

Mã độc

Chương 1. Tổng quan về mã độc

Mục tiêu

- Cung cấp một số kiến thức cơ bản về mã độc
- Giới thiệu cơ chế hoạt động của một số loại mã độc chính

Tài liệu tham khảo

[1] TS. Lương Thế Dũng, KS. Hoàng Thanh Nam,
2013, Giáo trình Mã độc, Học viện kỹ thuật Mật mã

Nội dung

1. Mã độc
2. Phân loại mã độc
3. Cơ chế hoạt động của mã độc

Nội dung

- 1. Mã độc**
- 2. Phân loại mã độc**
- 3. Cơ chế hoạt động của mã độc**

Mã độc

- Định nghĩa mã độc**
- Lịch sử của mã độc**
- Mục đích của mã độc**
- Con đường lây nhiễm mã độc**

Mã độc

- Định nghĩa mã độc**
- Lịch sử của mã độc**
- Mục đích của mã độc**
- Con đường lây nhiễm mã độc**

Định nghĩa mã độc

- Mã độc (Malwares) là những chương trình máy tính độc hại với mục tiêu là đánh cắp thông tin, phá hủy hay làm hư hỏng hệ thống.
- Những chương trình này xâm nhập hệ thống một cách trái phép (không có sự cho phép của người quản trị).

Định nghĩa mã độc

☐ **Mã độc** (Tên tiếng Anh là Malware hay Malicious software) là các chương trình máy tính được tạo ra với mục đích làm hại đến tính bí mật, tính toàn vẹn hoặc tính sẵn sàng của dữ liệu, ứng dụng và hệ điều hành của của hệ thống.

Mã độc

- Định nghĩa mã độc**
- Lịch sử của mã độc**
- Mục đích của mã độc**
- Con đường lây nhiễm mã độc**

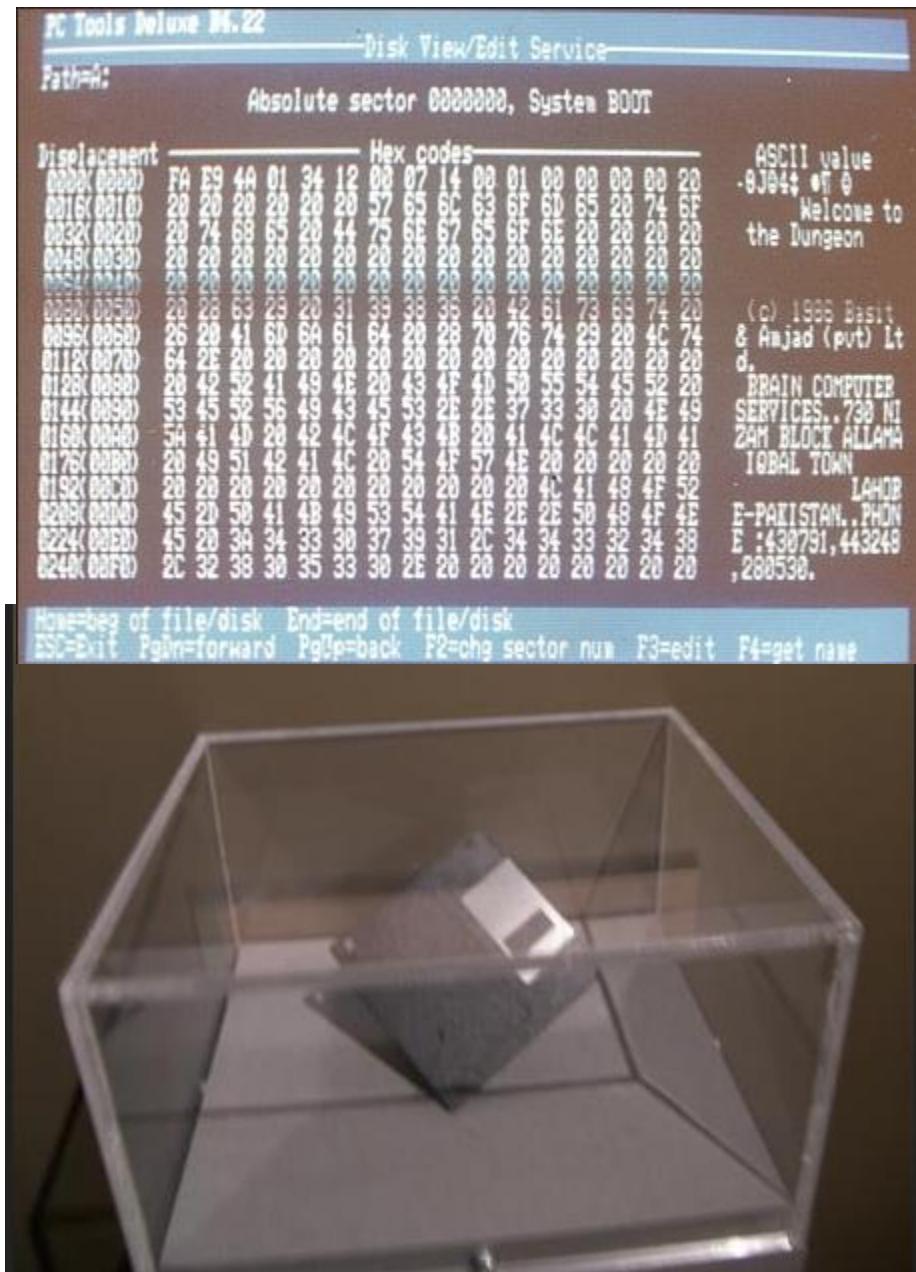
Lịch sử của mã độc

- Lịch sử của mã độc có thể coi được bắt đầu từ năm 1949 khi lý thuyết đầu tiên về các chương trình tự sao chép ra đời.
- Năm 1981 loại mã độc đầu tiên gọi là virus mới xuất hiện, virus này có tên là **Apple II**.

Lịch sử của mã độc

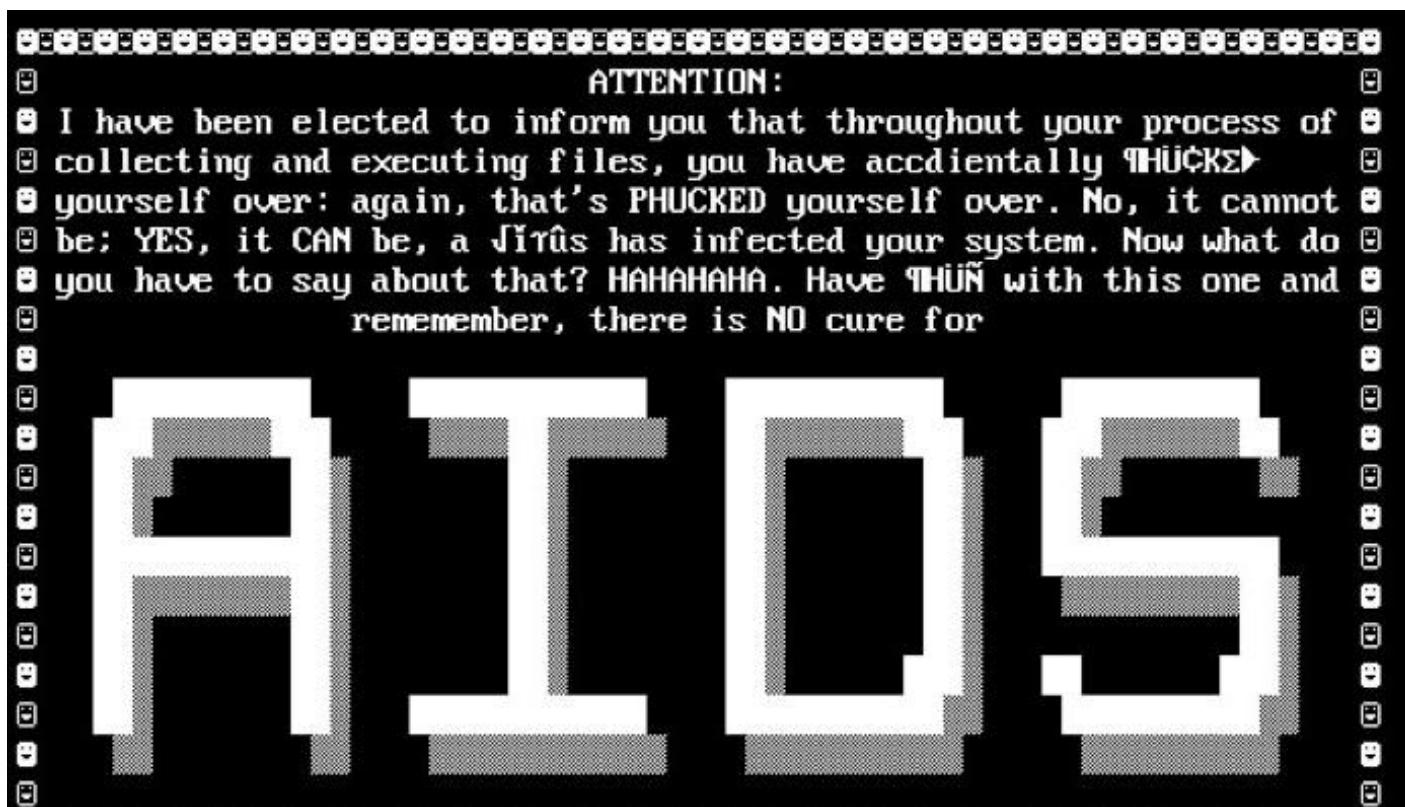
□ Năm 1986 virus Brain âm thầm đỗ bộ từ Pakistan vào nước Mỹ với mục tiêu đầu tiên là Trường Đại học Delaware.

□ 2/11/1988: Robert Morris đưa virus vào mạng máy tính quan trọng nhất của Mỹ, gây thiệt hại lớn. .



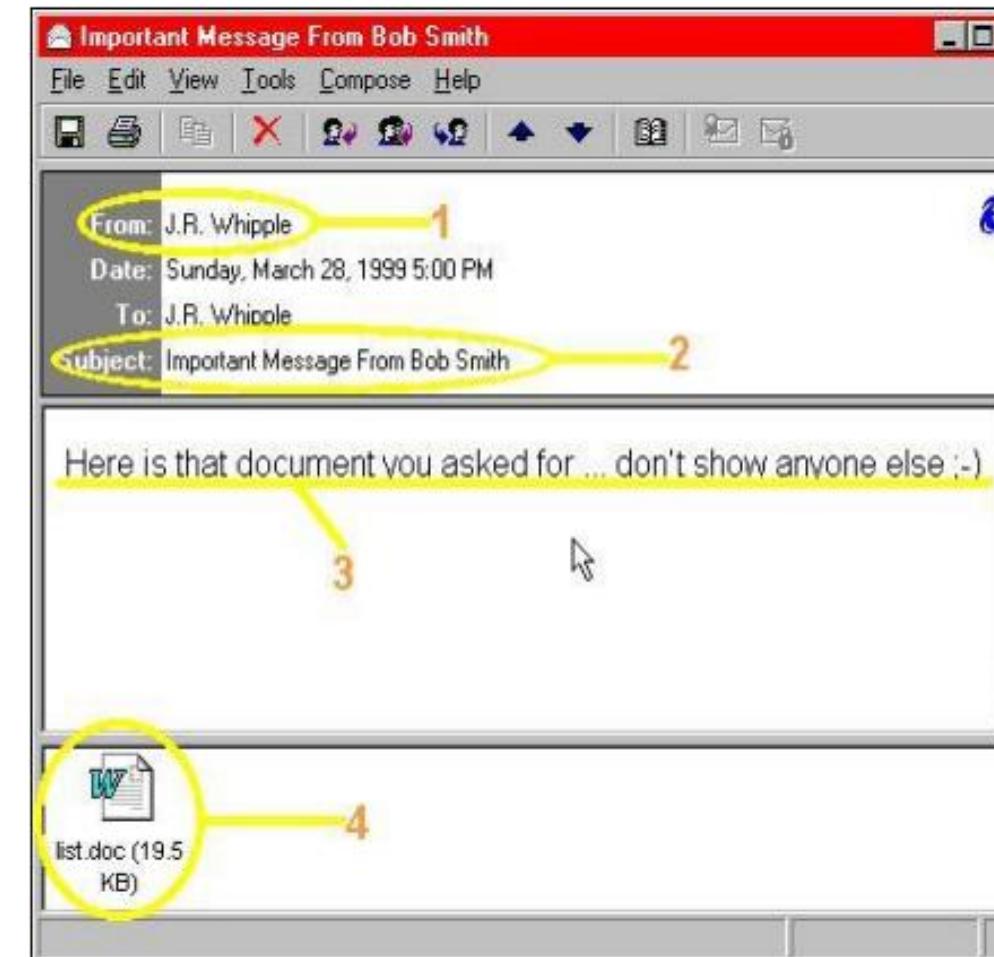
Lịch sử của mã độc

- Năm 1988: Virus Jerusalem xuất hiện, được kích hoạt vào thứ sáu ngày 13.
- Năm 1989: Xuất hiện chương trình Trojan có tên AIDS.



Lịch sử của mã độc

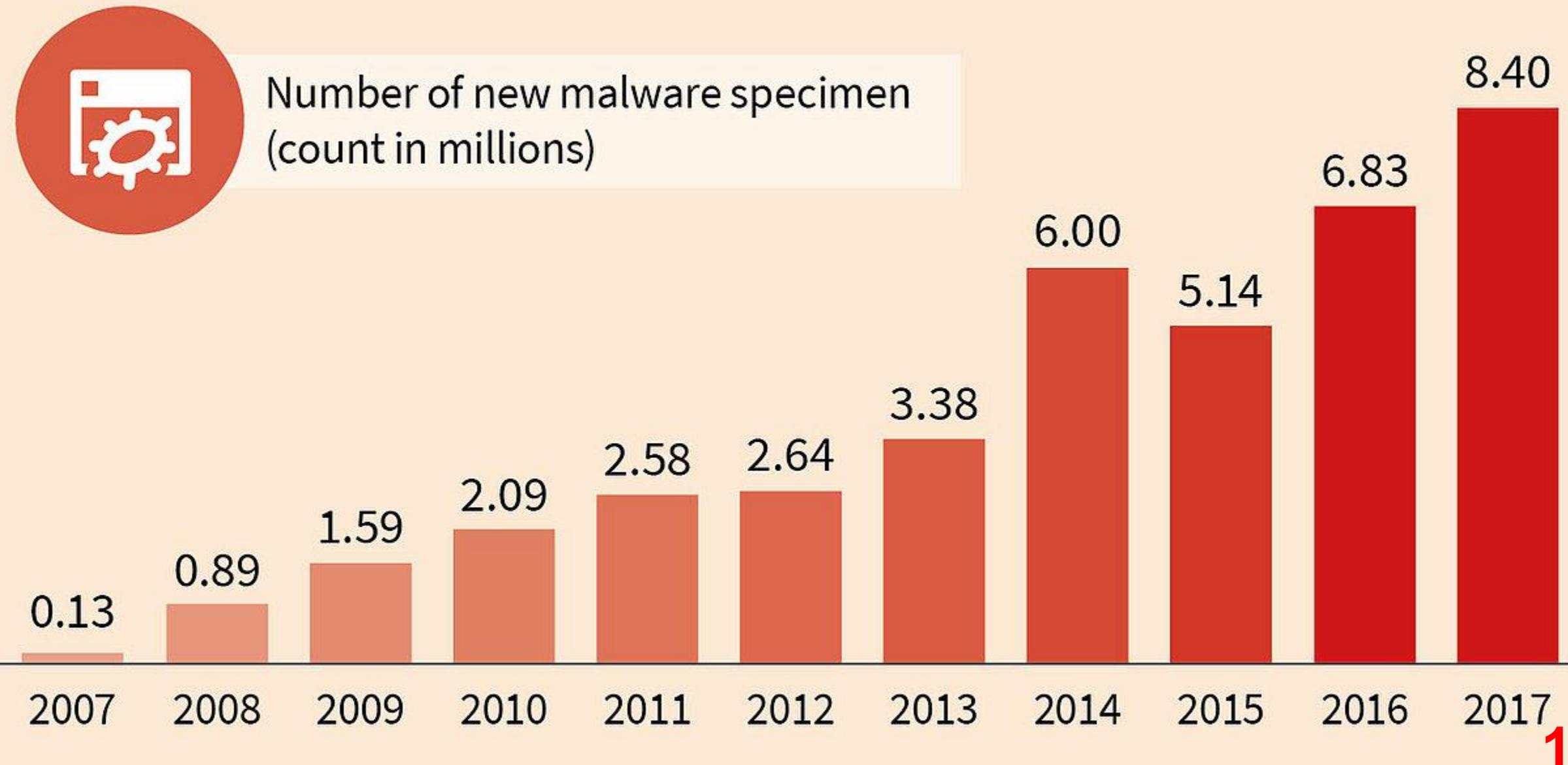
- Năm 1991: Tequila, một trong những virus phát tán dưới nhiều hình dạng đầu tiên được phát hiện.
- Năm 1996: Virus macro và virus Staog xuất hiện lần đầu tiên.
- Năm 1996: Virus Boza. Virus đầu tiên trên hệ điều hành Windows95.



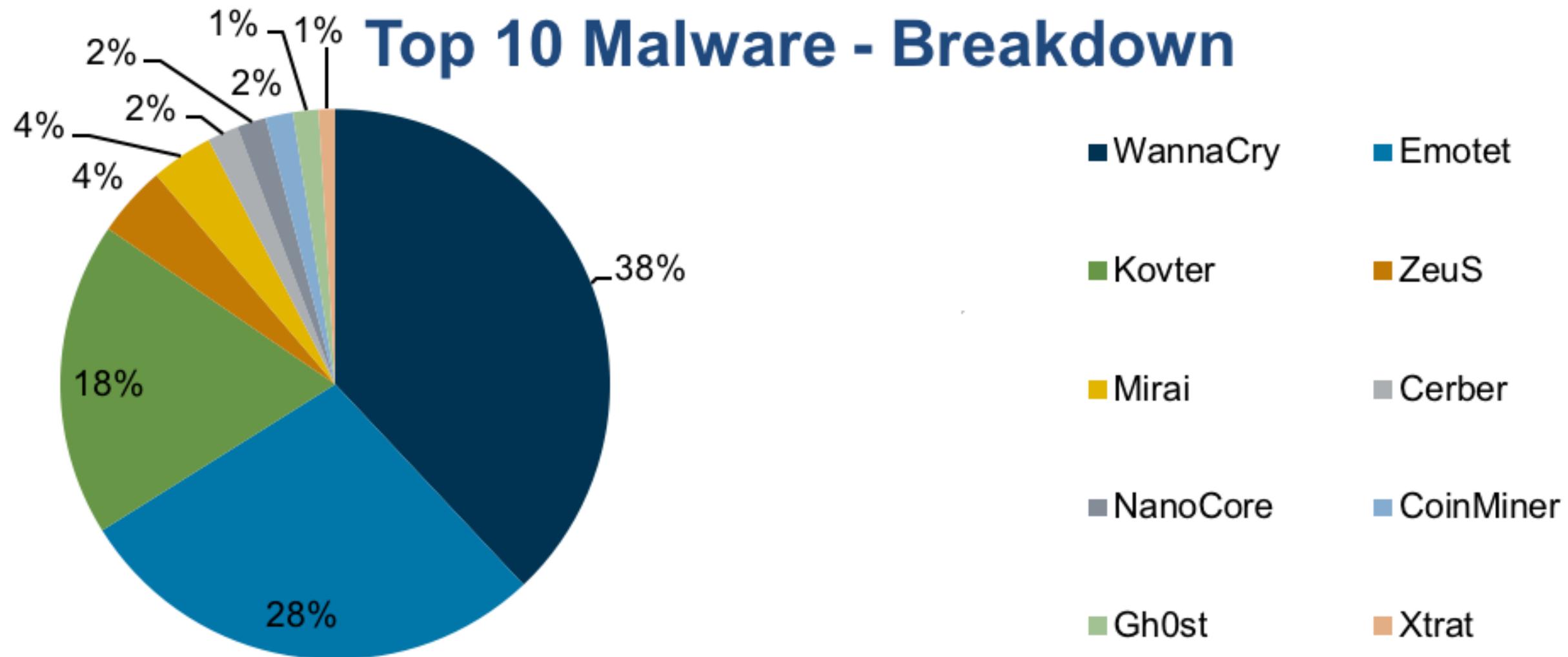
Lịch sử của mã độc

- Năm 1998: Strange Brew là virus đầu tiên lây nhiễm vào file Java.
- Năm 2001: Virus Winux (Windows/Linux), Nimda, Code Red. Virus Winux đánh dấu dòng virus có thể lây được trên các hệ điều hành Linux chứ không chỉ Windows.

Lịch sử của mã độc



Lịch sử của mã độc





Mã độc

- Định nghĩa mã độc
- Lịch sử của mã độc
- Mục đích của mã độc
- Con đường lây nhiễm mã độc

Mục đích của mã độc

- Hiển thị các quảng cáo;**
- Gian lận, lừa đảo;**
- Theo dõi hoạt động, lấy cắp thông tin của người dùng;**
- Chiếm quyền điều khiển máy tính;**
- Phá hoại hệ thống...**

Mã độc

- Định nghĩa mã độc**
- Lịch sử của mã độc**
- Mục đích của mã độc**
- Con đường lây nhiễm mã độc**

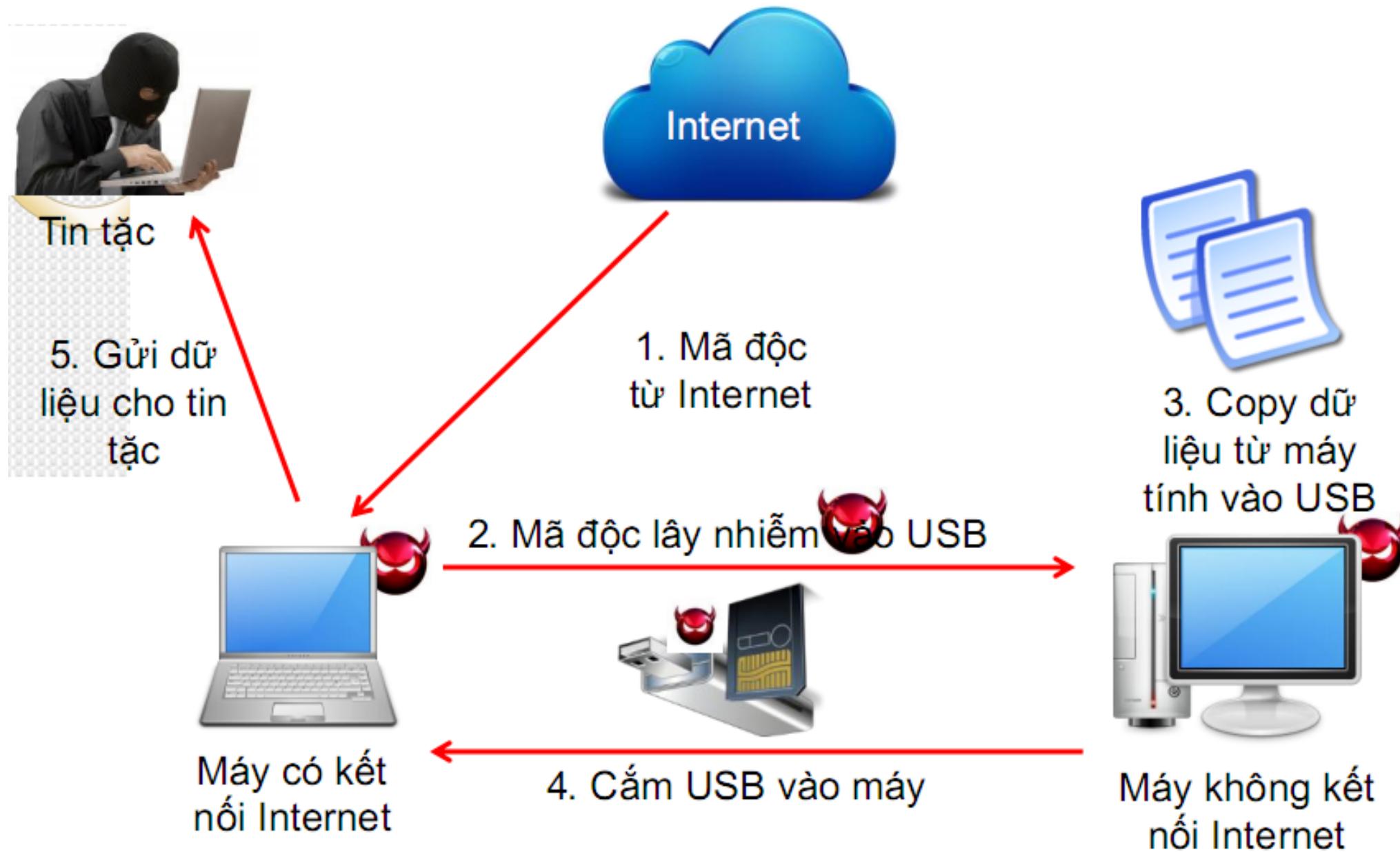
Con đường lây nhiễm mã độc

- Qua các thiết bị lưu trữ di động
- Qua thư điện tử
- Qua trình duyệt web
- Lây nhiễm từ smartphone sang máy tính

Con đường lây nhiễm mã độc

- Qua các thiết bị lưu trữ di động**
- Qua thư điện tử**
- Quá trình duyệt web**
- Lây nhiễm từ smartphone sang máy tính**

Qua các thiết bị lưu trữ di động



Qua các thiết bị lưu trữ di động

- **Hình thức lây nhiễm:** Mã độc lây nhiễm từ máy có kết nối Internet sang máy không kết nối Internet hoặc lây nhiễm từ máy tính này sang máy tính khác thông qua USB.
- **Cơ chế lây nhiễm:**
 - Khi cắm USB vào máy kết nối Internet, mã độc lây nhiễm vào USB (bằng các đường lây nhiễm kể trên).
 - Cắm USB sang máy không kết nối Internet, mã độc lây nhiễm vào máy này.

Qua các thiết bị lưu trữ di động

- Cơ chế lấy cắp dữ liệu: Mã độc tự động copy dữ liệu từ máy không nối Internet vào USB ở dạng ẩn.
 - Khi cắm USB sang máy có nối Internet, mã độc gửi tài liệu từ
 - USB đến hòm thư hoặc máy tính đích của tin tặc.

Ví dụ: **W32.XFileUSB**

Con đường lây nhiễm mã độc

- Qua các thiết bị lưu trữ di động
- Qua thư điện tử
- Quá trình duyệt web
- Lây nhiễm từ smartphone sang máy tính

Qua thư điện tử

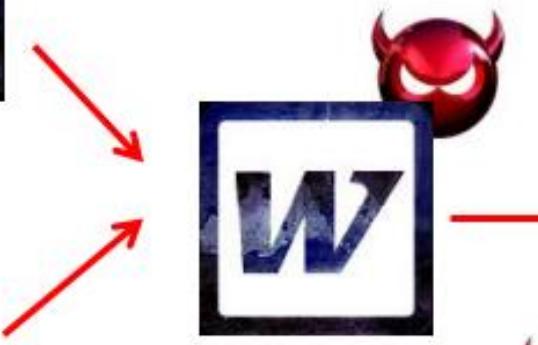


Tin tặc



.doc

+



Tệp .doc có
chứa mã độc

Mã độc



Đính kèm
vào thư
điện tử



Fake Mailer

Fake Mailer



Nạn nhân



Nạn nhân

Qua thư điện tử

Ngày Thứ 4, 25/07/12, NguyenThi LanHuong <lhuong...@...com> đã viết:

Từ: NguyenThi LanHuong <lhuong...@...com>

Chủ đề: Danh sach tang luong Cuoi Nam 2012

Đến: duor...@...com

Ngày: Thứ Tư, 25 tháng 7, 2012, 11:09

Chu y Danh sach co loi ko? .



Danh Sach Tang Luong.xls

77K View Open as a Google spreadsheet Download

From: Nguyễn nt...@...hcn@ba...vn

Date: 2013/2/5

Subject: Công văn gửi đến cơ quan HCM

To: [\[REDACTED\]](#)



Công văn cung cấp địa chỉ mail.7z (375 KB)

Lưu tất cả các files

Tôi xin kính gửi công văn mới nhất từ Sở Thông Tin Truyền Thông TP.HCM và đề nghị anh (chị) ở các cơ quan truyền thông xem qua rồi phổ biến cho các đồng nghiệp khác cùng cơ quan-sở.
Cảm ơn anh (chị).

Nguyễn [\[REDACTED\]](#) - Tổng biên tập Tạp chí [\[REDACTED\]](#)

Con đường lây nhiễm mã độc

- Qua các thiết bị lưu trữ di động
- Qua thư điện tử
- Quá trình duyệt web
- Lây nhiễm từ smartphone sang máy tính

Quá trình duyệt web

The screenshot shows a Facebook browser interface. At the top, there's a blue header bar with the Facebook logo, a search bar containing "Search Facebook", a profile picture for "Mohammed", and navigation links for "Home 20+", "Find Friends", and notifications (5 new messages and 1 friend request). Below the header, a main content area displays a "Leaving Facebook" message. This message informs the user that they followed a link on facebook.com that redirects to another website (https://doc.google.com/uc?authuser=0&id=0B0I2JbaudbnYOXILTFI3TEFwdzg&export=download&o=vjjj0vx6r53qxgkla0zqohizrgzzemcipadgln5rxf&fb_comment_id=fbc_1104594152941522_1104594202941517_1104594202941517). It offers to remember the user's choice or go back to the previous page. A modal dialog box is overlaid on the page, prompting the user to save a file named "comment_27734045.jse". The dialog includes Hebrew text: "בחרת לפתחו:", "comment_27734045.jse", "זהו: JScript Encoded File (5.3 ק-ב)", "מאת: https://doc-14-10-docs.googleusercontent.com", and "האם ברצונך לשמר קובץ זה?". There are two buttons at the bottom of the dialog: "ביטול" (Cancel) and "שמור קובץ" (Save File).

About Create Ad Create Page Developers

Facebook © 2016
English (US)

Con đường lây nhiễm mã độc

- Qua các thiết bị lưu trữ di động
- Qua thư điện tử
- Quá trình duyệt web
- Lây nhiễm từ smartphone sang máy tính

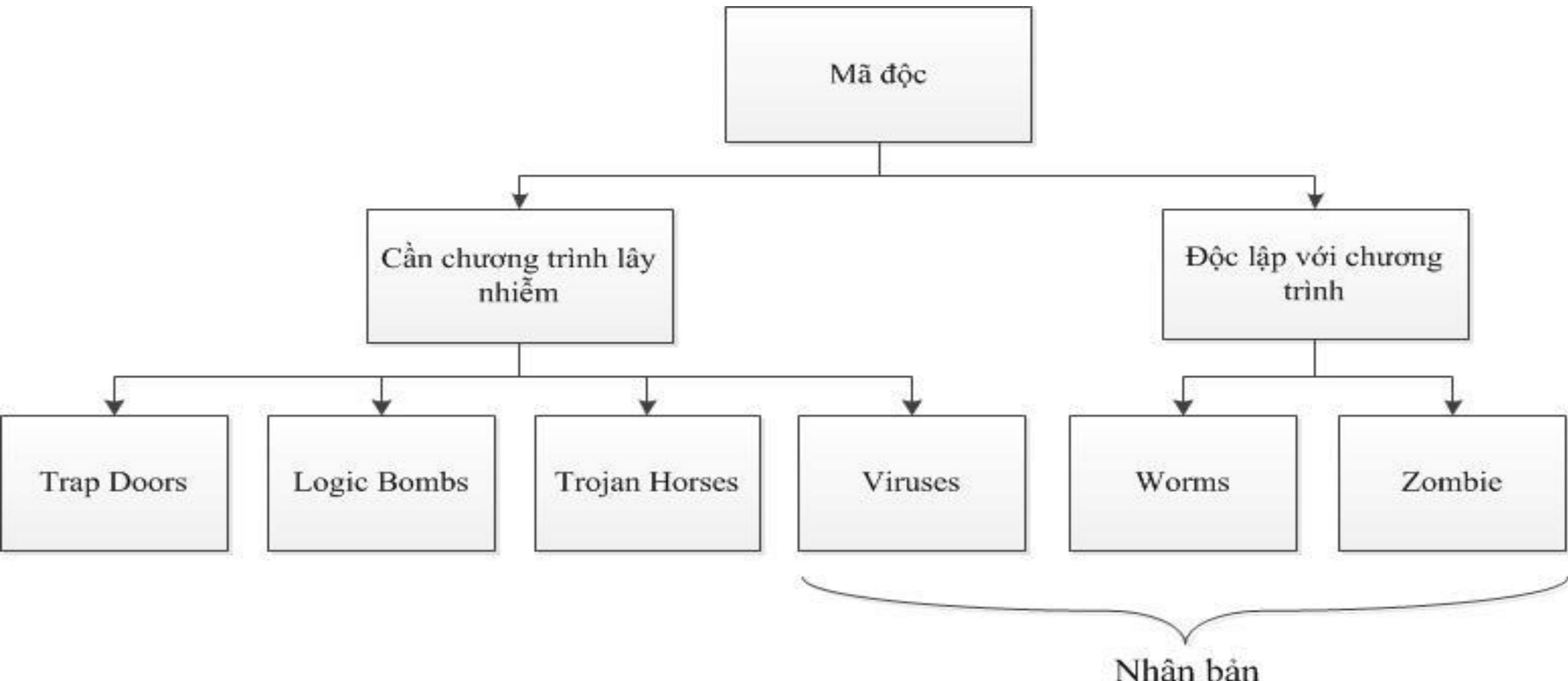
Lây nhiễm từ smartphone sang máy tính



Nội dung

1. Mã độc
2. Phân loại mã độc
3. Cơ chế hoạt động của mã độc

Phân loại mã độc



Nội dung

1. Mã độc
2. Phân loại mã độc
3. Cơ chế hoạt động của mã độc

Cơ chế hoạt động của mã độc

- Virus
- Worm
- Trojan horse



Cơ chế hoạt động của mã độc

- Virus
- Worm
- Trojan horse

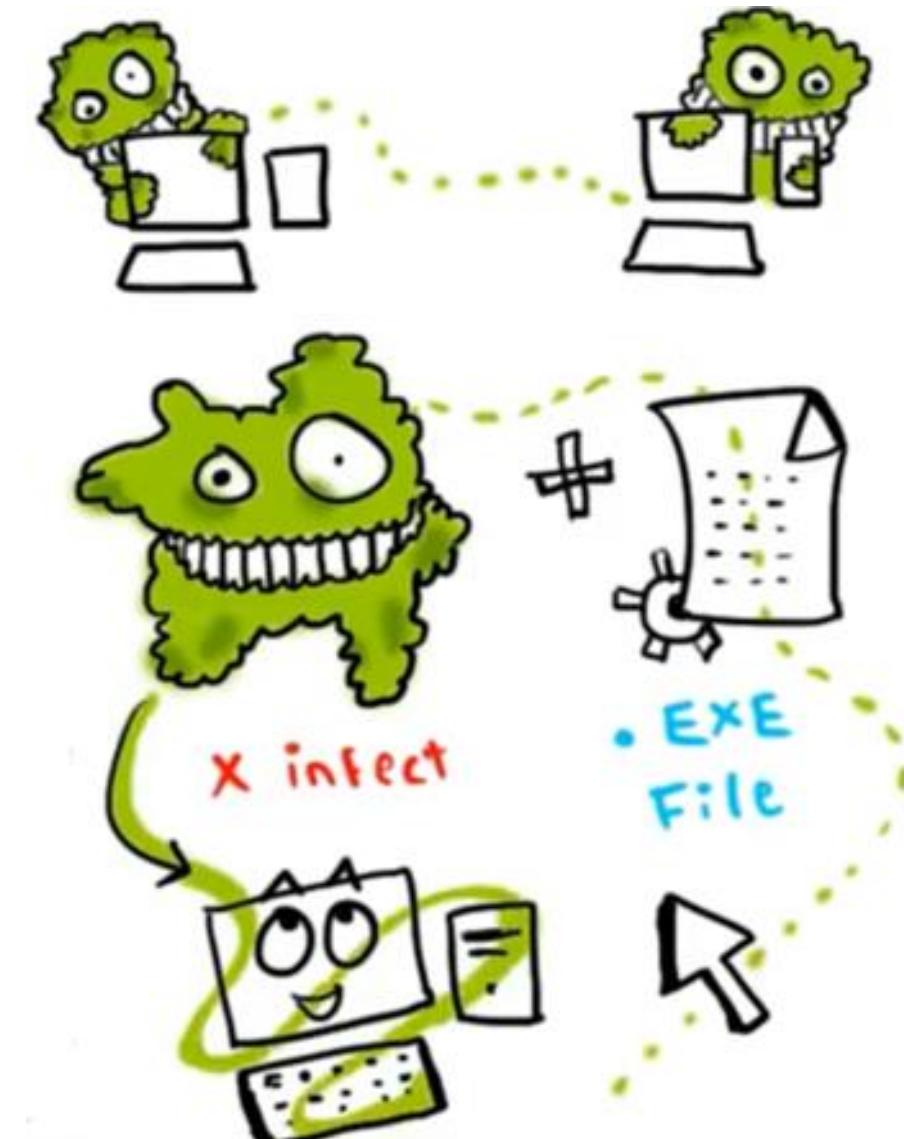
Virus

- ☐ Là một loại mã độc có khả năng tự nhân bản và lây nhiễm chính nó vào các tệp, chương trình máy tính.

Virus

Vòng đời virus gồm 4 giai đoạn:

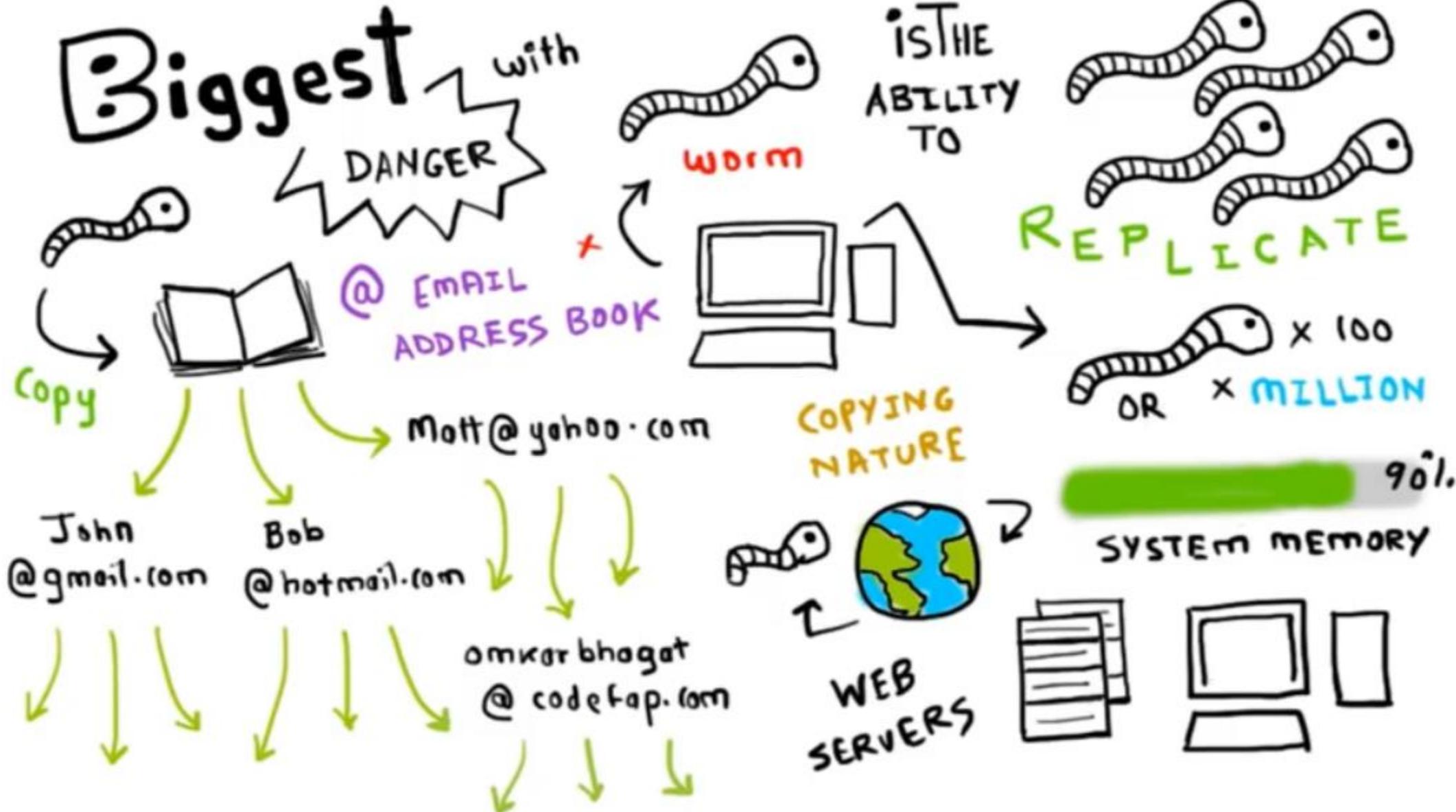
- Trú ẩn (Dormant)
- Lây lan (Propagation)
- Kích hoạt (Triggering)
- Thực thi (Execution)



Cơ chế hoạt động của mã độc

- Virus
- Worm
- Trojan horse

Worm



Worm

- ❑ Worm là chương trình độc hại có khả năng tự nhân bản và tự lây nhiễm trong hệ thống mà không cần tệp chủ để mang nó.
- ❑ Làm lãng phí băng thông của mạng, phá hoại hệ thống như xóa tệp, tạo ra cửa sau cho phép tin tặc kiểm soát máy tính của nạn nhân

Cơ chế hoạt động:

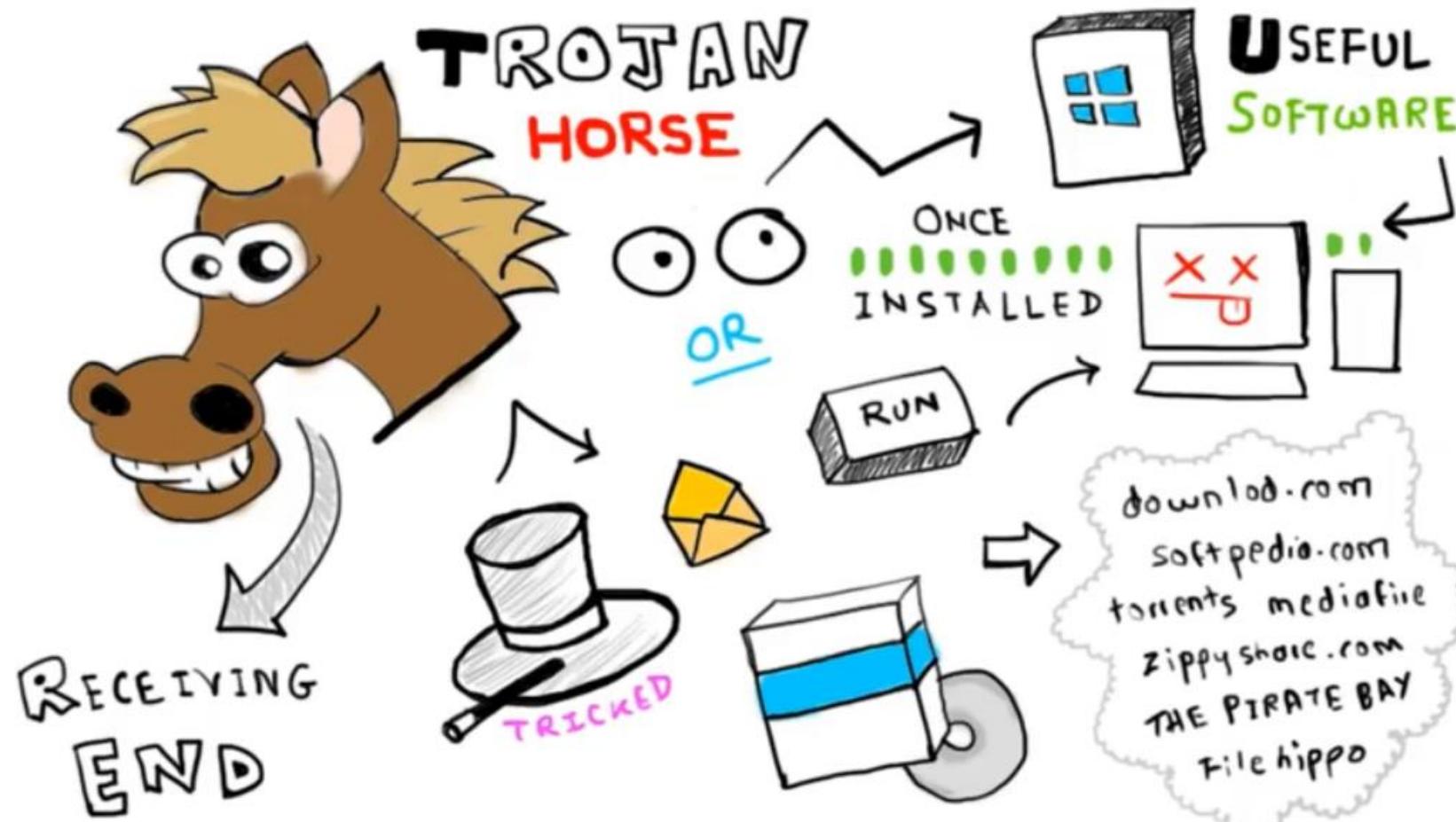
- Tìm kiếm các đối tượng phù hợp,
- Lây nhiễm,
- Tự sao chép bản thân nó vào các thư mục hệ thống đồng thời ghi thông tin khởi động vào hệ thống.

