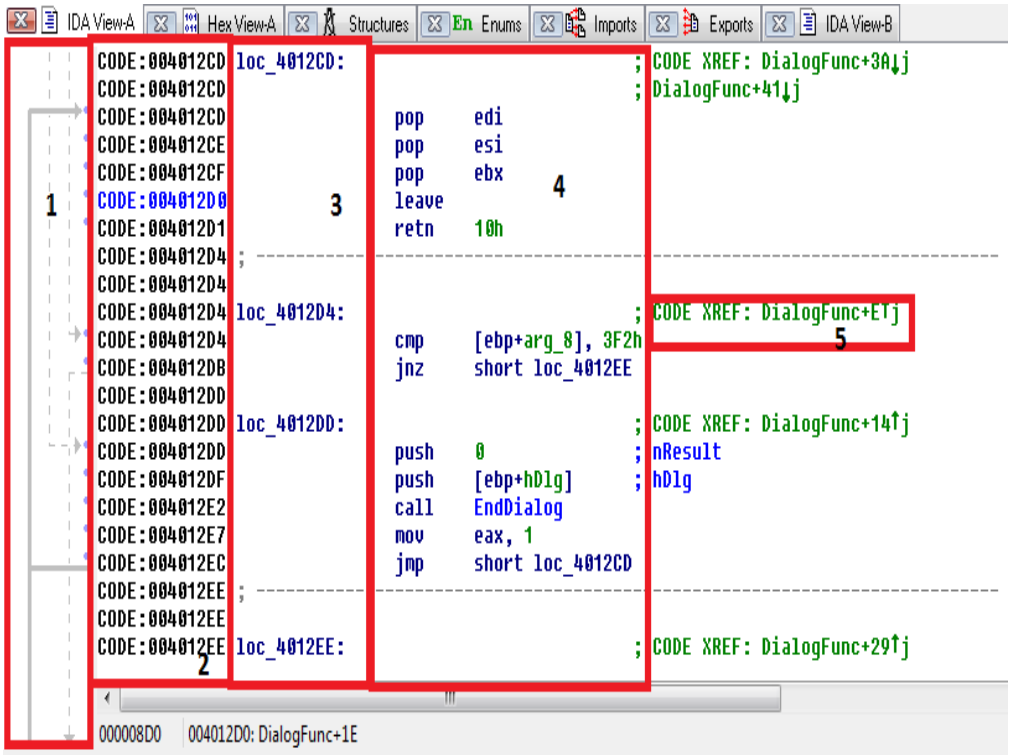
(4)Disassembly hiển thị dữ liệu để phân tích theo 2 loại text hoặc graph

(5)Graph overview một đồ thị thu nhỏ mô tả cấu trúc cơ bản của dữ liệu. Màn hình chữ nhật chấm nhỏ cho thấy màn hình hiển thị vị trí hiện tại đang làm việc.

(6) Output window nhận được các thông tin, tin nhắn từ IDA sau khi load file xong.

(7) Function window đây là cửa sổ hiện thị tất cả các hàm API các địa chỉ tìm thấy trong quá trình phân tích.

IDA Text view màn hình hiển thị kiểu text trình bày toàn bộ về công việc disassembler cũng như cung cấp địa chỉ để xem các dữ liệu thuộc vùng nào.



Hình 4-11: Cửa sổ IDA view

(1) Các mũi tên thể hiện nơi nhảy đến trong khối đoạn mã và nhận ra các vòng lặp nhỏ.

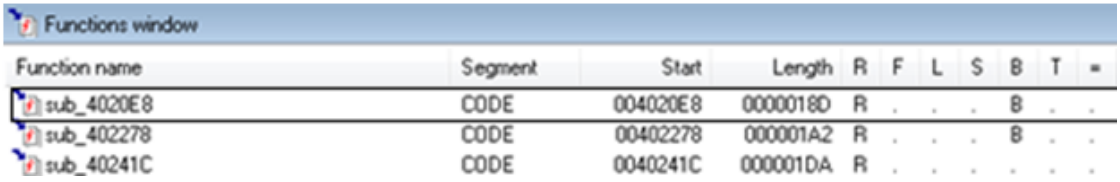
(2) Hiện ra các địa chỉ Virtual Address

(3) Các vị trí được so sánh để đánh dấu nhảy hoặc các biến tham chiếu trong stack

(4) Code của chương trình disassembled

(5) Code tham chiếu hiển thị các điểm đến khi truy cập, nhấp đúp vào sẽ đưa ta đến đoạn mã, hoặc hàm được gọi hoặc nhảy lên phía trên. Hoặc để nhảy đến vị trí tham chiếu khác ta nhấp chuột phải->jump to cross reference.

Fucntion Window hiển thị các hàm được nhận định bởi IDA

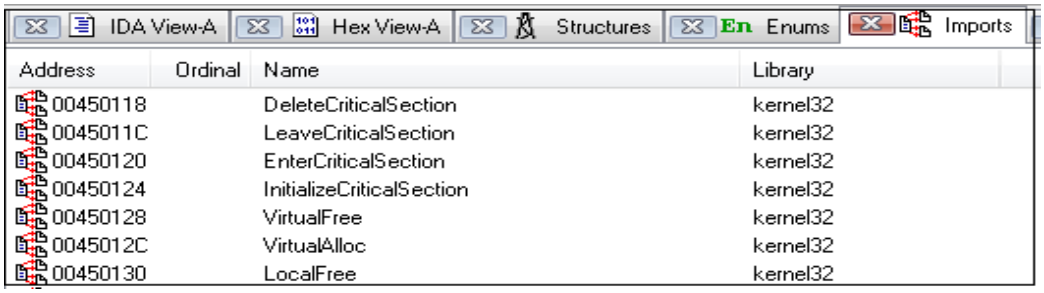


Hình 4-12: Cửa sổ Function

Bằng cách nhấp chuột hoặc enter là ta có thể di chuyển đc đến hàm đó trong cửa sổ chính disassembly. Ta nhìn thấy bên trong function window gồm có function name, segment, start, length, R(return hàm trả về khi gọi),F (far),L (libarary), S (static), B(BP tham chiếu đến biến cục bộ),T (type thông tin).