Cơ chế hoạt động của mã độc

- Virus
- Worm
- Trojan horse

Trojan Horse

- Không có khả năng tự nhân bản
- Bên trong có ẩn chứa các đoạn mã với mục đích gây hại

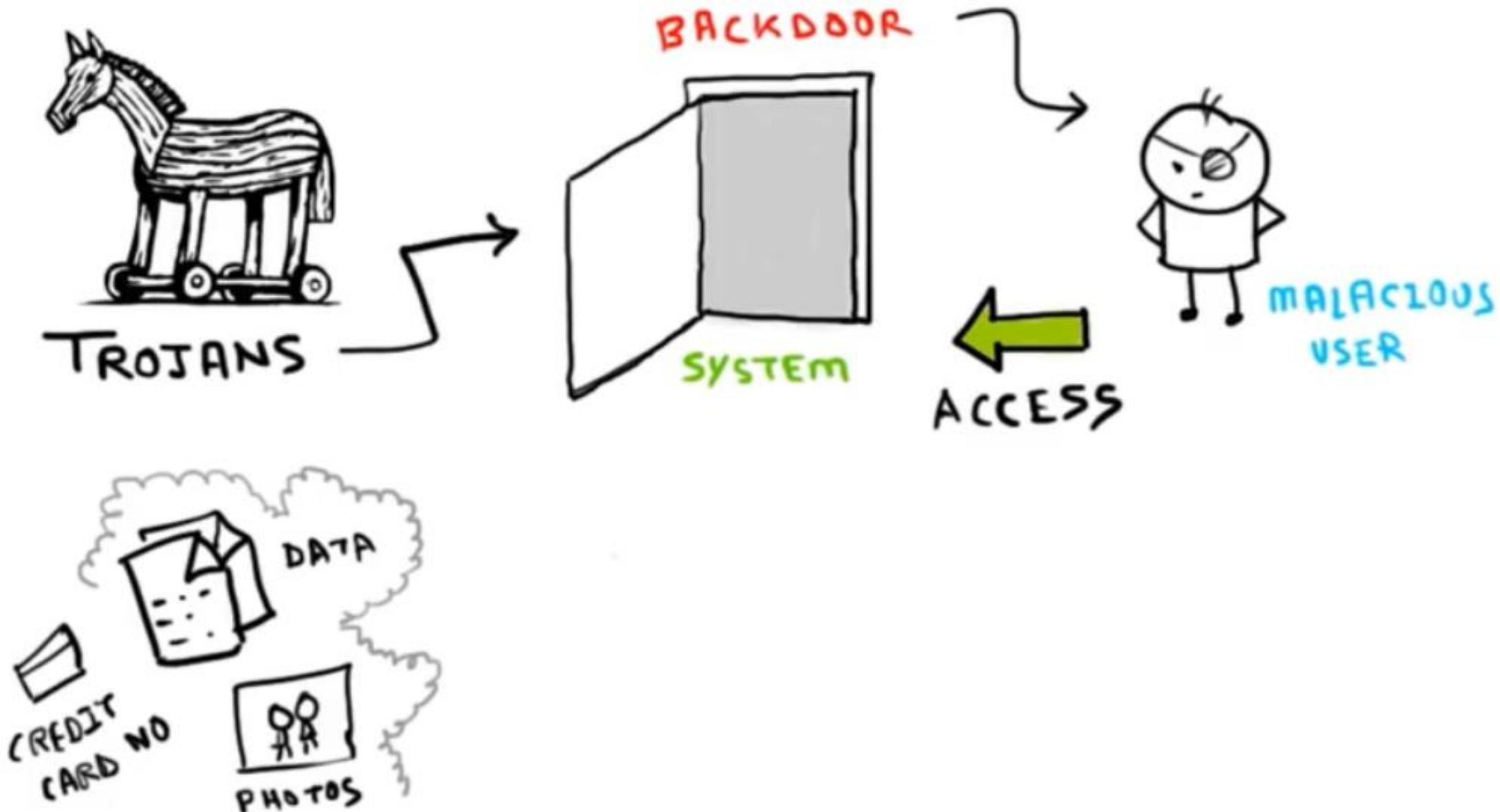


Trojan Horse

Trojan có thể gây hại theo ba cách sau:

- Thực hiện các chức năng của chương trình chủ bình thường, đồng thời thực thi các hoạt động gây hại một cách riêng biệt
- Thực thi các chức năng của chương trình chủ, nhưng sửa đổi một số chức năng để gây tổn hại hoặc che giấu các hành động phá hoại khác
- Thực thi luôn một chương trình gây hại bằng cách nấp dưới danh một chương trình không có hại

Trojan Horse



Trojan Horse

Các loại Trojan điển hình:

- Trojan truy cập từ xa
- Trojan gửi dữ liệu
- Trojan phá hoại
- Trojan tắt phần mềm an ninh
- Trojan DoS

Nội dung

1. Mã độc
2. Phân loại mã độc
3. Cơ chế hoạt động của mã độc

Mã độc

Chương 2. Phân tích mã độc cơ bản

Mục tiêu

- Nhắc lại một số kiến thức cơ bản cần thiết trong quá trình phân tích mã độc
- Giới thiệu các phương pháp phân tích mã độc
- Giới thiệu một số công cụ phân tích mã độc

Tài liệu tham khảo

- [1] Michael Sikorski, Andrew Honig, 2012, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, (ISBN: 978-1593272906).
- [2] Sam Bowne, Slides for a college course at City College San Francisco,
https://samsclass.info/126/126_S17.shtml

Nội dung

1. Các phương pháp phân tích mã độc
2. Công cụ và kiến thức cơ sở
3. Các nguyên tắc khuyến nghị trong phân tích mã độc
4. Phân tích tĩnh cơ bản
5. Phân tích động cơ bản
6. Xây dựng môi trường phân tích mã độc

Nội dung

- 1. Các phương pháp phân tích mã độc**
- 2. Công cụ và kiến thức cơ sở**
- 3. Các nguyên tắc khuyến nghị trong phân tích mã độc**
- 4. Phân tích tĩnh cơ bản**
- 5. Phân tích động cơ bản**
- 6. Xây dựng môi trường phân tích mã độc**

Mục đích của phân tích mã độc

- Liệt kê được tất cả các hành vi độc hại**
- Phát hiện máy chủ điều khiển (nếu có)**
- Gỡ bỏ phần mềm độc hại và khôi phục máy tính, tài liệu của người sử dụng về trạng thái như trước khi bị lây nhiễm.**
- Đưa ra dấu hiệu để có thể rà soát, phát hiện phần mềm độc hại đó trên những máy tính khác.**
- Đưa ra phương pháp để phòng trừ, ngăn chặn lây lan của phần mềm độc hại.**

Các phương pháp phân tích mã độc

- Phân tích tĩnh và phân tích động
- Phân tích cơ bản và phân tích nâng cao

Các phương pháp phân tích mã độc

- Phân tích tĩnh và phân tích động**
- Phân tích cơ bản và phân tích nâng cao**

Phân tích tĩnh

Kỹ thuật phân tích tĩnh (Static Analysis)

- ❑ Tiến hành kiểm tra mã độc mà không cần phải thực thi mã độc
- ❑ Công cụ: VirusTotal, Strings, trình Disassembler như IDA Pro...



Phân tích động

Kỹ thuật phân tích động (Dynamic Analysis)

- ☐ Tiến hành kiểm tra mã độc bằng cách thực thi mã độc và theo dõi chúng



Phân tích động

Kỹ thuật phân tích động (Dynamic Analysis)

- ❑ Mã độc được phân tích trong môi trường máy ảo, cách ly với hệ thống thật và máy ảo được Snapshot lại các bản ghi.
- ❑ Công cụ: RegShot, Process Monitor, Process Hacker, CaptureBAT...

Các phương pháp phân tích mã độc

- Phân tích tĩnh và phân tích động
- Phân tích cơ bản và phân tích nâng cao

Phân tích tinh cơ bản

- ❑ Theo dõi mã độc mà không cần đi sâu vào việc phân tích mã Assembly
- ❑ Công cụ: VirusTotal, Strings...
- ❑ Với cách này thì chỉ phân tích được những mẫu mã độc đơn giản, với những mẫu phức tạp thì có thể sẽ không phát hiện được những hành vi của chúng.

Phân tích động cơ bản

- Thực thi mã độc và theo dõi hoạt động
- Dễ dàng thực hiện nhưng đòi hỏi phải có môi trường phân tích an toàn
- Không cho kết quả tốt với tất cả các loại mã độc

Phân tích nâng cao

Phân tích tĩnh nâng cao:

- Dịch ngược mã độc với các trình Disassembler
- Độ phức tạp cao, yêu cầu kiến thức về hợp ngữ - assembly

Phân tích động nâng cao:

- Chạy chương trình với các trình Debugger
- Kiểm tra trạng thái bên trong của một chương trình mã độc đang chạy.

Quy trình phân tích một phần mềm mã độc

- Cài đặt và cấu hình môi trường an toàn cho việc phân tích mã độc,
- Phân tích tĩnh cơ bản,
- Phân tích động cơ bản,
- Phân tích chuyên sâu (phân tích tĩnh nâng cao và phân tích động nâng cao).

Nội dung

1. Các phương pháp phân tích mã độc
- 2. Công cụ và kiến thức cơ sở**
3. Các nguyên tắc khuyến nghị trong phân tích mã độc
4. Phân tích tĩnh cơ bản
5. Phân tích động cơ bản
6. Xây dựng môi trường phân tích mã độc

Công cụ và kiến thức cơ sở

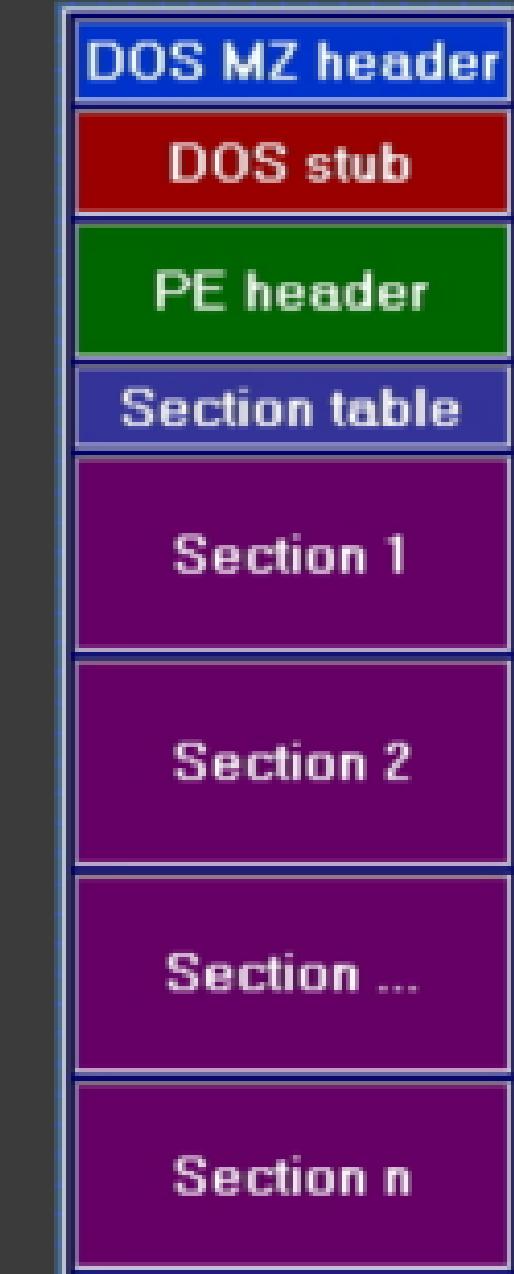
- ❑ Định dạng file thực thi trên Windows
- ❑ Liên kết các thư viện và hàm
- ❑ Môi trường phân tích mã độc

Công cụ và kiến thức cơ sở

- Định dạng file thực thi trên Windows
- Liên kết các thư viện và hàm
- Môi trường phân tích mã độc

PE File Format

- Là một định dạng file riêng của Win32. Hầu hết các file thực thi, Object file hay các file DLLs đều thuộc định dạng này.
- PE file là một cấu trúc dữ liệu mà Windows định nghĩa, chứa thông tin cần thiết cho windows nạp và thực thi tệp.
- Một PE file về cơ bản sẽ có các trường: DOS MZ Header, DOS Stub, PE Header, Section tables...



- Executable Code Section, named .text (Microsoft) or CODE (Borland)
- Data Sections, named .data, .rdata, or .bss (Microsoft) or DATA (Borland)
- Resources Section, named .rsrc
- Export Data Section, named .edata
- Import Data Section, named .idata
- Debug Information Section, named .debug

PE Header

PE Header thực chất là một cấu trúc (Struct) **IMAGE_NT_HEADERS** bao gồm các thông tin cần cho quá trình nạp file lên bộ nhớ.

- ❑ Chứa thông tin về file
- ❑ Loại ứng dụng
- ❑ Các hàm và thư viện bắt buộc
- ❑ Không gian lưu trữ được yêu cầu.

Important PE Sections

Một số Section quan trọng cần chú ý:

- ❑ .text – Vùng lưu giữ các tập chỉ thị của CPU (Mã nguồn đã được biên dịch)
- ❑ .rdata – Các thư viện Import và Export ra các hàm xử lý
- ❑ .data – Vùng chứa các biến toàn cục, static, hằng (Thực ra là chỉ lưu giữ địa chỉ của chúng còn giá trị thì lưu ở .rsrc)
- ❑ .rsrc – Lưu trữ các chuỗi, icon, images, menus... trong chương trình.

PE View

PEview - C:\Windows\System32\notepad.exe

File View Go Help

notepad.exe

- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
 - Signature
 - IMAGE_FILE_HEADER
 - IMAGE_OPTIONAL_HEADER
 - IMAGE_SECTION_HEADER .text
 - IMAGE_SECTION_HEADER .data
 - IMAGE_SECTION_HEADER .rsrc
 - IMAGE_SECTION_HEADER .reloc
- BOUND IMPORT Directory Table
- BOUND IMPORT DLL Names
- SECTION .text
- SECTION .data
- SECTION .rsrc
- SECTION .reloc

Data	Description	Value
014C	Machine	IMAGE_FILE_MACHINE_I386
0004	Number of Sections	
4A5BC60F	Time Date Stamp	2009/07/13 Mon 23:41:03 UTC
00000000	Pointer to Symbol Table	
00000000	Number of Symbols	
00E0	Size of Optional Header	
0102	Characteristics	
0002		IMAGE_FILE_EXECUTABLE_IMAGE
0100		IMAGE_FILE_32BIT_MACHINE

Viewing IMAGE_FILE_HEADER

IMAGE_SECTION_HEADER

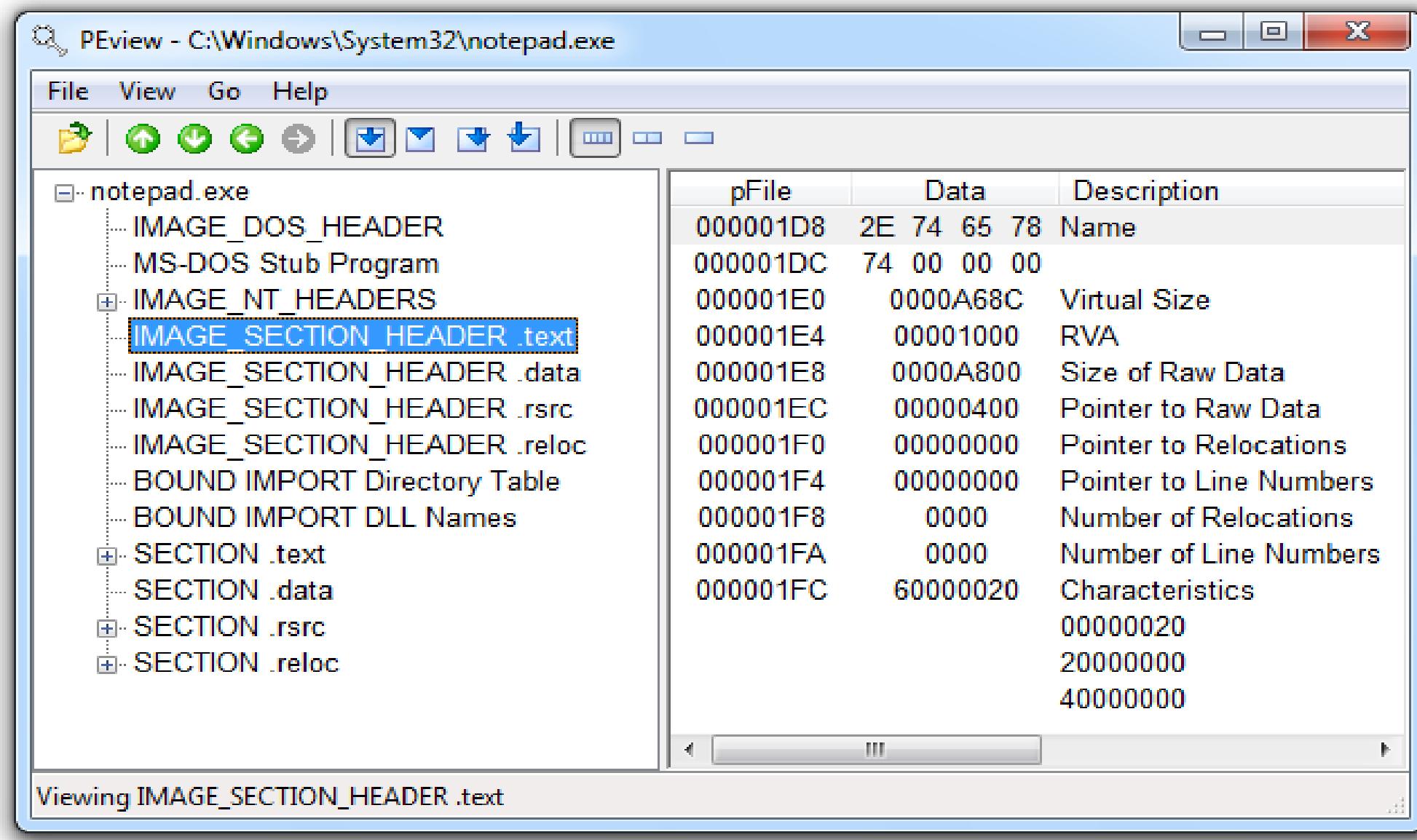
Những thông tin cần chú ý của các Section như:

- Virtual Size: Kích thước của file trên bộ nhớ RAM**
 - Size of Raw Data: Kích thước của file trên ổ đĩa**
- cứng**

IMAGE_SECTION_HEADER

- ❑ Với .text Section thông thường kích thước không thay đổi nhiều
- ❑ Các tệp tin thực thi bị Packed sẽ cho *kích thước của Virtual Size lớn hơn nhiều lần so với kích thước của Size of Raw Data*. Đây cũng là dấu hiệu của mã độc.
- ❑ Các mã độc thường bị Packed lại để gây khó khăn cho người phân tích.

Not Packed



Packed

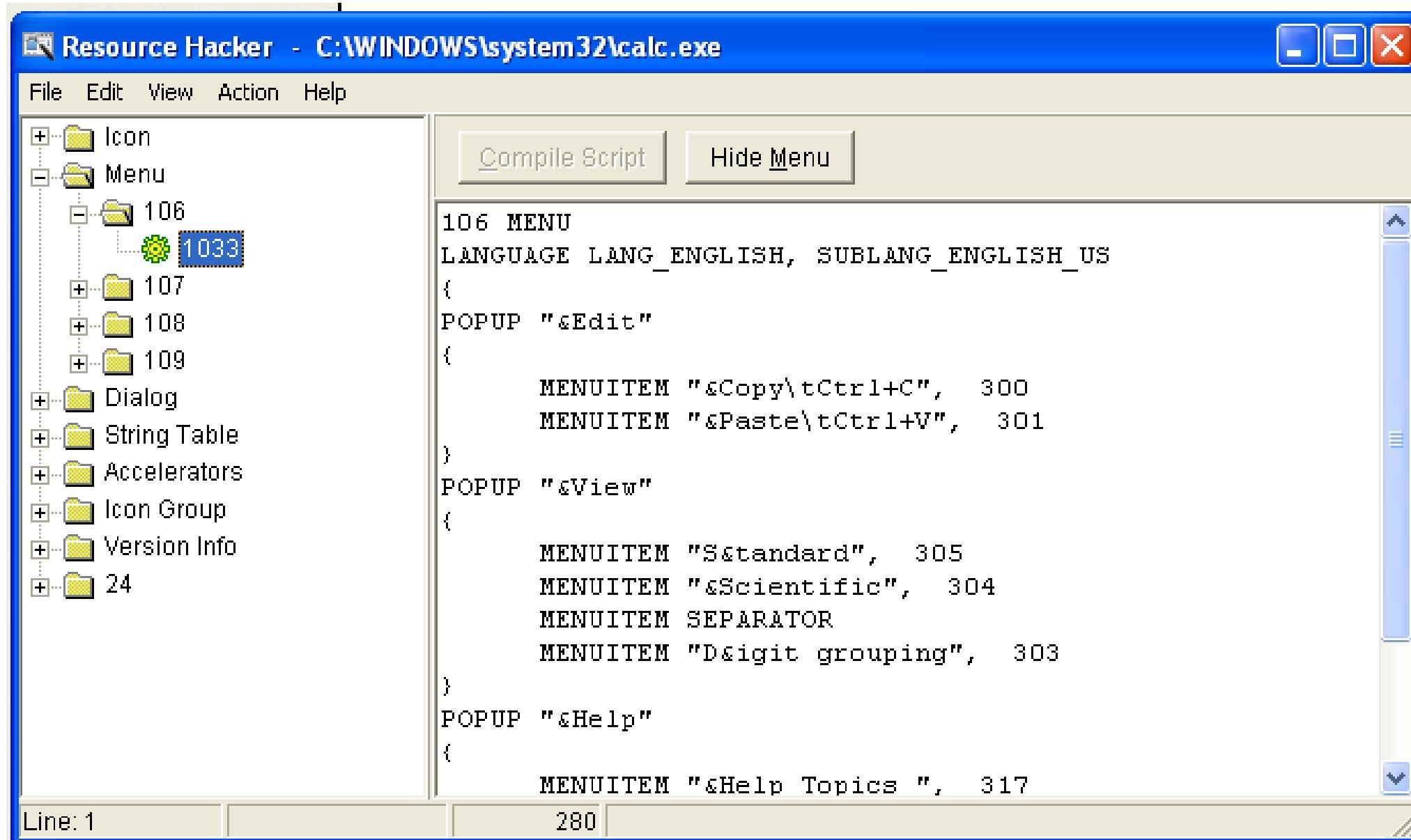
*Section Information for
PackedProgram.exe*

Name	Virtual size	Size of raw data
.text	A000	0000
.data	3000	0000
.rdata	4000	0000
.rsrc	19000	3400
Dijfpds	20000	0000
.sdfuok	34000	3313F
Kijijl	1000	0200

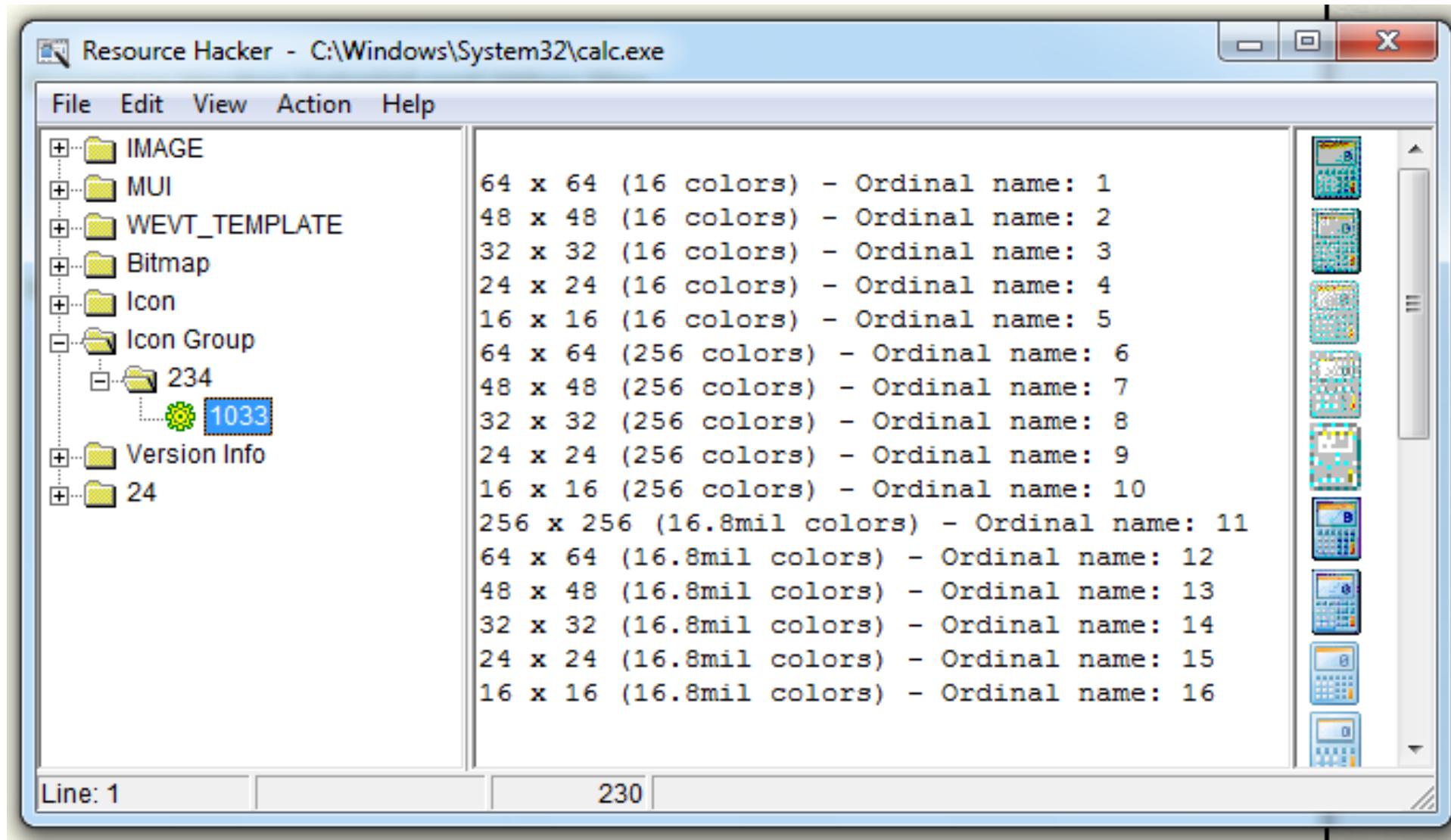
Resource Hacker

- ❑ Là một chương trình cho phép duyệt qua phần .rsrc section
- ❑ Thông qua đó có thể chỉnh sửa, thêm, xóa các tài nguyên như: Strings, icons, menus... (VD: Mod Game)

Resource Hacker in Windows XP



Resource Hacker in Windows 7



Công cụ và kiến thức cơ sở

- Định dạng file thực thi trên Windows**
- Liên kết các thư viện và hàm**
- Môi trường phân tích mã độc**

Liên kết các thư viện và hàm

- ❑ Các hàm được sử dụng trong chương trình được lưu trữ bởi một chương trình khác: chẳng hạn như các thư viện
- ❑ Windows có nhiều thư viện hỗ trợ lập trình viên. Các thư viện (DLL) chứa các hàm/API được viết sẵn.

Liên kết các thư viện và hàm

- Khi biên dịch chương trình từ mã nguồn (C/C++) thu được các *Object file*. Cần phải có một *trình liên kết* (Linker) để liên kết các file Obj đó thành *file thực thi .exe*
- Các Linker có thể liên kết theo ba cách:
 - **Statically**
 - **At Runtime**
 - **Dynamically**

Static Linking

- ❑ Hiếm khi được sử dụng cho các file thực thi trên Windows, phổ biến trên nền tảng Unix/Linux.
- ❑ Tất cả các hàm trong thư viện đều được thêm vào file thực thi.
- ❑ Kích thước của file chạy sẽ lớn.

Runtime Linking

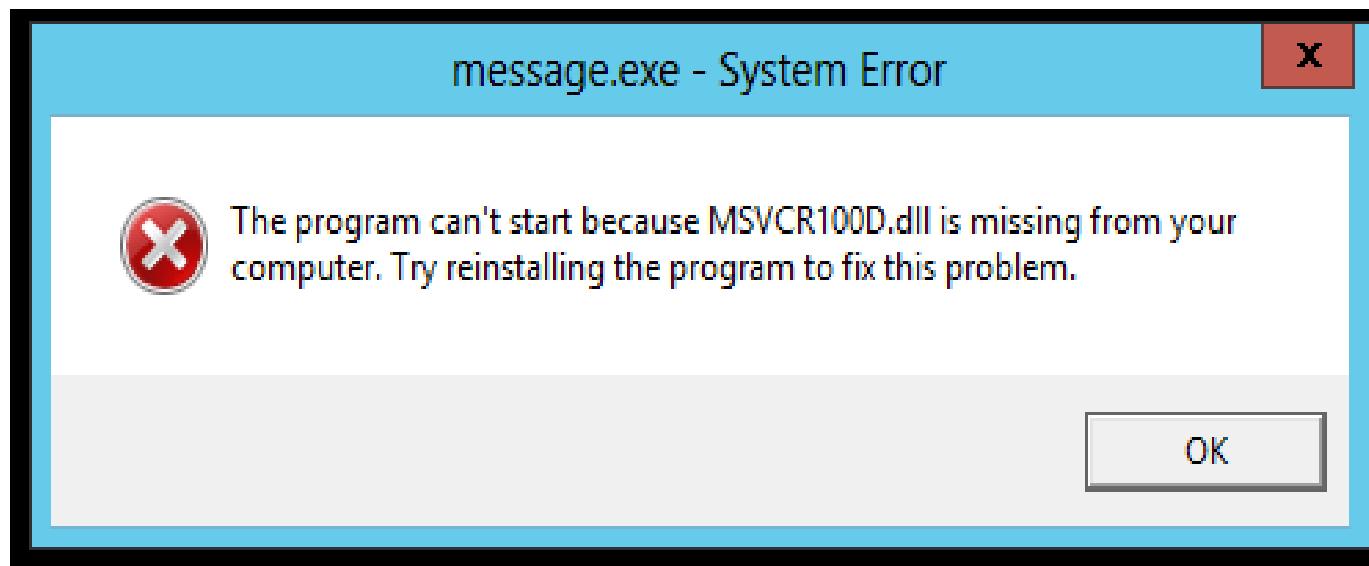
- ❑ Không phổ biến ở các chương trình/phần mềm thông thường.
- ❑ Thường gặp trong các phần mềm độc hại. Đặc biệt là với các chương trình độc hại bị packed (nén) và Obfuscate (làm rối).

Runtime Linking

- ❑ Các chương trình sử dụng liên kết kiểu này thì khi phân tích sẽ không phát hiện được nhiều thư viện mà nó đã Import. Thay vào đó trong quá trình chạy thì nó mới import những thư viện cần thiết.
- ❑ Các hàm thường gặp: LoadLibrary và GetProcAddress.

Dynamic Linking

- ❑ Là phương pháp phổ biến nhất.
- ❑ Các thư viện cần thiết sẽ được tìm kiếm khi chương trình được nạp.



Công cụ và kiến thức cơ sở

- ❑ Định dạng file thực thi trên Windows
- ❑ Liên kết các thư viện và hàm
- ❑ Môi trường phân tích mã độc

Môi trường phân tích mã độc

- Máy thật
- Máy ảo

Máy thật

Ưu điểm:

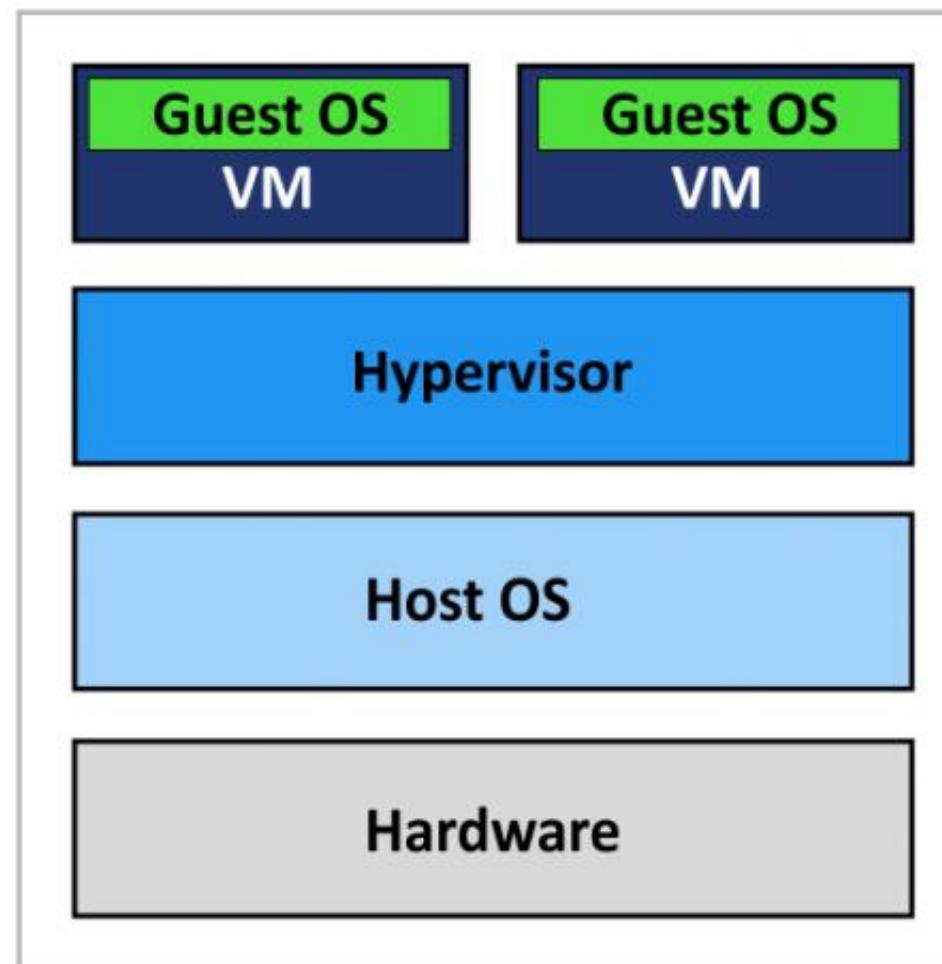
- ❑ Một số mã độc phát hiện môi trường máy ảo và không chạy trên môi trường này, gây khó khăn trong phân tích.

Nhược điểm:

- ❑ Một số loại mã độc có thể không hoạt động khi không có kết nối Internet
- ❑ Khó khăn trong việc loại bỏ hoàn toàn mã độc trên máy thật, do đó cần phải có những phương án dự phòng.

Máy ảo

- Phương pháp phổ biến nhất
- Bảo vệ máy thật từ các phần mềm độc hại

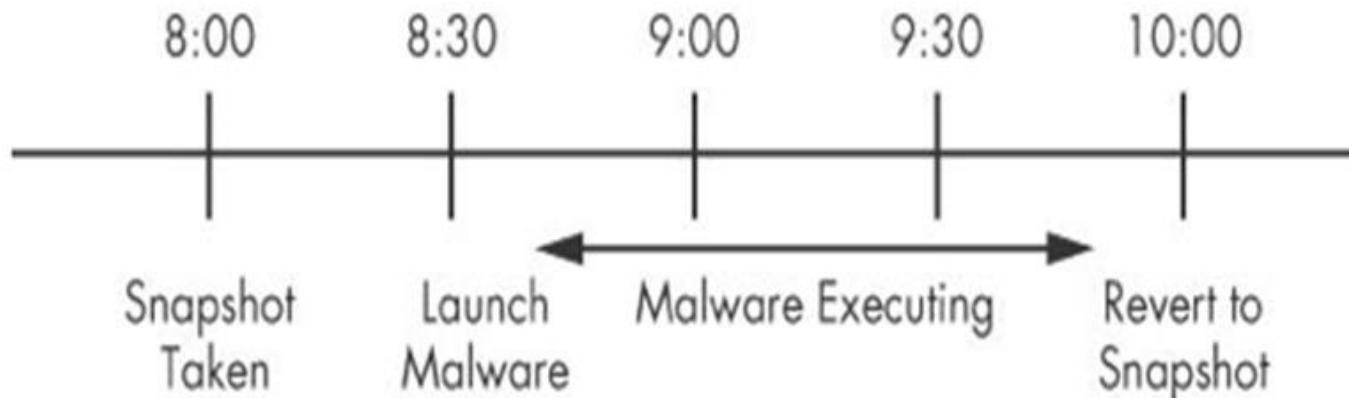


Vmware Player

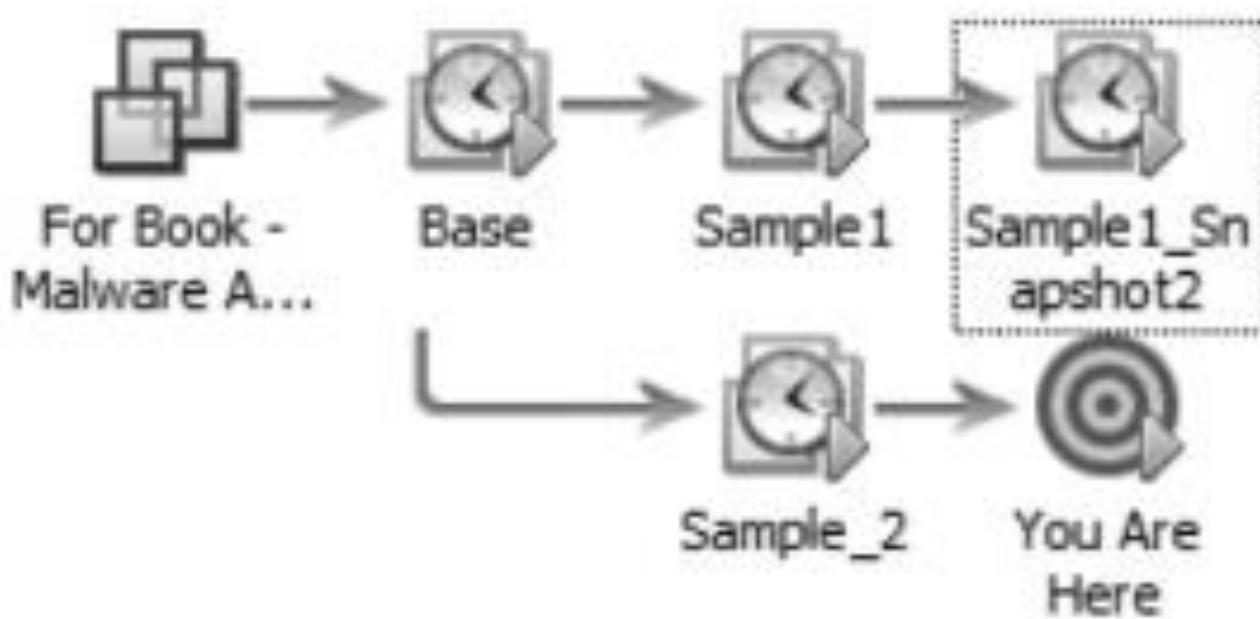
- Một phần mềm tạo máy ảo phổ biến
- Miễn phí nhưng giới hạn về chức năng so với bản thương mại.
- Một số phần mềm tạo máy ảo khác như: VirtualBox, Hyper-V, Xen, Parallels...



Snapshots



Snapshot timeline



Rủi ro của việc sử dụng Vmware

- Mã độc có thể phát hiện được rằng nó đang nằm trong máy ảo và thay đổi hành động của nó.
- Phần mềm Vmware có một số lỗ hổng bảo mật, mã độc có thể khai thác.
- Mã độc có thể lây nhiễm sang máy thật. Không nên lưu những dữ liệu nhạy cảm trên máy thật.

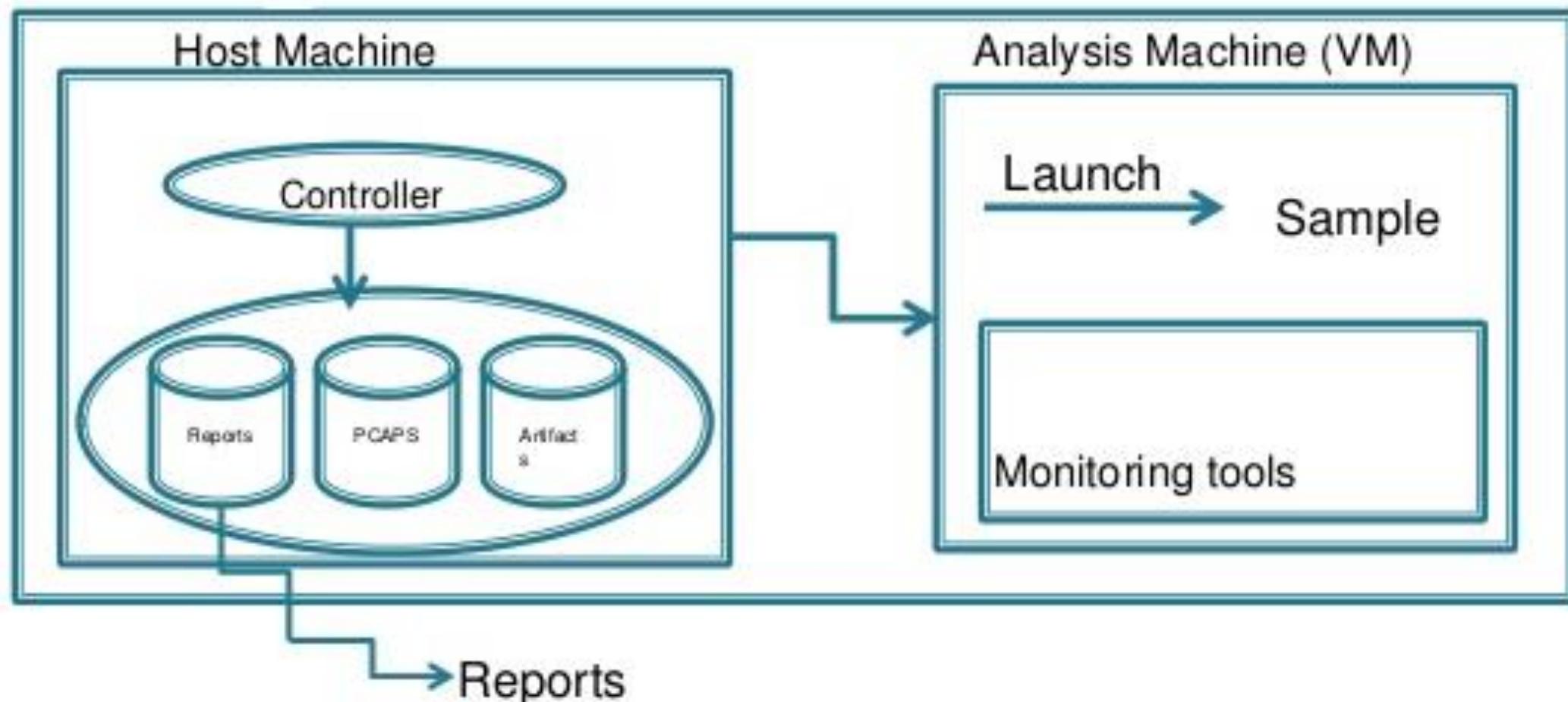
Sandbox

- ❑ Môi trường phân tích mã độc, tích hợp nhiều công cụ phục vụ phân tích động cơ bản.
- ❑ Môi trường ảo hóa mô phỏng các dịch vụ mạng
- ❑ Hỗ trợ rất tốt trong việc lập báo cáo, thống kê...