String Window tại cửa sổ này hiển thị ra thông tin tất cả các chuỗi xuất hiện trong file thực thi. Cửa sổ này hiện thị chi tiết về địa chỉ, độ dài, kiểu, tên chuỗi đó là gì. Ta có thể truy cập đến các chuỗi này bằng cách nhấn đúp chuột nó sẽ di chuyển đến cửa sổ Disassembly để chúng ta thực hiện.

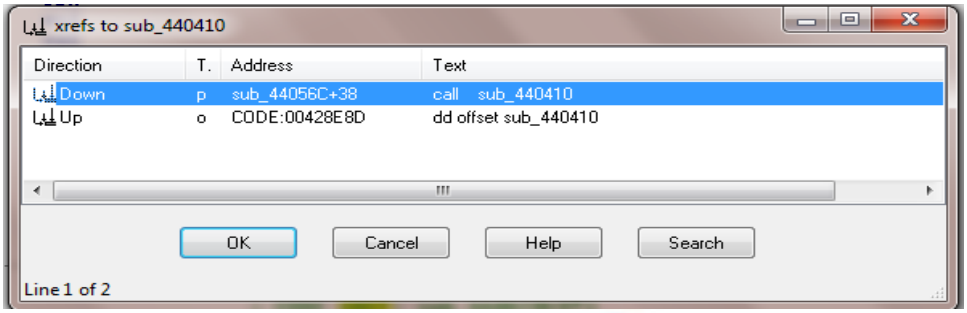
Import window cửa sổ hiển thị chi tiết địa chỉ, tên hàm được import và thư viện chứa hàm import đó. Đây là một cửa sổ rất quan trọng bởi ta có thể thấy được chương trình sử dụng các DLL khác nhau và chức năng của các hàm được gọi như đọc, viết hoặc registry .... Tại đây ta có thể thập được các thông tin về các hàm import mà mã độc hại hay dùng để chèn vào với mục đích xấu như lấy cắp thông tin, theo dõi …



Hình 4-13: Cửa sổ Import

Export window là cửa sổ liệt kê các entrypoint của tập tin. Trong cửa sổ này gồm có liệt kê theo tên, địa chỉ ảo và thứ tự(nếu có). Đối với các file thực thi export window cần phải chứa ít nhất một entry point đây là địa chỉ điểm đầu vào đầu tiên trong lúc thực thi.

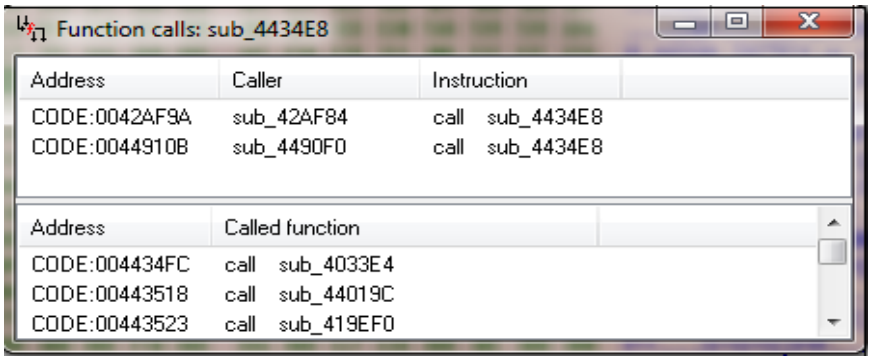
Cross-references cửa sổ hiện thi tất cả các tham chiếu , là tất cả các điểm code nơi mà hàm được gọi. Để mở được cửa sổ này ta chỉ nhấn vào header của hàm, kích chuột phải chọn jump to reference xref hoặc chọn trên thanh công cụ view-> Open subview-> cross reference.



Hình 4-14: Cửa sổ tham chiếu Cross- references

Name Window là cửa sổ cung cấp danh sách các tên được sử dụng, tên có thể được sắp xếp theo bảng chữ cái hoặc để virtual address. Hiển thị tên có các chữ cái in hoa khác nhau A(string data), D(data), C(Name code), I(import name), L (library fuction), F(regular function). Bằng việc nhấp đúp chuột vào tên bất kì ta muốn tìm, chương trình sẽ chuyển đến vị trí đó trong cửa sổ chính hiển thị để làm việc. Chọn view-> option subview-> name.