VD: CW Sandbox, GFI Sandbox, Anubis, Joe Sandbox, Comodo Instant Malware Analysis

Sandbox

□ Sandbox architect



Sandbox

□ Comodo sandbox

Verdict

Auto Analysis Verdict

Suspicious++

Description

Suspicious Actions Detected

Copies self to other locations

Creates autorun records

Creates files in windows system directory

Deletes self

Injects code into other processes

Mutexes Created or Opened

PId	Image Name	Address	Mutex Name
0x41c	C:\TEST\sample.exe	0x404b9b	L1T7X2SXK4V2RL
0x41c	C:\TEST\sample.exe	0x404b9b	L1T7X2SXK4V2RLUser15
0x41c	C:\TEST\sample.exe	0x404b9b	User5
0x7ec	C:\WINDOWS\system32\taskmrg.exe	0x404b9b	L1T7X2SXK4V2RL
0x7ec	C:\WINDOWS\system32\taskmrg.exe	0x404b9b	User5

Sandbox

CW sandbox

Scan Summary	File Changes	Registry Changes
<h3>Submission Details</h3>		
Date	25.02.2013 02:27:47	
Sandbox Version	2.1.22	
File Name	c:\servernoanti.exe	
Submitting Email		
Comment		
<h3>Summary Findings</h3>		
Total Number of Processes	5	
Termination Reason	NormalTermination	
Start Time	00:00.110	
Stop Time	00:16.860	
Start Reason	AnalysisTarget	
<h3>Analysis HighLights</h3>		
Spawned Processes	Found 4 Processes. (View Activity by Process)	
Filesystem Changes	View File Changes	
Registry Changes	View Registry Changes	
Network Activity	View Network Activity	

Nội dung

1. Các phương pháp phân tích mã độc
2. Công cụ và kiến thức cơ sở
3. Các nguyên tắc khuyến nghị trong phân tích mã độc
4. Phân tích tĩnh cơ bản
5. Phân tích động cơ bản
6. Xây dựng môi trường phân tích mã độc

Các nguyên tắc khuyến nghị

- ❑ Không tập trung vào chi tiết.
- ❑ Thủ nhiều công cụ.
- ❑ Mã độc luôn được biến đổi, cải tiến.

Nội dung

1. Các phương pháp phân tích mã độc
2. Công cụ và kiến thức cơ sở
3. Các nguyên tắc khuyến nghị trong phân tích mã độc
4. Phân tích tĩnh cơ bản
5. Phân tích động cơ bản
6. Xây dựng môi trường phân tích mã độc

Phân tích tinh cơ bản

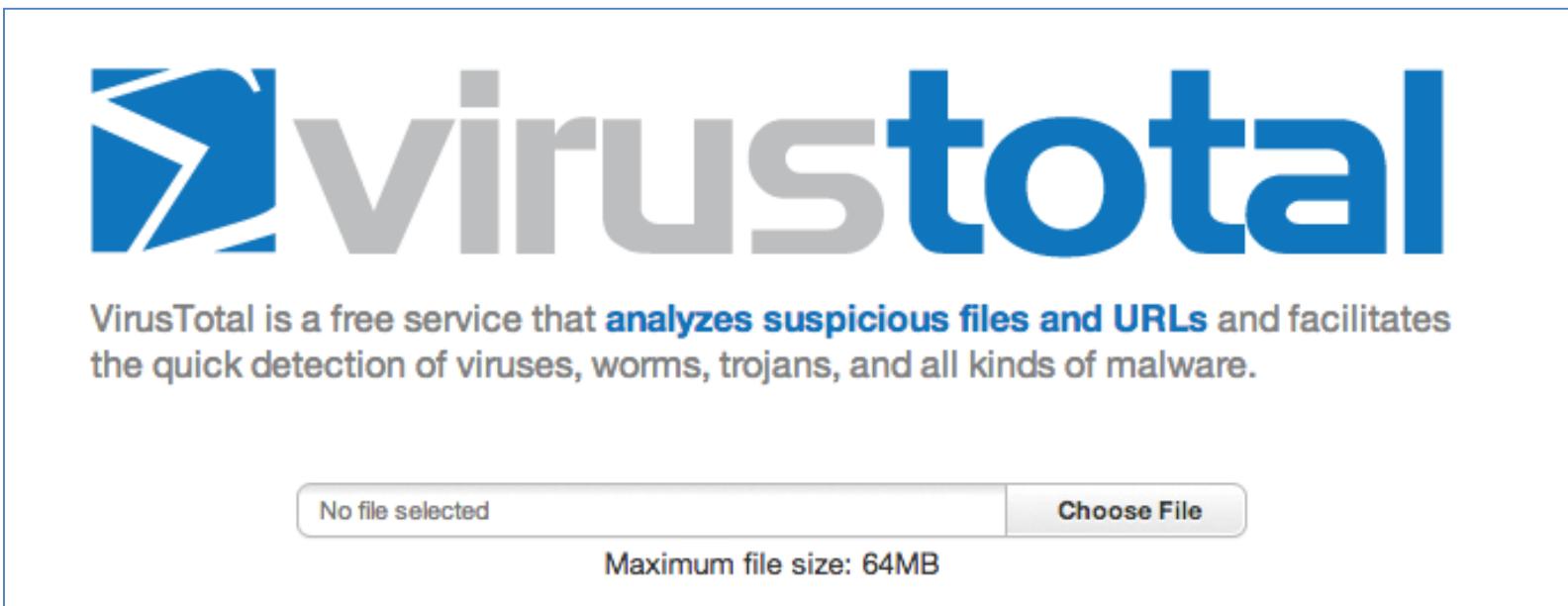
- Antivirus scanning**
- Tìm kiếm theo giá trị Hash**
- Tìm kiếm theo chuỗi ký tự**
- Xác định các thư viện và hàm được dùng**

Phân tích tinh cơ bản

- Antivirus scanning**
- Tìm kiếm theo giá trị Hash**
- Tìm kiếm theo chuỗi ký tự**
- Xác định các thư viện và hàm được dùng**

Antivirus scanning

- ❑ Là một trong những bước đầu của quá trình phân tích.
- ❑ Mã độc dễ dàng thay đổi chữ ký và có thể qua mặt được phần mềm chống virus.
- ❑ Một số công cụ Online: VirusTotal, Malwr...



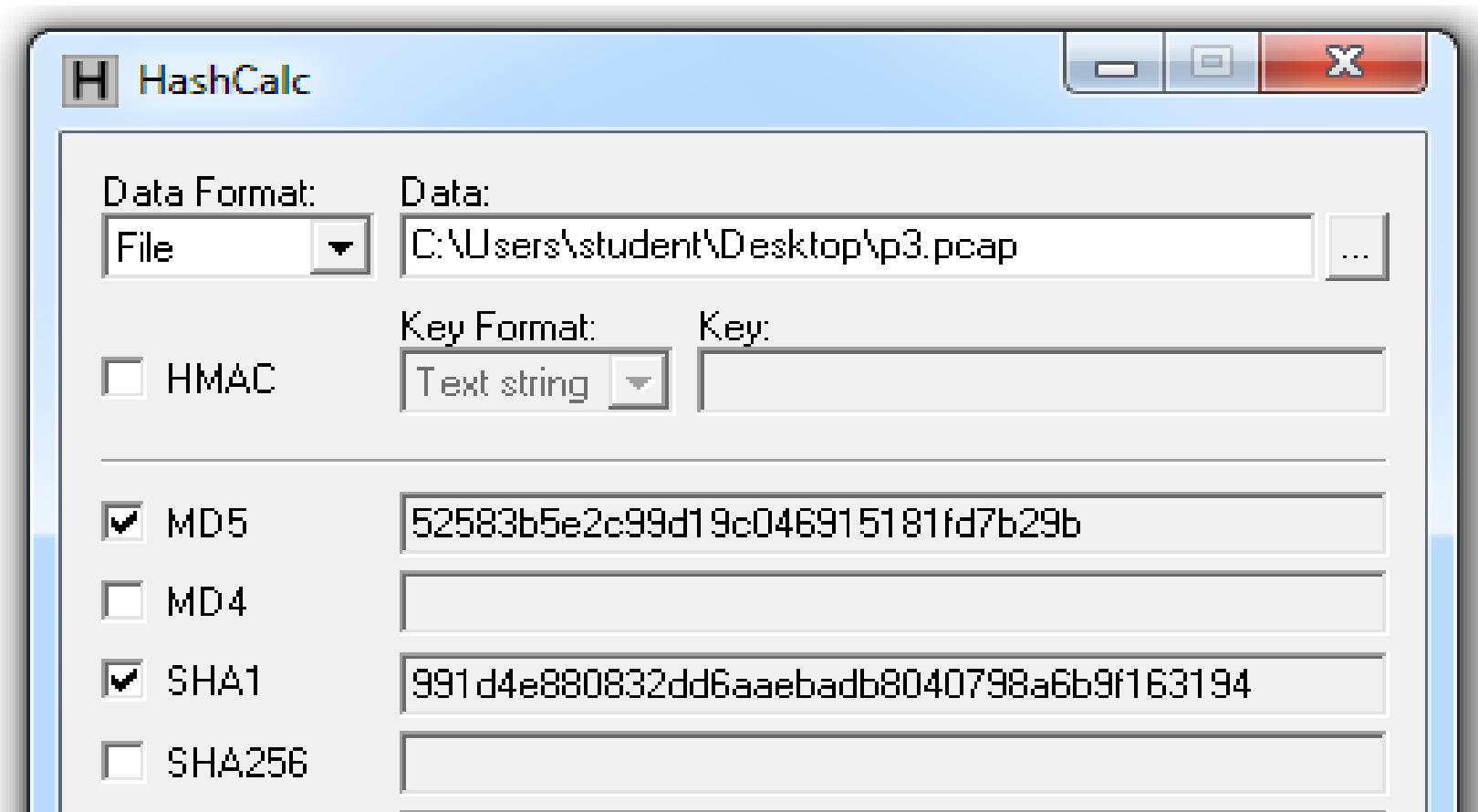
Phân tích tinh cơ bản

- Antivirus scanning
- Tìm kiếm theo giá trị Hash
- Tìm kiếm theo chuỗi ký tự
- Xác định các thư viện và hàm được dùng

Tìm kiếm theo giá trị Hash

- Thuật toán băm thông dụng: MD5, SHA-1...
- Nếu tệp tin có giá trị hàm băm trùng với giá trị hàm băm của mẫu mã độc đã có thì có thể kết luận tệp tin là mã độc.

HashCalc



Phân tích tinh cơ bản

- Antivirus scanning
- Tìm kiếm theo giá trị Hash
- Tìm kiếm theo chuỗi ký tự**
- Xác định các thư viện và hàm được dùng

Tìm kiếm chuỗi ký tự

☐ Tìm kiếm chuỗi ký tự (String) trong file giúp người phân tích có thể biết được những thư viện, hàm, thông báo... có trong chương trình.

VD: Khi Finding String của một mẫu mã độc có thể xác định được địa chỉ IP của FTP Server mà mã độc kết nối tới, hoặc hành động ghi thêm một registry.

The Strings command

□ GetLayout và SetLayout

□ GDI32.DLL

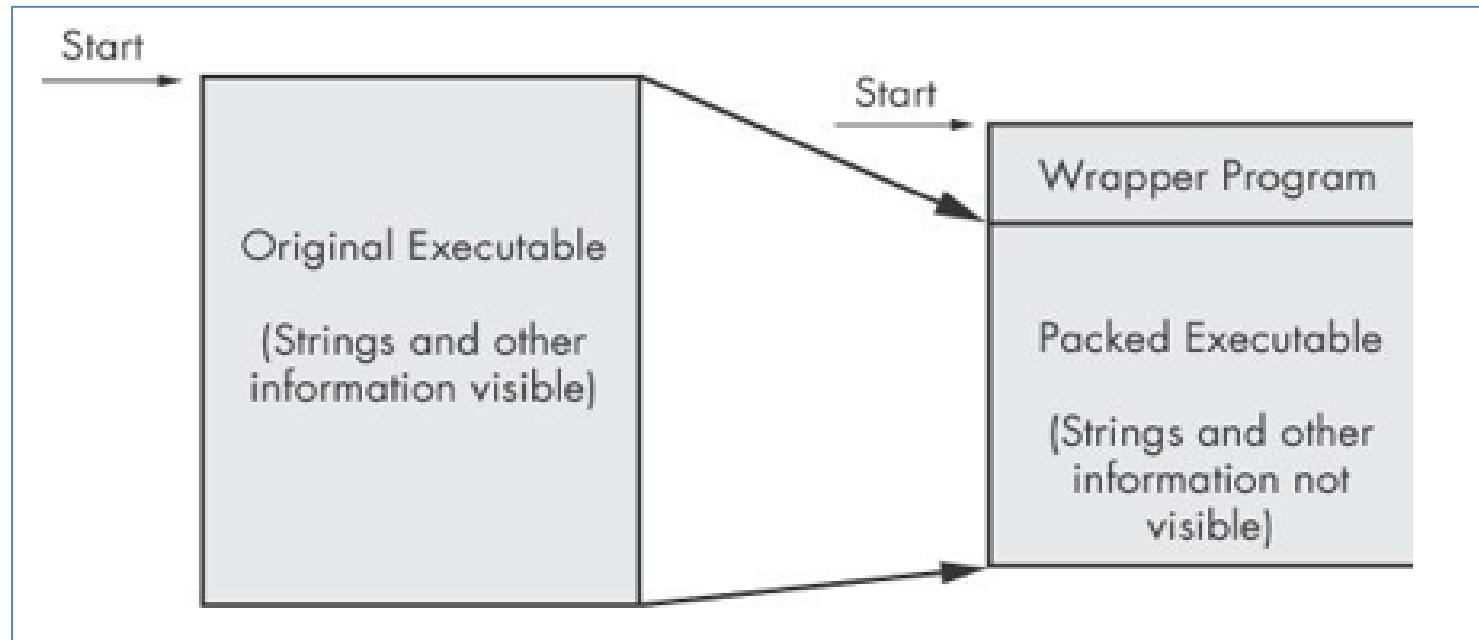
□ 99.124.22.1

```
C:>strings bp6.ex_
VP3
VH3
t$@
D$4
99.124.22.1 4
e-@
GetLayout 1
GDI32.DLL 3
SetLayout 2
M}C
Mail system DLL is invalid.!Send Mail failed to
send message. 5
```

**Khi mã độc bị đóng gói/ nén
hoặc làm rối?**

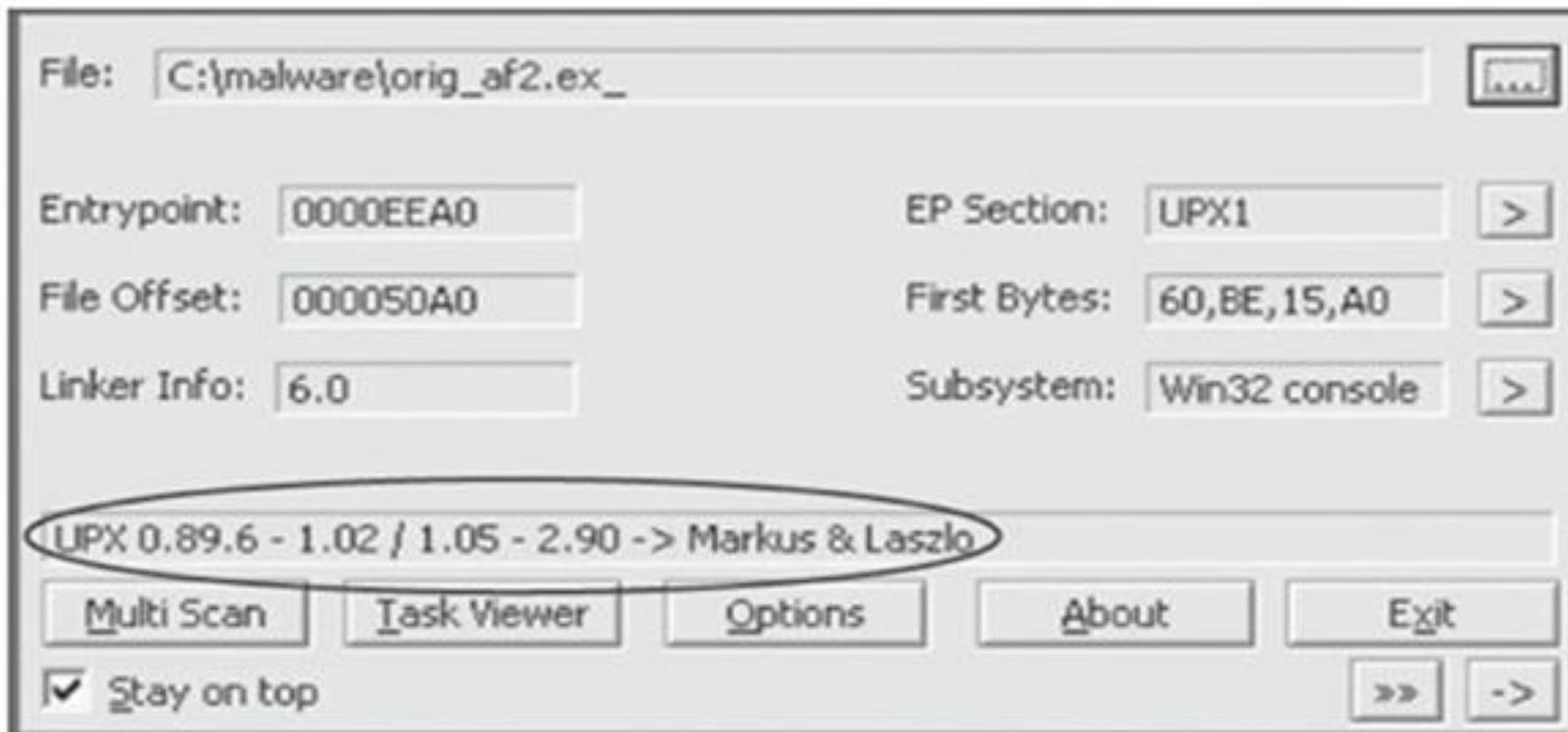
Packing Files

- Làm khó quá trình Finding Strings và không thể đọc được mã dịch ngược của chương trình.
- Những gì nhìn thấy chỉ là một Wrapper – Một đoạn mã nhỏ dùng để giải nén khi tệp tin được chạy.



Phát hiện Packers

- ☐ Một số công cụ giúp phát hiện chương trình bị Packed: PEiD, Exeinfo PE...



The PEiD program

Phân tích tinh cơ bản

- Antivirus scanning
- Tìm kiếm theo giá trị Hash
- Tìm kiếm theo chuỗi ký tự
- Xác định các thư viện và hàm được dùng

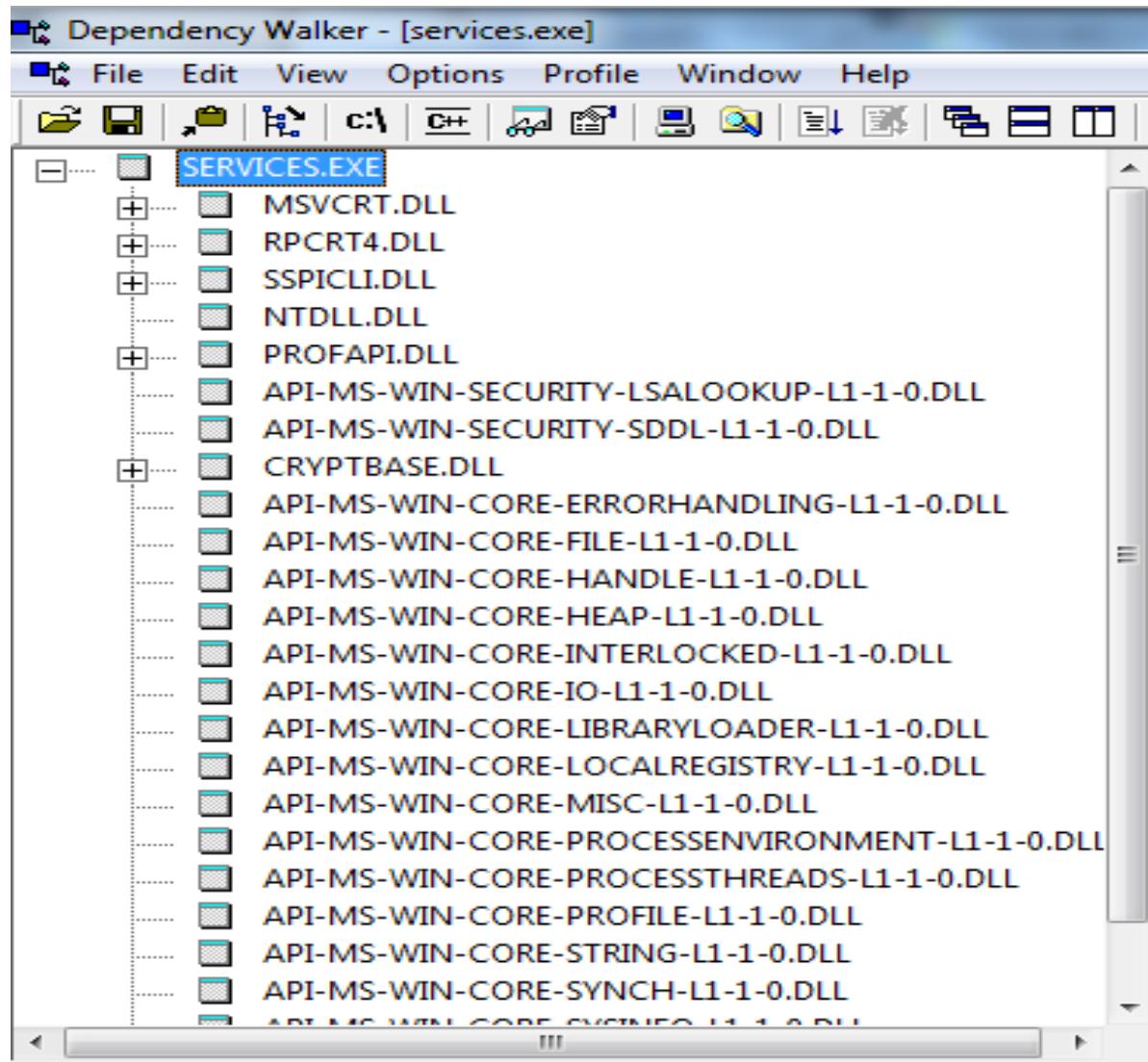
Xác định các thư viện và hàm được dùng

- Trong PE Header liệt kê các thư viện và hàm mà chương trình sử dụng.
- Dựa vào tên và các hàm được gọi có thể phỏng đoán được chức năng của chương trình.
- VD: Hàm **URLDownloadToFile** chỉ ra rằng chương trình tải một cái gì đó.

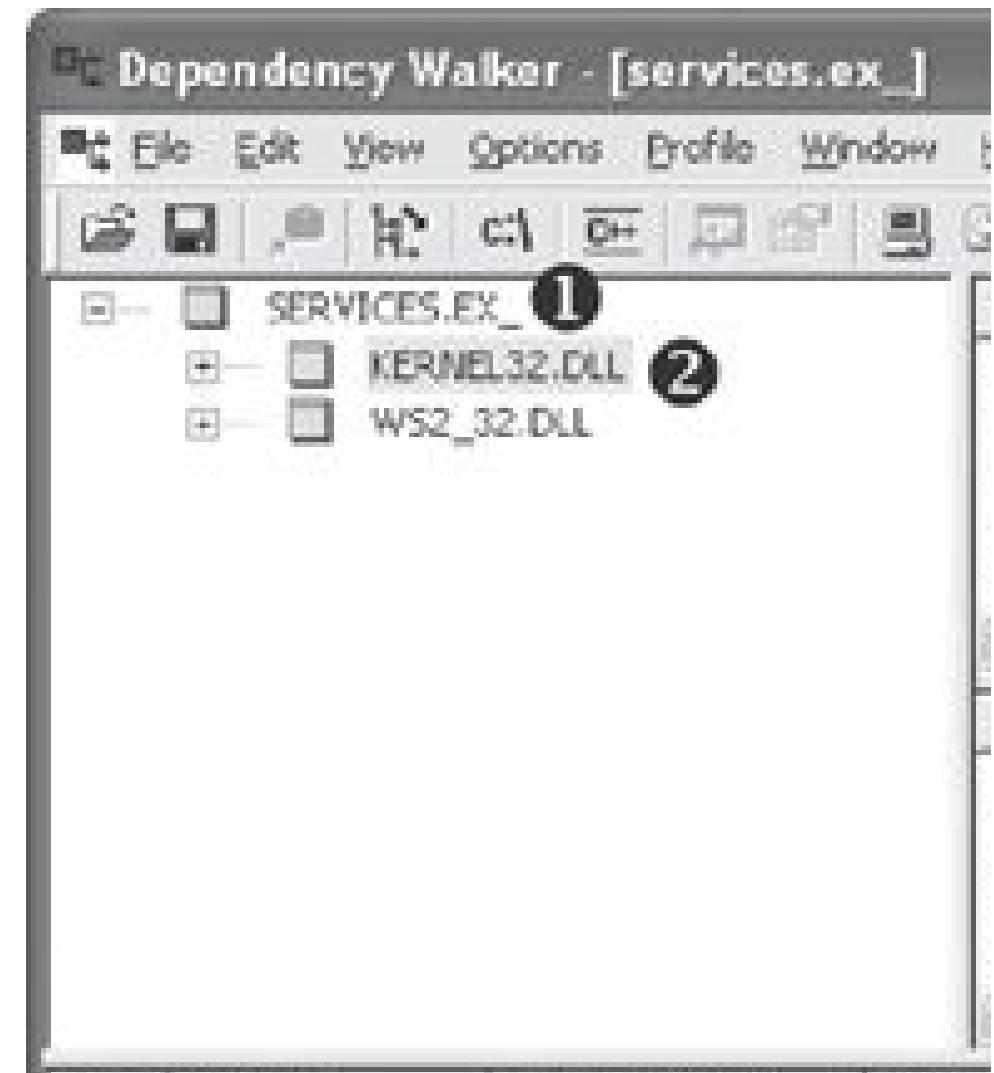
Dependency Walker

- ❑ Công cụ giúp hiển thị các hàm trong **Dynamic Linked**.
- ❑ Một chương trình bình thường sẽ có rất nhiều các DLL được nạp.
- ❑ Phần mềm độc hại/ mã độc thường có rất ít các DLL được nạp, nó sẽ nạp trong quá trình thực thi.

Dependency Walker



Services.exe (Normal)



Services.ex_ (Malware) 68

Imports & Exports in Dependency Walker

Imports (PI)

Exports (E)

The screenshot shows the Dependency Walker interface for the file Lab01-01.exe. The left pane displays the file structure: LAB01-01.EXE depends on KERNEL32.DLL and MSVCRT.DLL. The right pane contains two tables: 'Imports (PI)' and 'Exports (E)'. The 'Imports (PI)' table lists functions imported from MSVCRT.DLL, while the 'Exports (E)' table lists functions exported by LAB01-01.EXE.

PI^	Ordinal	Hint	Function
C	N/A	657 (0x0291)	malloc
C	N/A	585 (0x0249)	exit
C	N/A	211 (0x00D3)	_exit
C	N/A	72 (0x0048)	_XcptFilter
C	N/A	100 (0x0064)	_P__initenv
C	N/A	88 (0x0058)	_getmainarg
C	N/A	271 (0x010F)	_initterm
F	N/A	121 (0x000021)	_setusermath

E^	Ordinal	Hint	Function
I C	108 (0x006C)	106 (0x006A)	_XcptFilter
I C	147 (0x0093)	145 (0x0091)	_getmainarg
I C	181 (0x00B5)	179 (0x00B3)	_P__initenv
I C	187 (0x00BB)	185 (0x00B9)	_P__common
I C	192 (0x00C0)	190 (0x00BE)	_P_fmode
I C	212 (0x00D4)	210 (0x00D2)	_set_app_type
I C	214 (0x00D6)	212 (0x00D4)	_setusermath

Các DLL phổ biến

DLL	Description
<i>Kernel32.dll</i>	This is a very common DLL that contains core functionality, such as access and manipulation of memory, files, and hardware.
<i>Advapi32.dll</i>	This DLL provides access to advanced core Windows components such as the Service Manager and Registry.
<i>User32.dll</i>	This DLL contains all the user-interface components, such as buttons, scroll bars, and components for controlling and responding to user actions.
<i>Gdi32.dll</i>	This DLL contains functions for displaying and manipulating graphics.

Các DLL phổ biến

<i>Ntdll.dll</i>	This DLL is the interface to the Windows kernel. Executables generally do not import this file directly, although it is always imported indirectly by <i>Kernel32.dll</i> . If an executable imports this file, it means that the author intended to use functionality not normally available to Windows programs. Some tasks, such as hiding functionality or manipulating processes, will use this interface.
<i>WSock32.dll</i> and <i>Ws2_32.dll</i>	These are networking DLLs. A program that accesses either of these most likely connects to a network or performs network-related tasks.
<i>Wininet.dll</i>	This DLL contains higher-level networking functions that implement protocols such as FTP, HTTP, and NTP.

Imports & Exports

- ❑ Các file thư viện DLL sẽ Export ra các hàm.
- ❑ Các file thực thi *.EXE sẽ Import các hàm vào từ thư viện.
- ❑ Các hàm, thư viện được Import hay Export đều được liệt kê trong PE Header của file.

Ví dụ: Keylogger

- Imports **User32.dll** và sử dụng hàm **SetWindowsHookEx**. Nó là một hàm rất phổ biến trong các keylogger để nhận đầu vào từ bàn phím.
- Nó Exports **LowLevelKeyboardProc** và **LowLevelMouseProc** để gửi dữ liệu đi nơi khác.
- Nó sử dụng **RegisterHotKey** để xác định một hành động gõ phím đặc biệt như: Ctrl + Shift + P để thu thập dữ liệu.

Ví dụ: Chương trình bị Packed

- Rất ít các hàm, thư viện được gọi
- Tất cả những gì thấy được chỉ là những hàm dùng cho việc giải nén/unpacker

DLLs and Functions Imported from PackedProgram.exe

Kernel32.dll User32.dll

GetModuleHandleA MessageBoxA

LoadLibraryA

GetProcAddress

ExitProcess

VirtualAlloc

VirtualFree

Nội dung

1. Các phương pháp phân tích mã độc
2. Công cụ và kiến thức cơ sở
3. Các nguyên tắc khuyến nghị trong phân tích mã độc
4. Phân tích tĩnh cơ bản
- 5. Phân tích động cơ bản**
6. Xây dựng môi trường phân tích mã độc

Phân tích động cơ bản

Một số lưu ý khi tiến hành phân tích động:

- Thực thi/ chạy phần mềm độc hại đồng thời theo dõi kết quả,
- Yêu cầu cần có một môi trường an toàn cho việc phân tích,
- Cách ly với môi trường máy thật, tránh lây nhiễm sang các máy khác,
- Máy thật có thể ngắt kết internet hoặc kết nối với các máy khác.

Phân tích động

- ❑ Giám sát hệ thống và theo dõi các tiến trình độc hại đồng thời theo dõi kết quả
- ❑ So sánh Registry

Phân tích động

- Giám sát hệ thống và theo dõi các tiến trình độc hại đồng thời theo dõi kết quả**
- So sánh Registry**

Process Monitor

- ❑ Theo dõi Registry, tệp tin hệ thống, mạng, tiến trình và các hoạt động của luồng...
- ❑ Ghi lại tất cả các sự kiện, dễ dàng tìm kiếm và lọc kết quả.
- ❑ Khi chạy lâu sẽ chiếm dụng RAM càng nhiều, dẫn đến treo hoặc tắt máy.

Process Monitor Toolbar

Start/Stop
Capture

Erase Filter

Default Filters
Registry, File system,
Network, Processes

The screenshot shows the Process Monitor application window. At the top is a toolbar with several icons: a folder (File), a magnifying glass (Edit), a gear (Event), a trash can (Erase), a funnel (Filter), a red A (Default Filters), a globe (Registry, File system, Network, Processes), a clipboard (Options), and a question mark (Help). Below the toolbar is a menu bar with File, Edit, Event, Filter, Tools, Options, and Help. The main area displays a table of captured events:

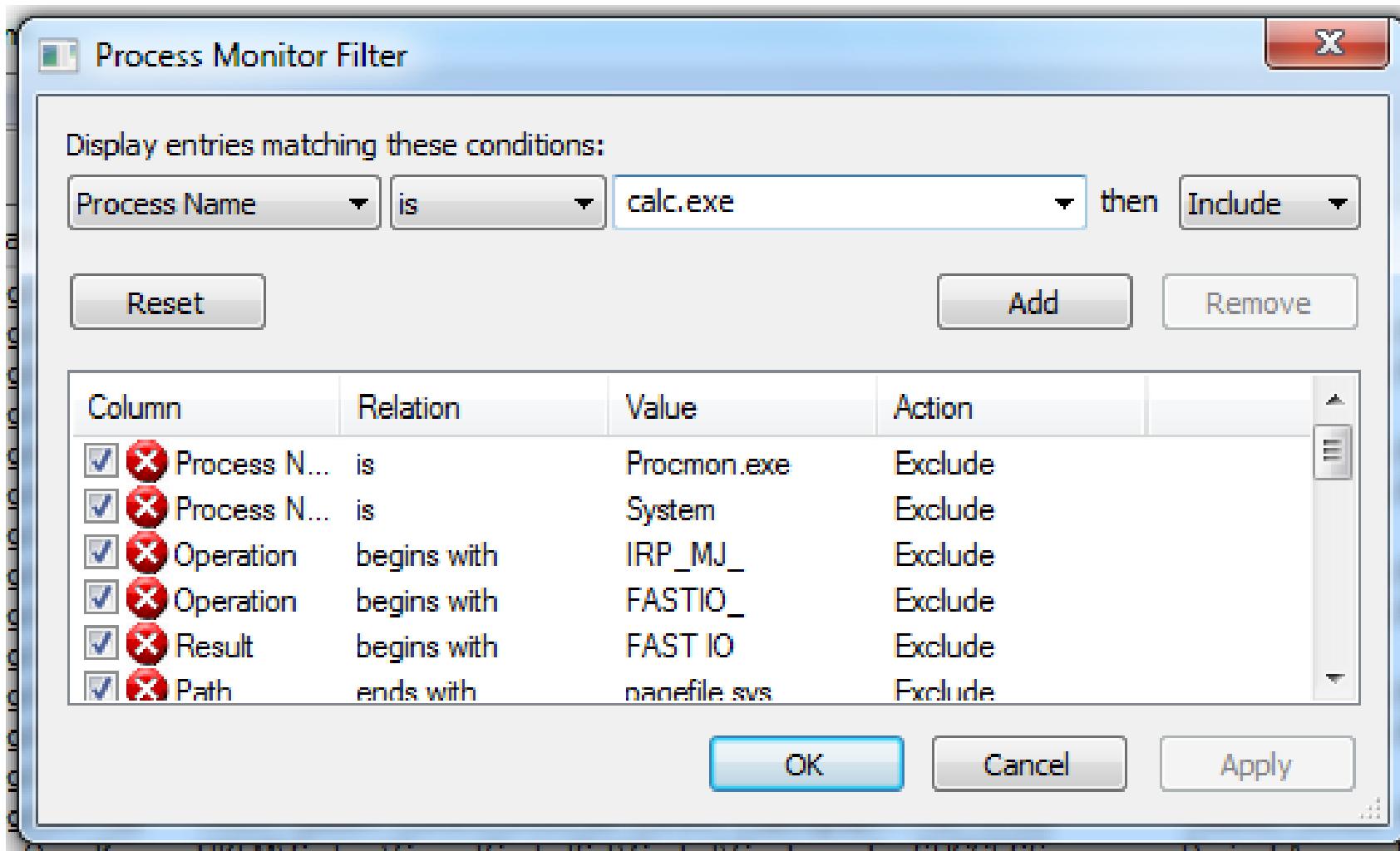
Time of Day	Process Name	PID	Operation	Path
1:17:48.599	1893 PM Explorer.EXE	3188	RegOpenKey	HKLM\Software\Microsoft\Win
1:17:48.599	2018 PM Explorer.EXE	3188	RegCloseKey	HKLM\SOFTWARE\Microsoft\
1:17:48.599	8061 PM Explorer.EXE	3188	CloseFile	C:\Windows\winsxs\x86_microsoft

Lọc và loại trừ kết quả

- Ẩn các hoạt động thông thường/tin cậy trước khi thực thi mã độc**
- Click chuột phải chọn tiến trình và chọn Exclude**
- Không phải lúc nào cũng hoạt động ổn với tất cả các mẫu mã độc**

Filtering with Include

- ❑ Những bộ lọc hữu ích: Process Name, Operation and Detail



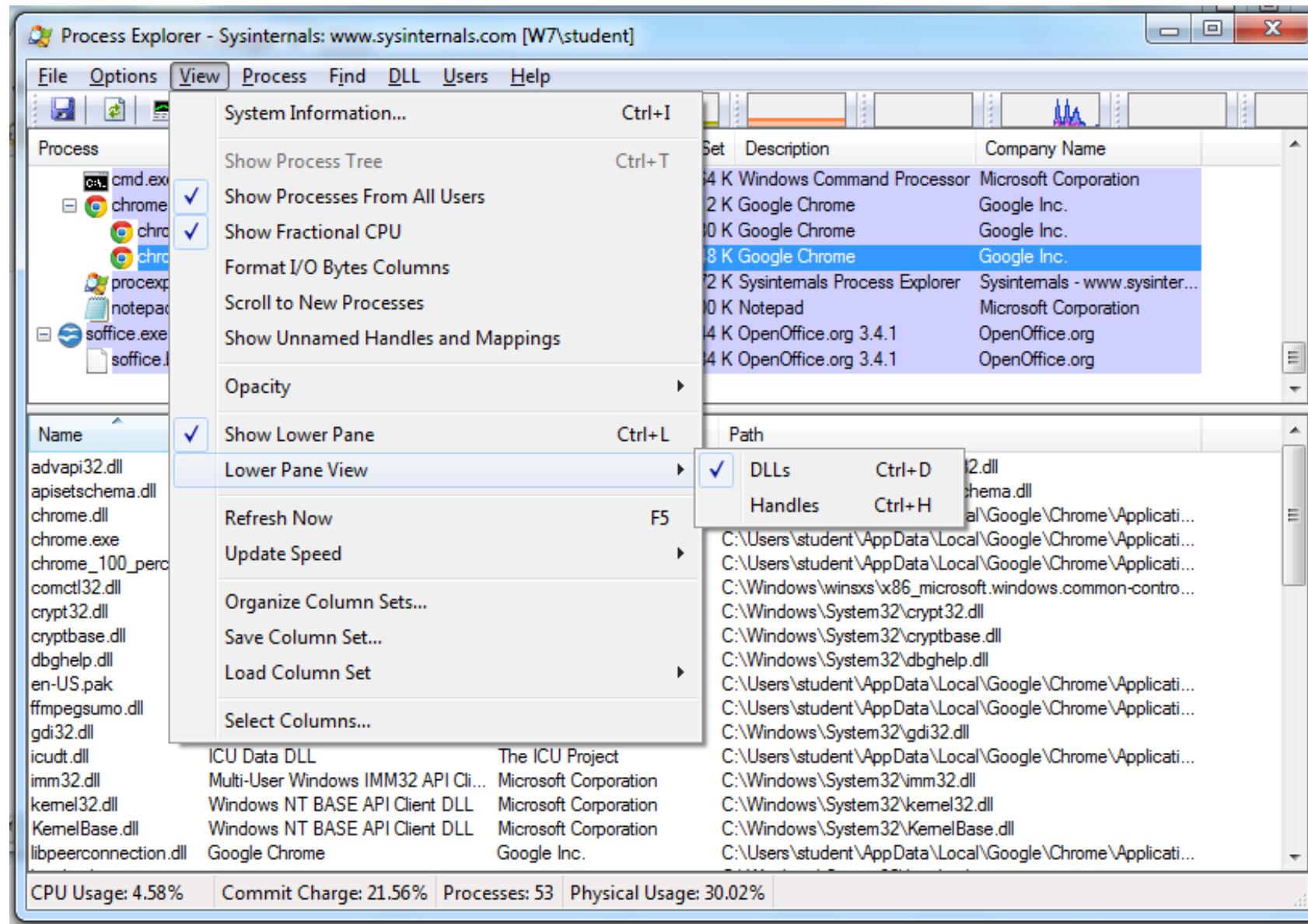
Process Explorer

The screenshot shows the Process Explorer application window. The title bar reads "Process Explorer - Sysinternals: www.sysinternals.com [W7\student]". The menu bar includes File, Options, View, Process, Find, Users, and Help. The toolbar contains icons for Process, Thread, CPU, Private Bytes, Working Set, Description, and Company Name. The main table lists system processes with columns: Process, PID, CPU, Private Bytes, Working Set, Description, and Company Name. The table shows the following data:

Process	PID	CPU	Private Bytes	Working Set	Description	Company Name
System Idle Process	0	96.81	0 K	24 K		
System	4	0.09	48 K	560 K		
Interrupts	n/a	0.88	0 K	0 K	Hardware Interrupts and DPCs	
smss.exe	260		224 K	748 K	Windows Session Manager	Microsoft Corporation
csrss.exe	348	< 0.01	1,252 K	3,164 K	Client Server Runtime Process	Microsoft Corporation
wininit.exe	400		892 K	3,084 K	Windows Start-Up Application	Microsoft Corporation
services.exe	504	0.01	3,972 K	6,640 K	Services and Controller app	Microsoft Corporation
svchost.exe	652		2,700 K	6,024 K	Host Process for Windows S...	Microsoft Corporation
dllhost.exe	1716		6,176 K	4,804 K	COM Surrogate	Microsoft Corporation
WmiPrvSE.exe	740		1,804 K	4,736 K	WMI Provider Host	Microsoft Corporation
svchost.exe	724	< 0.01	2,972 K	6,012 K	Host Process for Windows S...	Microsoft Corporation
svchost.exe	772		13,776 K	11,760 K	Host Process for Windows S...	Microsoft Corporation
audiogd.exe	3200		14,960 K	13,972 K	Windows Audio Device Grap...	Microsoft Corporation
svchost.exe	912		37,940 K	42,292 K	Host Process for Windows S...	Microsoft Corporation
dwm.exe	3248	0.74	61,892 K	27,976 K	Desktop Window Manager	Microsoft Corporation
svchost.exe	936	0.02	20,836 K	29,900 K	Host Process for Windows S...	Microsoft Corporation
svchost.exe	1116	0.03	5,136 K	8,340 K	Host Process for Windows S...	Microsoft Corporation
svchost.exe	1260	0.06	10,840 K	11,960 K	Host Process for Windows S...	Microsoft Corporation
spoolsv.exe	1352		5,392 K	7,436 K	Spooler SubSystem App	Microsoft Corporation
svchost.exe	1388		6,752 K	8,720 K	Host Process for Windows S...	Microsoft Corporation
svchost.exe	1500		2,472 K	4,712 K	Host Process for Windows S...	Microsoft Corporation
gogoc.exe	1592	< 0.01	1,216 K	3,920 K	gogoCLIENT	gogo6, Inc.
vmtoolsd.exe	1728	0.07	7,260 K	10,368 K	VMware Tools Core Service	VMware, Inc.
svchost.exe						

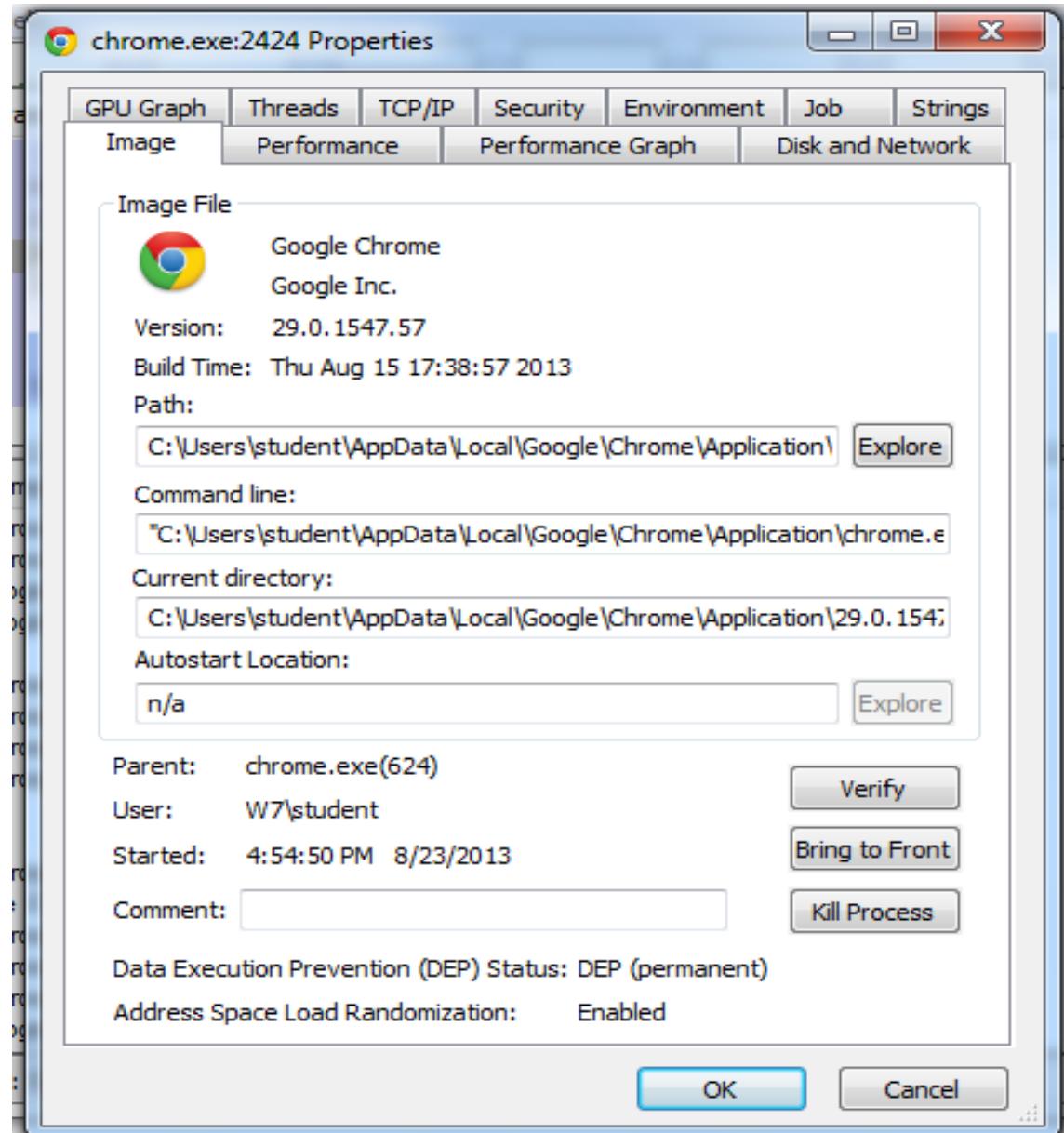
CPU Usage: 3.19% | Commit Charge: 21.92% | Processes: 57 | Physical Usage: 30.24%

DLL Mode - Process Explorer



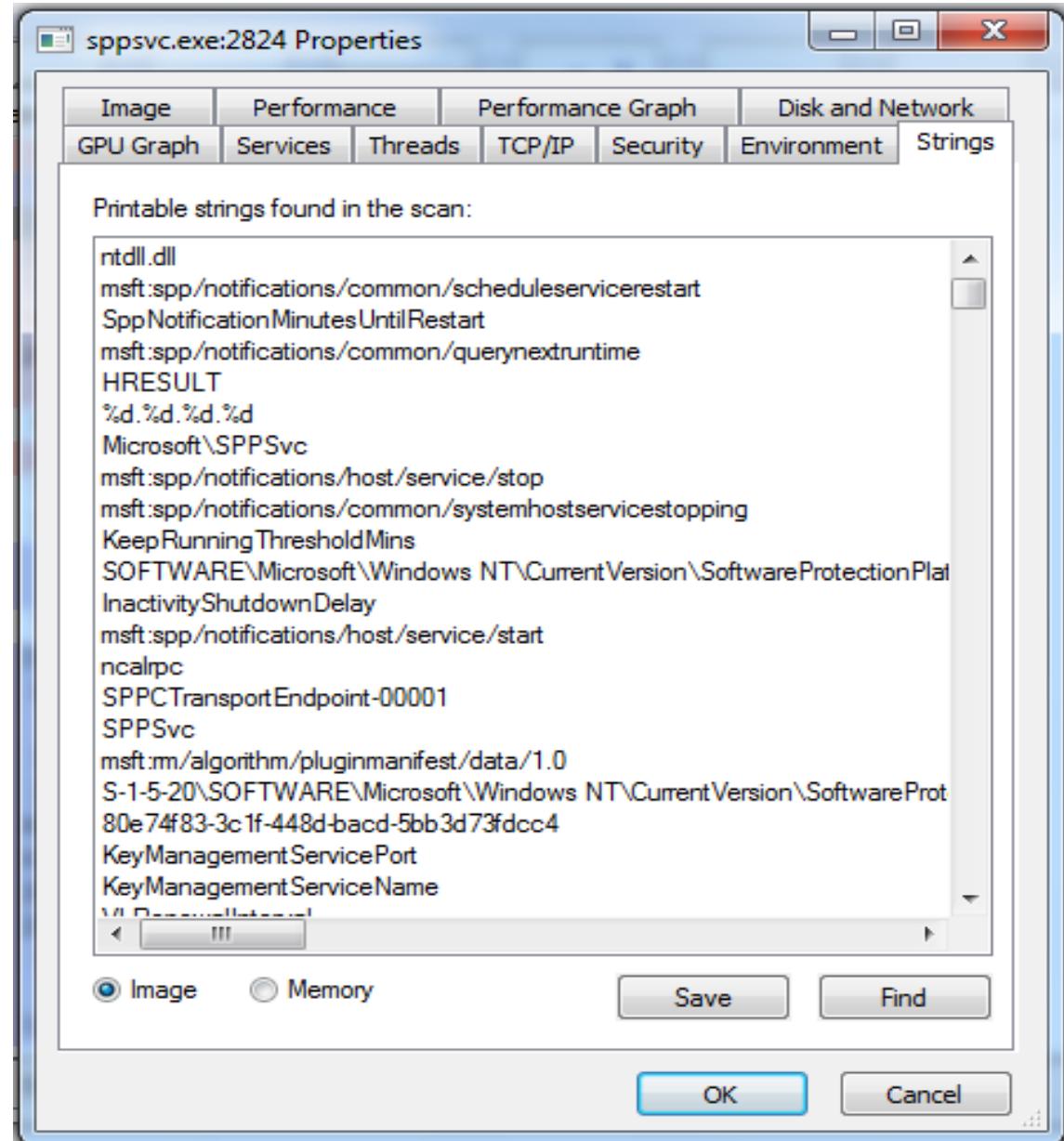
Properties - Process Explorer

- Hiển thị trạng thái DEP và ASLR có được bật hay không.**
- Kiểm tra chữ ký của Windows trên tệp tin.**



Strings - Process Explorer

- So sánh ảnh với chuỗi được nạp trên bộ nhớ, nếu chúng khác xa nhau, nó có thể chỉ ra quá trình thay thế.



Phát hiện mã độc dạng tài liệu

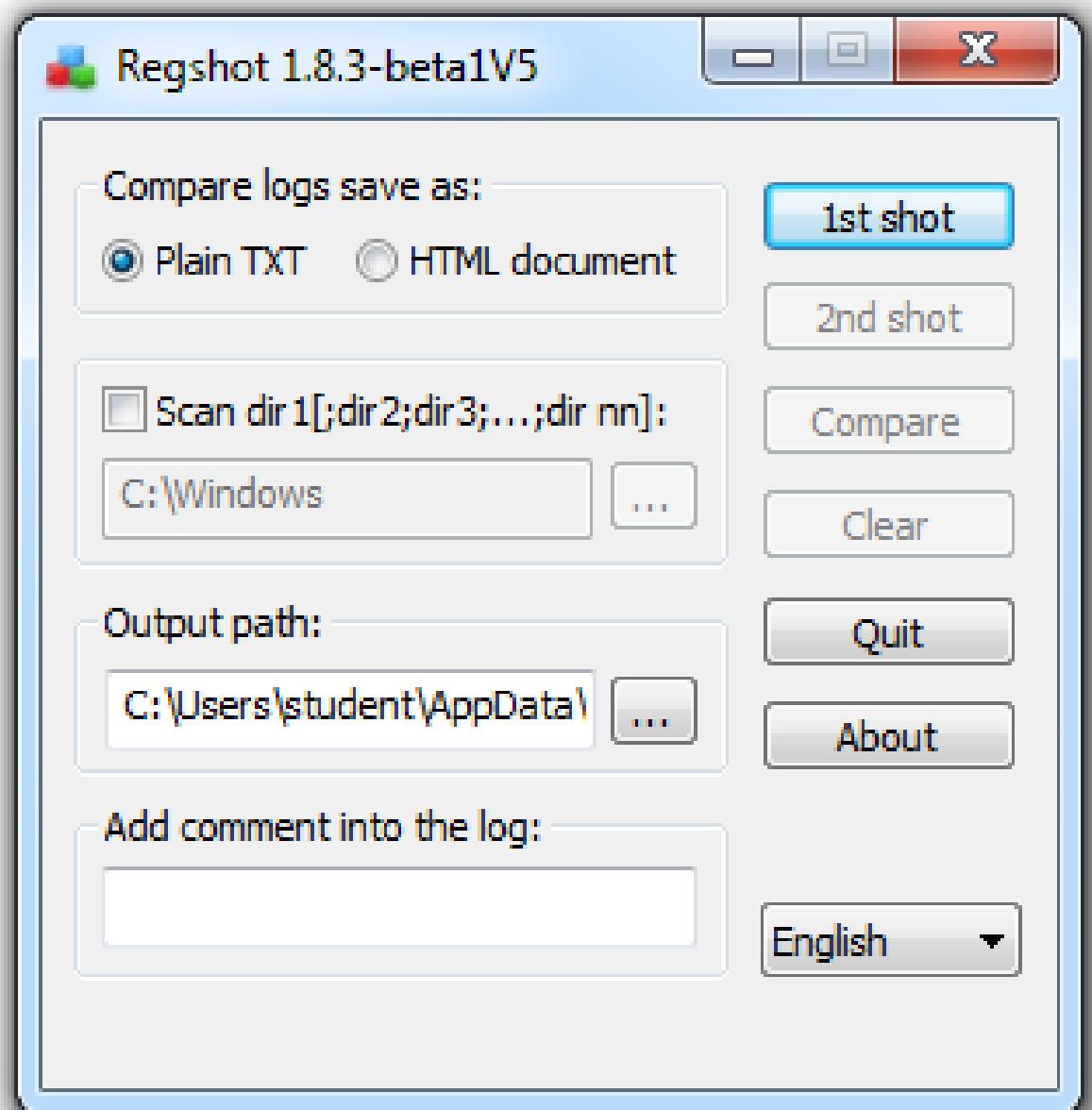
- Các tệp tài liệu có thể chứa mã độc hay đoạn mã khai thác chính những chương trình đọc tài liệu (nếu có lỗ hổng bảo mật).
- Có thể thông qua Process Explorer để xem nó có khởi chạy một tiến trình nào đó hay không.
- Tab Image của Process Explorer Properties sẽ hiển thị vị trí của phần mềm độc hại.

Phân tích động

- ❑ Giám sát hệ thống và theo dõi các tiến trình độc hại đồng thời theo dõi kết quả
- ❑ So sánh Registry

Regshot

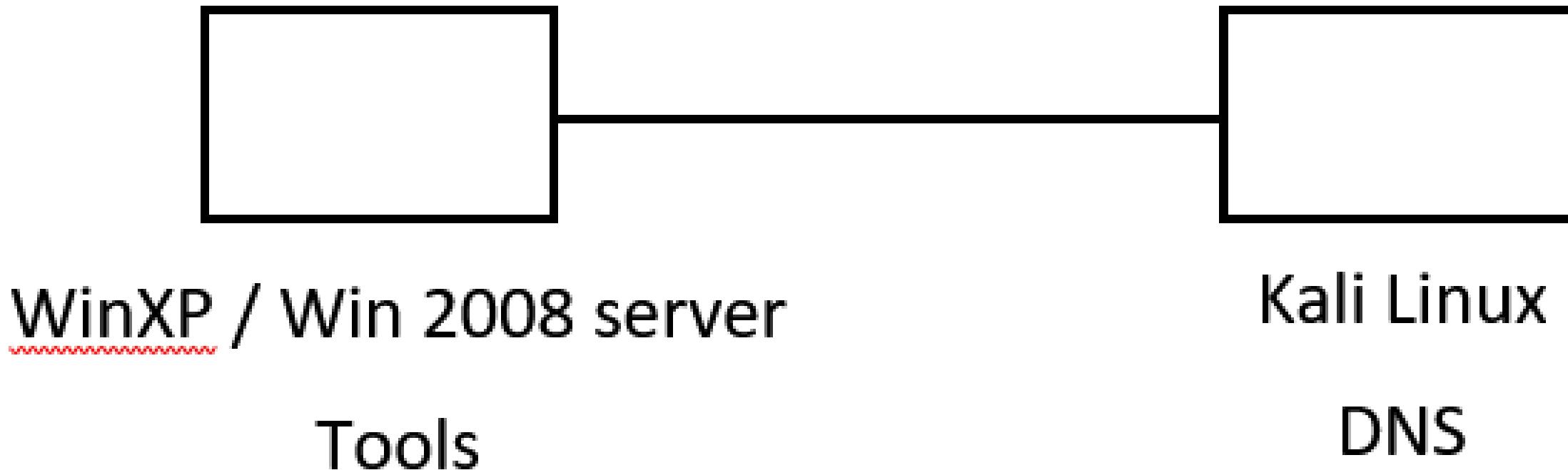
- Regshot hỗ trợ chụp lại Registry trước và sau khi chạy mã độc
- Quan sát và so sánh xem các khóa Registry đã bị thay đổi



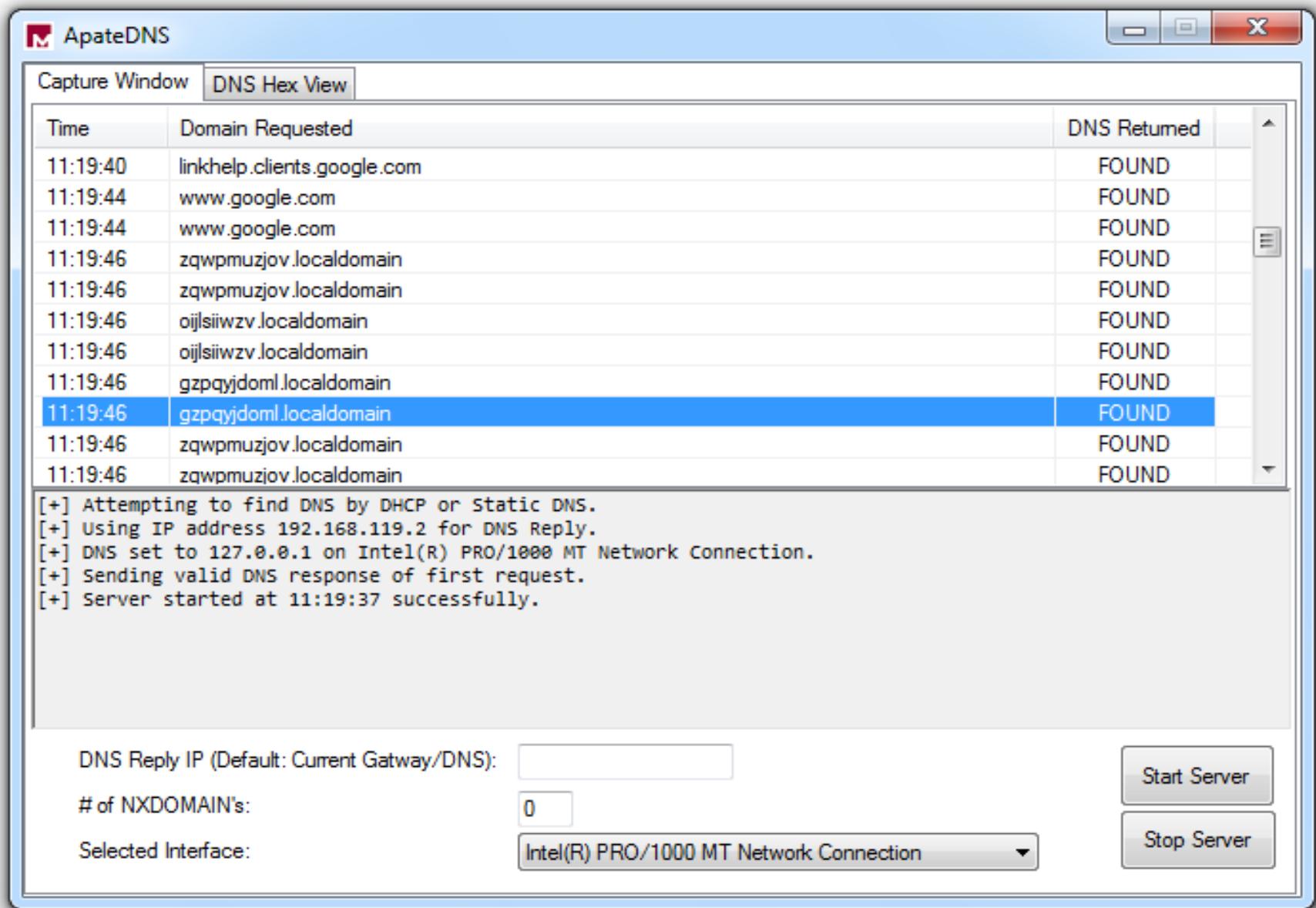
Nội dung

1. Các phương pháp phân tích mã độc
2. Công cụ và kiến thức cơ sở
3. Các nguyên tắc khuyến nghị trong phân tích mã độc
4. Phân tích tĩnh cơ bản
5. Phân tích động cơ bản
6. Xây dựng môi trường phân tích mã độc

Xây dựng môi trường phân tích mã độc

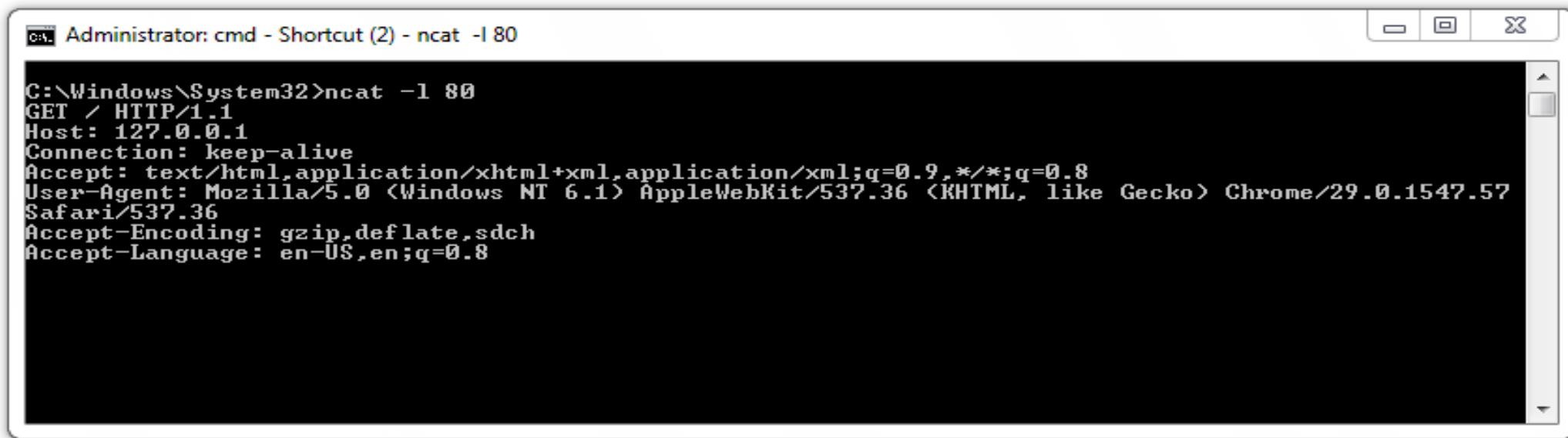


Sử dụng ApateDNS để chuyển hướng luồng DNS



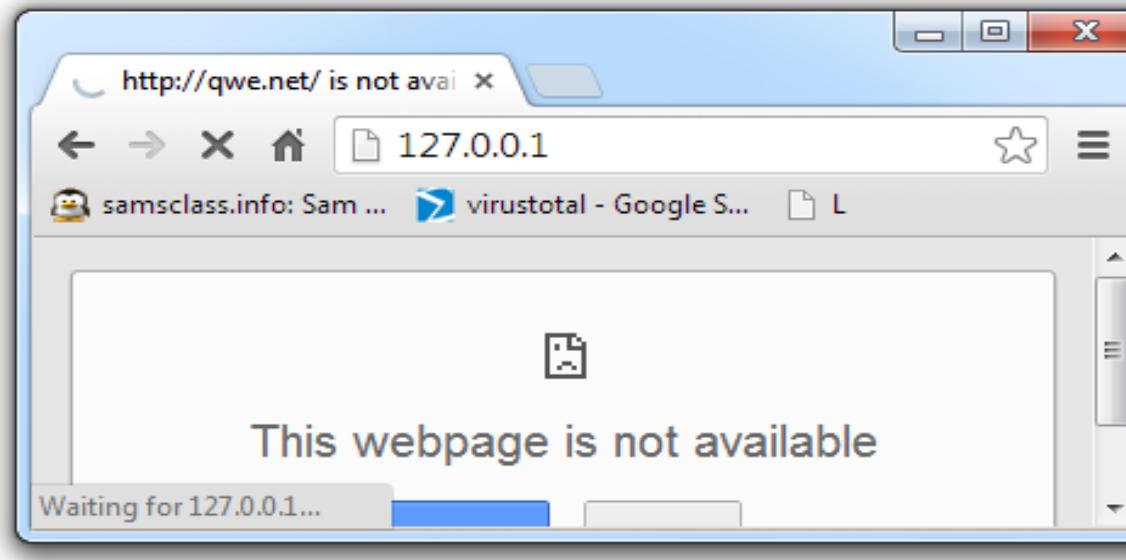
- ❑ Không hoạt động trên Windows XP và Windows 7
- ❑ Nslookup có thể hoạt động nhưng không thể nhìn thấy bất cứ điều gì trong trình duyệt hoặc với công cụ ping
- ❑ Công cụ thay thế: INetSim

Monitoring with Ncat

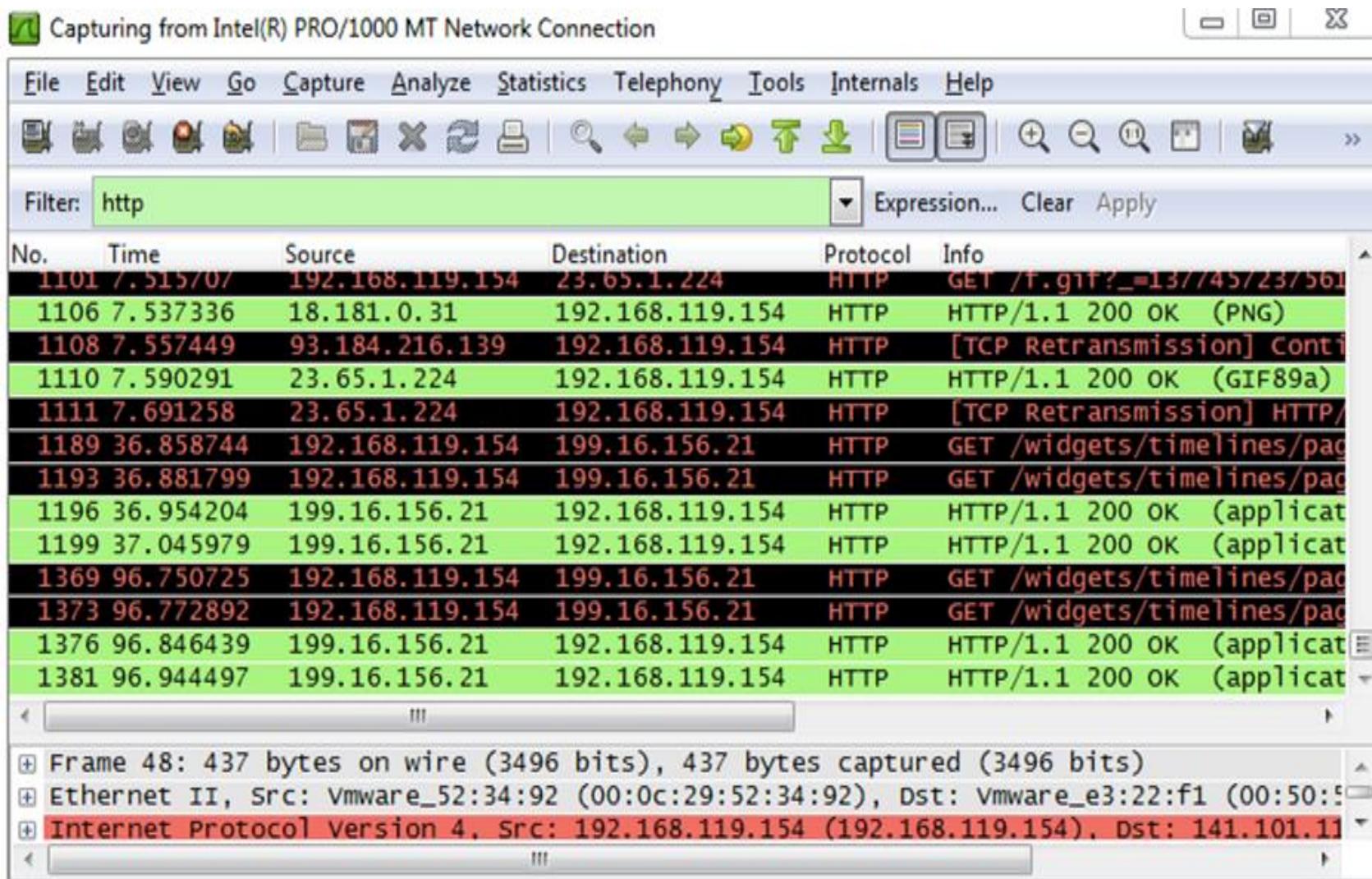


Administrator: cmd - Shortcut (2) - ncat -l 80

```
C:\>Windows\System32>ncat -l 80
GET / HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/29.0.1547.57
Safari/537.36
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
```

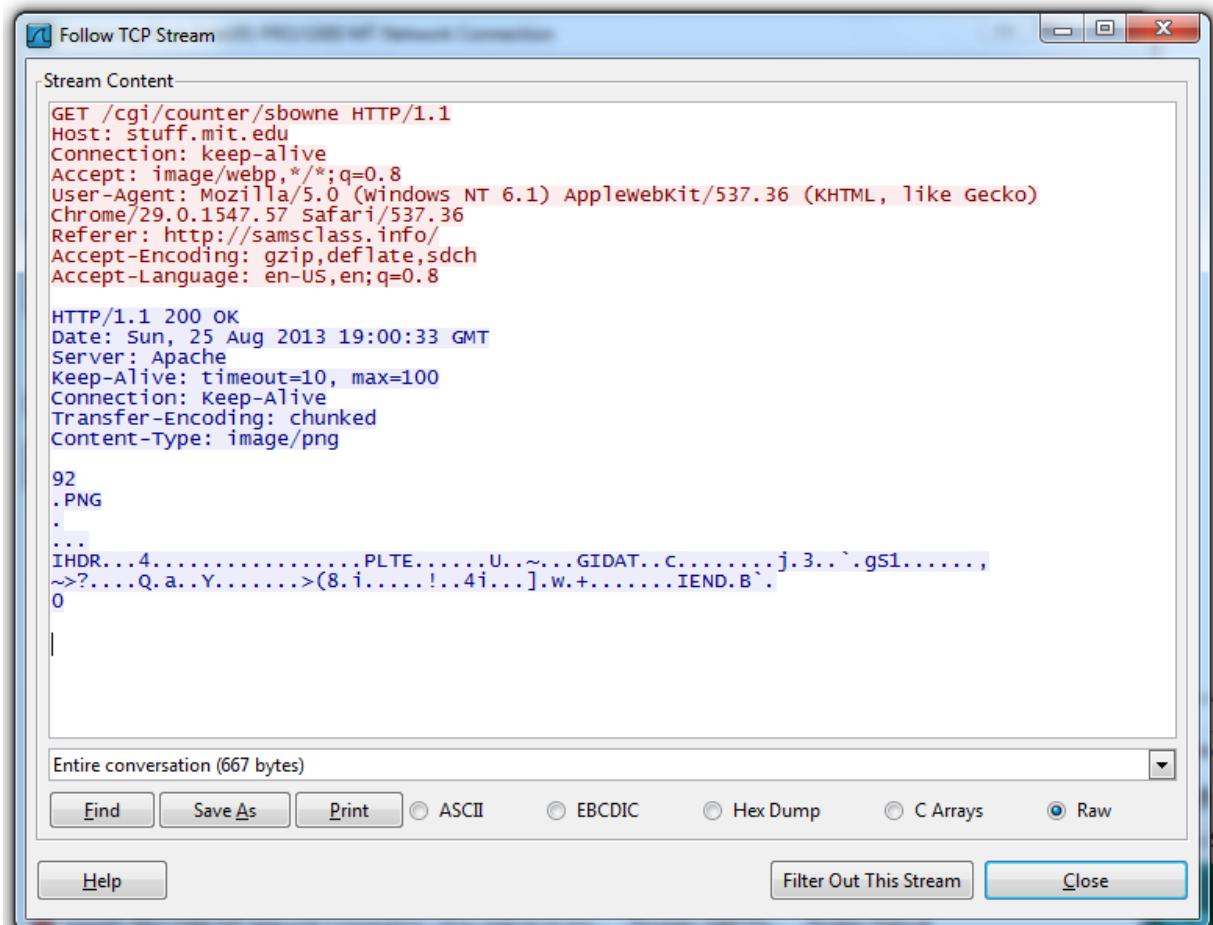


Packet Sniffing with Wireshark

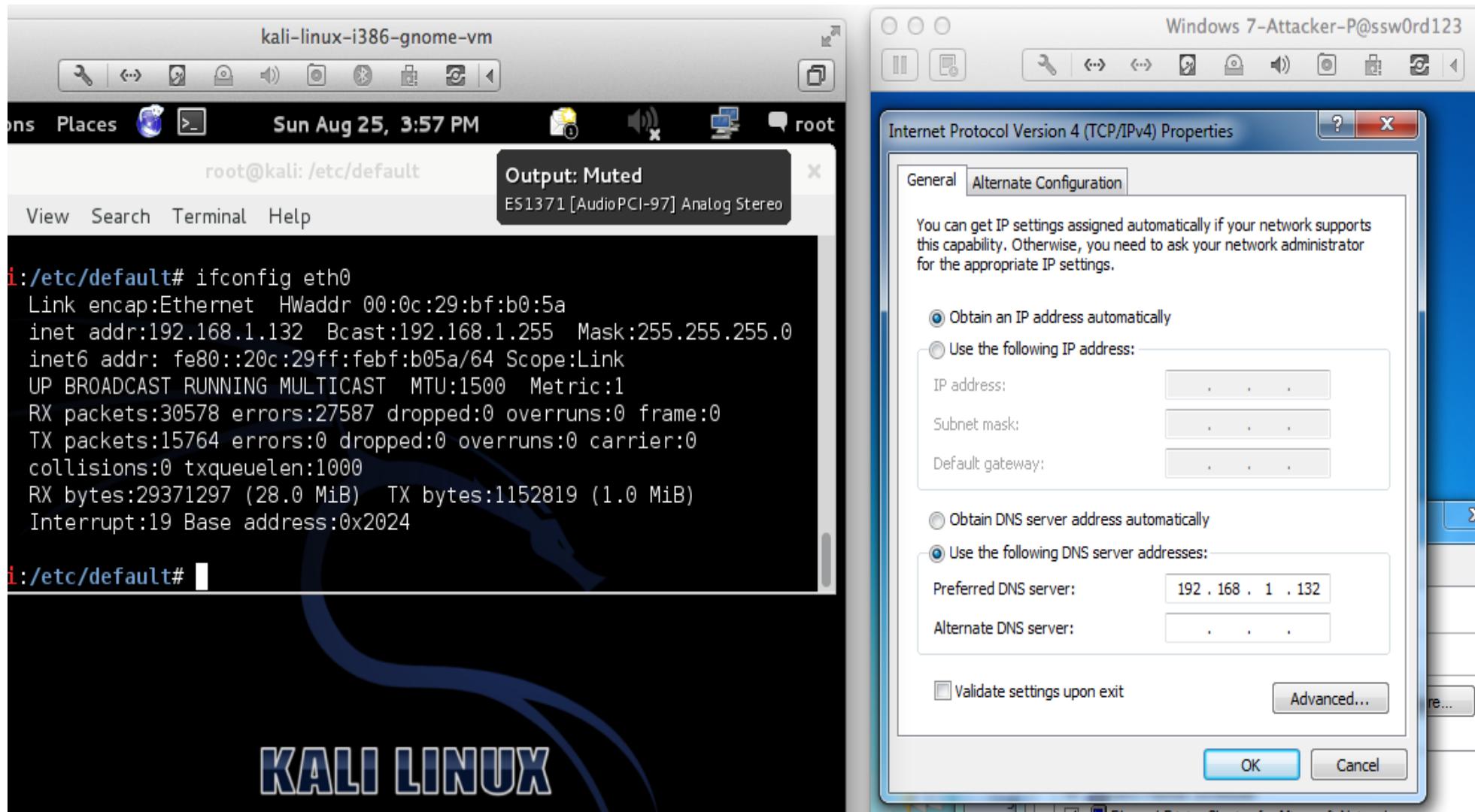


Follow TCP Stream

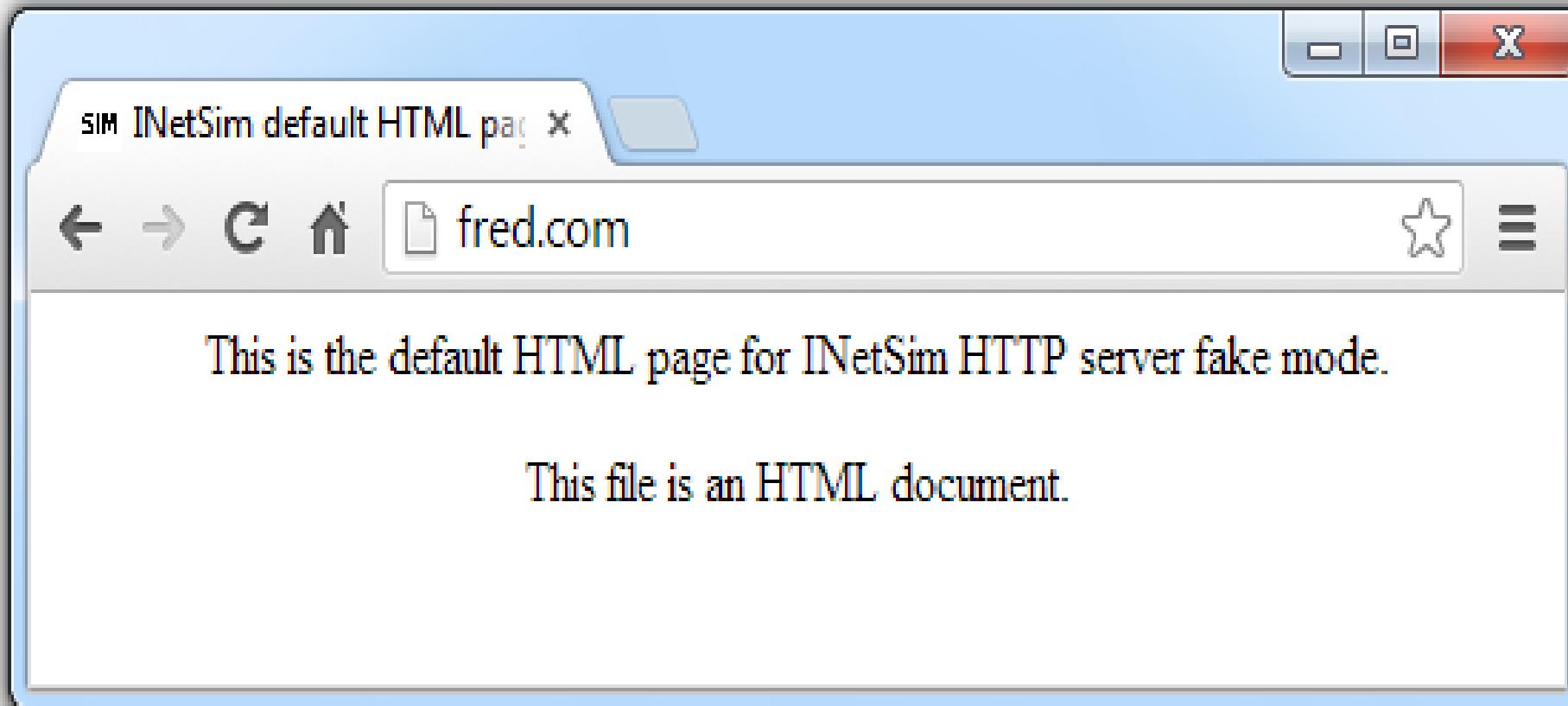
Có thể lưu lại file từ
stream



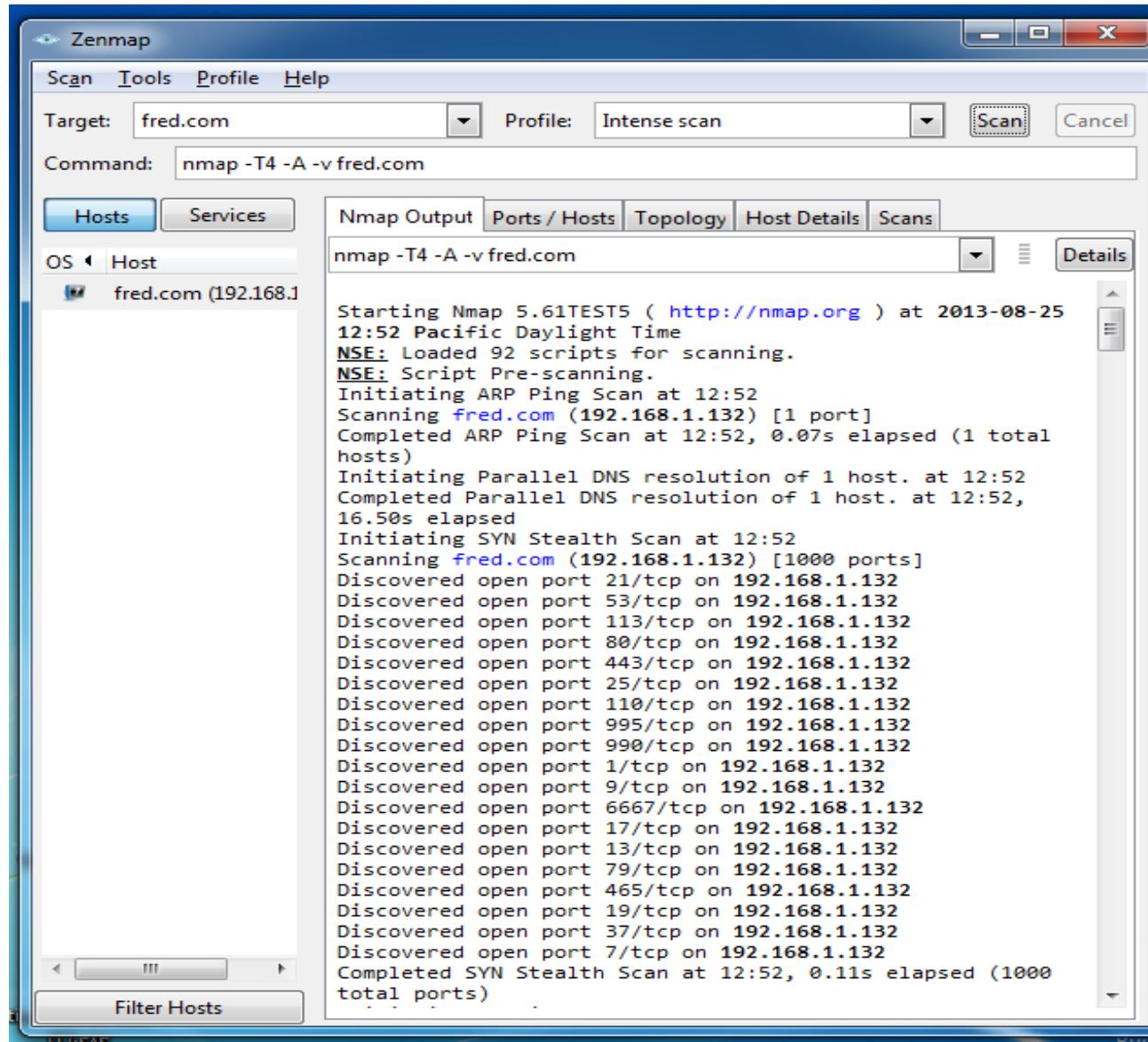
INetSim



INetSim Fools a Browser



INetSim Fools Nmap



Nội dung

1. Các phương pháp phân tích mã độc
2. Công cụ và kiến thức cơ sở
3. Các nguyên tắc khuyến nghị trong phân tích mã độc
4. Phân tích tĩnh cơ bản
5. Phân tích động cơ bản
6. Xây dựng môi trường phân tích mã độc

Mã độc

Chương 3. Dịch ngược mã độc

Mục tiêu

- Nhắc lại một số kiến thức cơ bản về hợp ngữ
- Giới thiệu, hướng dẫn sinh viên sử dụng công cụ IDA pro và các chức năng chính

Tài liệu tham khảo

- [1] Michael Sikorski, Andrew Honig, 2012, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, (ISBN: 978-1593272906).
- [2] Sam Bowne, Slides for a college course at City College San Francisco,
https://samsclass.info/126/126_S17.shtml

Nội dung

- 1. Nhắc lại về Assembly**
- 2. Sử dụng IDA pro để dịch ngược mã độc**
- 3. Sử dụng đối sánh chéo**
- 4. Phân tích hàm**
- 5. Sử dụng biểu đồ hàm**
- 6. Một số lưu ý**

Nội dung

- 1. Nhắc lại về Assembly**
- 2. Sử dụng IDA pro để dịch ngược mã độc**
- 3. Sử dụng đối sánh chéo**
- 4. Phân tích hàm**
- 5. Sử dụng biểu đồ hàm**
- 6. Một số lưu ý**

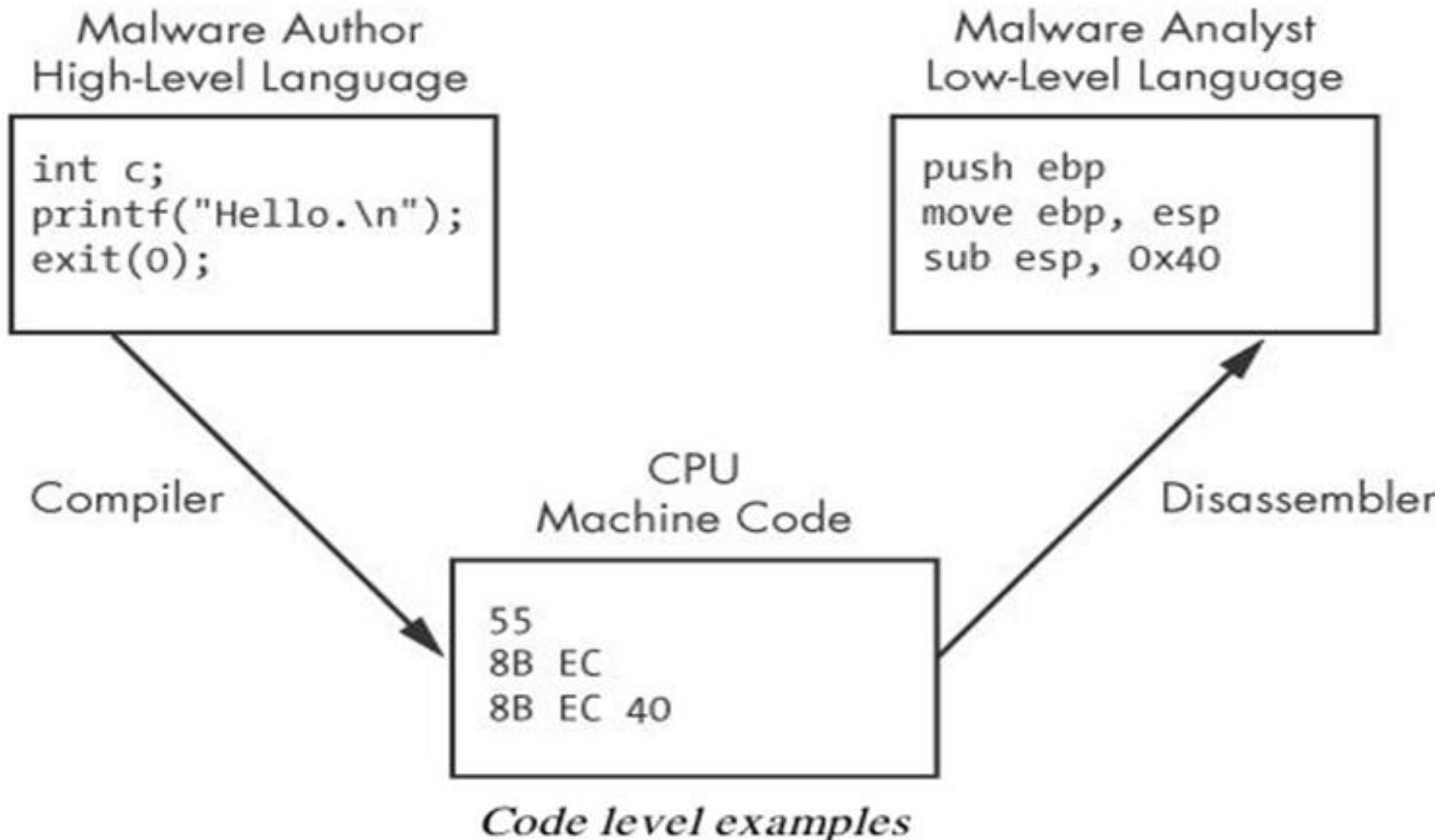
Nhắc lại về Assembly

- Các mức trừu tượng của ngôn ngữ
- Reverse Engineering
- Kiến trúc x86
- Một số cấu trúc đơn giản

Nhắc lại về Assembly

- Các mức trừu tượng của ngôn ngữ
- Reverse Engineering
- Kiến trúc x86
- Một số cấu trúc đơn giản

Các mức trừu tượng của ngôn ngữ



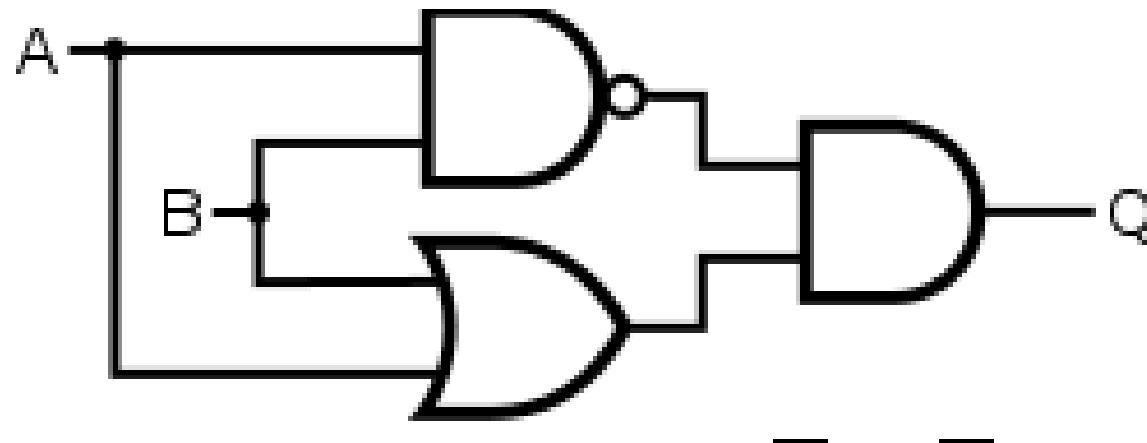
Các mức trừu tượng của ngôn ngữ

Sáu mức trừu tượng:

- Hardware**
- Microcode**
- Machine code**
- Low-level languages**
- High-level languages**
- Interpreted languages**

Hardware

- ❑ Các mạch số
- ❑ Cổng logic: AND, OR, NOT, XOR...
- ❑ Không dễ dàng thao tác được bằng phần mềm



Microcode

- Còn được gọi là Firmware (phần sụn)
- Chỉ hoạt động trên một vài phần cứng cụ thể

Machine code

Mã máy: Dạng ngôn ngữ mà chỉ máy tính mới hiểu được
(Binary, Hexa...)