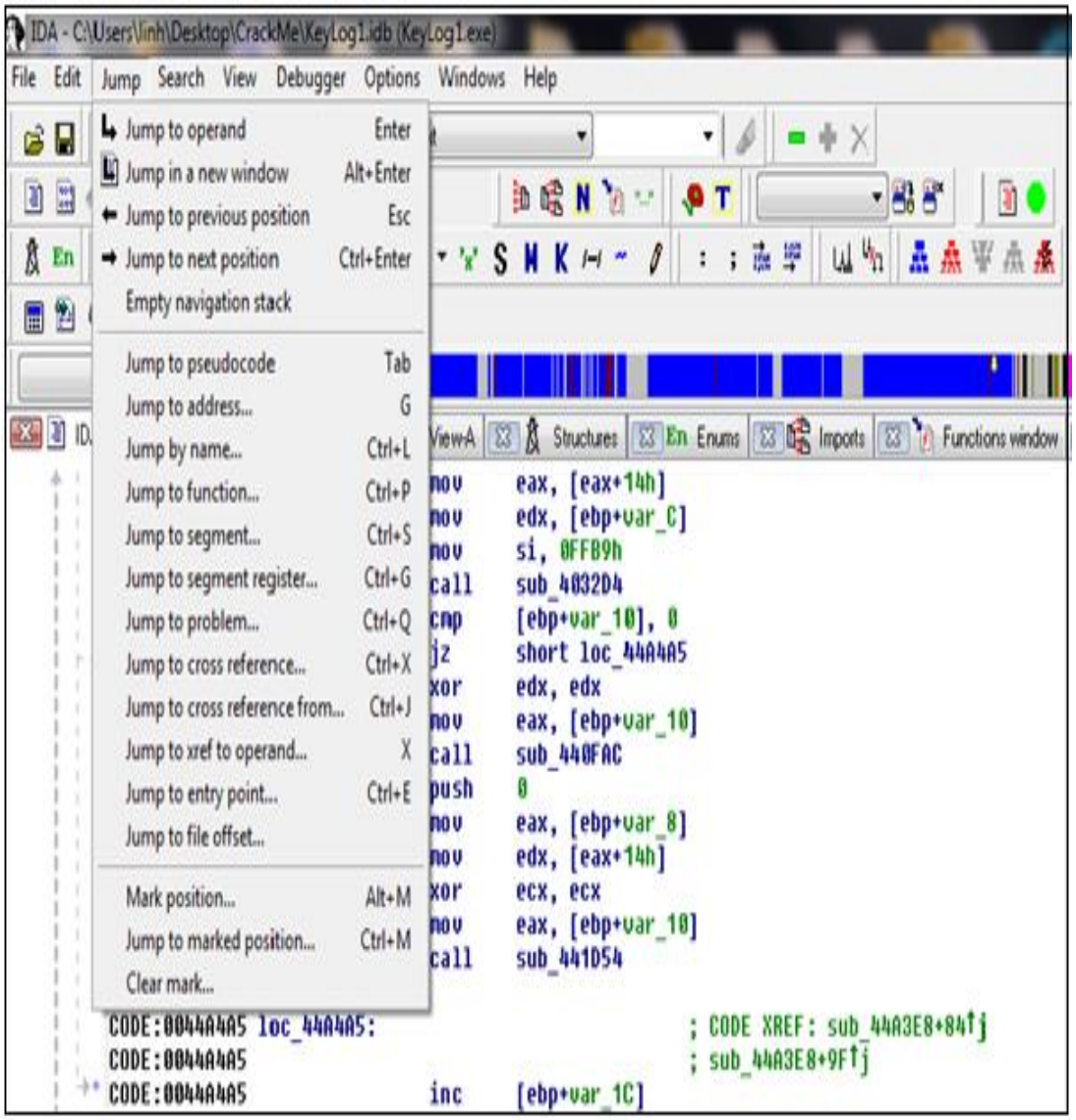
Function-Calls cửa sổ thể hiện được 2 chức năng caller và called function. Xác định được các hàm được gọi xung quanh đó là gì. View-> option subview-> function call.



Hình 4-15: Cửa sổ Function calls

Còn rất nhiều những cửa sổ khác với các chức năng khác nữa, những cửa sổ giới thiệu bên trên là những cửa sổ quan trọng nhất trong việc làm việc với công cụ IDA pro.

Menu Jump



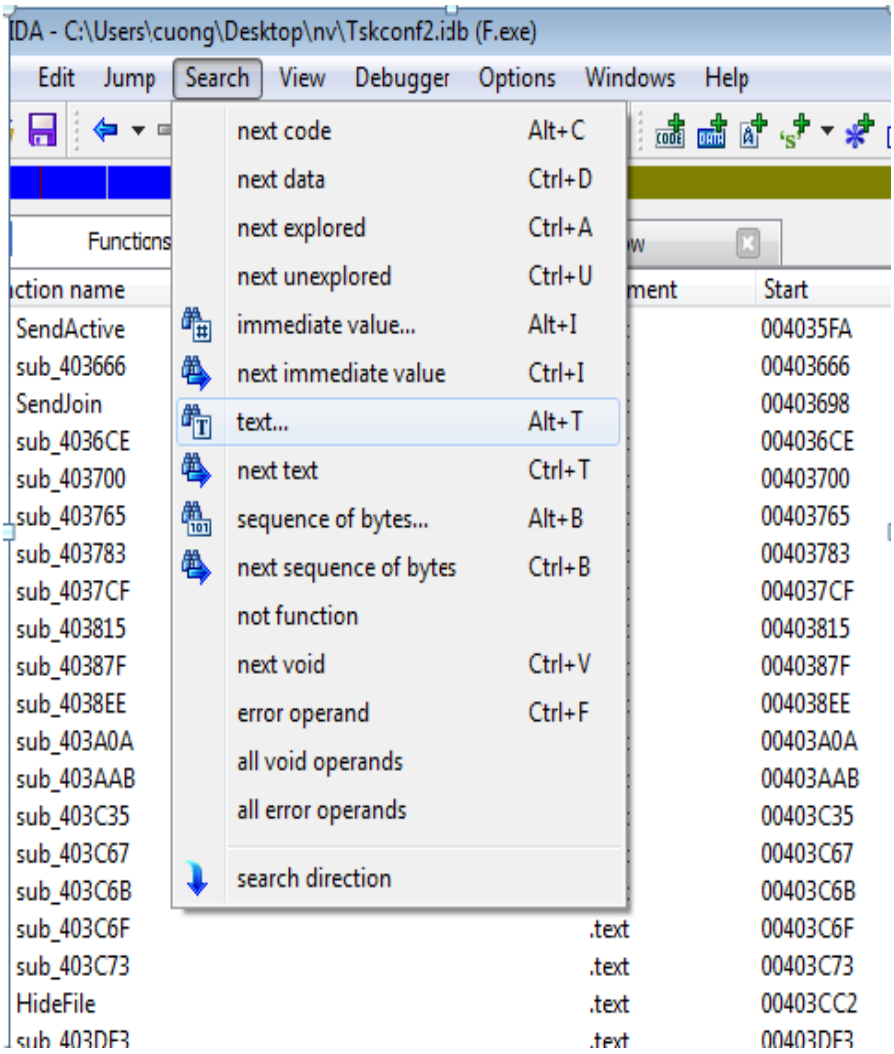
Hình4-16: Menu Jump

Jump to address(G) nhiều lúc ta sẽ biết được địa chỉ chính xác mà ta muốn đến để thuận tiện cho việc này IDA cũng tạo ra 1 cửa sổ để nhập vào địa chỉ muốn jump.