Opcodes (cũng được hiểu là mã máy):

- Được tạo ra khi biên dịch một chương trình từ ngôn ngữ bậc cao (như C/C++...)
- Là những chỉ thị để ra lệnh cho CPU làm một công việc
- Có thể được tái tạo lại thành dạng hợp ngữ - assembly để con người có thể đọc hiểu
- VD: 90 – NOP, 88 – MOV, FF – PUSH...

Low-level languages

- Dạng ngôn ngữ mà con người có thể đọc hiểu được
- Hợp ngữ - **Assembly** language
Vd: MOV, NOP, PUSH, POP, JMP...
- **Disassembly** là quá trình tái tạo lại ngôn ngữ máy thành Assembly, để con người có thể đọc hiểu
- Quá trình này cần một trình **Disassembler**.

Low-level languages

- ❑ Kết quả tái tạo không phải lúc nào cũng chính xác tuyệt đối.
- ❑ **Assembly** là dạng ngôn ngữ cao nhất có thể tái tạo từ các chương trình sử dụng các ngôn ngữ biên dịch khi mà mã nguồn của phần mềm độc hại không phải lúc nào cũng có.

High-level languages

- Là dạng ngôn ngữ lập trình phổ biến
VD: C, C++, etc...
- Quá trình chuyển đổi sang mã máy cần thông qua
một **trình biên dịch (Compiler)**.

Interpreted languages

- Thuộc dạng ngôn ngữ bậc cao
- VD: Java, C#, Perl, .NET, Python...
- Mã nguồn **không** biên dịch trực tiếp sang mã máy mà nó được biên dịch sang **bytecode**. Sau đó trình thông dịch của ngôn ngữ đó mới dịch bytecode sang mã máy.

Interpreted languages

- ❑ Bytecode còn được gọi là mã trung gian, độc lập với phần cứng và hệ điều hành
- ❑ Trình thông dịch sẽ dịch bytecode sang mã máy trong quá trình chạy và máy tính sẽ thực thi chúng
- ❑ VD: Java Bytecode sẽ chạy trong JVM (Java Virtual Machine)

Nhắc lại về Assembly

- Các mức trừu tượng của ngôn ngữ
- Reverse Engineering
- Kiến trúc x86
- Một số cấu trúc đơn giản

Reverse Engineering

- ❑ Các phần mềm độc hại tồn tại trên ổ cứng ở dạng tệp nhị phân (nằm ở mức Machine code).
- ❑ Quá trình Disassembly sẽ tái tạo và chuyển đổi phần mềm độc hại ở dạng nhị phân về dạng hợp ngữ (Assembly).
- ❑ IDA Pro là một trình Disassembler phổ biến trong việc dịch ngược.

Assembly Language

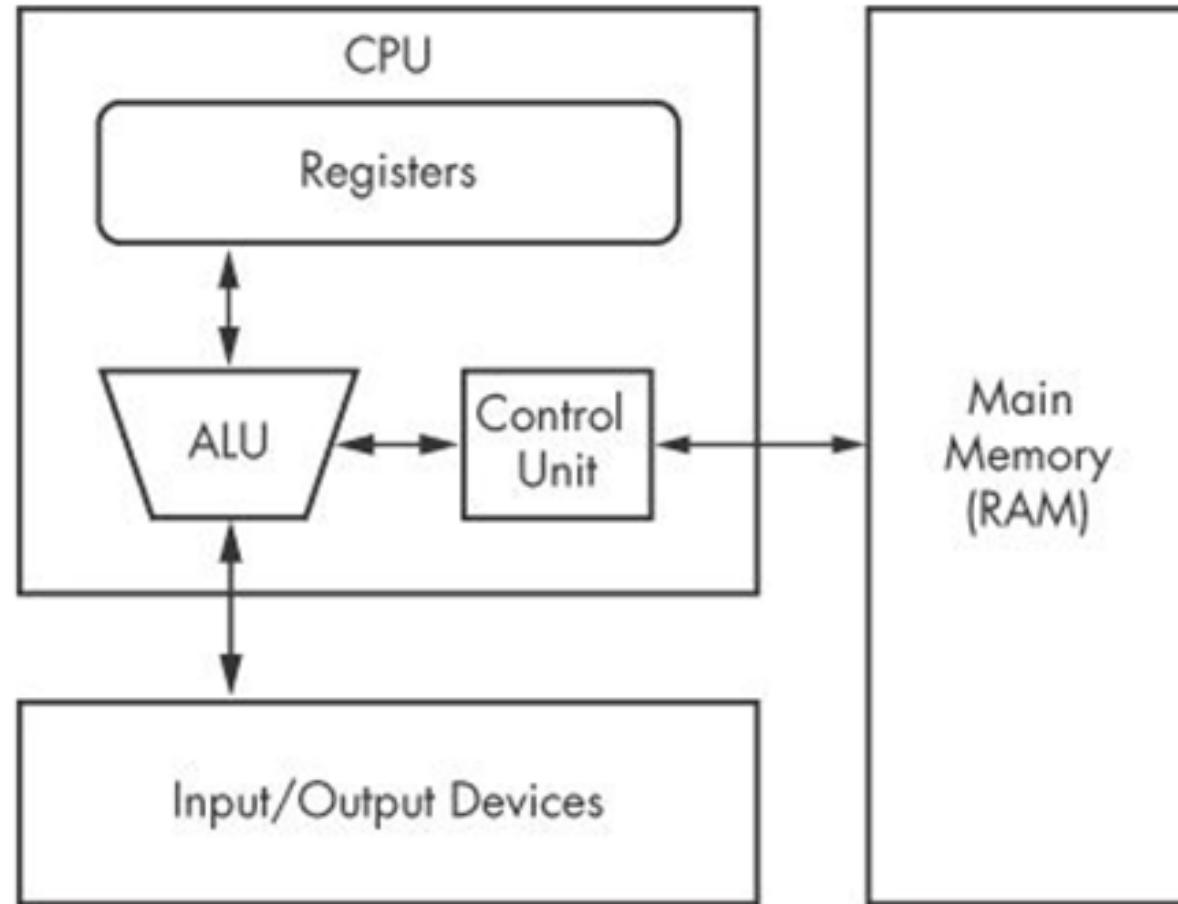
- Mỗi bộ vi xử lý khác nhau sẽ có những tập lệnh assembly khác nhau
- Kiến trúc x86 – 32 bits, Kiến trúc x64 – 64 bits
- Hệ điều hành Windows chạy x86 hoặc x64
- Hầu hết các mã độc đều được thiết kế chạy trên x86.

Nhắc lại về Assembly

- Các mức trừu tượng của ngôn ngữ
- Reverse Engineering
- **Kiến trúc x86**
- Một số cấu trúc đơn giản

Kiến trúc máy tính Von Neumann

- CPU (Central Processing Unit) sẽ thực thi các mã lệnh,
- RAM lưu trữ dữ liệu và mã lệnh,
- Hệ thống vào-ra kết nối với các thiết bị ngoại vi: bàn phím, màn hình, đĩa cứng...



Von Neumann architecture

CPU Components

□ Khối CU - Control Unit

- Tìm và nạp các mã lệnh từ bộ nhớ RAM, sử dụng thanh ghi có tên là IP/EIP (Instruction Poiter)

□ Khối ALU - Arithmetic Logic Unit

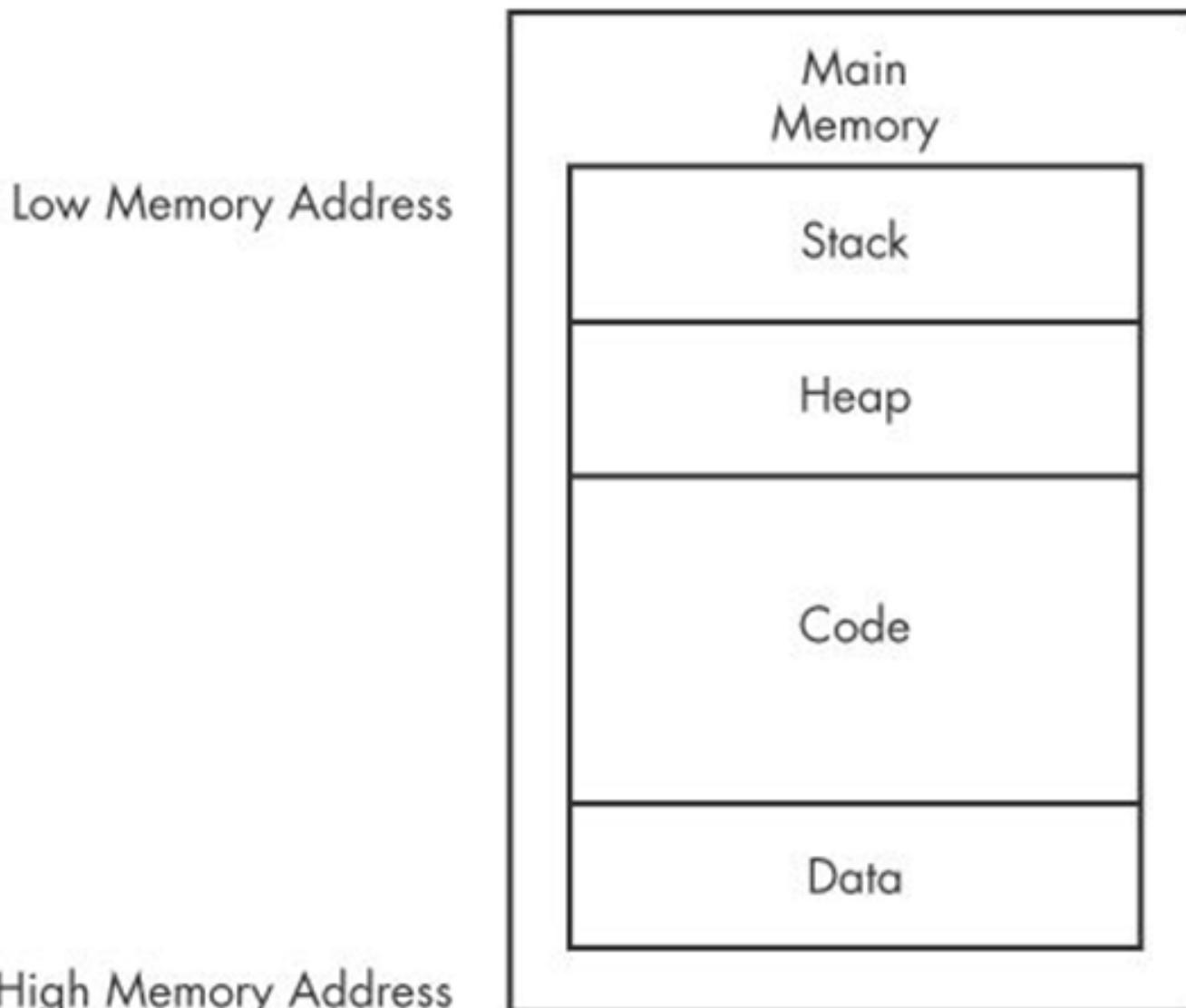
- Thực hiện việc tính toán số học, đưa kết quả vào thanh ghi hoặc bộ nhớ RAM

CPU Components

□ Thanh ghi - Register

- Vùng nhớ nằm bên trong CPU, lưu trữ dữ liệu đợi CPU xử lý
- Tốc độ của Register là rất nhanh, nhanh hơn nhiều lần so với bộ nhớ RAM

Main Memory (RAM)



Basic memory layout for a program

Data Segment (RAM)

- ❑ Lưu các giá trị của chương trình khi được nạp vào RAM.
- ❑ Các giá trị tĩnh (biến static), không thể thay đổi trong khi chương trình đang chạy.
- ❑ Chứa các biến toàn cục, hằng số... (phạm vi toàn chương trình).

Code Segment (RAM)

- ❑ Chứa mã nguồn đã được biên dịch của chương trình (Các chỉ dẫn CPU thực thi)
- ❑ Kiểm soát chương trình làm việc

Heap Segment (RAM)

- Vùng nhớ dùng trong cấp phát động (malloc, new...),
- Vùng nhớ này thay đổi thường xuyên trong quá trình thực hiện chương trình,
- Khi không có nhu cầu sử dụng cần phải giải phóng (free, delete...).

Stack Segment (RAM)

- Vùng nhớ lưu chứa các biến cục bộ trong hàm, các tham số truyền vào cho hàm, các giá trị trả về của hàm...
- Vùng nhớ quản lý bởi CPU.

Instructions

- Một lệnh (Instruction) bao gồm: toán hạng đích, nguồn và tên lệnh
- VD: **MOV ECX, 0x42**
 - Tên lệnh: MOV – Lệnh di chuyển giá trị 0x42 (hexa) vào thanh ghi ECX,
 - Giá trị 0x42 ở dạng thập lục phân (hex).
- Lệnh này ở dạng nhị phân (mã máy) sẽ có dạng:
0xB942000000

Endianness

□ Thuật ngữ Big-Endian và Little-Endian diễn tả sự khác nhau về cách đọc và ghi dữ liệu giữa các nền tảng máy tính.

□ Big-Endian

- Bit LSB lưu ở ô nhớ có địa chỉ lớn nhất (ngoài cùng bên phải)
- Bit MSB lưu ở ô nhớ có địa chỉ nhỏ nhất (ngoài cùng bên trái)
- VD: 0x42 được biểu diễn: 0x00000042

Endianness

□ Little-Endian

- Bit LSB lưu ở ô nhớ có địa chỉ nhỏ nhất (ngoài cùng bên trái)
- Bit MSB lưu ở ô nhớ có địa chỉ lớn nhất (ngoài cùng bên phải)
- VD: 0x42 được biểu diễn: 0x42000000

□ Dữ liệu mạng biểu diễn dạng Big-Endian

□ Chương trình trên Windows x86 sử dụng kiểu Little-Endian

IP Address

IP: 127.0.0.1 chuyển sang dạng hex: 7F 00 00 01

- Gửi qua mạng dưới dạng: 0x7F000001
- Lưu trữ trong RAM dưới dạng: 0x0100007F

Operands

Toán tử trong một lệnh của hợp ngũ có thể là:

- Một giá trị trực tiếp: 0x42h
- Toán tử là một thanh ghi: EAX, EBX, ECX...
- Toán tử lấy địa chỉ: [EAX], [ECX + 10h]...

Registers

The x86 Registers

General registers Segment registers Status register Instruction pointer

EAX (AX, AH, AL) CS

EFLAGS

EIP

EBX (BX, BH, BL) SS

ECX (CX, CH, CL) DS

EDX (DX, DH, DL) ES

EBP (BP) FS

ESP (SP) GS

ESI (SI)

Registers

- Nhóm thanh ghi chung: Được CPU sử dụng như bộ nhớ siêu tốc trong việc tính toán, đặt biến tạm, tham số: EAX, EBX.
- Nhóm thanh ghi xử lý chuỗi: Sao chép, tính độ dài chuỗi: EDI.
- Thanh ghi ngăn xếp: Thanh ghi được dùng trong quản lý cấu trúc bộ nhớ ngăn xếp: ESP, EBP.

Registers

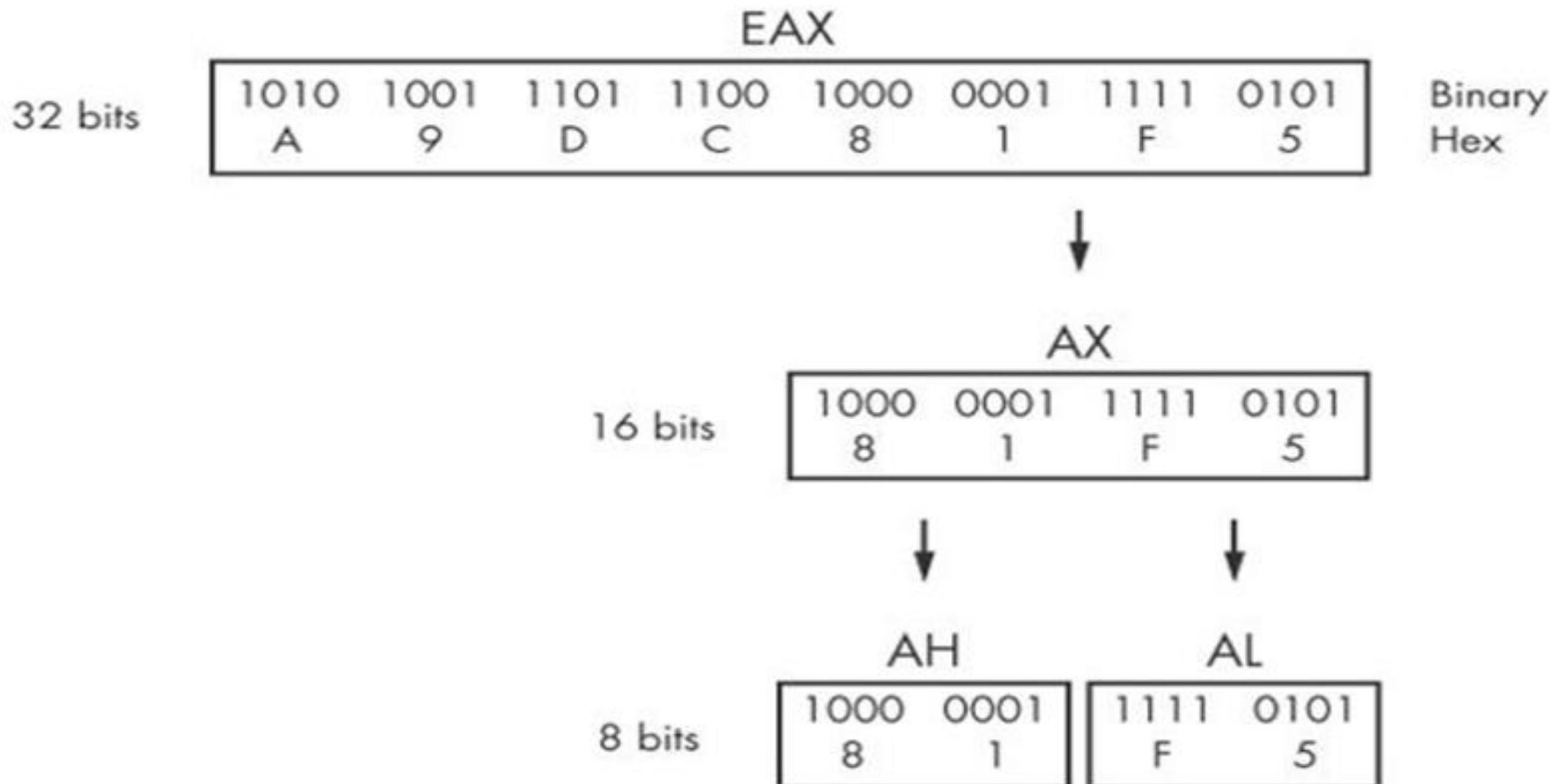
Thanh ghi đặc biệt

- ❑ EIP: Thanh ghi con trỏ lệnh, luôn trỏ đến lệnh tiếp theo mà CPU sẽ thực thi
- ❑ EFLAGS: Thanh ghi cờ, các bit cờ được bật hay không thể hiện trạng thái sau khi thực hiện một lệnh của CPU.

Kích thước Register

- Các thanh ghi trên kiến trúc x86 phổ biến có kích thước là 32 bits.
 - Các thanh ghi có thể tham chiếu dưới dạng 32 bit (VD: EAX) hoặc dưới dạng 16 bit (VD: AX).
- Bốn thanh ghi chung EAX, EBX, ECX, EDX cũng có thể tham chiếu dưới dạng các giá trị 8 bits:
 - AL: Biểu diễn giá trị 8 bit thấp
 - AH: Biểu diễn giá trị 8 bit cao

Kích thước Register



x86 EAX register breakdown

General Registers

- Thường lưu trữ dữ liệu hoặc địa chỉ bộ nhớ
- Một số quy định riêng về việc sử dụng các thanh ghi
 - Phép nhân và chia thường dùng thanh ghi EAX và EDX
 - Trình biên dịch sử dụng các thanh ghi một cách nhất quán (theo chuẩn)
 - Các giá trị trả về trong một hàm thường lưu vào thanh ghi EAX

Flags

- ❑ EFLAGS là một thanh ghi trạng thái
- ❑ Kích thước 32 bits
- ❑ Mỗi bit là một cờ (Chỉ có vài cờ: SF, ZF, AF, PF, CF..., chứ không phải có tận 32 cờ)
- ❑ Các cờ được bật (1) hoặc không được bật (0)

Flags

Cờ Zero – ZF (cờ không):

Cờ này được bật khi kết quả của phép toán là 0

Cờ Carry – CF (cờ nhớ):

Cờ này được bật khi có mượn hoặc nhớ bit MSB

Cờ Sign – SF (cờ dấu):

Cờ này được bật khi bit MSB của kết quả = 1 tức đây là một kết quả âm

Cờ Trap – TF (cờ bẫy):

Cờ này được bật để sử dụng chế độ gõ lỗi, CPU sẽ chỉ thực hiện một lệnh tại một thời điểm

Flags

Cờ Overflow – OF (cờ tràn):

Cờ này được bật khi thực hiện phép tính với hai số cùng dấu mà kết quả là số có dấu => tràn

Cờ Parity – PF (cờ chẵn lẻ):

Cờ này được bật = 1 khi tổng số bit 1 trong kết quả là chẵn

Cờ này được bật = 0 khi tổng số bit 1 trong kết quả là lẻ

VD cờ tràn

EAX = 7fffffff (Giá trị số dương lớn nhất dạng 32 bit). Ta thực hiện lệnh:
add eax, 1. Kết quả sau khi thực hiện: EAX = 80000000h, OF = 1

EIP (Extended Instruction Pointer)

- Thanh ghi chứa địa chỉ trên bộ nhớ của lệnh tiếp theo mà CPU sẽ thực thi
- Nếu EIP chứa dữ liệu sai, CPU sẽ tìm ra lệnh không hợp lệ và sẽ gây lỗi chương trình
- EIP là mục tiêu của lỗi tràn bộ đệm (Buffer Overflow)

Nhắc lại về Assembly

- Các mức trừu tượng của ngôn ngữ
- Reverse Engineering
- Kiến trúc x86
- Một số cấu trúc đơn giản

Một số cấu trúc đơn giản

□ Cú pháp lệnh mov cơ bản:

MOV **Toán hạng đích**, **Toán hạng nguồn**

- Di chuyển dữ liệu từ toán hạng nguồn sang toán hạng đích

□ Toán hạng được biểu diễn trong cặp dấu [toán hạng] hiểu đó là toán hạng lấy địa chỉ. VD:

- **mov giá trị**: mov eax, ebx hoặc mov eax, 19h
- **mov địa chỉ**: mov eax, [ebx] hoặc mov eax, [ebx+10]

Một số cấu trúc đơn giản

mov Instruction Examples

Instruction	Description
<code>mov eax, ebx</code>	Copies the contents of EBX into the EAX register
<code>mov eax, 0x42</code>	Copies the value 0x42 into the EAX register
<code>mov eax, [0x4037C4]</code>	Copies the 4 bytes at the memory location 0x4037C4 into the EAX register
<code>mov eax, [ebx]</code>	Copies the 4 bytes at the memory location specified by the EBX register into the EAX register
<code>mov eax, [ebx+esi*4]</code>	Copies the 4 bytes at the memory location specified by the result of the equation <code>ebx+esi*4</code> into the EAX register

LEA (Load Effective Address)

LEA EAX, [EBX+8]

- So sánh MOV EAX, [EBX+8] và LEA EAX, [EBX+8]
 - MOV: chuyển giá trị tại địa chỉ EBX + 8 vào EAX
 - LEA: Chuyển địa chỉ mà EBX đang giữ + 8 vào EAX

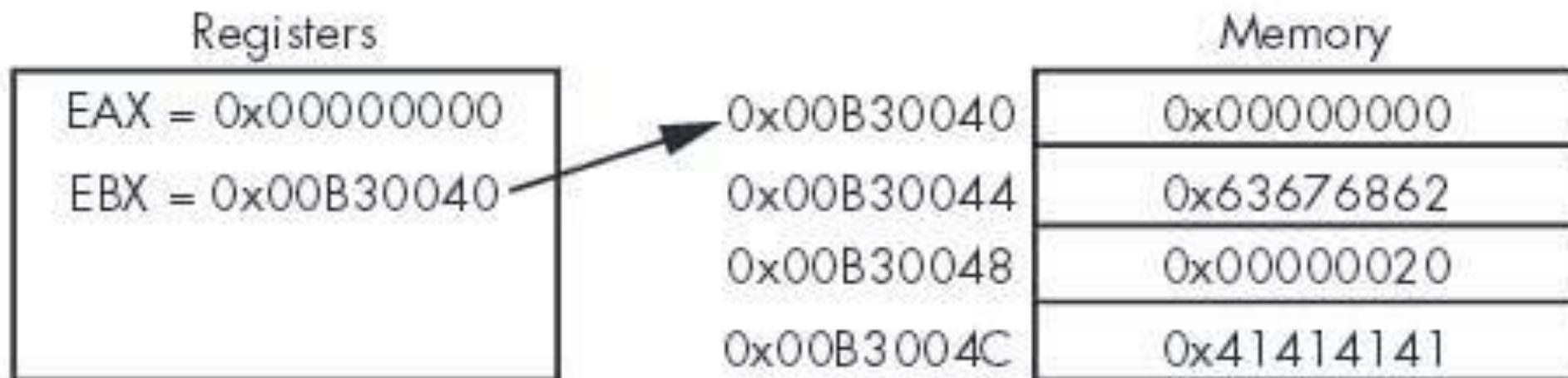
LEA (Load Effective Address)

LEA EAX, [EBX+8]

- So sánh MOV EAX, [EBX+8] và LEA EAX, [EBX+8]
 - MOV: chuyển giá trị tại địa chỉ EBX + 8 vào EAX
 - LEA: Chuyển địa chỉ mà EBX đang giữ + 8 vào EAX

LEA (Load Effective Address)

- Các giá trị của thanh ghi EAX và EBX ở bên trái, dữ liệu được lưu trữ trong bộ nhớ ở bên phải. EBX được set giá trị **0xb30040**. Ở địa chỉ **0xb30048** là giá trị **0x20**. Lệnh **mov eax, [ebx+8]** sẽ truyền giá trị **0x20** (lấy từ bộ nhớ) vào thanh ghi EAX, còn lệnh **lea eax, [ebx+8]** sẽ set giá trị **0xb30048** trên thanh ghi EAX.



Các phép toán đại số

- SUB:** Subtracts
- ADD:** Adds
- INC:** Increments
- DEC:** Decremens
- MUL:** Multiplies
- DIV:** Divides

NOP

- ❑ Lệnh đặc biệt, nó không làm gì cả
- ❑ Có mã opcode là 90
- ❑ Thường sử dụng NOP như để “trượt” qua

So sánh

☐ TEST

- So sánh hai toán hạng mà không làm thay đổi chúng
- TEST EAX, EAX => Kiểm tra EAX có bằng 0 hay không, nếu bằng 0 thì cờ ZF được bật

☐ CMP EAX, EBX

Lệnh này thực hiện việc so sánh hai toán hạng bằng cách lấy toán hạng đích – toán hạng nguồn

- Nếu đích = nguồn => CF = 0, ZF = 1, SF = 0
- Nếu đích > nguồn => CF = 0, ZF = 0, SF = 0
- Nếu đích < nguồn => CF = 1, ZF = 0, SF = 1

Rẽ nhánh

- JZ loc: Nhảy đến nhãn loc nếu cờ ZF được set (= 1)
- JNZ loc: Nhảy đến nhãn loc nếu cờ ZF không được set (= 0)

The Stack

- Vùng nhớ cho các biến cục bộ, tham số của hàm, theo dõi luồng điều khiển.
- Hoạt động theo nguyên lý “Vào sau ra trước - LIFO”
- Thanh ghi ESP (Extended Stack Pointer): Luôn trỏ vào đỉnh Stack
- Thanh ghi EBP (Extended Base Pointer): Thanh ghi con trỏ cơ sở, đáy Stack
- Lệnh PUSH: Đẩy dữ liệu vào ngăn xếp
- Lệnh POP: Lấy dữ liệu ra khỏi ngăn xếp

Function Calls

- Việc tổ chức chương trình theo các hàm thuận tiện cho việc tái sử dụng trong một chương trình.
- Việc gọi hàm có hai cơ chế là **Prologue** và **Epilogue**.
- Prologue hay Epilogue: mô tả quá trình gọi hàm - cần chuẩn bị Stack như thế nào và các thanh ghi làm việc ra sao

Function Prologue

- Lưu thanh ghi con trỏ cơ sở (EBP) vào ngăn xếp. Sau khi thực hiện xong và quay trở lại hàm trước đó thì phải trả lại EBP ban đầu.
- Cập nhập giá trị mới cho EBP là bằng giá trị của ESP.
- Dịch chuyển ESP một khoảng phù hợp dành cho các biến cục bộ của hàm.

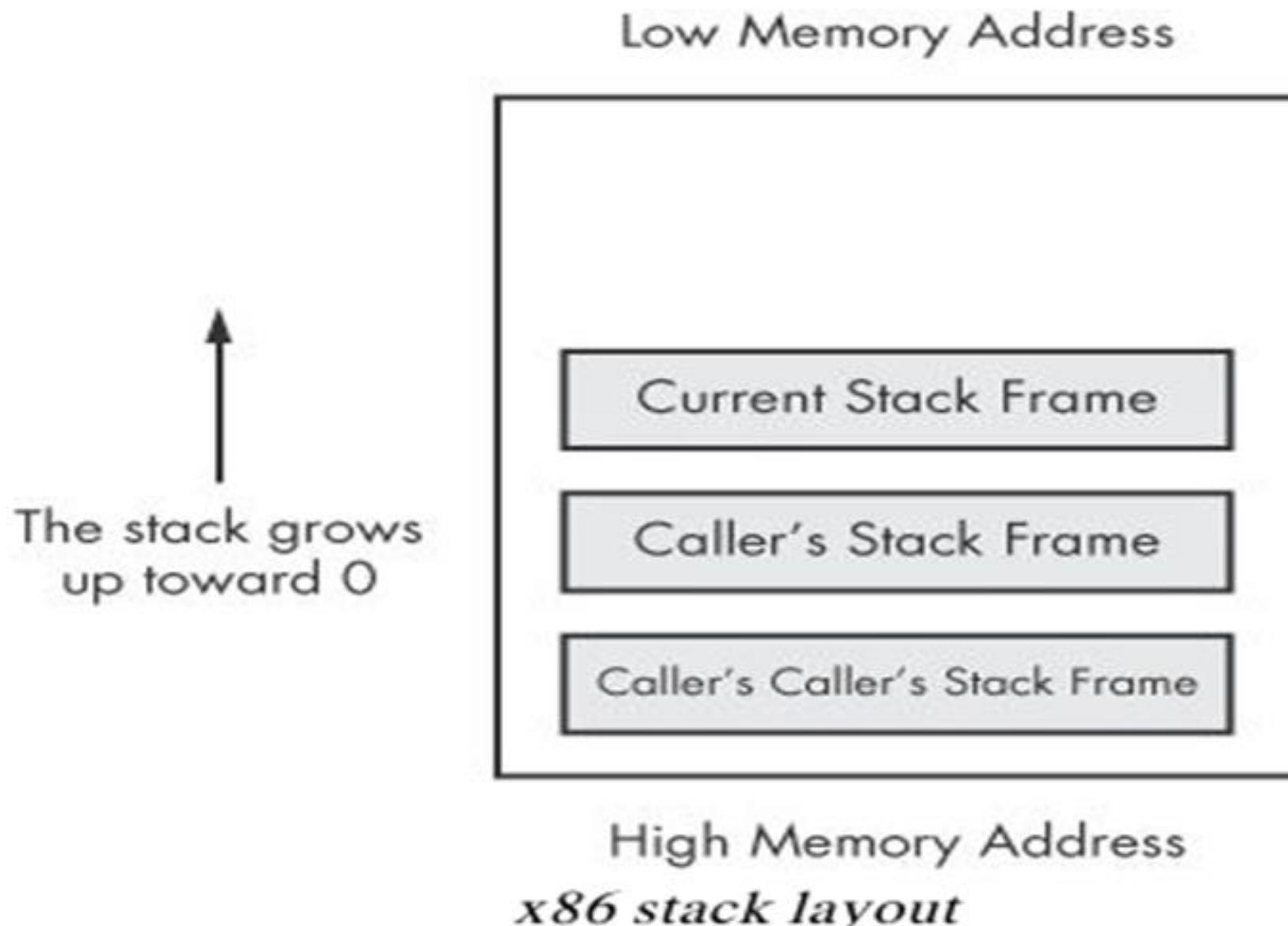
```
push ebp  
mov ebp, esp  
sub esp, N
```

Function Epilogue

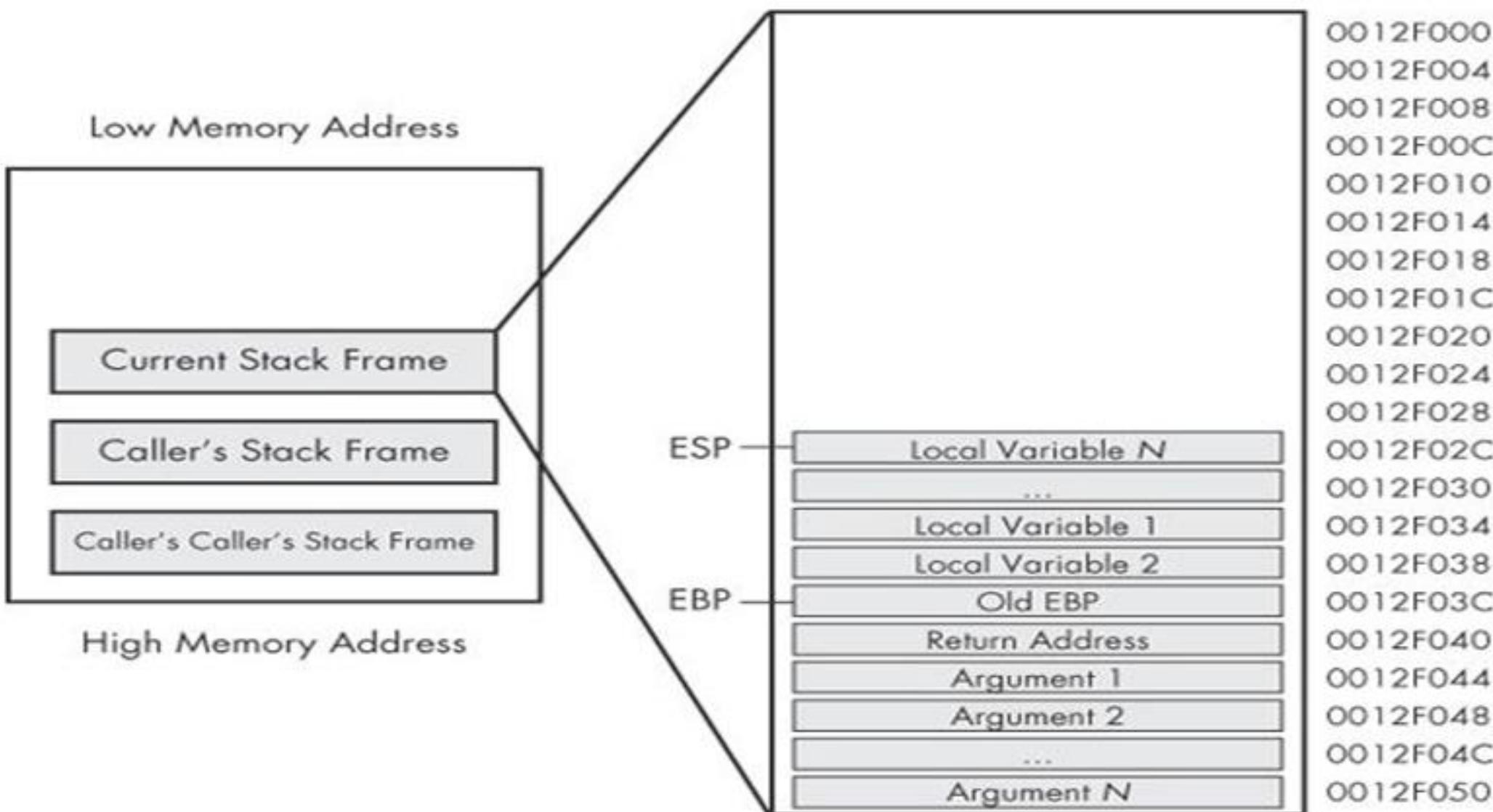
- ☐ Các lệnh ở cuối hàm sẽ khôi phục lại Stack và thanh ghi ở trạng thái trước khi hàm được gọi.

```
mov esp, ebp  
pop ebp  
ret
```

Function Calls



Function Calls



Individual stack frame

Function Calls

Danh sách sau tóm tắt luồng thực hiện của lời gọi hàm trong hầu hết trường hợp:

1. Các tham số đầu vào được đưa vào stack bằng lệnh **push**.
2. Một hàm được gọi bởi call **memory_location**. Địa chỉ lệnh hiện tại (giá trị của thanh ghi EIP) sẽ được push vào stack. Địa chỉ này sẽ được dùng để trở về chương trình chính khi hàm đã hoàn tất thực thi. Khi hàm bắt đầu, EIP được set giá trị **memory_location** (điểm bắt đầu của hàm).

Function Calls

3. Qua mở đầu hàm (prologue), không gian được chỉ định trên stack cho các biến cục bộ và EBP được push vào stack.
4. Hàm bắt đầu thực hiện chức năng của mình.
5. Qua kết thúc hàm (epilogue), stack được khôi phục về trạng thái ban đầu. ESP được điều chỉnh giá trị để giải phóng các biến cục bộ. EBP được khôi phục và hàm gọi đến có thể đánh đúng địa chỉ cho các biến của nó. Lệnh **leave** có thể được dùng như một kết thúc hàm vì nó set ESP bằng EBP và pop EBP khỏi stack.

Function Calls

6. Hàm trả về giá trị bằng lệnh **ret**. Lệnh này pop địa chỉ trả về khỏi stack và truyền giá trị đó vào EIP, chương trình sẽ tiếp tục thực thi từ đoạn lời gọi ban đầu được gọi.
7. Stack được điều chỉnh để xóa các tham số đã được gửi đi trừ khi các tham số đó sẽ được sử dụng lại về sau.

Nội dung

1. Nhắc lại về Assembly
2. Sử dụng IDA pro để dịch ngược mã độc
3. Sử dụng đối sánh chéo
4. Phân tích hàm
5. Sử dụng biểu đồ hàm
6. Một số lưu ý

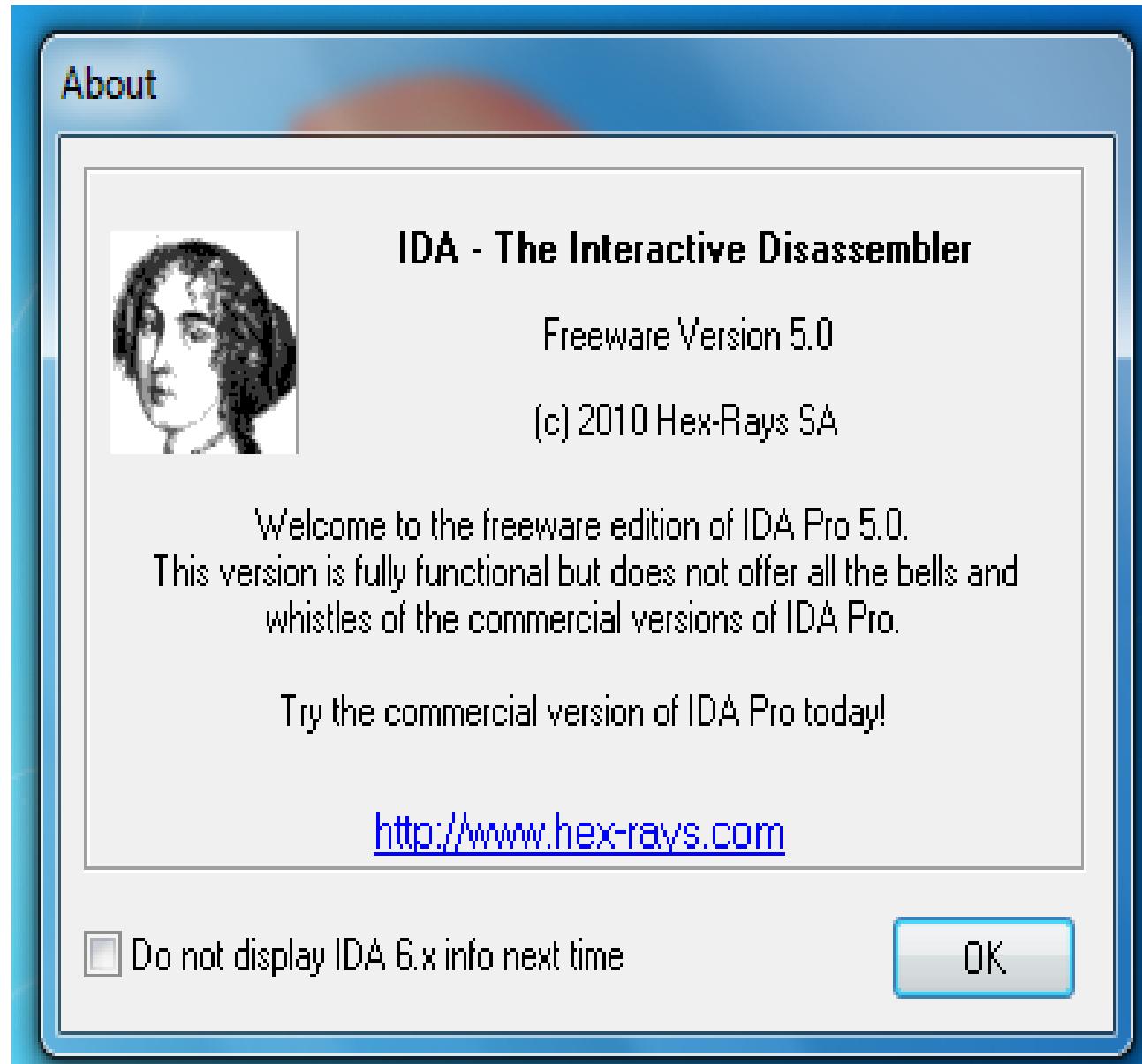
Sử dụng IDA pro để dịch ngược mã độc

- IDA Pro
- Một số tính năng hữu ích

Sử dụng IDA pro để dịch ngược mã độc

- IDA Pro**
- Một số tính năng hữu ích**

IDA pro



IDA Pro Versions

- Phiên bản thương mại sẽ đầy đủ tính năng hơn
- Phiên bản miễn phí thì hạn chế tính năng hơn
 - Cả hai phiên bản sẽ đều hỗ trợ kiến trúc x86
 - Với phiên bản thương mại thì hỗ trợ kiến trúc x64 và nhiều bộ vi xử lý khác nữa, ví dụ như với các dòng vi xử lý trên thiết bị di động.
- Cả hai phiên bản đều có những mẫu nhận dạng thư viện và những công nghệ nhận dạng giúp cho quá trình

Disassembly

Graph and Text Mode

☐ Sử dụng phím cách để
chuyển qua lại giữa hai
chế độ hiện thị

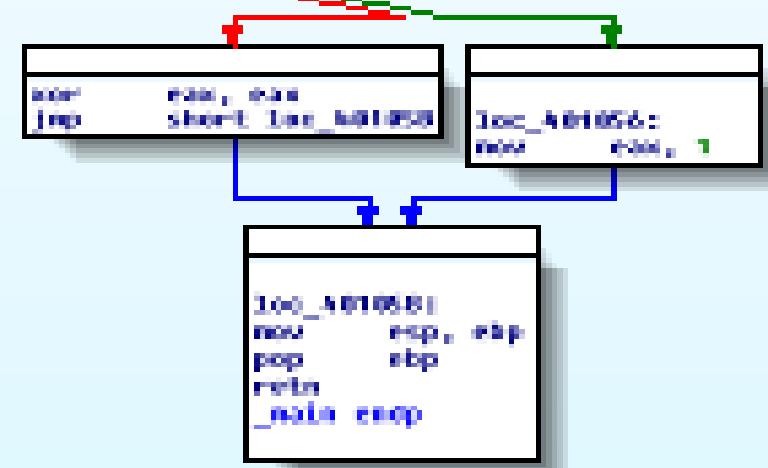
IDA View-A

```
.text:00401040
.text:00401040 ; Attributes: bp-based frame
.text:00401040
.text:00401040 ; int _cdecl main(int argc,const char **argv,const char *envp)
.text:00401040 _main           proc near             ; CODE XREF: start+AF↓p
.text:00401040
.text:00401040     var_4          = dword ptr -4
.text:00401040     argc           = dword ptr 8
.text:00401040     argv           = dword ptr 0Ch
.text:00401040     envp           = dword ptr 10h
.text:00401040
.text:00401040     push    ebp
.text:00401041     mov     ebp, esp
.text:00401043     push    ecx
```

```
; Attributes: bp-based frame
; int _cdecl main(int argc,const char **argv,const char *envp)
_main proc near

var_4 = dword ptr -4
argc = dword ptr 8
argv = dword ptr 0Ch
envp = dword ptr 10h

push    ebp
mov    ebp, esp
push    ebx
call    sub_401000
mov    [ebp+var_4], ebx
cmp    [ebp+var_4], 0
jne    short loc_401000
loc_401000:
        short loc_401000
```



Default Graph Mode Display

```
NUL

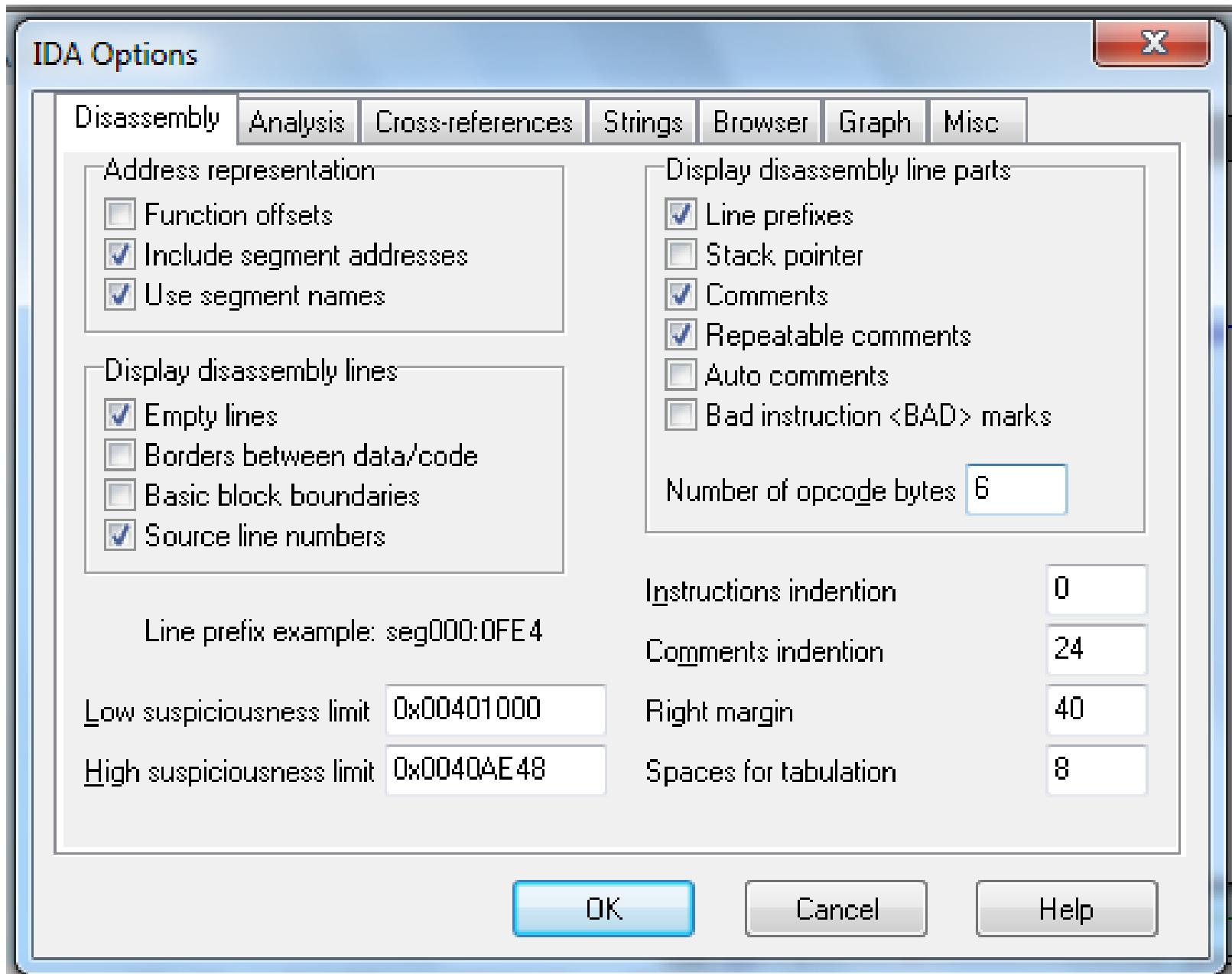
; Attributes: bp-based frame

; int __cdecl main(int argc,const char **argv,const char *envp)
_main proc near

var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
push    ecx
call    sub_401000
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jnz    short loc_401056
```

Graph Mode Options



Arrows

Các kiểu mũi tên trong chế độ Graph mode:

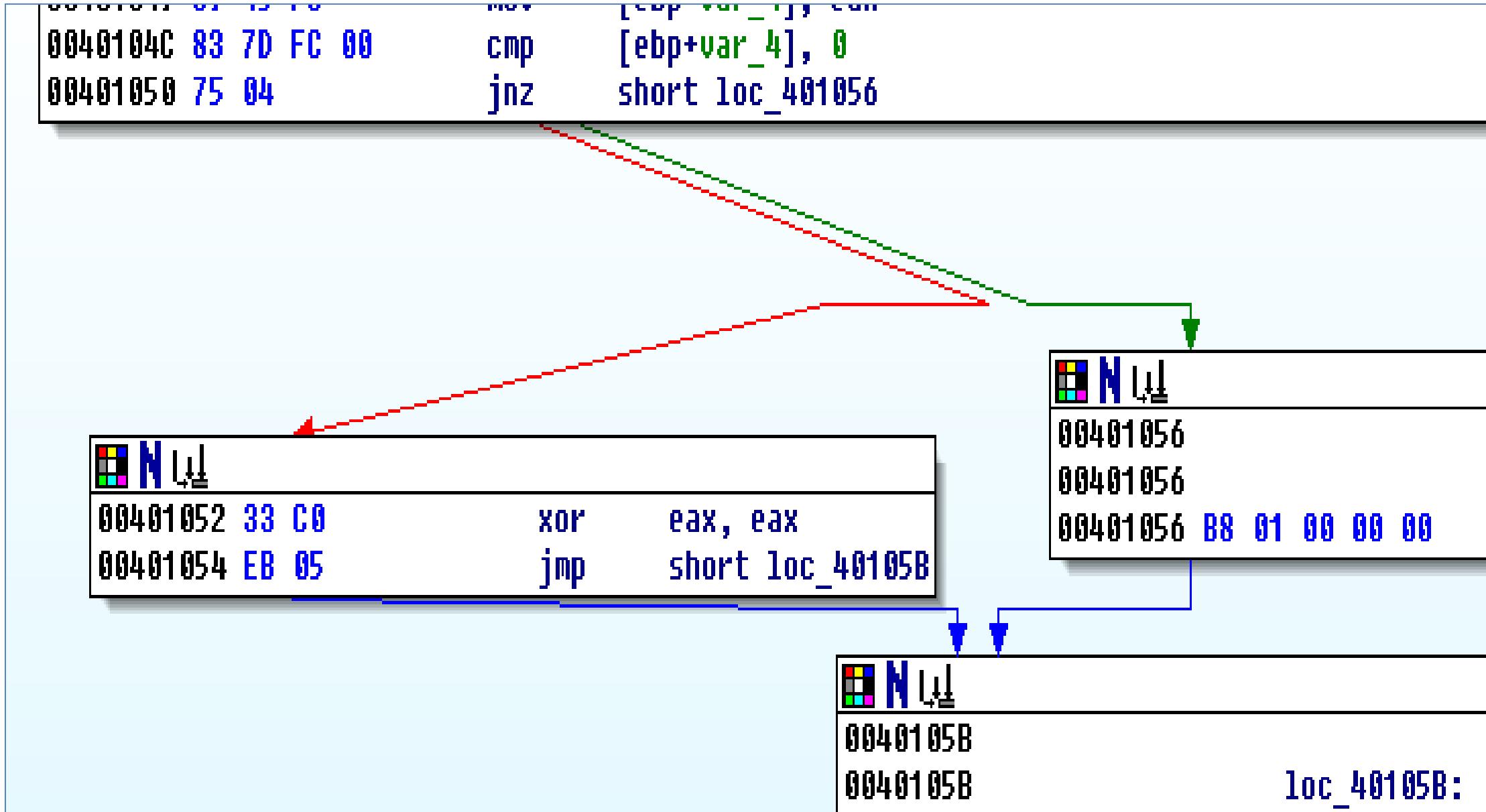
□ Màu

- Đỏ (Red): Nhảy có điều kiện nhưng chưa thực hiện,
- Xanh lá cây (Green): Nhảy có điều kiện và được thực hiện,
- Xanh dương (Blue): Nhảy không điều kiện.

□ Hướng

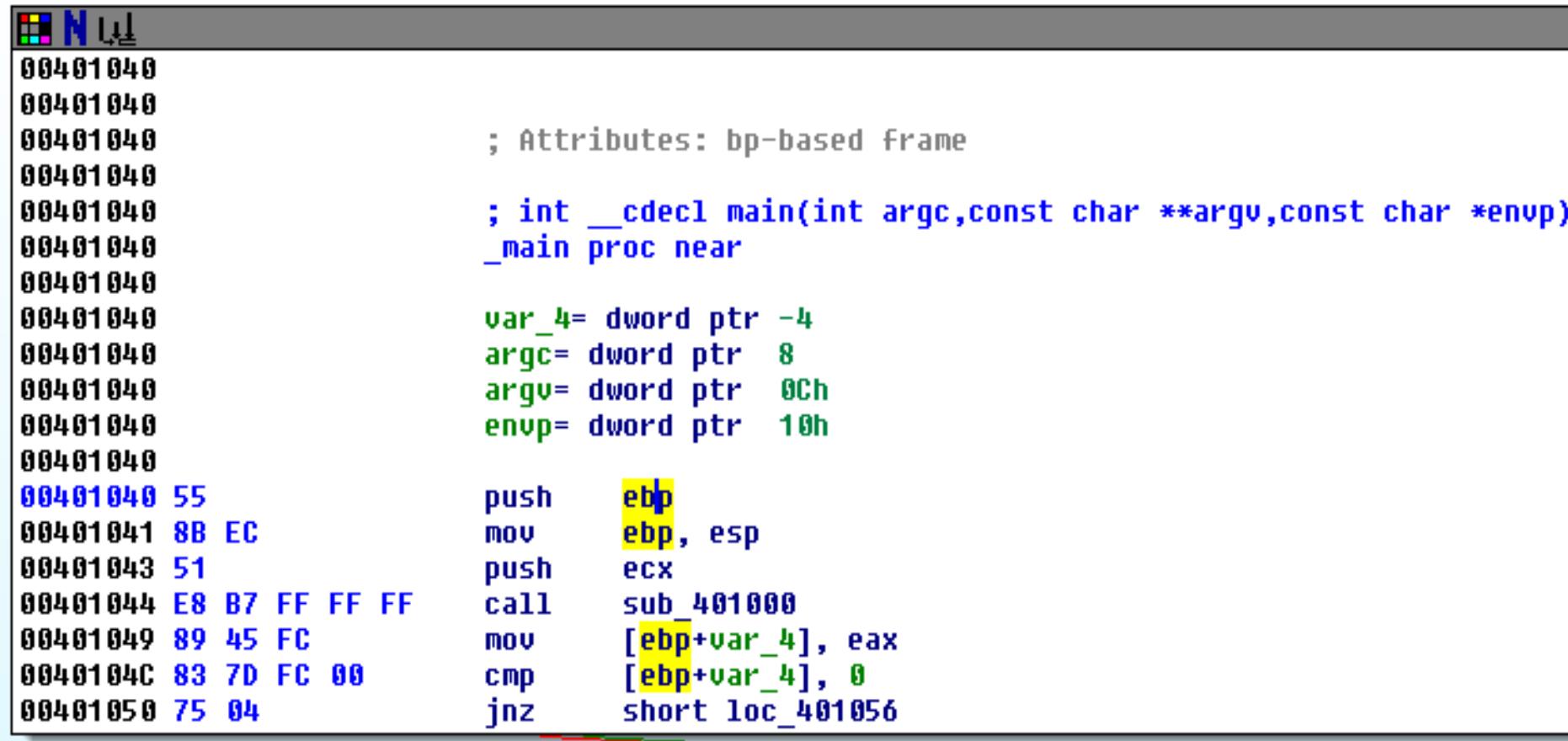
- Lên: Biểu diễn đây là một vòng lặp.

Arrow Color Example



Highlighting

- Đánh dấu tên một biến, thanh ghi... Trong chế độ Graph mode làm nổi bật những biến, thanh ghi giống nhau.



The screenshot shows a debugger interface with assembly code. Variable names are highlighted in blue, and memory addresses are in green. The assembly code is:

```
00401040 ; Attributes: bp-based frame
00401040 ; int __cdecl main(int argc,const char **argv,const char *envp)
00401040 _main proc near
00401040     var_4= dword ptr -4
00401040     argc= dword ptr  8
00401040     argv= dword ptr  0Ch
00401040     envp= dword ptr  10h
00401040
00401040 55     push    ebp
00401041 8B EC     mov     ebp, esp
00401043 51     push    ecx
00401044 E8 B7 FF FF FF     call    sub_401000
00401049 89 45 FC     mov     [ebp+var_4], eax
0040104C 83 7D FC 00     cmp     [ebp+var_4], 0
00401050 75 04     jnz    short loc_401056
```

Text Mode

Mũi tên nét liền = Nhảy không có điều kiện

Mũi tên nét đứt = Nhảy có điều kiện

Mũi tên đi lên = Vòng lặp

Những comment
được sinh ra bởi IDA Pro

Section

Address

```
* .text:00401015          jz    short loc_40102B
* .text:00401017          push   offset aSuccessInterne ; "Success: Internet Connection\n"
* .text:0040101C          call   sub_40105F
* .text:00401021          add    esp, 4
* .text:00401024          mov    eax, 1
* .text:00401029          jmp    short loc_40103A
.text:0040102B ; -----
.text:0040102B loc_40102B:
* .text:0040102B          push   offset aError1_1NoInte ; "Error 1.1: No Internet\n"
* .text:00401030          call   sub_40105F
* .text:00401035          add    esp, 4
* .text:00401038          xor    eax, eax
.text:0040103A loc_40103A:
* .text:0040103A          mov    esp, ebp
* .text:0040103C          pop    ebp
```

Thêm nhận xét cho mỗi lệnh

```
; .text:00401015          jz    short loc_40102B ; Jump if Zero (ZF=1)
; .text:00401017          push   offset aSuccessInterne ; "Success: Internet Connection\n"
; .text:0040101C          call   sub_40105F      ; Call Procedure
; .text:00401021          add    esp, 4        ; Add
; .text:00401024          mov    eax, 1        ; Add
; .text:00401029          jmp    short loc_40103A ; Jump
.text:0040102B ; -----
.text:0040102B
.text:0040102B loc_40102B:           ; CODE XREF: sub_401000+15↑j
; .text:0040102B          push   offset aError1_1NoInte ; "Error 1.1: No Internet\n"
; .text:00401030          call   sub_40105F      ; Call Procedure
; .text:00401035          add    esp, 4        ; Add
; .text:00401038          |     xor    eax, eax    ; Logical Exclusive OR
.text:0040103A
.text:0040103A loc_40103A:           ; CODE XREF: sub_401000+29↑j
; .text:0040103A          mov    esp, ebp
; .text:0040103C          pop    ebp
```

Sử dụng IDA pro để dịch ngược mã độc

- IDA Pro
- Một số tính năng hữu ích

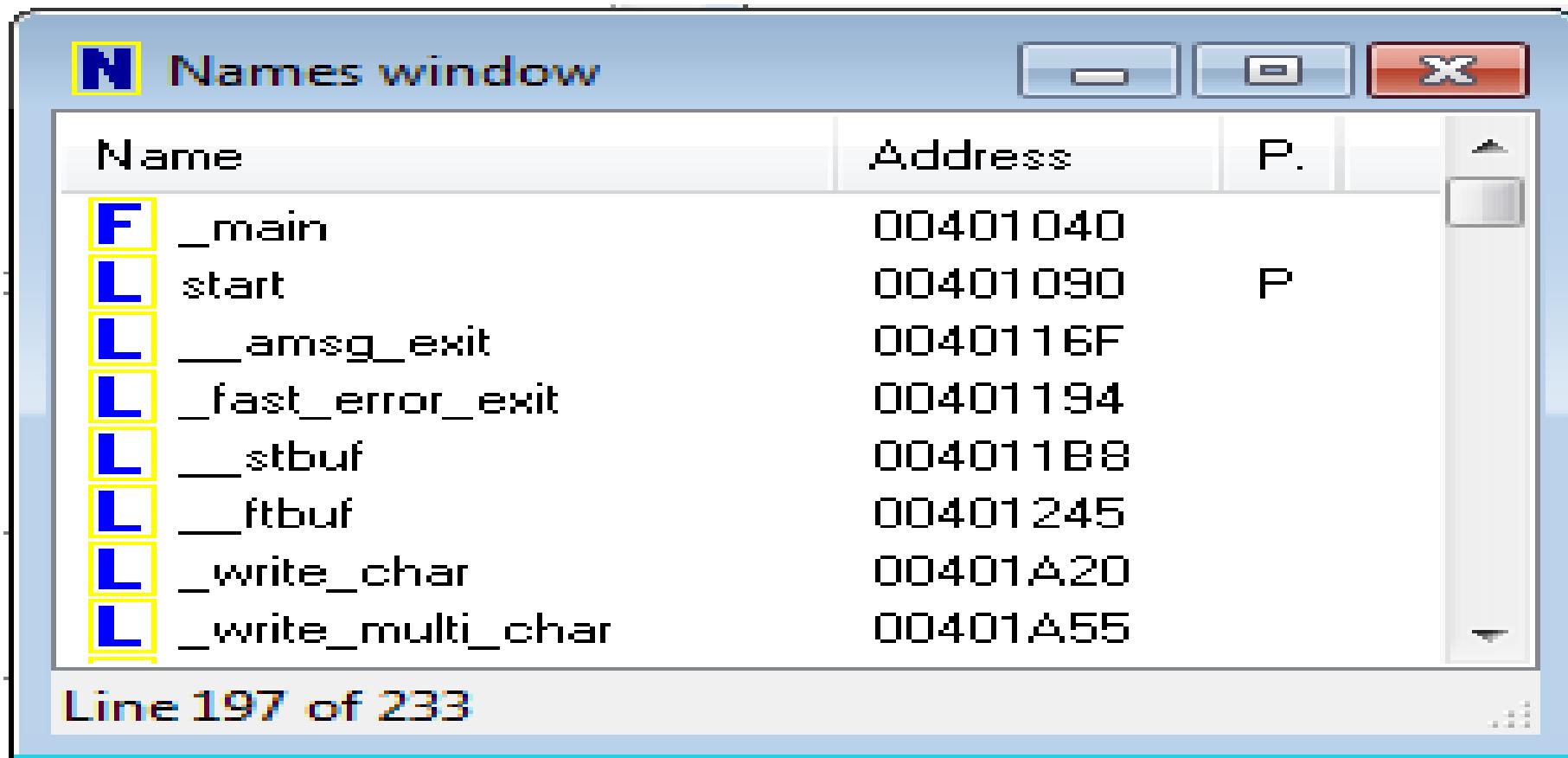
Functions

- ❑ Cửa sổ này hiển thị các hàm, độ dài và các cờ
L = Library function
- ❑ Windows > Function để hiển thị cửa sổ này

Function name	Segment	Start	Length	R	F	L	S	B	T	=
CheckWindowsGenuineStatus()	.text	010091F9	0000007C	R	.	.	.	B	.	.
ControlBackgroundBrushInfo::scalar deletingtext	01032897	00000029	R	.	.	.	B	.	.
CreateDecoderFromResource(WICImagingF...	.text	0101FB50	00000097	R	.	.	.	B	T	.

Names Window

- ☐ Cửa sổ này hiển thị tên các hàm, dữ liệu, chuỗi đi kèm với địa chỉ của chúng.

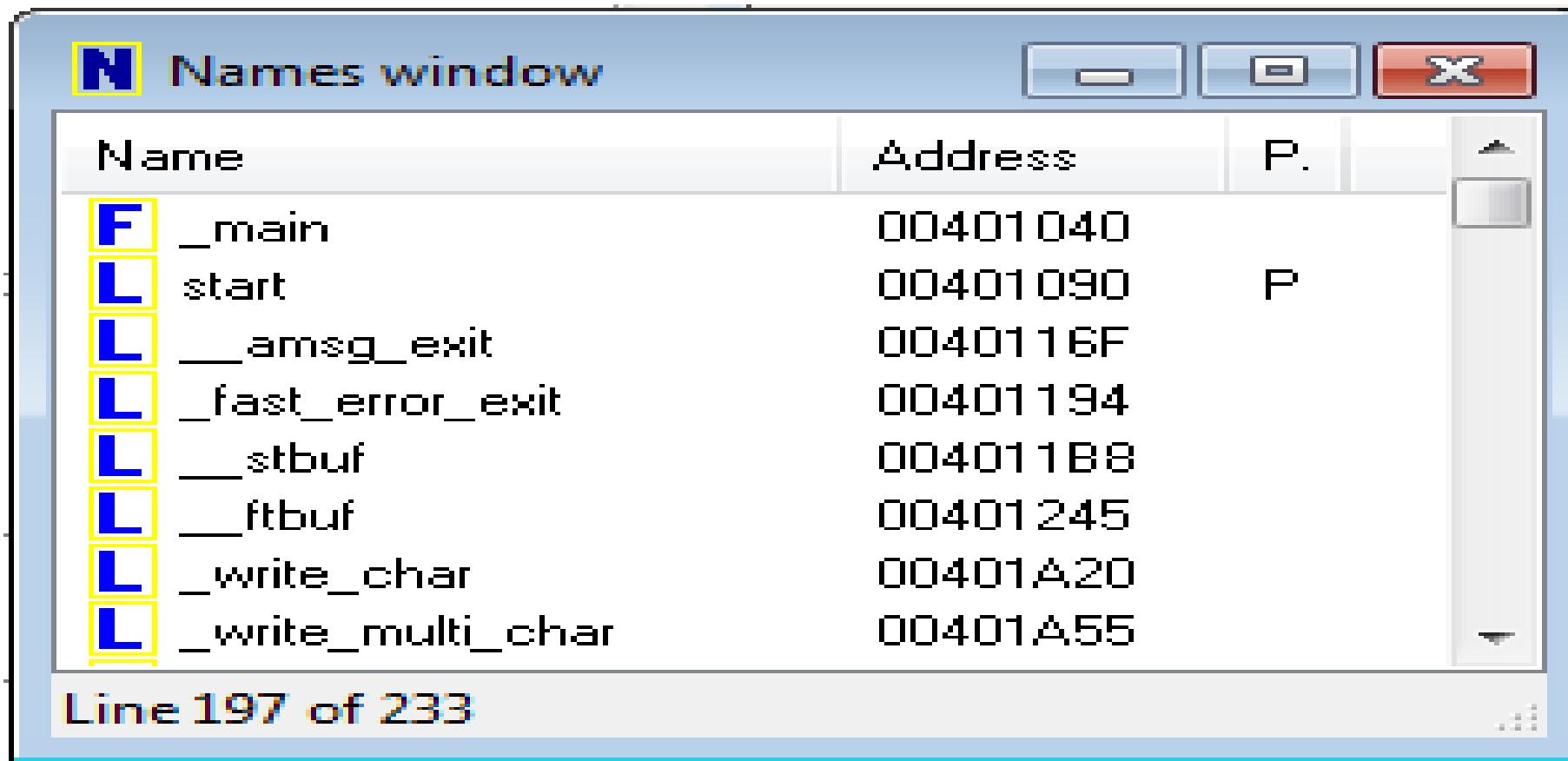


The screenshot shows a software window titled "Names window". The window has a standard title bar with minimize, maximize, and close buttons. The main area is a table with three columns: "Name", "Address", and "P.". The "Name" column contains symbols with blue "L" icons: "_main", "start", "__amsg_exit", "__fast_error_exit", "__stbuf", "__ftbuf", "__write_char", and "__write_multi_char". The "Address" column lists memory addresses: "00401040", "00401090", "0040116F", "00401194", "004011B8", "00401245", "00401A20", and "00401A55". The "P." column contains a single letter "P" next to the address of "start". At the bottom of the window, a status bar displays "Line 197 of 233".

Name	Address	P.
F _main	00401040	
L start	00401090	P
L __amsg_exit	0040116F	
L __fast_error_exit	00401194	
L __stbuf	004011B8	
L __ftbuf	00401245	
L __write_char	00401A20	
L __write_multi_char	00401A55	

Strings

- Cửa sổ này liệt kê các chuỗi trong đoạn .rdata của binary.



The screenshot shows a window titled "Names window" with a list of symbols and their addresses. The symbols are categorized by color: blue for function names and red for other symbols. The table has columns for Name, Address, and P.

Name	Address	P.
F _main	00401040	
L start	00401090	P
L __amsg_exit	0040116F	
L __fast_error_exit	00401194	
L __stbuf	004011B8	
L __ftbuf	00401245	
L __write_char	00401A20	
L __write_multi_char	00401A55	

Line 197 of 233

Imports & Exports

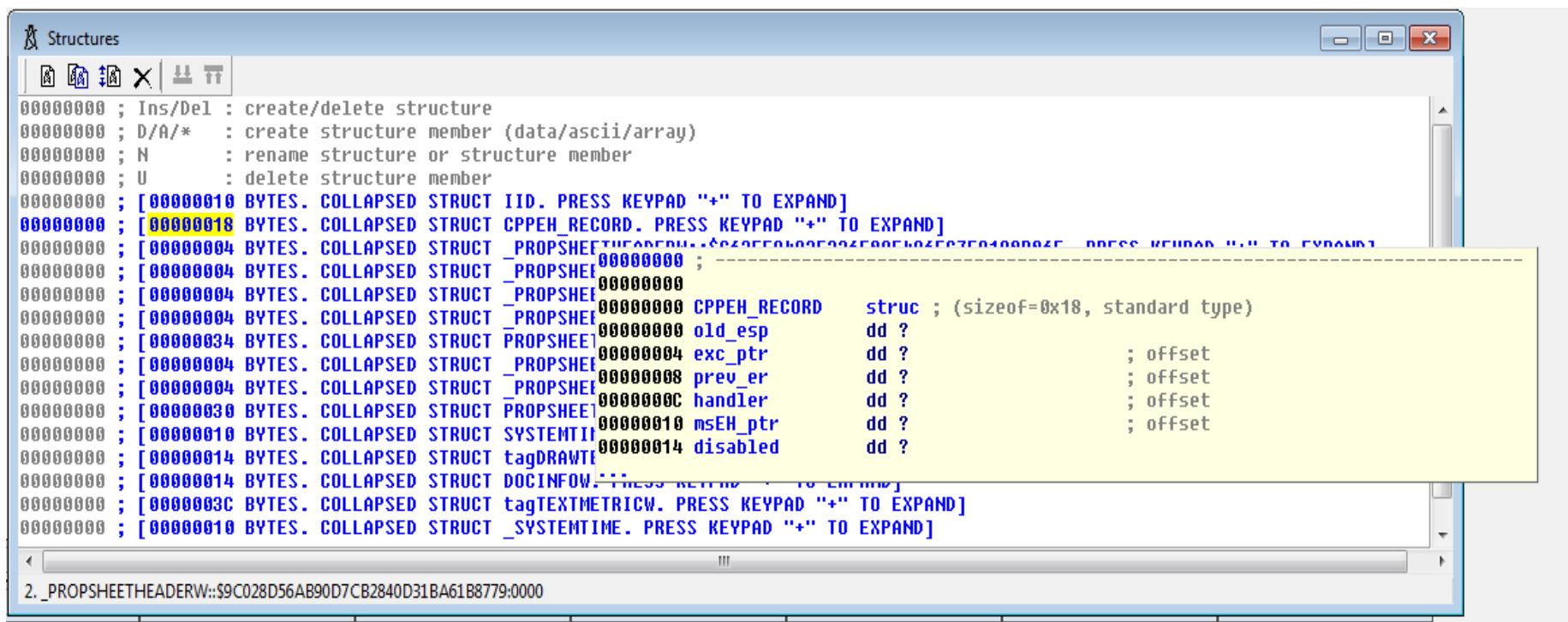
Address	Ordinal	Name	Library
004060A0		GetStringTypeA	KERNEL32
004060A4		GetStringTypeW	KERNEL32
004060A8		SetStdHandle	KERNEL32
004060B0		InternetGetConnectedState	WININET

Line 1 of 44

Name	Address	Ordinal
start	00401090	

Structures

- Hiển thị tất cả dữ liệu cấu trúc**
 - Di chuột vào để hiển thị một Pop-up màu vàng**



Function Call

- ❑ Các tham số được đẩy vào Stack
- ❑ Lệnh Call hiểu là sẽ gọi một thủ tục/hàm

The screenshot shows a debugger interface with assembly code. The assembly code is:

```
0100478B ; Attributes: bp-based frame
0100478B ; int __stdcall RegWriteString(HKEY hKey,LPCWSTR lpValueName,BYTE *lpData)
0100478B _RegWriteString@12 proc near
0100478B
0100478B     hKey= dword ptr  8
0100478B     lpValueName= dword ptr  0Ch
0100478B     lpData= dword ptr  10h
0100478B
0100478B 8B FF      mov    edi, edi
0100478D 55          push   ebp
0100478E 8B EC      mov    ebp, esp
01004790 FF 75 10    push   [ebp+lpData]    ; lpString
01004793 FF 15 10 11 00 01 call   ds:_imp__lstrlenW@4 ; lstrlenW(x)
```

The status bar at the bottom shows: 0.00% | (-30,-41) | (788,342) | 00003B8B | 0100478B: RegWriteString(x,x,x)

Imports or Strings

- ☐ Nhấn đúp chuột vào bất kỳ entry nào để hiển thị cửa sổ Disassembly.

The screenshot shows the IDA View-A interface. On the left, the assembly code is displayed:

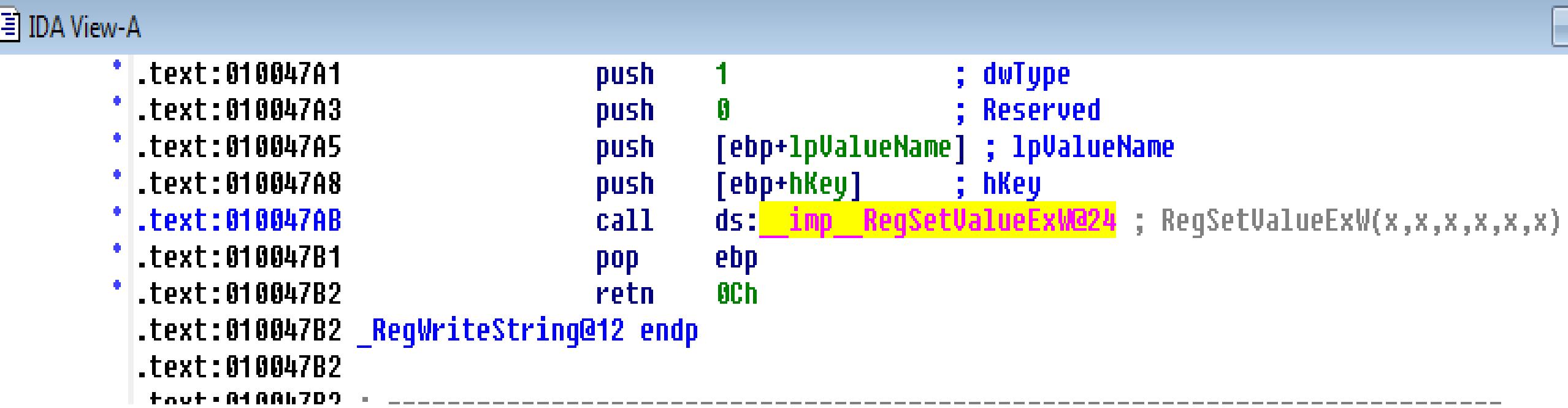
```
text:0100A779          jb    short near ptr loc_100A7DF+1
text:0100A77B          add   gs:[eax], al
text:0100A77E          retf
text:0100A77E ;-----+
text:0100A77F          db 2
text:0100A780 aHeapalloc db 'HeapAlloc',0
text:0100A78A          dw 24Ah
text:0100A78C aGetprocessheap db 'GetProcessHeap',0
text:0100A79B          align 4
text:0100A79C          db 0ECh ; 8
text:0100A79D          db 1, 47h, 65h
text:0100A7A0 aTfileinformati db 'tFileInformationByHandle',0
text:0100A7B9          align 2
```

On the right, the Strings window lists the strings found in the code:

Address	Length	Type	String
... .text:01...	0000000C	C	TextUnicode
... .text:01...	00000013	C	CloseServiceHandle
... .text:01...	00000014	C	QueryServiceConfig
... .text:01...	00000019	C	GetUserDefaultUIL
... .text:01...	00000004	C	HeapAlloc
... .text:01...	0000000F	C	GetProcessHeap
... .text:01...	00000019	C	tFileInformationByH

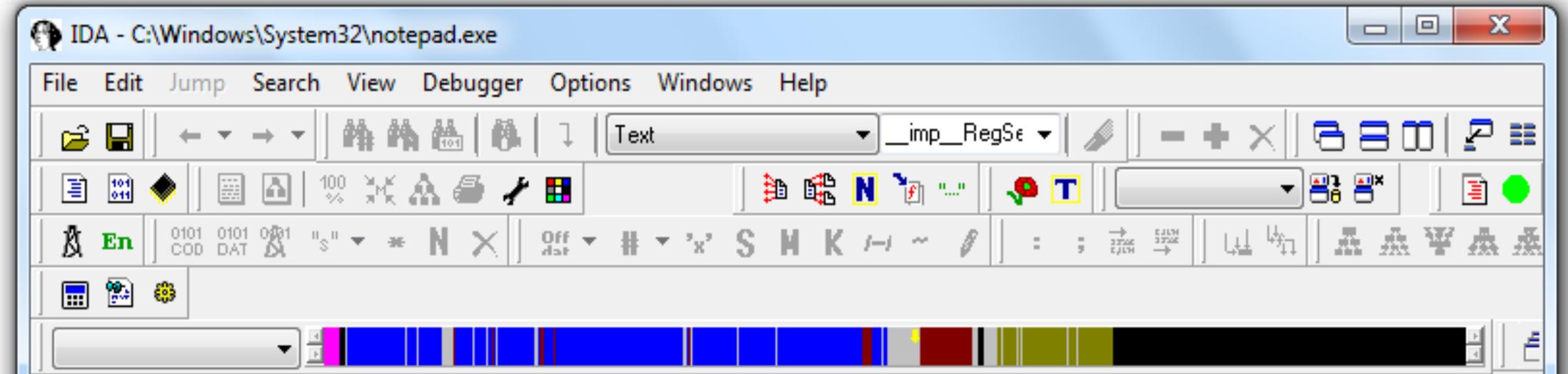
Using Links

- Nhấn đúp chuột vào bất kỳ địa chỉ nào trong cửa sổ Disassembly để hiển thị vị trí đó.



```
IDA View-A
.text:010047A1      push    1          ; dwType
.text:010047A3      push    0          ; Reserved
.text:010047A5      push    [ebp+lpValueName] ; lpValueName
.text:010047A8      push    [ebp+hKey]    ; hKey
.text:010047AB      call    ds:_imp_RegSetValueExW@24 ; RegSetValueExW(x,x,x,x,x,x)
.text:010047B1      pop     ebp
.text:010047B2      retn    0Ch
.text:010047B2 _RegWriteString@12 endp
.text:010047B2
� out - 010047D0 - -----
```

Dải điều hướng

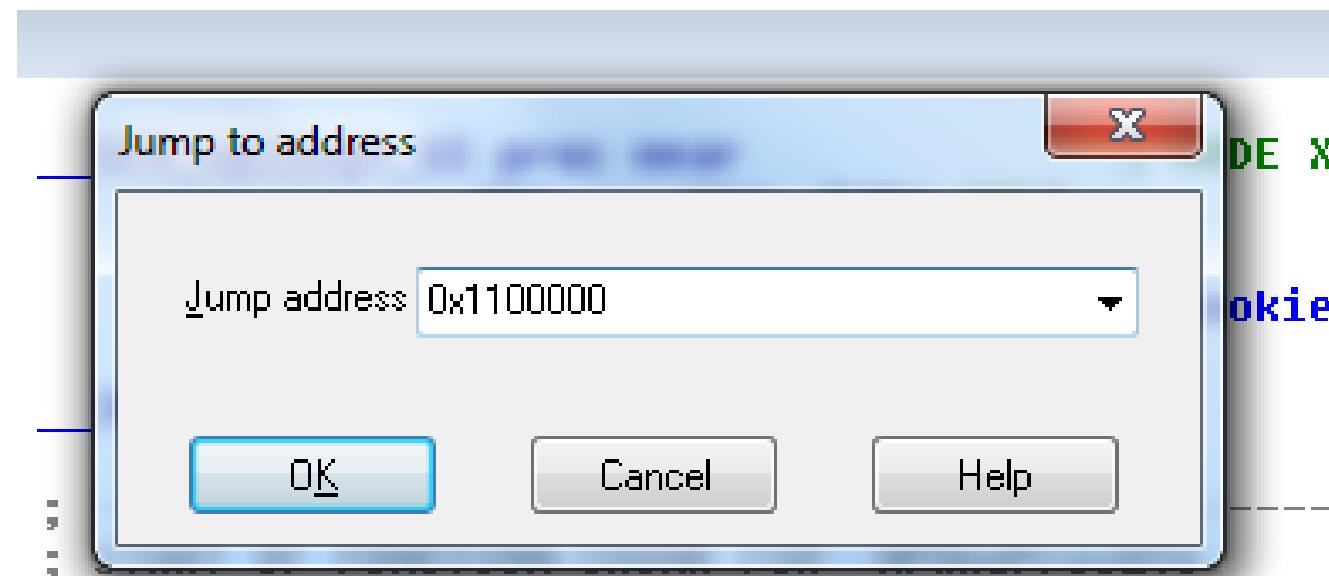


Thanh điều hướng hiển thị dải màu này rất hữu dụng, nó cho chúng ta biết được nên phân tích vùng nào:

- Light Blue** (Xanh nhạt): Vùng code của thư viện
- Red** (đỏ): Vùng code mà trình biên dịch sinh ra
- Dark Blue** (Xanh đậm): Vùng code của người dùng viết, đây là phần cần phân tích

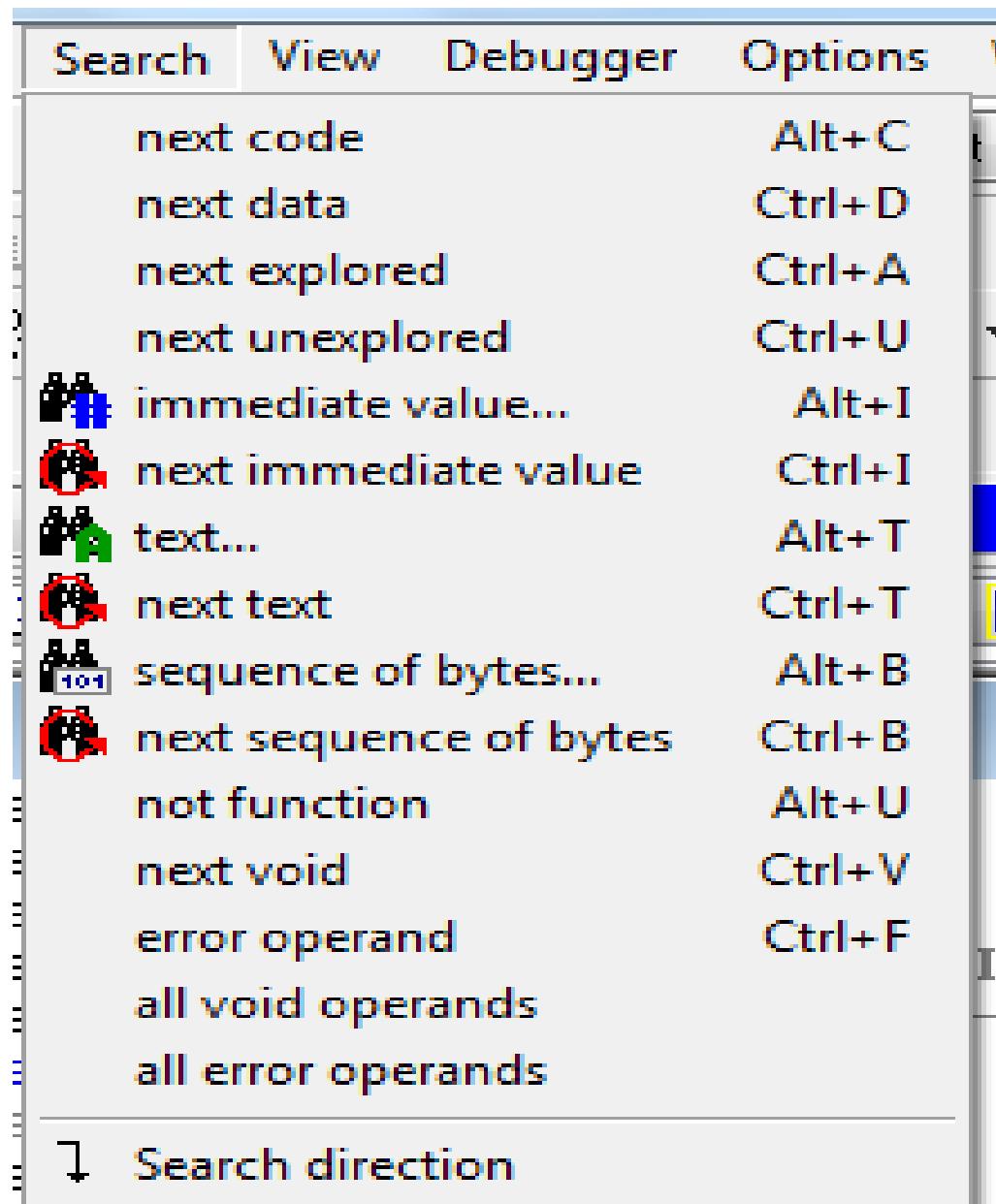
Nhảy tới địa chỉ

- ☐ Nhấn phím G và điền địa chỉ muốn nhảy đến vào trong khung nhập địa chỉ
- ☐ Nhấn Ok để chương trình nhảy đến vị trí đã điền



Tìm kiếm

- Tìm kiếm tên hàm
- Tìm kiếm tên biến
- Tìm kiếm địa chỉ
- Tìm kiếm comment
- .vv.