Jump to entrey point(ctrl-E) hiện lên vị trí các entry point ban đầu chỉ cần nhấp vào sẽ đưa ta đến vị trí làm việc ban đầu của nó.

Sử dụng forward/ backward arrows bằng cách sử dụng nút jump trên thanh công cụ chọn jump to previous(Esc) hoặc jump to next(ctrl-enter), hoặc sử dụng nút trên màn hình làm việc.

Menu Search với các tùy chọn tìm kiếm:



Hình 4-17: Menu Search

Next code sẽ đi tìm đến vùng code tiếp theo(vùng này đã được định nghĩa, Disasm.

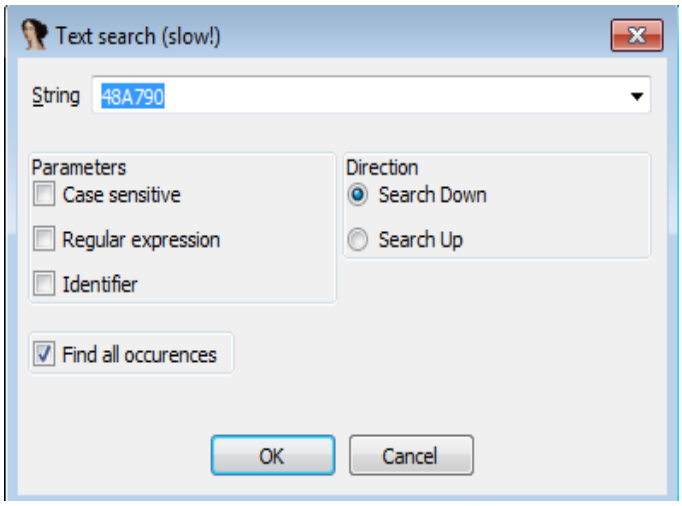
Next data là sẽ đi đến vùng chứa dữ liệu tiếp theo.

Next explored sẽ đi đến vùng dữ liệu tiếp theo được định nghĩa, sử dụng thành struct.

Next unexplored thì ngược với next explored.

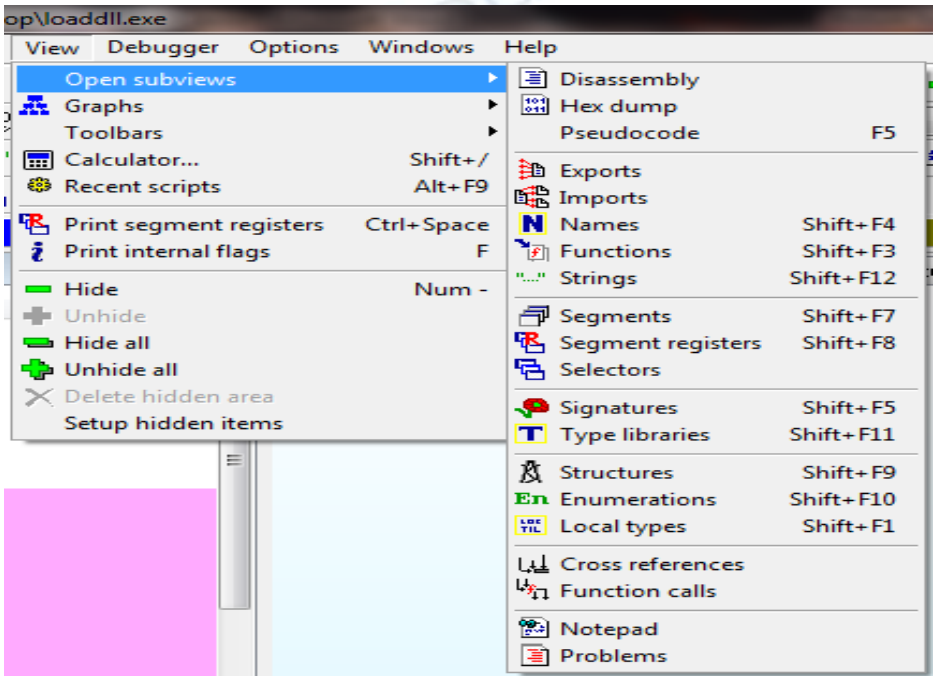
Immediate valua sẽ hiện ra cửa sổ tìm kiếm cá giá trị trong các struct, data.

Text tìm kiếm các chuỗi có ký tự cụ thể trong tất cả các hàm, dữ liệu.



Hình 4-18: Tìm kiếm chuỗi cụ thể

Menu View



Hình 4-19: Menu View

Với open subview là mở các cửa sổ view con trên các tab với: DisAssemBly, hexdump, export import … là các cửa sổ đã có sẵn. Có thể mở thêm các cửa số như:

PseudoCode là cửa sổ chứa mã C được tái tạo lại của function hiện tại của sổ Disaseembly.