Nội dung

1. Nhắc lại về Assembly
2. Sử dụng IDA pro để dịch ngược mã độc
3. Sử dụng đối sánh chéo
4. Phân tích hàm
5. Sử dụng biểu đồ hàm
6. Một số lưu ý

Nội dung

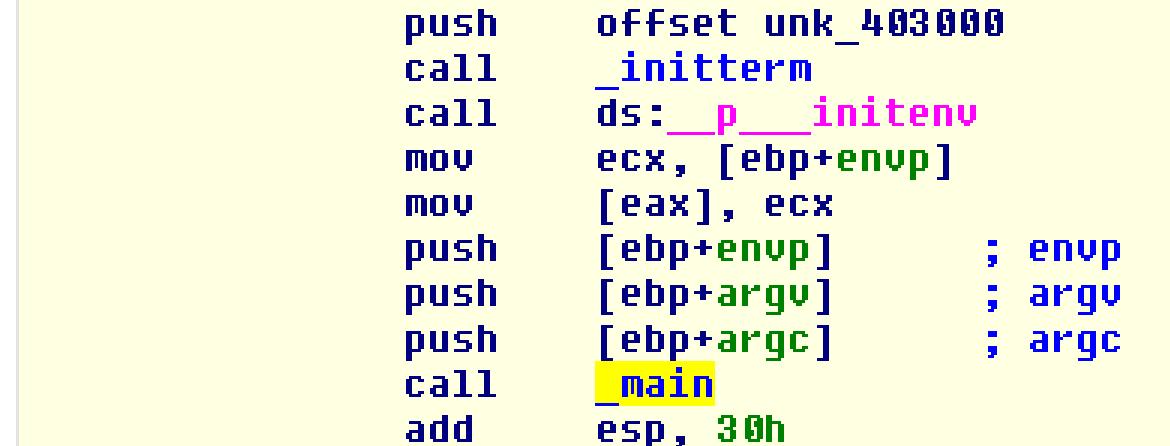
1. Nhắc lại về Assembly
2. Sử dụng IDA pro để dịch ngược mã độc
3. Sử dụng đối sánh chéo
4. Phân tích hàm
5. Sử dụng biểu đồ hàm
6. Một số lưu ý

Sử dụng đối sánh chéo

- ❑ Code Cross-References
- ❑ Data Cross-References

Code Cross-References

```
.text:00401440
.text:00401440 ; ┌───────── S U B R O U T I N E ─────────┐
.text:00401440
.text:00401440
.text:00401440 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:00401440 _main          proc near             ; CODE XREF: start+DE↓p
.text:00401440
.text:00401440 var_44         = dword ptr -44h
.text:00401440 var_40         = dword ptr -40h
.text:00401440 var_3C         = dword ptr -3Ch
.text:00401440 var_38         = dword ptr -38h
.text:00401440 var_34         = dword ptr -34h
.text:00401440 var_30         = dword ptr -30h
.text:00401440 var_2C         = dword ptr -2Ch
.text:00401440 var_28         = dword ptr -28h
.text:00401440 var_24         = dword ptr -24h
.text:00401440 var_20         = dword ptr -20h
.text:00401440 var_1C         = dword ptr -1Ch
.text:00401440 ..... 40       = dword ptr -40h
```



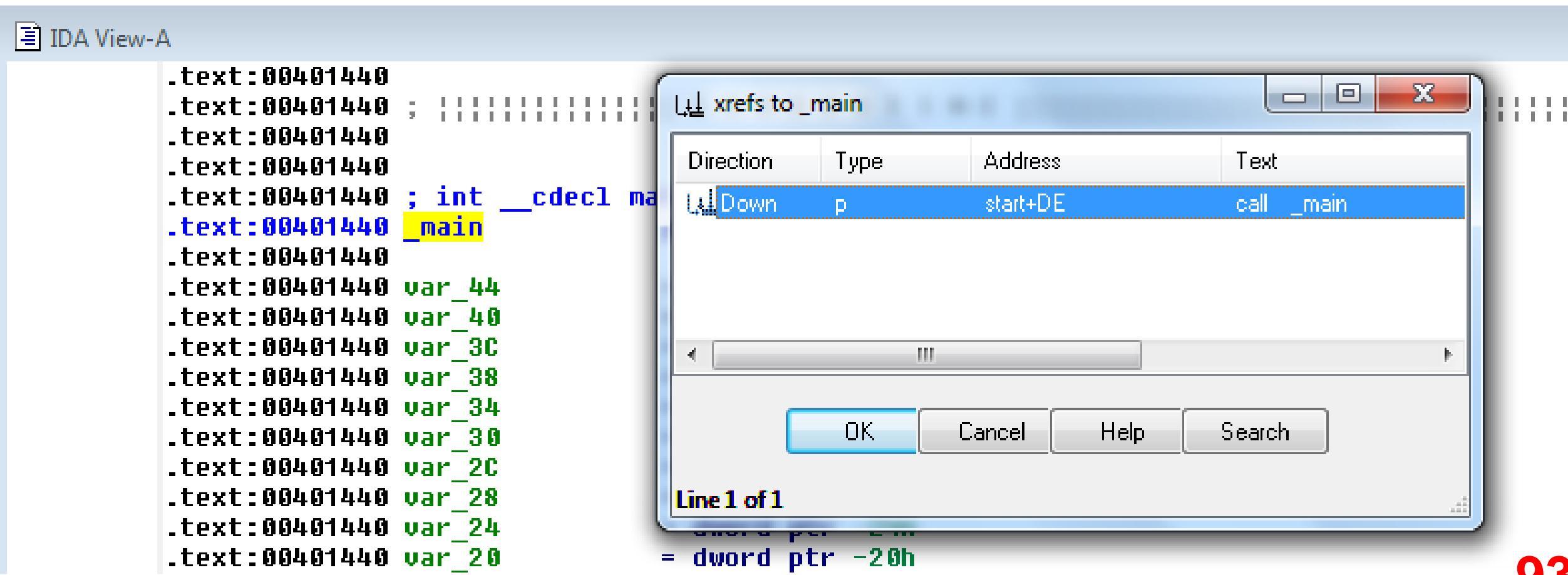
```
push    offset unk_403000
call   _initterm
call   ds:_p__initenv
mov    ecx, [ebp+envp]
mov    [eax], ecx
push   [ebp+envp]           ; envp
push   [ebp+argv]            ; argv
push   [ebp+argc]            ; argc
call   _main
add    esp, 30h
```

❑ XREF: Bình luận cho thấy hàm hiện tại đã được gọi từ đâu

❑ Chỉ hiện thị một vài tham chiếu chéo mặc định

Code Cross-References

□ Để xem tất cả Code Cross-References: Click vào một tên hàm và nhấn ‘X’



Data Cross-References

- Bắt đầu với chuỗi, nháy đúp chuột vào chuỗi
- Di chuột qua DATA XREF để xem chuỗi đó ở đâu sử dụng
- X hiển thị tất cả các tham chiếu

The screenshot shows a debugger interface with two main panes. The left pane displays assembly code and data definitions. The right pane shows the assembly instructions with their corresponding opcodes and comments.

Left Pane (Data Definitions):

```
* .data:0040304C ; char NewFileName[]  
.data:0040304C NewFileName db 'C:\windows\system32\kerne132.dll',0  
.data:0040304C ; DATA XREF: _main+3AA↑o  
* .data:0040306D align 10h  
* .data:00403070 dword_403070 dd 6E72654Bh  
* .data:00403074 dword_403074 dd 32336C65h  
* .data:00403078 byte_403078 db 2Eh  
* .data:00403079 align 4  
* .data:0040307C ; char ExistingFileName[]  
.data:0040307C ExistingFileName db 'Lab01-01.dll',0 ; D  
.data:0040307C ;  
* .data:00403089 align 4  
* .data:0040308C ; char FileName[]  
.data:0040308C FileName db 'C:\Windows\System32\Ker  
.data:0040308C ; DATA XREF: _main+67↑o
```

Right Pane (Assembly Instructions):

```
        mov    ecx, [esp+54h+hObject]  
        mov    esi, ds:CloseHandle  
        push   ecx          ; hObject  
        call   esi ; CloseHandle  
        mov    edx, [esp+54h+var_4]  
        push   edx          ; hObject  
        call   esi ; CloseHandle  
        push   0             ; bFailIfExists  
        push   offset NewFileName ; "C:\\windows\\system32\\kerne132.dll"  
        push   offset ExistingFileName ; "Lab01-01.dll"
```

Nội dung

- 1. Nhắc lại về Assembly**
- 2. Sử dụng IDA pro để dịch ngược mã độc**
- 3. Sử dụng đối sánh chéo**
- 4. Phân tích hàm**
- 5. Sử dụng biểu đồ hàm**
- 6. Một số lưu ý**

Phân tích hàm

- Hàm và tham số
- Biến toàn cục và biến cục bộ
- Các phép toán cơ bản

Phân tích hàm

- Hàm và tham số**
- Biến toàn cục và biến cục bộ**
- Các phép toán cơ bản**

Hàm và tham số

- IDA xác định các hàm, các tham số và đặt tên chúng
- Không phải luôn luôn đúng

IDA View-A

```
.text:00401040
.text:00401040
.text:00401040 sub_401040      proc near               ; CODE XREF: sub_4010A0+88↓p
.text:00401040
.text:00401040
.text:00401040
.text:00401040 arg_0          = dword ptr  4
.text:00401040 arg_4          = dword ptr  8
.text:00401040 arg_8          = dword ptr  0Ch
.text:00401040
• .text:00401040             mov     eax, [esp+arg_4]
• .text:00401044             push    esi
• .text:00401045             mov     esi, [esp+4+arg_0]
• .text:00401049             push    eax
```

Disassembly in IDA Pro

- ❑ Hàm printf() sẽ được xử lý theo thứ: tham số thứ n xử lý trước rồi đến n-1, n-2...

```
wmain proc near

var_C0= dword ptr -0C0h

push    ebp
mov     ebp, esp
sub     esp, 0C0h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_C0]
mov     ecx, 30h
mov     eax, 0CCCCCCCCCh
rep stosd
mov     esi, esp
push    3
push    2          printf("Hello! %d %d %d\n", 1, 2, 3);
push    1
push    offset aHelloDDD ; "Hello! %d %d %d\n"
call    ds:_imp__printf
add    esp, 10h
cmp    esi, esp
call    j__RTC_CheckEsp
xor    eax, eax
pop    edi
pop    esi
pop    ebx
add    esp, 0C0h
cmp    ebp, esp
call    j__RTC_CheckEsp
mov    esp, ebp
pop    ebp
retn
wmain endp
```

Phân tích hàm

- Hàm và tham số
- Biến toàn cục và biến cục bộ
- Các phép toán cơ bản

Biến toàn cục và biến cục bộ

Biến toàn cục (Global Variable)

- Được định nghĩa bên ngoài hàm, phạm vi toàn chương trình, sử dụng được ở tất cả các hàm.

Biến cục bộ (Local Variable)

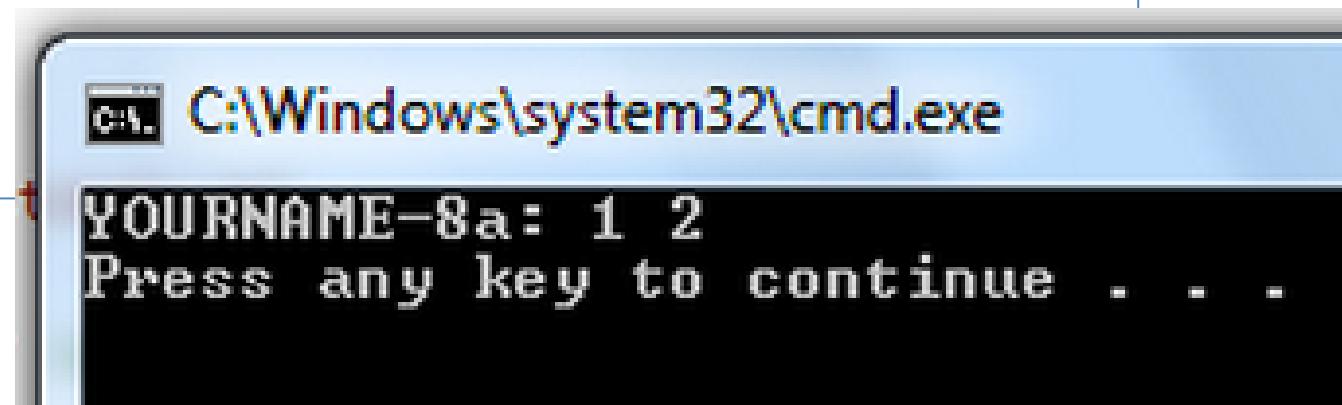
- Được định nghĩa trong một hàm và phạm vi nằm trong hàm/khoi lệnh đó.

Biến toàn cục và biến cục bộ

```
#include "stdafx.h"

int i=1; // GLOBAL VARIABLE

int _tmain(int argc, _TCHAR* argv[])
{
    int j=2; // LOCAL VARIABLE
    printf("YOURNAME-8a: %d %d\n", i, j);
    return 0;
}
```



Biến toàn cục và biến cục bộ

- ☐ **mov [ebp+var_8], 2**
tương ứng với biến j là
biến cục bộ

```
mov    [ebp+var_8], 2
mov    esi, esp
mov    eax, [ebp+var_8]
push   eax
mov    ecx, i
push   ecx
push   offset aYourname8aDD ; "YOURNAME-8a: %d %d\n"
call   ds:_imp_printf
```

```
#include "stdafx.h"

int i=1; // GLOBAL VARIABLE

int _tmain(int argc, _TCHAR* argv[])
{
    int j=2; // LOCAL VARIABLE
    printf("YOURNAME-8a: %d %d\n", i, j);
    return 0;
}
```

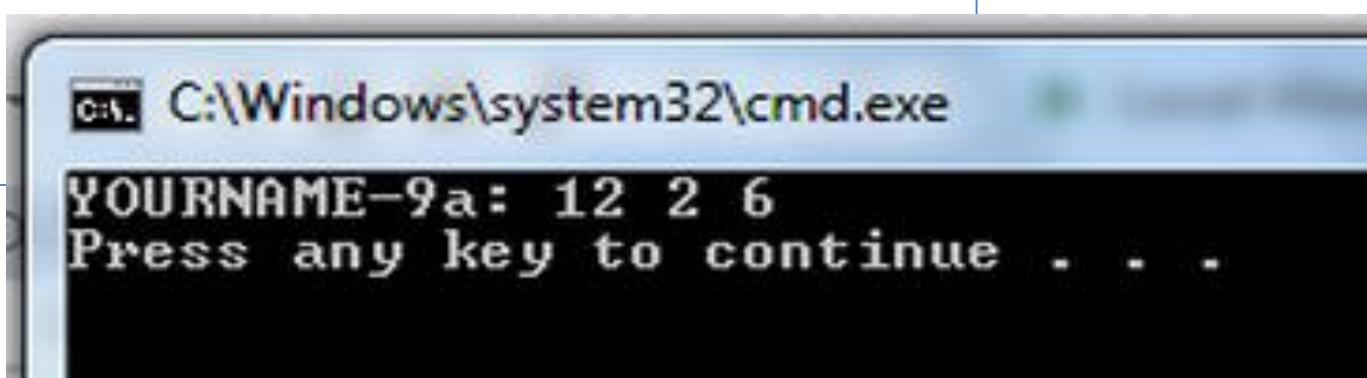
Phân tích hàm

- Hàm và tham số
- Biến toàn cục và biến cục bộ
- Các phép toán cơ bản

Các phép toán cơ bản

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int i=10;
    int j=2;
    int k;
    i = i + 2;
    k = i / j;
    printf("YOURNAME-9a: %d %d %d\n", i, j, k);
    return 0;
}
```



Các phép toán cơ bản

```
mov    [ebp+var_8], 0Ah  
mov    [ebp+var_14], 2  
mov    eax, [ebp+var_8]  
add    eax, 2  
mov    [ebp+var_8], eax  
mov    eax, [ebp+var_8]  
cdq  
idiv   [ebp+var_14]  
mov    [ebp+var_20], eax
```

```
int i=10;  
int j=2;  
  
i = i + 2;  
  
k = i / j;
```

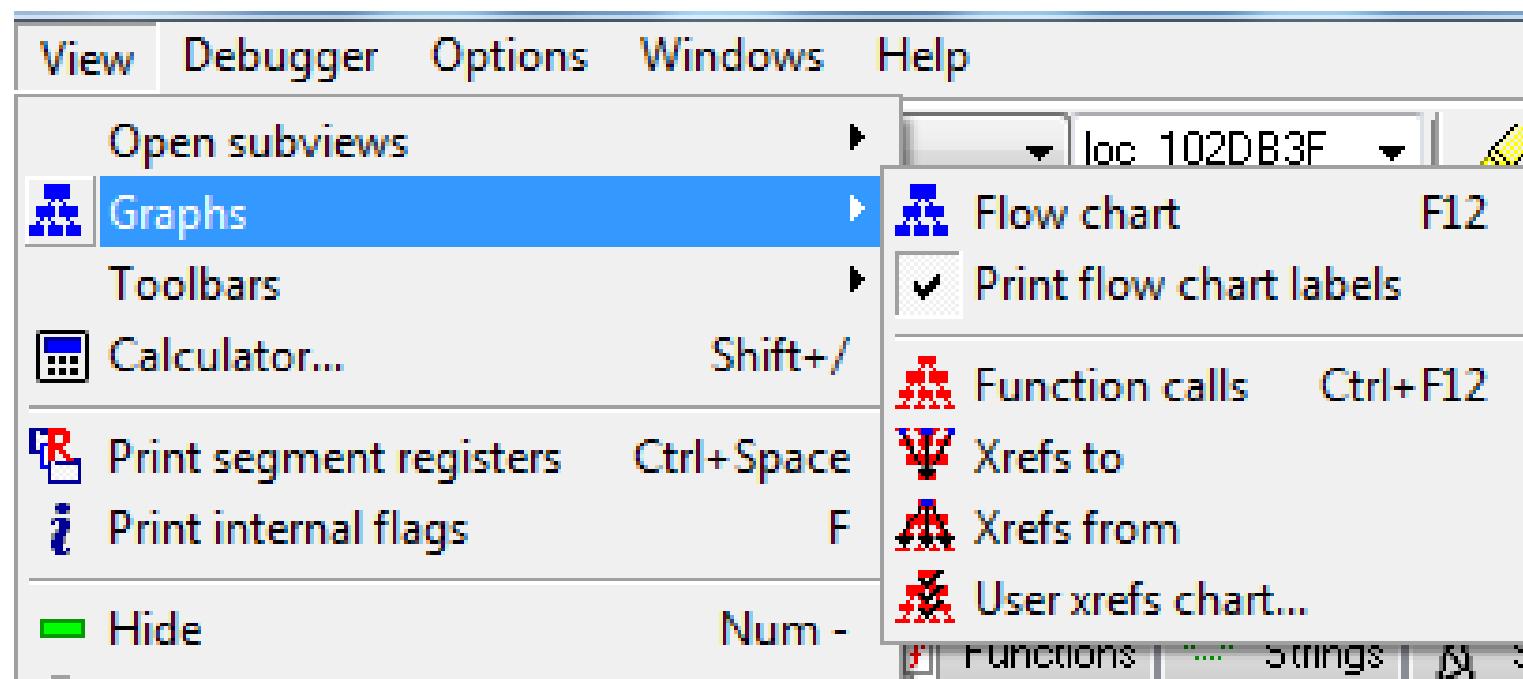
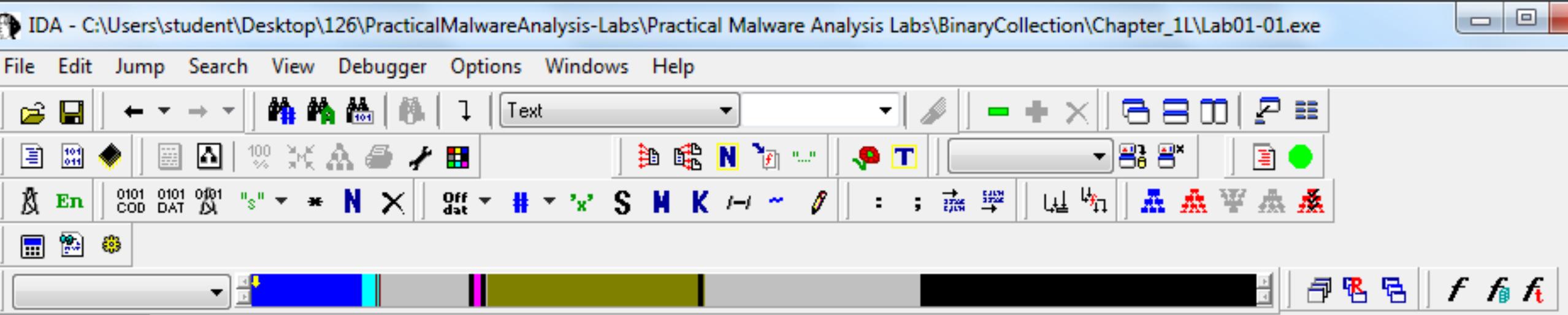
Các phép toán cơ bản

ASM Code	Explanation	C Code
mov [ebp+var_8], 0Ah	Put the number 10 into a local variable (i)	int i=10;
mov [ebp+var_14], 2	Put the number 2 into a local variable (j)	int j=2;
mov eax, [ebp+var_8]	Put i into eax	
add eax, 2	Add 2 to eax	i = i + 2;
mov [ebp+var_8], eax	Put eax (the result) into a local variable (i)	
mov eax, [ebp+var_8]	Put i into eax	
cdq	Convert double to quad (required for division)	k = i / j;
idiv [ebp+var_14]	Divide the value in eax by a local variable (j)	
mov [ebp+var_20], eax	Put eax (the result) into a local variable (k)	

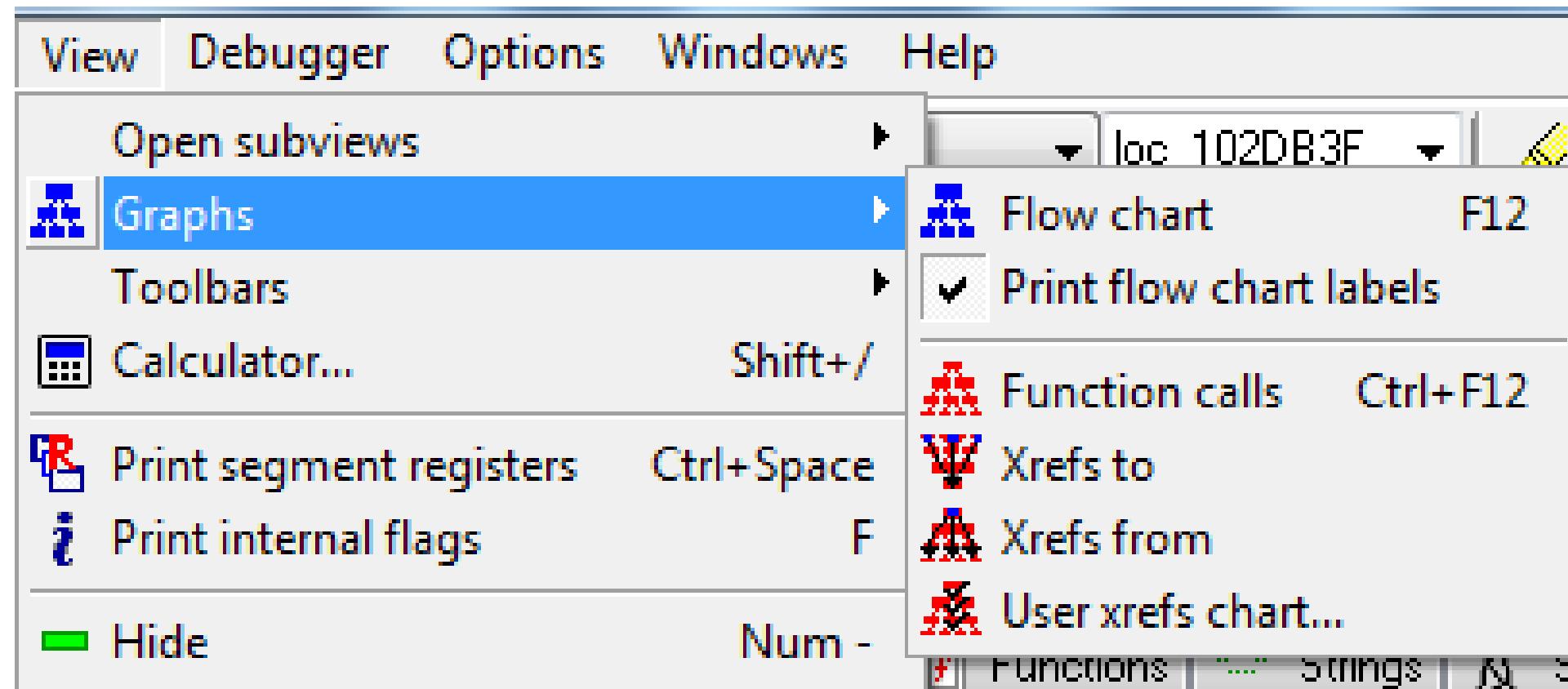
Nội dung

1. Nhắc lại về Assembly
2. Sử dụng IDA pro để dịch ngược mã độc
3. Sử dụng đối sánh chéo
4. Phân tích hàm
5. Sử dụng biểu đồ hàm
6. Một số lưu ý

Sử dụng biểu đồ hàm

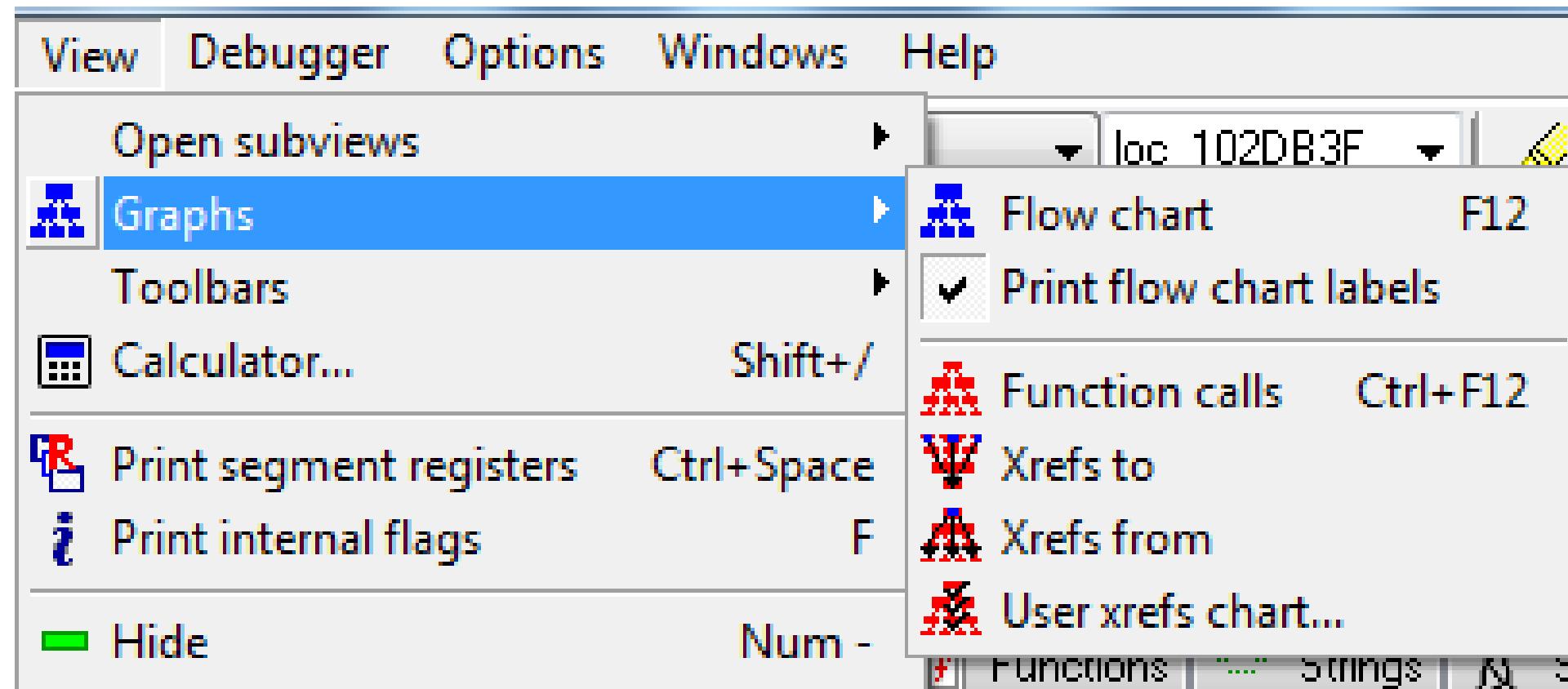


Sử dụng biểu đồ hàm



- Flow chart: Tạo biểu đồ theo dõi của hàm hiện tại
- Function call: Đồ thị hàm gọi toàn bộ chương trình

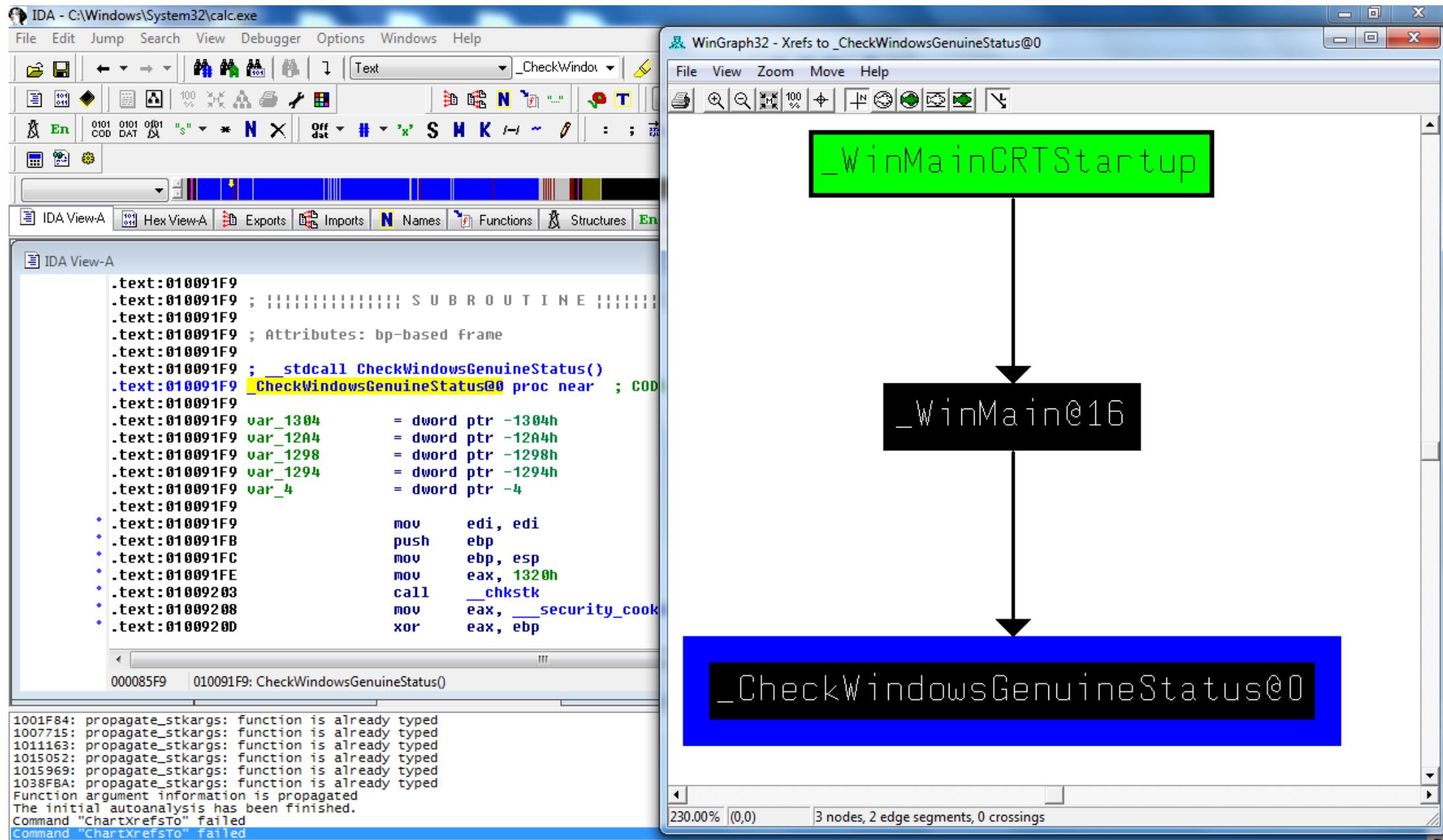
Sử dụng biểu đồ hàm



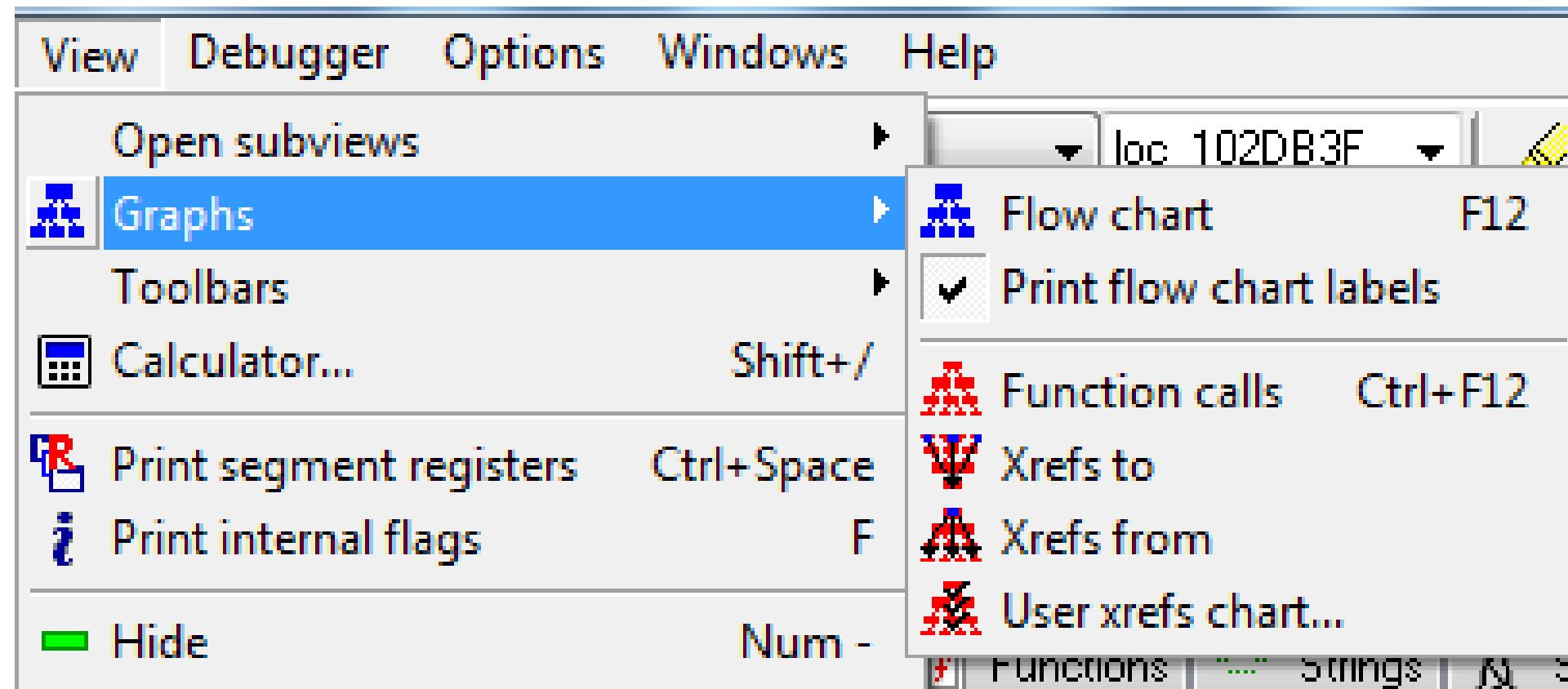
Xrefs to: Đồ thị XREFs đến XREF đã chọn

Có thể hiển thị tất cả các paths đến một hàm

Xrefs to

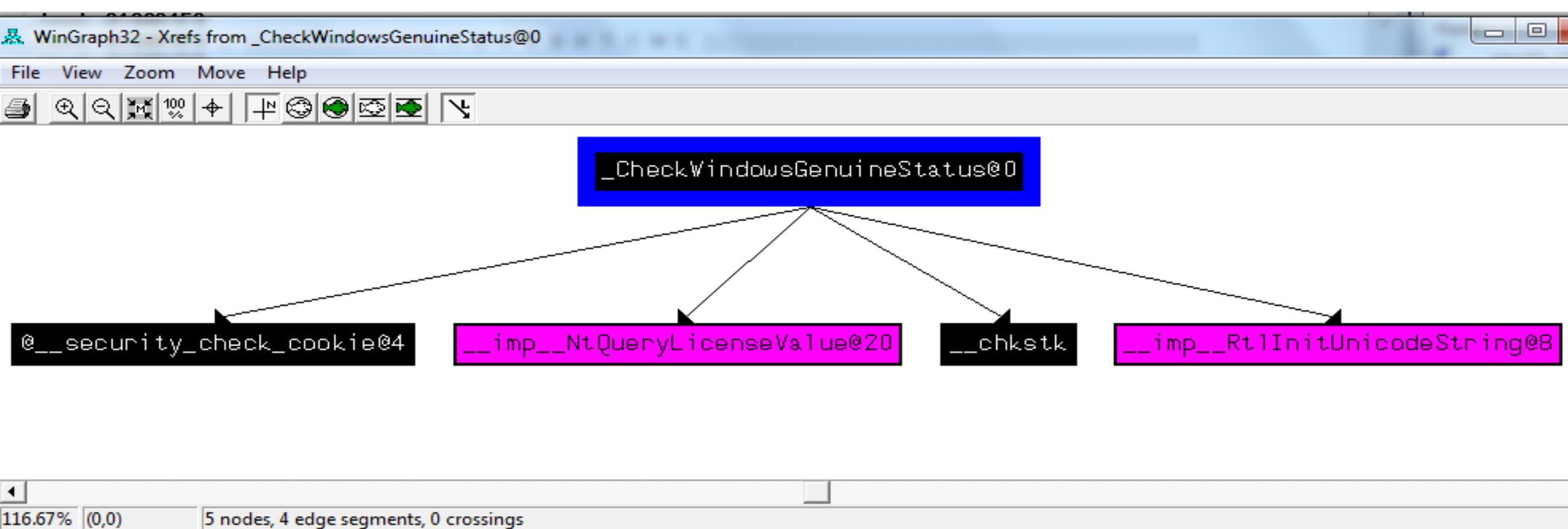


Sử dụng biểu đồ hàm

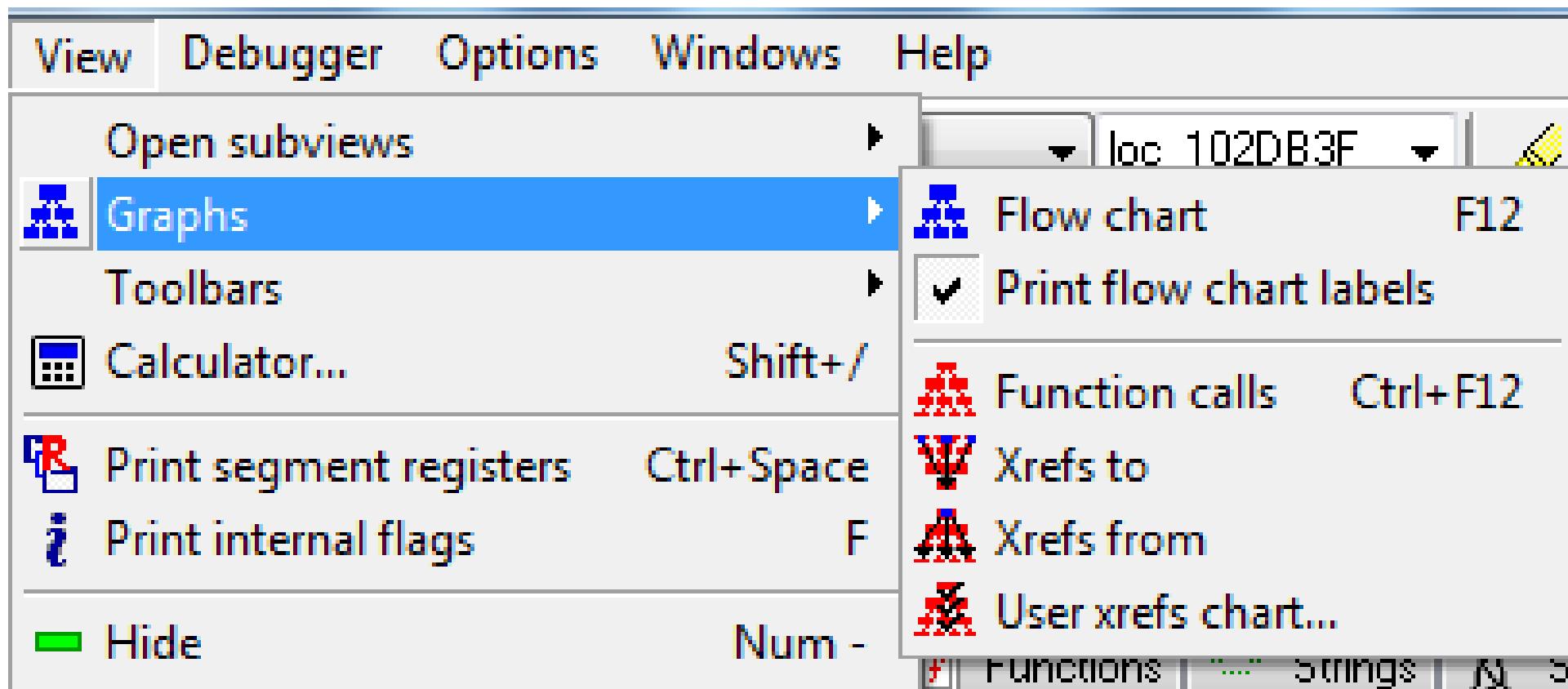


- Xrefs from: Đồ thị XREFs từ XREF đã chọn
- Có thể hiển thị tất cả các đường đi thoát khỏi một hàm

Xrefs from



Sử dụng biểu đồ hàm



User xrefs chart:

- Tùy chỉnh đệ quy của biểu đồ, ký hiệu được sử dụng...
- Cách duy nhất để sửa đổi các đồ thị kế thừa

Nội dung

- 1. Nhắc lại về Assembly**
- 2. Sử dụng IDA pro để dịch ngược mã độc**
- 3. Sử dụng đối sánh chéo**
- 4. Phân tích hàm**
- 5. Sử dụng biểu đồ hàm**
- 6. Một số lưu ý**

Một số lưu ý

Một số tùy chọn sau đây giúp việc phân tích dễ dàng hơn:

- Đổi tên hàm**
- Comments**
- Kiểu hiển thị các toán tử**
- Sử dụng hằng số được đặt tên**

Đổi tên hàm

- ❑ Có thể thay đổi một tên như: sub_401000 thành ReverseBackdoorThread
- ❑ Khi thay đổi tên ở một nơi thì IDA sẽ tự động đồng bộ tên mới ở tất cả những nơi khác.

Đổi tên hàm

Function Operand Manipulation

Without renamed arguments

```
004013C8 mov    eax, [ebp+arg_4]
004013CB push   eax
004013CC call   _atoi
004013D1 add    esp, 4
004013D4 mov    [ebp+var_598], ax
004013DB movzx  ecx, [ebp+var_598]
004013E2 test   ecx, ecx
004013E4 jnz    short loc_4013F8
004013E6 push   offset aError
004013EB call   printf
004013F0 add    esp, 4
004013F3 jmp    loc_4016FB
004013F8 ; -----
004013F8
004013F8 loc_4013F8:
004013F8 movzx  edx, [ebp+var_598]
004013FF push   edx
00401400 call   ds:htons
```

With renamed arguments

```
004013C8 mov    eax, [ebp+port_str]
004013CB push   eax
004013CC call   _atoi
004013D1 add    esp, 4
004013D4 mov    [ebp+port], ax
004013DB movzx  ecx, [ebp+port]
004013E2 test   ecx, ecx
004013E4 jnz    short loc_4013F8
004013E6 push   offset aError
004013EB call   printf
004013F0 add    esp, 4
004013F3 jmp    loc_4016FB
004013F8 ; -----
004013F8
004013F8 loc_4013F8:
004013F8 movzx  edx, [ebp+port]
004013FF push   edx
00401400 call   ds:htons
```

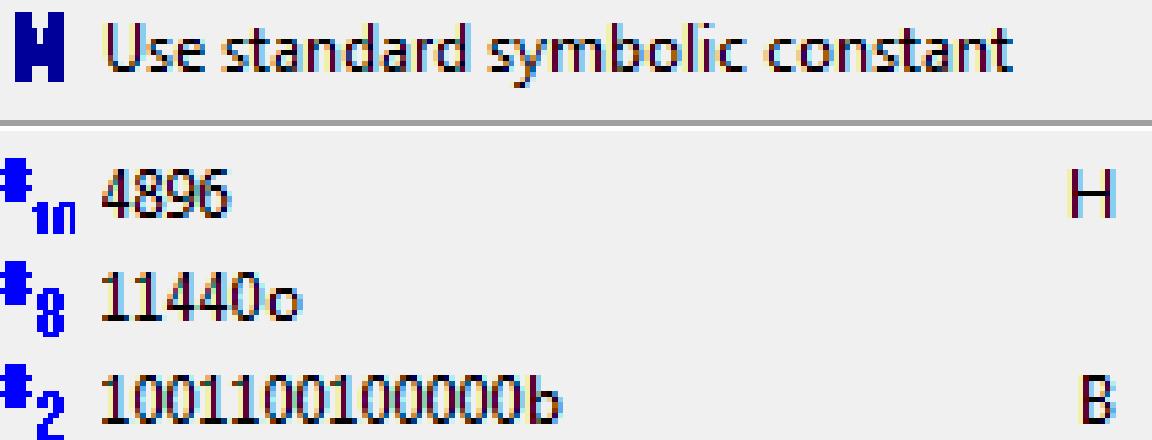
Comments

- Nhập dấu hai chấm (:) để thêm một comment
- Comment được đặt sau dấu chấm phẩy (;) cho tất cả các Xrefs

Kiểu hiển thị các toán tử

- Các toán tử mặc định kiểu thập lục phân (Hex)
- Có thể sử dụng định dạng kiểu khác bằng cách nhấn chuột phải và chọn kiểu.