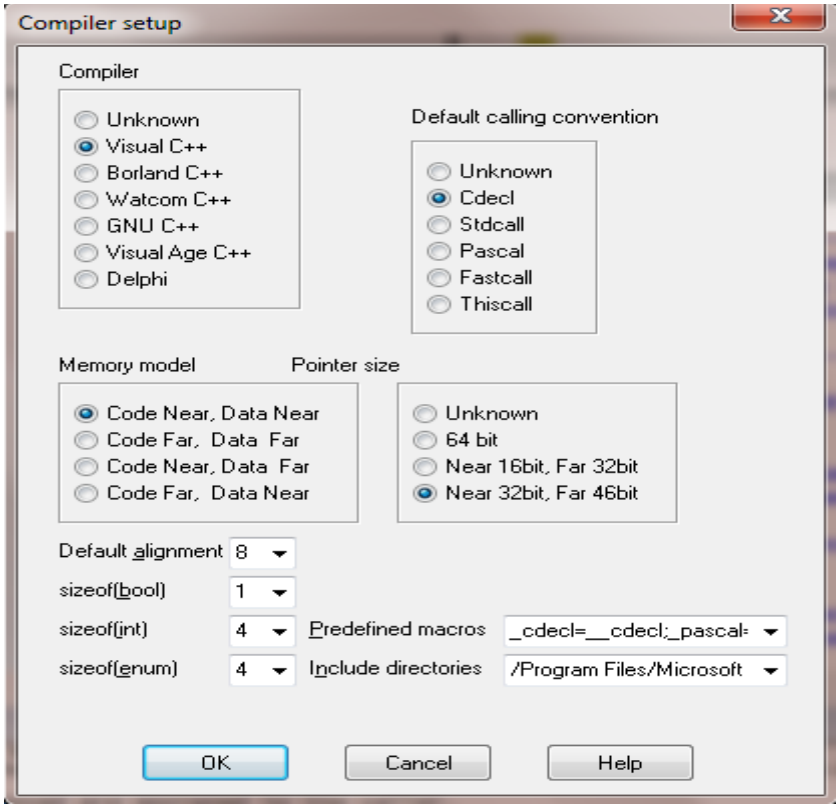
Segmen,SegmenRegister chứa các thông tin về các segmen, các thanh gi đoạn, các vùng dữ liệu.

Signature,Type Lib chứa các chữ ký để xác định các chương trình dùng thư viện gì, xây dựng trên ngôn ngữ, cơ sở gì.

Tiếp theo sẽ là menu debuger chứa các tùy chọn liện quan đến debuger gắn kèm theo IDA, nhưng ta tạm thời không xét vì các debuger này khá phức tạp, ta sẽ tìm hiểu debuger đơn giản hơn là OllyDebuger.

Menu Options

Menu này chứa các tùy chọn về complier:



Hình 4-20: Cửa sổ tùy chọn về complier

Disasembly cho phù hợp với các trình biên dịch

Tùy chọn string style chọn các kiểu string tùy theo trình biên dịch, kiểu dữ liệu.

Ngoài ra còn rất nhiều các tùy chọn liên quan dến cách dặt tên cách hiển thị biểu đồ, chú thích…

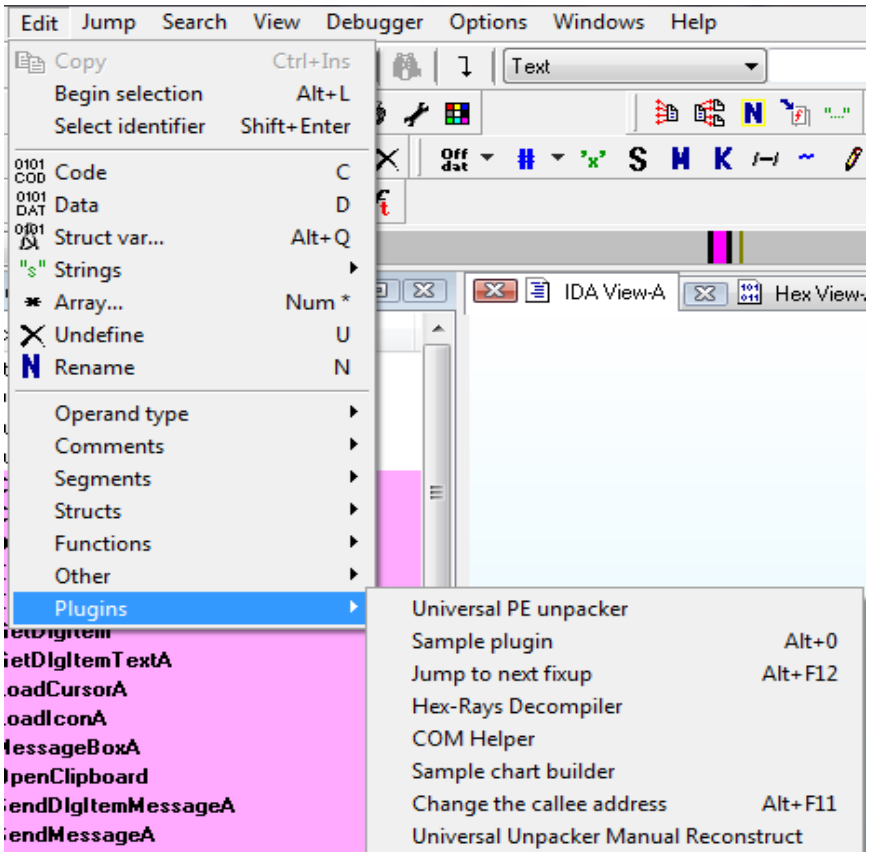
Menu Edit

Chứa các tùy chọn:

Code để định nghĩa một vùng dữ liệu bytecode thành AsmCode.

Data để địch nghĩa một vùng dữ liệu từ AsmCode thành các bytecode.

String sẽ định nghĩa lại kiểu của các string.



Hình 4-21: Menu Edit

Và nhiều tùy chọn khác liên quan đến việc chỉnh sửa các segments,struct, function…

Nhưng quan trọng nhất là Plugin nơi chứa các Pluign mà ta cài thêm vào.

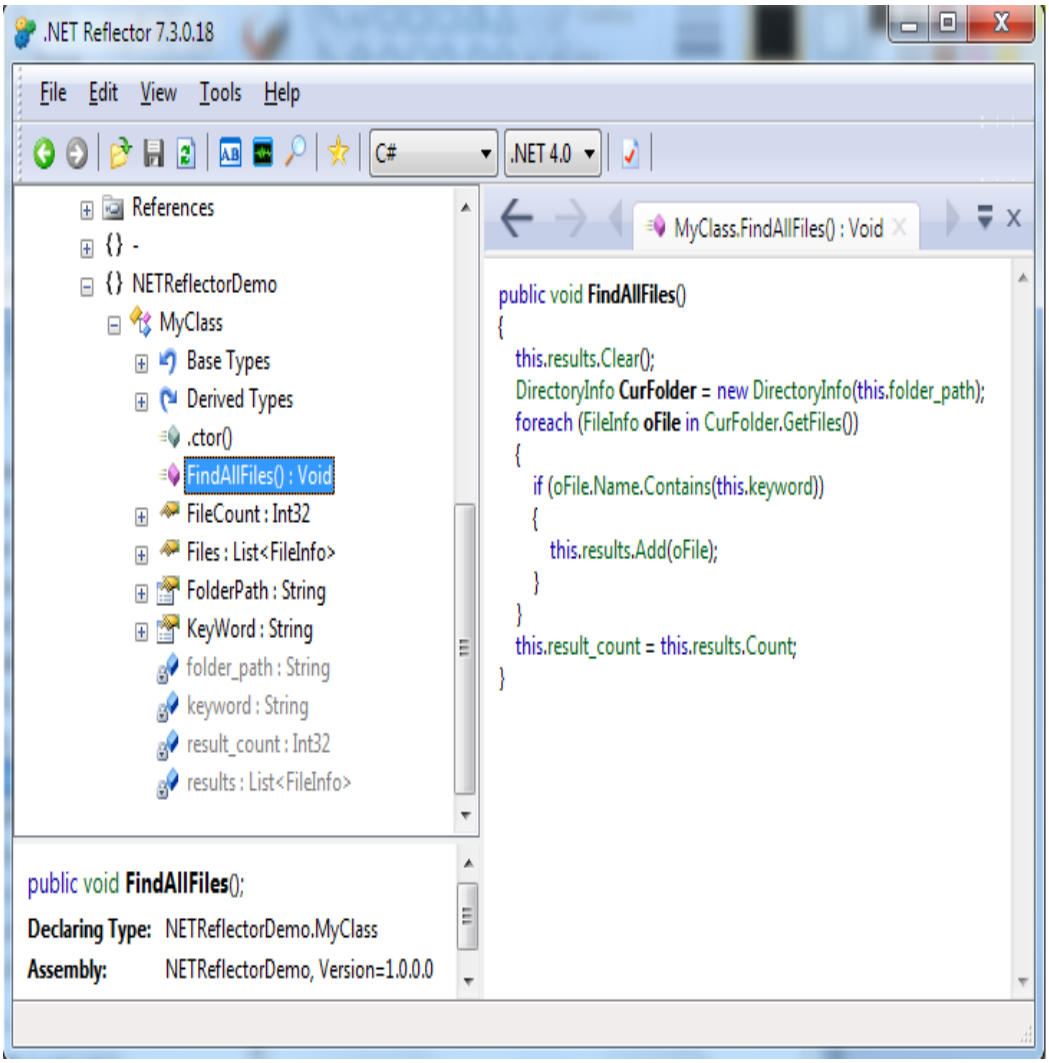
Đáng chú ý nhất là Plugin HexRays Decomplier với phím tắt là F5 sẽ dịch một đoạn Assembly thành một đoạn C.

Plugin BinDiff cũng là Một plugin rất quan trọng cho phép ta so sánh cở sở dữ liệu hiện tại với một cơ sở dữ liệu khác. Chỉ ra các function tương tự, giống nhau…

4.1.6 Công cụ Reflector

Đối với những loại được viết trên nền .NET cũng có các công cụ Decomplier riêng cho nó như Reflector, công cụ này cho phép ta dễ dàng quan sát, tìm kiếm, biên dịch ngược chương trình viết trên nền .NET trở về mã nguồn C#, VB, C++, Delphi, IL, F#, Oxygene…

Link download : http://shop.reflector.net/download

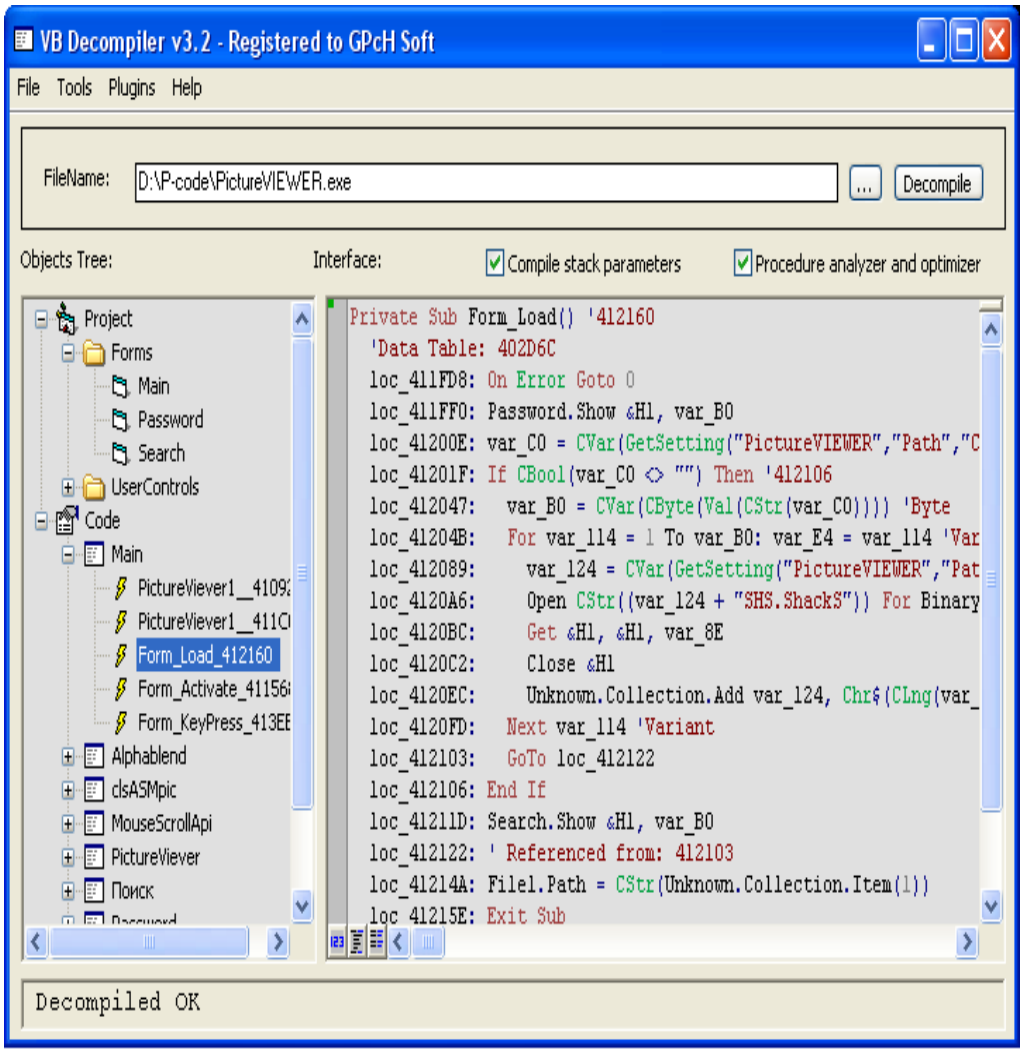


Hình 4-22: Giao diện phần mềm Reflector

4.1.7 Công cụ VB Decomplier

Link download http://www.vb-decompiler.org/

Như ta đã viết các chương trình trong Visual Basic có thể được dịch thành p-code(mã giả) đã được dịch hoặc thành code gốc. Kể từ khi p-code bao gồm nhiều lệnh cấp độ cao thì đã có một khả năng thực sự để dịch ngược nó thành mã nguồn. Vì vậy ta cần VB Decomplier một trình dịch ngược cho các chương trình được viết trên chương trình Visual Basic(EXE,DLL hoặc OCX) để khôi phục nhiều lệnh p-code. Ngoài ra một bộ phân tích mã giúp tìm kiếm tất cả các chướng gọi chức năng API và tham chiếu chuỗi trong mã được phân tác và rồi thay đổi chúng thành các diễn giải cho các chuỗi phân tích. Nó là công cụ cần thiết nếu như mất mã nguồn và cần khôi phục lại.



Hình 4-23: VB Decomplier

Ngoài các công cụ đã kể trên vẫn còn nhiều loại công cụ khác để phục vụ cho việc cho việc phân tích tĩnh.

Ưu điểm phương pháp phân tích tĩnh này cho ta xem được code để xem mã độc hại làm gì chứ không cần phải chạy trên ngay hệ thống, hiểu được rõ nhất cũng như biết được các hoạt động chính xác của mã độc hại là làm gì. Tuy nhiên để thực hiện được phương pháp này đòi hỏi người phân tích phải hiểu rõ về lập trình và hệ thống.

4.1.8 Công cụ Ollydebug

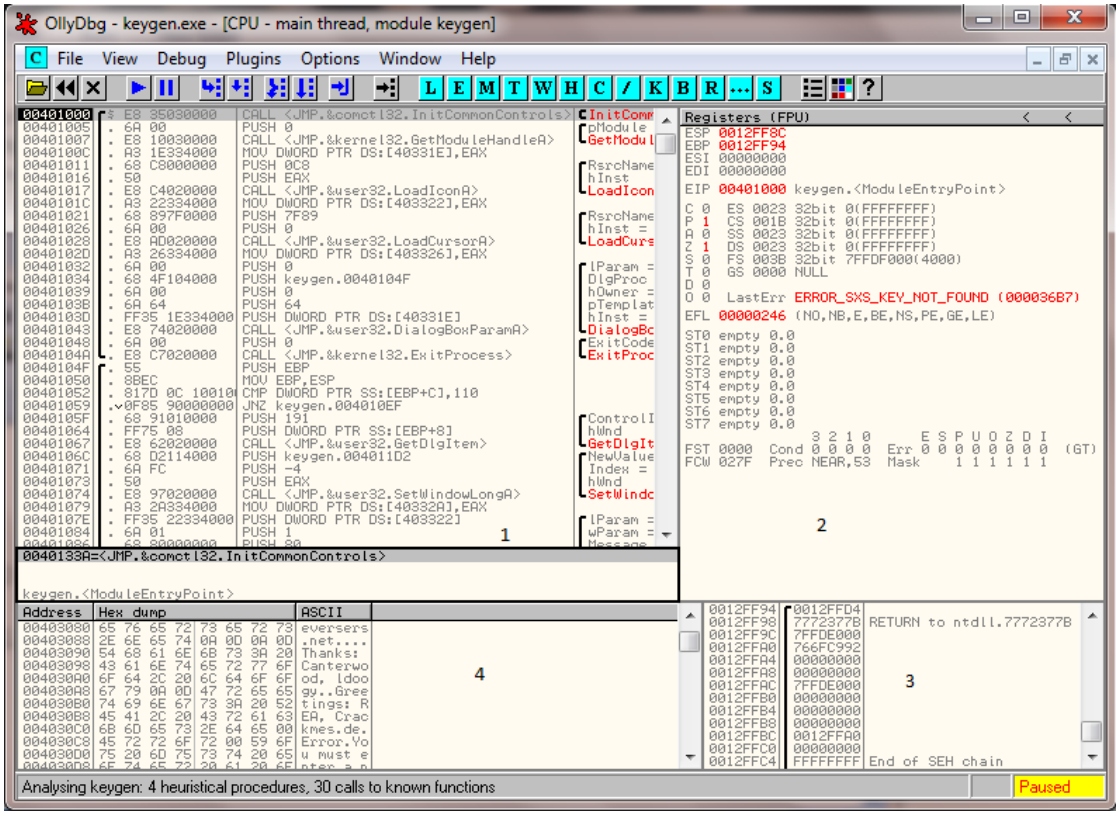
4.1.8.1 Giới thiệu chung

Phần mềm này giúp người phân tích có thể debug, xem quá trình chạy từng bước của chương trình.

Dowload phần mềm tại trang http://www.ollydbg.de/

4.1.8.2 Các chức năng cơ bản

Đầu tiên người phân tích load file vào trong ollydbg ta sẽ được giao diện chương trình như sau:



Hình 4-24 Giao diện làm việc Ollydbg

(1) The Dissembler Window: ở cửa sổ này ta nhìn thấy các đoạn chương trình ở dạng asm cùng với những lời giải thích cho các đoạn mã asm.

(2) The Registry Window: đây là cửa sổ chứa thông tin chi tiết về các thanh ghi ,các cờ trạng thái.

(3) The Stack Window: hiển thị trạng thái của stack, lưu dữ tạm thời các dữ liệu và địa chỉ.

(4) The Dump Window: cửa sổ hiển thị nội dung của bộ nhớ hoặc file, cửa sổ này cho phép ta tìm kiếm thực hiện các chức năng chỉnh sửa…

Các tùy chọn với View

View->log (cửa sổ L) cho chúng ta biết thông tin mà Olly ghi lại. Theo mặc định cửa sổ này sẽ lưu thông tin về các module, import libarary hoặc các Plugins được load cùng chương trình tại thời điểm đầu tiên khi

load chương trình vào Olly, cùng với ghi lại các thông tin về BP. Và 1 chức năng nữa là khi ta muốn lưu lại thông tin về file Log ta chỉ việc nhấp chuột phải trong cửa sổ L và chọn Log to file

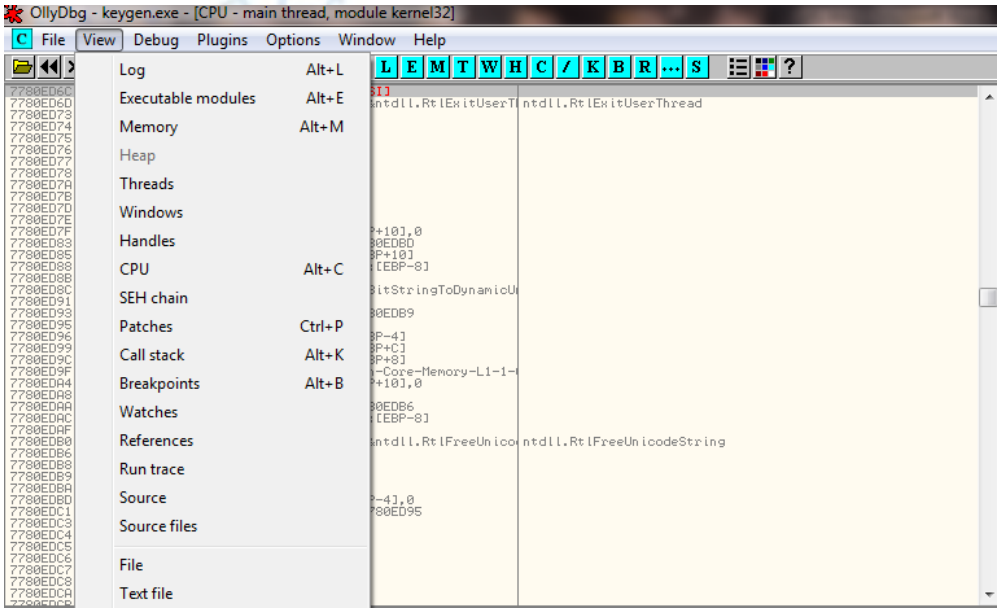
View-> Executable module(cửa sổ E) cửa sổ đưa ra danh sách những file có khả năng thực thi được chương trình sử dụng như file exe, dlls, …

View->Memory (cửa sổ M) cửa sổ này cho ta biết thông tin về bộ nhớ đang được sử dụng. Tại cửa sổ này chúng ta có thể sử dụng tính năng search để tìm kiếm thông tin về các string, các đoạn hexa cụ thể hay unicode… thêm vào đó cùng cấp cho chúng ta những kiểu thiết đặt BP khác nhau tại section.

View-> Threads (cửa sổ T) liệt kê các thread của chương trình.

View->window(cửa sổ W) để mở cửa sổ Window.

View-> Handles(cửa sổ H) để mở cửa sổ Handles.



Hình 4-25:Menu View

View->CPU(cửa sổ C) cửa sổ đang làm việc hiện tại.

View->Patches(cửa sổ /) cửa sổ này cho chúng ta biết thông tin về những gì mà chúng ta đã edit trong chương trình.

View-> Call stack(cửa sổ K) hiển thị một danh sách các lệnh Call mà chương trình của chúng ta đã thực hiện khi chúng ta Run bằng F9 và dùng F12 để tạm dừng chương trình.