```
mov      edi, edi
push    ebp
mov      ebp, esp
mov      eax, 1320h
call    __chkstk
mov      eax, ___se
xor      eax, ebp
mov      [ebp+var_4], eax
push    offset aSe
```



Sử dụng hằng số được đặt tên

- ☐ Làm cho các tham số của Windows API được rõ ràng hơn

Before symbolic constants

```
mov    esi, [esp+1Ch+argv]
mov    edx, [esi+4]
mov    edi, ds>CreateFileA
push   0      ; hTemplateFile
push   80h    ;
dwFlagsAndAttributes
push   3      ;
dwCreationDisposition
push   0      ;
lpSecurityAttributes
push   1      ; dwShareMode
```

After symbolic constants

```
mov    esi, [esp+1Ch+argv]
mov    edx, [esi+4]
mov    edi, ds>CreateFileA
push   NULL   ; hTemplateFile
push   FILE_ATTRIBUTE_NORMAL ;
dwFlagsAndAttributes
push   OPEN_EXISTING      ;
dwCreationDisposition
push   NULL              ;
lpSecurityAttributes
push   FILE_SHARE_READ    ; dwShareMode
```

Nội dung

1. Nhắc lại về Assembly
2. Sử dụng IDA pro để dịch ngược mã độc
3. Sử dụng đối sánh chéo
4. Phân tích hàm
5. Sử dụng biểu đồ hàm
6. Một số lưu ý

Mã độc

**Chương 4. Kỹ thuật phân tích mã
độc dựa trên gõ rối**

Mục tiêu

- **Giới thiệu kỹ thuật gõ rối**
- **Giới thiệu, hướng dẫn sinh viên sử dụng trình gõ rối**

OllyDbg trong phân tích mã độc

Tài liệu tham khảo

- [1] Michael Sikorski, Andrew Honig, 2012, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, (ISBN: 978-1593272906).
- [2] Sam Bowne, Slides for a college course at City College San Francisco,
https://samsclass.info/126/126_S17.shtml

Nội dung

1. Gõ rối
2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly
3. Gõ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gõ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gõ rối
7. Phân tích mã độc với OllyDbg

Nội dung

1. Gõ rối
2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly
3. Gõ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gõ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gõ rối
7. Phân tích mã độc với OllyDbg

Trình dịch ngược và gỡ rối

- Trình gỡ rối (debugger) là một chương trình hoặc thiết bị phần cứng cho phép kiểm tra và thực thi một chương trình khác.
- Trình dịch ngược (IDA Pro) dịch ngược file thực thi từ mã máy về mã Assembly

Trình dịch ngược và gõ rối

- Trình gõ rối dừng chương trình tại một điểm bất kỳ, đồng thời hiển thị
 - Các vùng nhớ
 - Register
 - Đối số của mọi hàm
 - Cho phép thay đổi giá trị

Trình gõ rối

❑ Ollydbg

- Phổ biến trong phân tích mã độc
- Chỉ gõ rối ở chế độ người dùng

❑ Windbg

- Hỗ trợ gõ rối mức nhân

Nội dung

1. Gõ rối
2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly
3. Gõ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gõ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gõ rối
7. Phân tích mã độc với OllyDbg

Gõ rối mức mã nguồn

- ❑ Thường được tích hợp trong ngôn ngữ lập trình
- ❑ Có thể đặt breakpoints (dừng tại dòng mã nguồn nhất định)
- ❑ Có thể chạy chương trình theo từng dòng mã nguồn

Gỡ rối mức mã Assembly

- Hoạt động trên mã Assembly**
- Đặt breakpoints tại một cấu trúc Assembly**
- Dùng trong quá trình phân tích mã độc vì không có mã nguồn của mã độc**

Nội dung

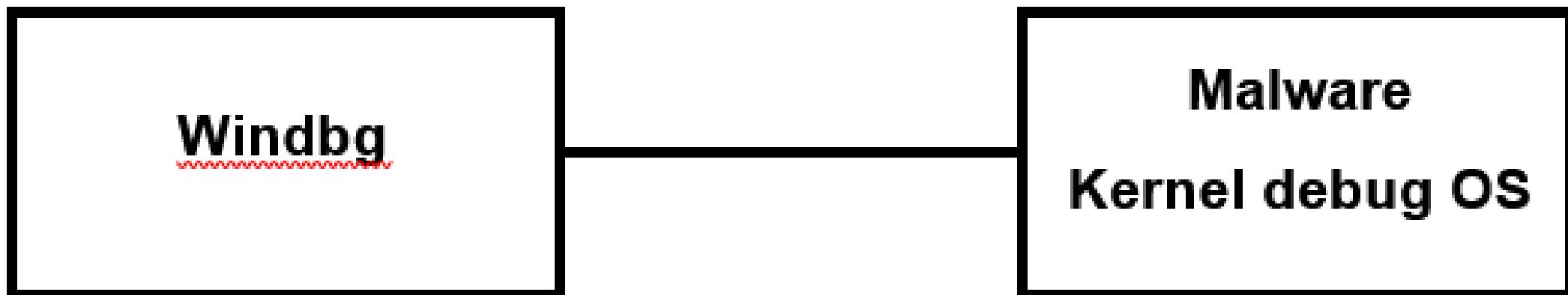
1. Gõ rối
2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly
3. Gõ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gõ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gõ rối
7. Phân tích mã độc với OllyDbg

Gõ rối chế độ người dùng

- Trình gõ rối chạy trên cùng hệ thống với mã được phân tích
- Gõ rối một file thực thi duy nhất
- Tách khỏi các tệp thực thi khác của HĐH

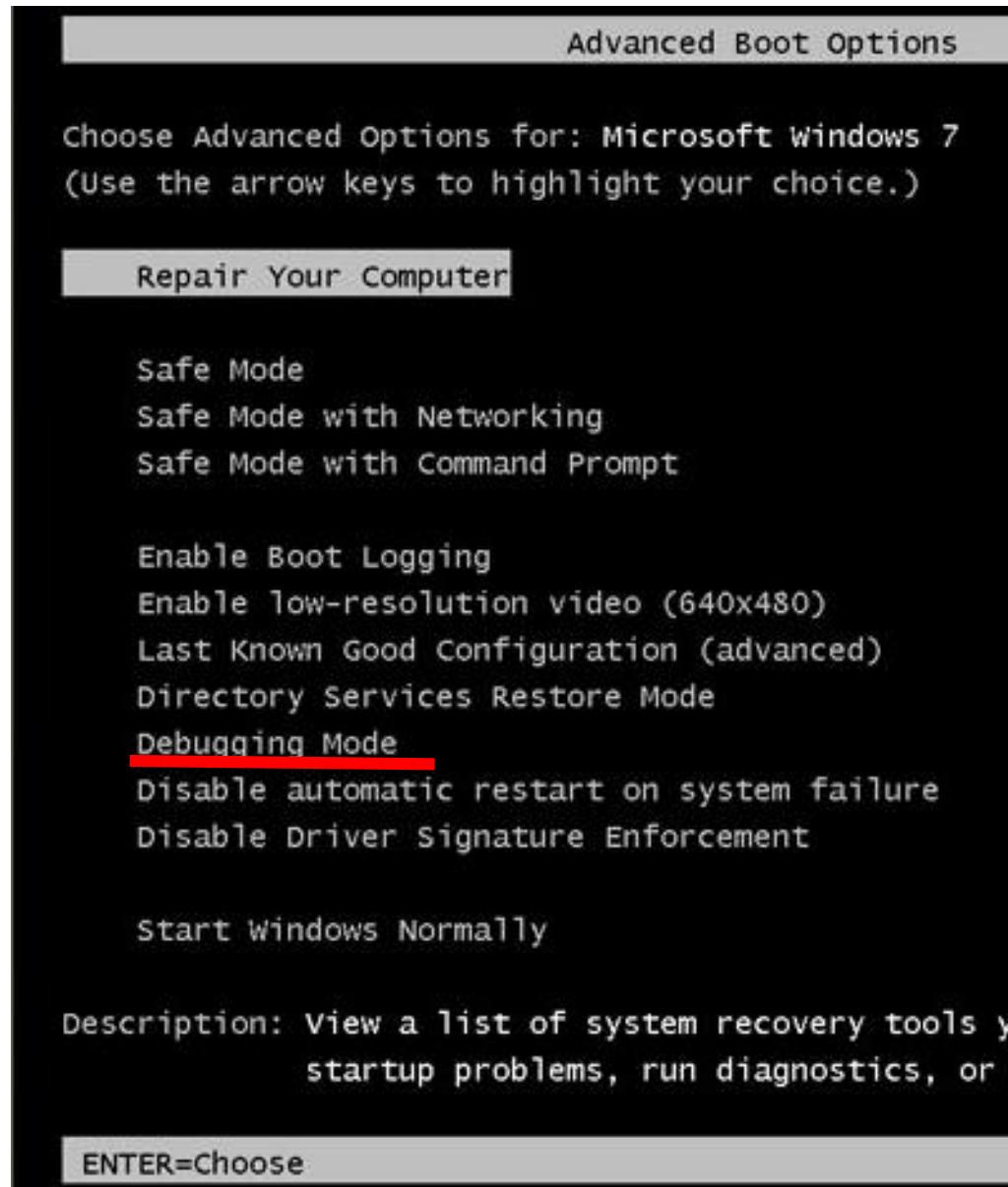
Gõ rối chế độ nhân

- Yêu cầu hai máy tính có kết nối mạng với nhau, một máy chạy đoạn mã được gõ rối, máy khác chạy trình gõ rối
- HĐH cần được cấu hình cho phép gõ rối ở chế độ nhân



Gỡ rối chế độ nhân

- ☐ Án F8 trong quá trình khởi động
- ☐ "Debugging Mode"



Nội dung

1. Gõ rối
2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly
3. Gõ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gõ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gõ rối
7. Phân tích mã độc với OllyDbg

Sử dụng trình gõ rối

- ❑ Có thể tải các file EXEs hoặc DLLs trực tiếp vào Ollydbg
- ❑ Nếu một phần mềm độc hại đang được chạy, có thể sử dụng chức năng Attach Process để thực hiện Debug một tiến trình đang chạy

Mở một file thực thi

- Thao tác mở một binary: File > Open
- Thêm các đối số dòng lệnh nếu cần thiết
- Ollydbg sẽ dừng lại ở Entry Point, WinMain.. Nếu nó có thể xác định được
- Nếu không nó sẽ break tại điểm vào của chương trình được xác định trong PE Header

Mở một tiến trình đang chạy

- ❑ Thao tác: File > Attach
- ❑ Ollydbg sẽ break và tạm dừng các chương trình và tất cả các luồng
- ❑ Nếu bắt nó trong DLL, thiết lập một breakpoint để truy cập vào bên trong những đoạn code và quan sát.

Sử dụng trình gõ rối

- ☐ Từng bước (Single-step): chạy từng lệnh một và quan sát mọi thứ diễn ra trong một chương trình.

Single-stepping through a section of code to see how it changes memory

```
D0F3FDF8 D0F5FEEE FDEEE5DD 9C (.....)
4CF3FDF8 D0F5FEEE FDEEE5DD 9C (L.....)
4C6FFDF8 D0F5FEEE FDEEE5DD 9C (Lo.....)
4C6F61F8 D0F5FEEE FDEEE5DD 9C (Loa.....)
. . . SNIP . .
4C6F6164 4C696272 61727941 00 (LoadLibraryA.)
```

Stepping through code

```
mov     edi, DWORD_00406904
mov     ecx, 0x0d
LOC_040106B2
xor     [edi], 0x9C
inc     edi
loopw   LOC_040106B2
...
DWORD:00406904: F8FDF3D01
```

Stepping-over v. Stepping-Into

□ Step-over

- Thực hiện một hàm mà không dừng
- Giảm khối lượng code cần phân tích
- Có thể bỏ qua một số chức năng, đặc biệt khi hàm không có các returns

□ Step-into

- Di chuyển đến lệnh đầu tiên của hàm và dừng lại tại đó

Dùng thực thi với Breakpoints

- ❑ Các breakpoint được dùng để làm điểm dừng thực thi và cho phép ta quan sát trạng thái của chương trình
- ❑ Một chương trình khi dừng tại breakpoint được gọi là **broken**

Call to EAX

00401008	mov	ecx, [ebp+arg_0]
0040100B	mov	eax, [edx]
0040100D	call	eax

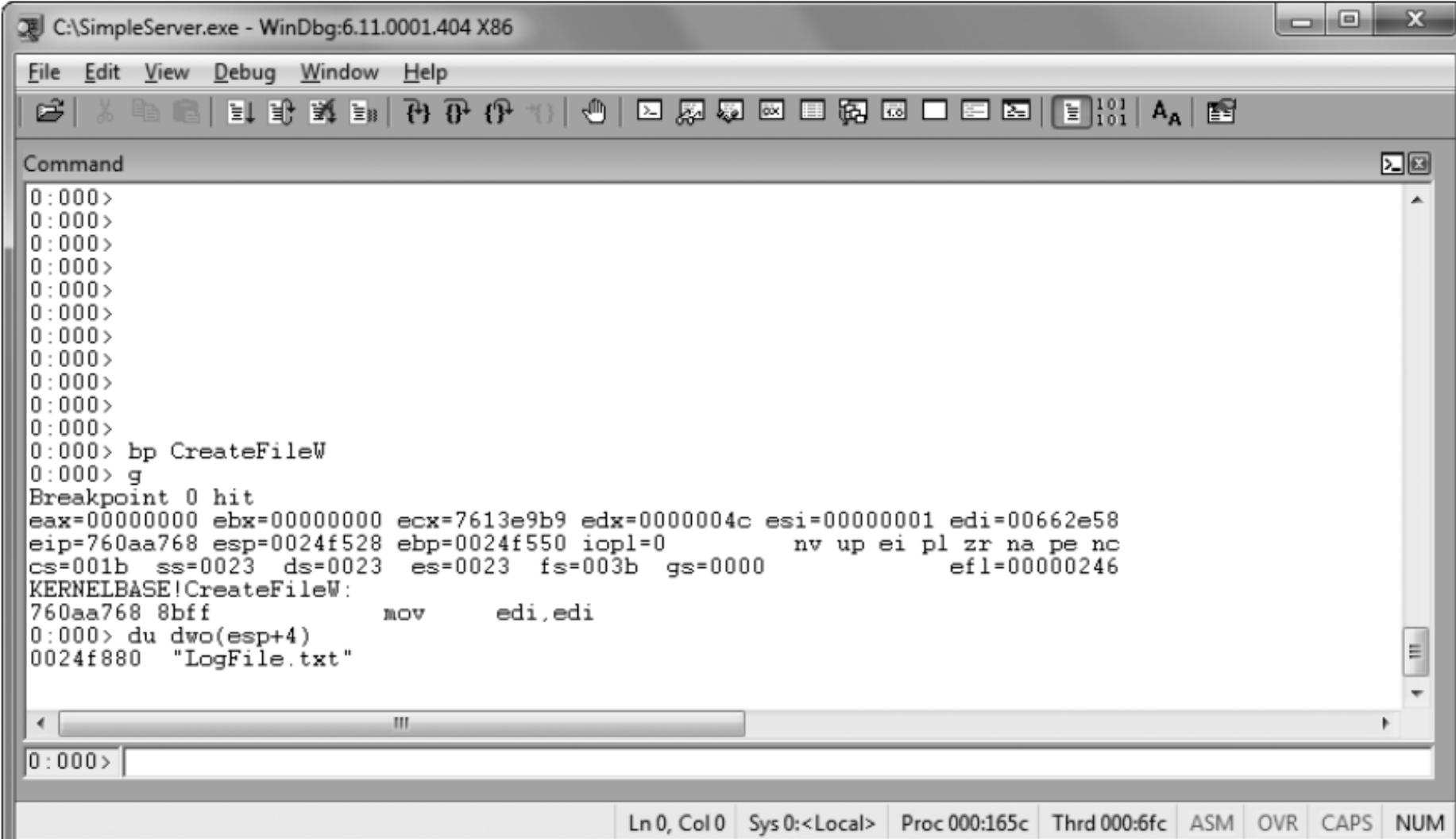
Dừng thực thi với Breakpoints

- Đoạn chương trình tín tên file, sau đó tạo file
- Đặt một breakpoint tại **CreateFileW** và xem trong stack tên file

Using a debugger to determine a filename

```
0040100B xor    eax, esp
0040100D mov    [esp+0D0h+var_4], eax
00401014 mov    eax, edx
00401016 mov    [esp+0D0h+NumberOfBytesWritten], 0
0040101D add    eax, 0FFFFFFFEh
00401020 mov    cx, [eax+2]
00401024 add    eax, 2
00401027 test   cx, cx
0040102A jnz    short loc_401020
0040102C mov    ecx, dword ptr ds:a_txt ; ".txt"
00401032 push   0                      ; hTemplateFile
00401034 push   0                      ; dwFlagsAndAttributes
00401036 push   2                      ; dwCreationDisposition
00401038 mov    [eax], ecx
0040103A mov    ecx, dword ptr ds:a_txt+4
00401040 push   0                      ; lpSecurityAttributes
00401042 push   0                      ; dwShareMode
00401044 mov    [eax+4], ecx
00401047 mov    cx, word ptr ds:a_txt+8
0040104E push   0                      ; dwDesiredAccess
00401050 push   edx                   ; lpFileName
00401051 mov    [eax+8], cx
00401055 1call  CreateFileW ; CreateFileW(x,x,x,x,x,x,x)
```

Dùng thực thi với Breakpoints



The screenshot shows the WinDbg debugger interface with the title bar "C:\SimpleServer.exe - WinDbg:6.11.0001.404 X86". The command window displays the following session:

```
0:000>
0:000>
0:000>
0:000>
0:000>
0:000>
0:000>
0:000>
0:000>
0:000>
0:000>
0:000> bp CreateFileW
0:000> g
Breakpoint 0 hit
eax=00000000 ebx=00000000 ecx=7613e9b9 edx=0000004c esi=00000001 edi=00662e58
eip=760aa768 esp=0024f528 ebp=0024f550 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
KERNELBASE!CreateFileW:
760aa768 8bff          mov     edi,edi
0:000> du dwo(esp+4)
0024f880  "LogFile.txt"
```

The status bar at the bottom shows "Ln 0, Col 0" and "Proc 000:165c Thrd 000:6fc ASM OVR CAPS NUM".

Using a breakpoint to see the parameters to a function call. We set a breakpoint on CreateFileW and then examine the first parameter of the stack.

Các loại breakpoints

- Software breakpoints – breakpoints mềm
- Hardware breakpoints – breakpoints cứng
- Conditional breakpoints – breakpoints có điều kiện
- Breakpoints on memory – breakpoints trên bộ nhớ

Software Breakpoints

- ☐ Khi được đặt, trình gõ rối sẽ ghi đè lệnh 0xCC (INT 3)
- ☐ Thường là loại breakpoint mặc định của các trình gõ rối

Disassembly and Memory Dump of a Function with a Breakpoint Set

Disassembly view	Memory dump
00401130 55 ① push ebp	00401130 ② CC 8B EC 83
00401131 88 EC mov ebp, esp	00401134 E4 F8 81 EC
00401133 83 E4 F8 and esp, 0FFFFFFF8h	00401138 A4 03 00 00
00401136 81 EC A4 03 00 00 sub esp, 3A4h	0040113C A1 00 30 40
0040113C A1 00 30 40 00 mov eax, dword_403000	00401140 00

Software Breakpoints

□ Hữu dụng cho string decoders

A string decoding breakpoint

```
push offset "4NNpTNHLKIXoPm7iBhUAjvRKNaUVBlr"  
call String_Decoder
```

...

```
push offset "ugKLdNlLT6emldCeZi72mUjieuBqdfZ"  
call String_Decoder
```

...

Hardware Breakpoints

- Không biến đổi code, stack hoặc bất kỳ tài nguyên nào
- Không làm chậm quá trình thực thi
- Hoạt động dựa vào thanh ghi debug (DR7) của CPU, có thể đặt tối đa 4 vị trí trong một thời gian (DR0-DR3)

Conditional Breakpoints

- ❑ Là các breakpoint mềm và chỉ ngắt khi thỏa mãn một điều kiện logic
- ❑ Giúp hạn chế các hành động thừa

Conditional Breakpoints

Poison Ivy backdoor

- ❑ Poison Ivy cấp phát vùng nhớ để đặt Shellcode, nó nhận lệnh từ các máy chủ C&C
- ❑ Nhiều hàm cấp phát bộ nhớ
- ❑ Đặt một Conditional breakpoint tại hàm VirtualAlloc trong thư viện kernel32.dll

00C3F0B0	0095007C	CALL to VirtualAlloc from 00950079
00C3F0B4	00000000	Address = NULL
00C3F0B8	00000029	Size = 29 (41.)
00C3F0BC	00001000	AllocationType = MEM_COMMIT
00C3F0C0	00000040	Protect = PAGE_EXECUTE_READWRITE

Memory Breakpoints

- ❑ Chương trình bị ngắt khi truy cập vào vị trí bộ nhớ đã định
- ❑ Thay đổi các thuộc tính của khối nhớ
- ❑ Không đáng tin cậy, ít sử dụng

Nội dung

- 1. Gõ rối**
- 2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly**
- 3. Gõ rối chế độ nhân và chế độ người dùng**
- 4. Sử dụng trình gõ rối trong quá trình phân tích mã độc**
- 5. Ngoại lệ**
- 6. Chỉnh sửa ngoại lệ với trình gõ rối**
- 7. Phân tích mã độc với OllyDbg**

Ngoại lệ (Exception)

- Sử dụng bởi trình gõ rối để chiếm quyền điều khiển chương trình
- Breakpoints tạo ra ngoại lệ (INT 3)
- Ngoại lệ cũng được gây ra bởi
 - Truy cập bộ nhớ không hợp lệ
 - Chia cho 0
 - Lý do khác

First- and Second-Chance Exceptions

□ First-Chance

- INT 3
- Các lỗi đã được ghi chú và xử lý trong chương trình

□ Second-Chance

- Các lỗi chưa được ghi chú và xử lý trong chương trình

Danh sách các ngoại lệ

The following chart lists the exceptions that can be generated by the Intel 80286, 80386, 80486, and Pentium processors:

Exception (dec/hex)	Description
0 00h	Divide error: Occurs during a DIV or an IDIV instruction when the divisor is zero or a quotient overflow occurs.
1 01h	Single-step/debug exception: Occurs for any of a number of conditions: <ul style="list-style-type: none">- Instruction address breakpoint fault- Data address breakpoint trap- General detect fault- Single-step trap- Task-switch breakpoint trap
2 02h	Nonmaskable interrupt: Occurs because of a nonmaskable hardware interrupt.
3 03h	Breakpoint: Occurs when the processor encounters an INT 3 instruction.

Nội dung

1. Gõ rối
2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly
3. Gõ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gõ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gõ rối
7. Phân tích mã độc với OllyDbg

BỎ QUA MỘT HÀM

- ☐ Có thể thay đổi cò điều khiển, con trỏ lệnh hoặc mã nguồn
- ☐ BỎ qua một hàm bằng cách đặt một breakpoint trong quá trình gọi hàm, và sau đó thay đổi con trỏ lệnh đến lệnh sau nó

Có thể gây crash chương trình

Kiểm tra một hàm

Chạy một hàm trực tiếp, không thông qua hàm main
bằng cách

- Đặt các giá trị tham số
- Hủy ngăn xếp của chương trình

Nội dung

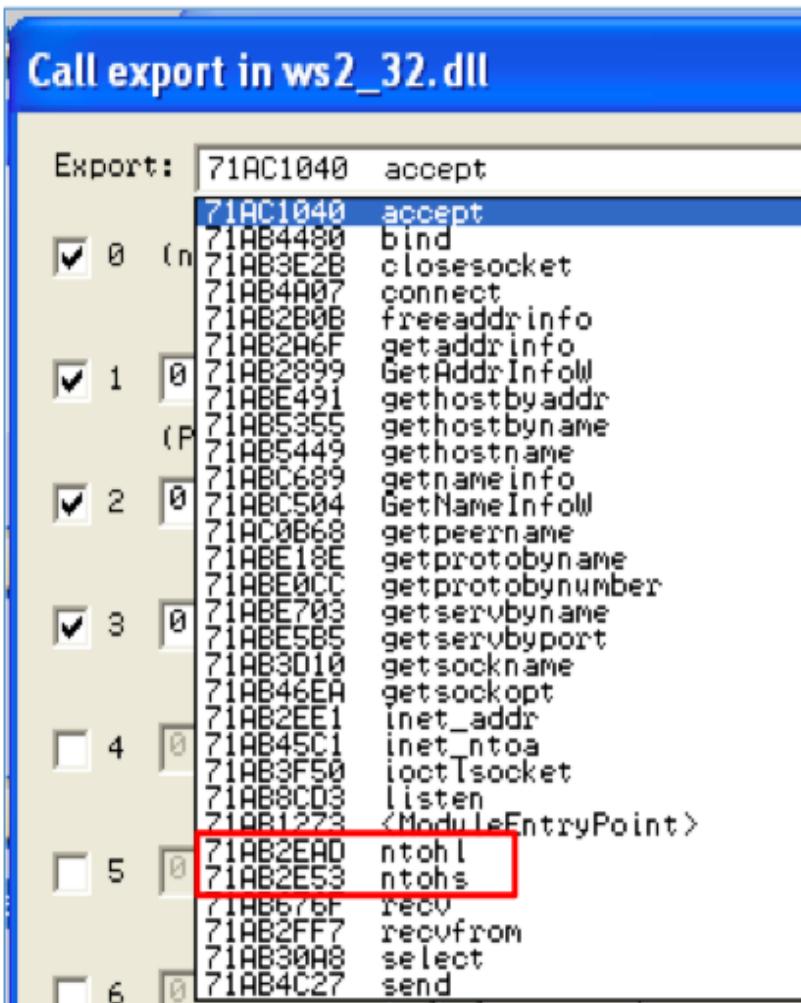
- 1. Gõ rối**
- 2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly**
- 3. Gõ rối chế độ nhân và chế độ người dùng**
- 4. Sử dụng trình gõ rối trong quá trình phân tích mã độc**
- 5. Ngoại lệ**
- 6. Chỉnh sửa ngoại lệ với trình gõ rối**
- 7. Phân tích mã độc với OllyDbg**

OllyDbg

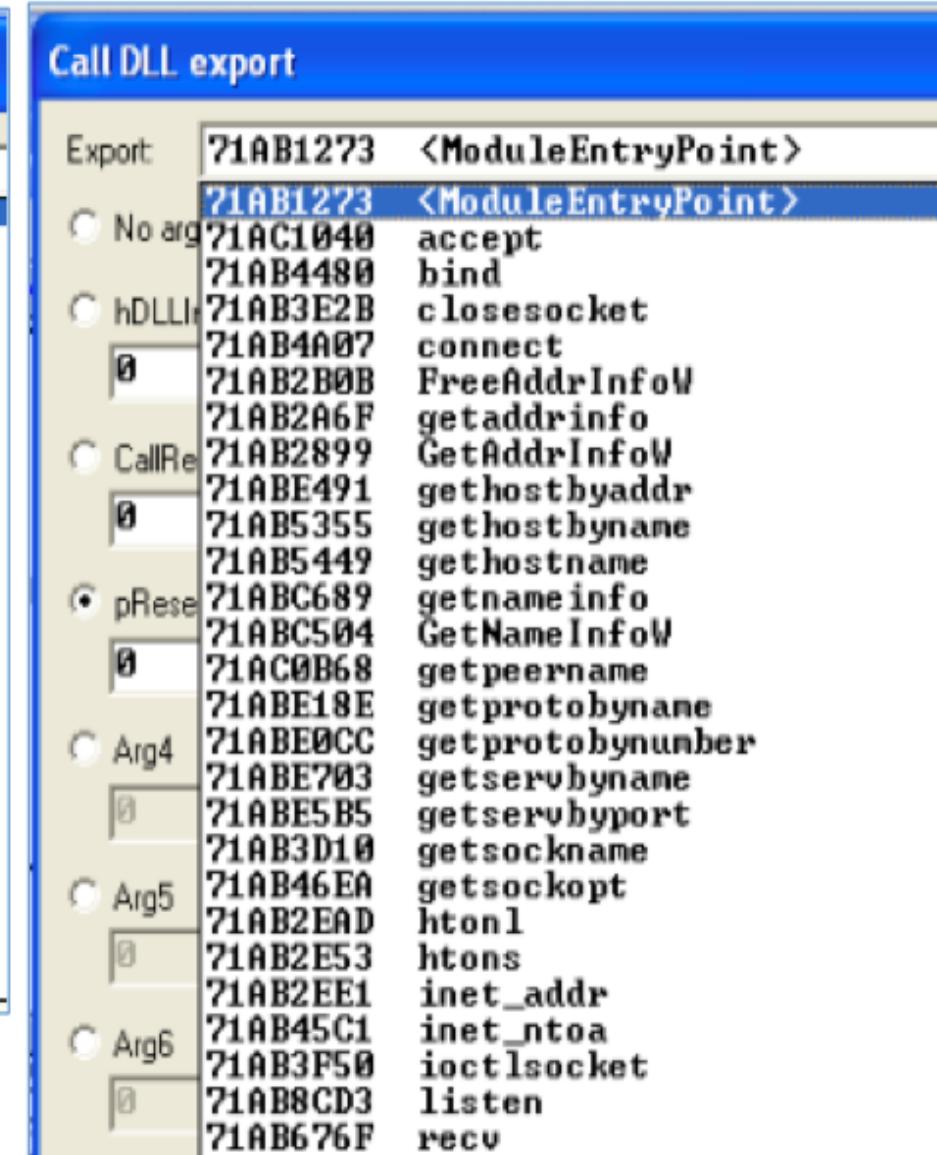
- ❑ Ollydbg đã được phát triển cách đây hơn một thập kỷ.
- ❑ Ban đầu nó chủ yếu được dùng trong việc crack các phần mềm và khai thác.
- ❑ Mã nguồn của phiên bản Ollydbg 1.1 được mua lại bởi Immunity và được đặt lại với cái tên Immunity Debugger

OllyDbg

OllyDbg 1.10



OllyDbg 2.01



Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Phân tích mã độc với OllyDbg

- Tải mã độc**
- Giao diện OllyDbg**
- Bản đồ bộ nhớ**
- Chạy tiến trình**
- Tải DLLs**
- Tracing**
- Ngoại lệ**
- Patching**

Tải mã độc

- ❑ Có thể tải các file EXEs hoặc DLLs trực tiếp vào Ollydbg
- ❑ Nếu một phần mềm độc hại đang được chạy, có thể sử dụng chức năng Attach Process để thực hiện Debug một tiến trình đang chạy

Mở file exe

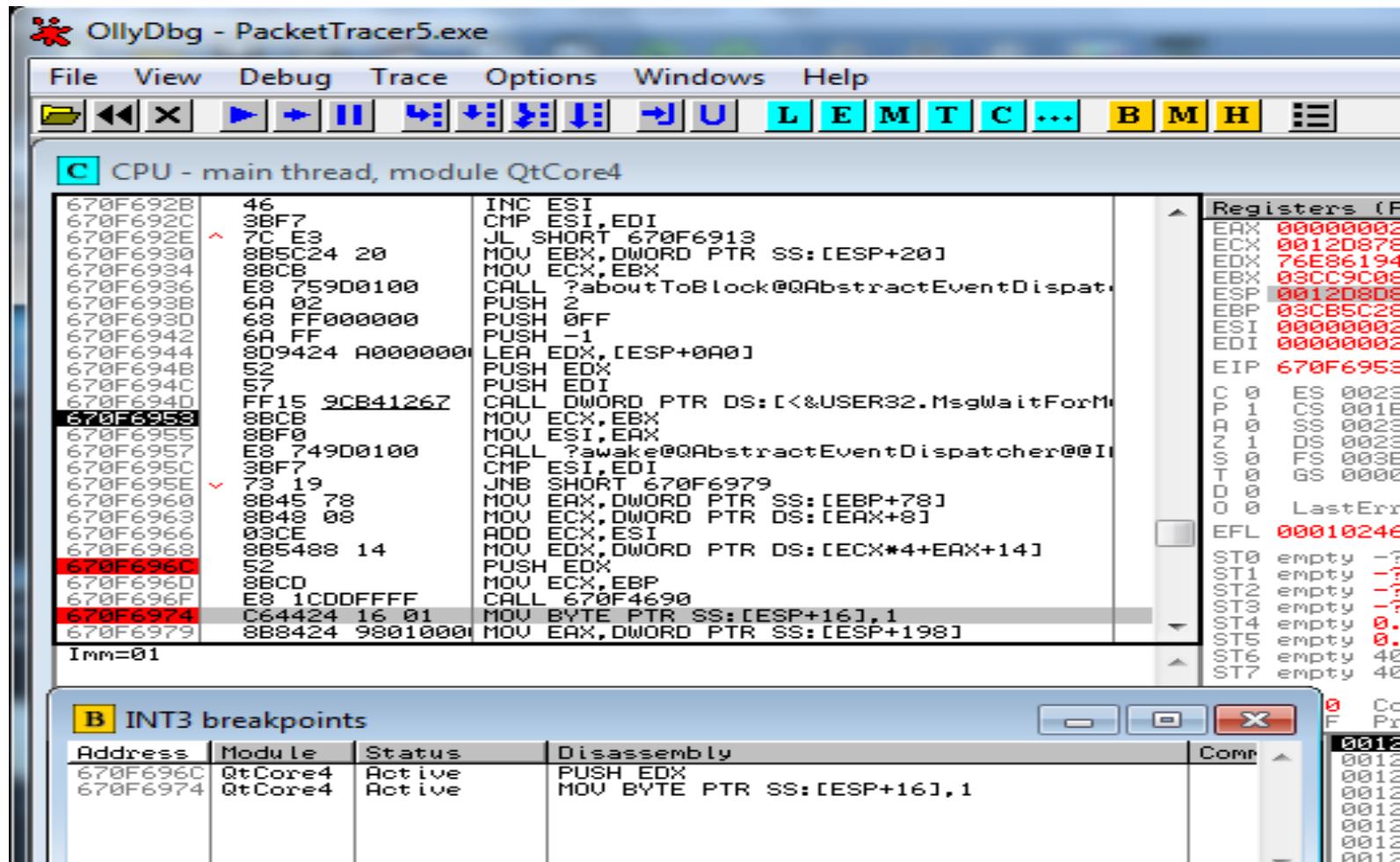
- ❑ Thao tác mở một binary: File > Open
- ❑ Thêm các đối số dòng lệnh nếu cần thiết
- ❑ Ollydbg sẽ dừng lại ở Entry Point, WinMain.. Nếu nó có thể xác định được
- ❑ Nếu không nó sẽ break tại điểm vào của chương trình được xác định trong PE Header
 - Cấu hình tại Option > Debugging Options

Tải một tiến trình đang chạy

- Thao tác: File > Attach**
- Ollydbg sẽ break và tạm dừng các chương trình và tất cả các luồng**
- Nếu bắt nó trong DLL, thiết lập một breakpoint để truy cập vào bên trong những đoạn code và quan sát.**

Xem các breakpoints đang có

□ Thao tác: View, Breakpoints hoặc click vào icon có biểu tượng B trên thanh toolbar



Xem các breakpoints đang có

OllyDbg Breakpoint Options

Function	Right-click menu selection	Hotkey
Software breakpoint	Breakpoint ▶ Toggle	F2
Conditional breakpoint	Breakpoint ▶ Conditional	SHIFT-F2
Hardware breakpoint	Breakpoint ▶ Hardware, on Execution	
Memory breakpoint on access (read, write, or execute)	Breakpoint ▶ Memory, on Access	F2 (select memory)
Memory breakpoint on write	Breakpoint ▶ Memory, on Write	

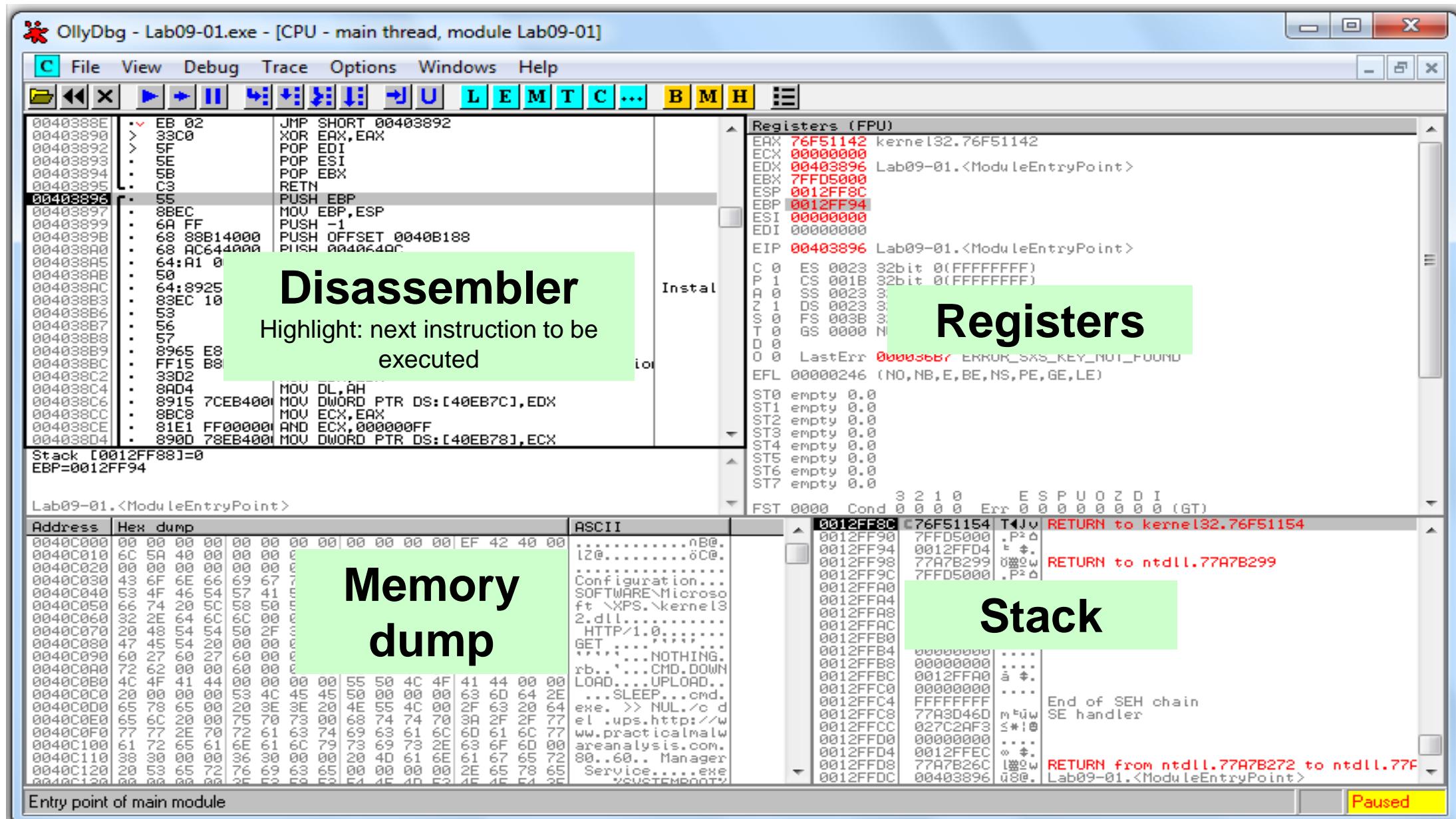
Lưu breakpoints

- ❑ Khi đóng chương trình Ollydbg thì nó sẽ lưu lại các breakpoints đã đặt
- ❑ Khi thực hiện việc mở lại tệp đó thì các breakpoints vẫn giữ nguyên như lúc trước.

Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Giao diện OllyDbg



Chỉnh sửa dữ liệu

☐ Cửa sổ Disassembler

- Nhấn phím Space để chỉnh sửa

☐ Cửa sổ Register hoặc Stack

- Right-click, Modify

☐ Cửa sổ Memory dump

- Right-click, Binary, Edit

- Ctrl+G để đi tới một vị trí trên bộ nhớ

- Right-click vào một địa chỉ bộ nhớ và chọn “Follow in dump”

Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Bản đồ bộ nhớ

M Memory map

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00010000				Map	RW	RW	
00020000	00010000				Map	RW	RW	
00120000	00001000			Stack of main thr.	Priv	RW Gua:	RW Gua:	
0012E000	00002000				Priv	RW	RW	
00130000	00004000				Map	R	R	
00140000	00001000				Priv	RW	RW	
00150000	00067000				Map	R	R	
001C0000	00001000				Priv	RW	RW	\Device\HarddiskVolume1\Windows\System32\locale.nls
001D0000	00001000				Priv	RW	RW	
00240000	00003000				Priv	RW	RW	
002A0000	00008000				Priv	RW	RW	
00400000	00001000	Lab09-01		PE header	Img	R	RWE Cop:	
00401000	0000A000	Lab09-01	.text	Code	Img	R E	RWE Cop:	
0040B000	00001000	Lab09-01	.rdata	Imports	Img	R	RWE Cop:	
0040C000	00005000	Lab09-01	.data	Data	Img	RW Cop:	RWE Cop:	
00420000	00005000				Map	R	R	
004E0000	00003000				Map	R	R	
004F0000	00101000			GDI handles	Map	R	R	
00600000	00088B0000				Map	R	R	
75C60000	00001000	KERNELBA:		PE header	Img	R	RWE Cop:	
75C61000	00044000	KERNELBA:			Img	R E	RWE Cop:	
75CA5000	00002000	KERNELBA:			Img	RW	RWE Cop:	
75CA7000	00004000	KERNELBA:			Img	R	RWE Cop:	
75EB0000	00001000	NSI		PE header	Img	R	RWE Cop:	
75EB1000	00002000	NSI			Img	R E	RWE Cop:	
75EB3000	00001000	NSI			Img	RW	RWE Cop:	
75EB4000	00002000	NSI			Img	R	RWE Cop:	
7SEC0000	00001000	SHELL32		PE header	Img	R	RWE Cop:	
7SEC1000	003C8000	SHELL32			Img	R E	RWE Cop:	
76289000	00007000	SHELL32			Img	RW Cop:	RWE Cop:	
76290000	00879000	SHELL32			Img	R	RWE Cop:	
76B10000	00001000	USER32		PE header	Img	R	RWE Cop:	
76B11000	00068000	USER32			Img	R E	RWE Cop:	
76B79000	00001000	USER32			Img	RW	RWE Cop:	
76B7A000	0005F000	USER32			Img	R	RWE Cop:	
76BE0000	00001000	sechost		PE header	Img	R	RWE Cop:	
76BE1000	00013000	sechost			Img	R E	RWE Cop:	
76BF4000	00003000	sechost			Img	RW Cop:	RWE Cop:	
76BF7000	00002000	sechost			Img	R	RWE Cop:	

Rebasing

- Cơ chế Rebasing xảy ra khi một module không load được địa chỉ cơ sở mà nó ưu tiên
- Các PE file thường có một địa chỉ cơ sở ưu tiên.
 - Hầu hết các EXE được thiết kế để nạp vào địa chỉ ưu tiên của nó là **0x00400000**.
- Các tệp nhị phân EXEs hỗ trợ Address Space Layout Randomization (ASLR) thường được relocated (cấp lại)

DLL Rebasing

- ❑ Các DLLs thường được relocated
- ❑ Vì một ứng dụng có thể sẽ import nhiều DLLs
- ❑ Các Windows DLLs thường có địa chỉ cơ sở ưu tiên khác nhau
- ❑ Các DLLs của bên thứ ba thường có cùng địa chỉ cơ sở ưu tiên

Địa chỉ tuyệt đối và địa chỉ tương đối

Assembly code that requires relocation

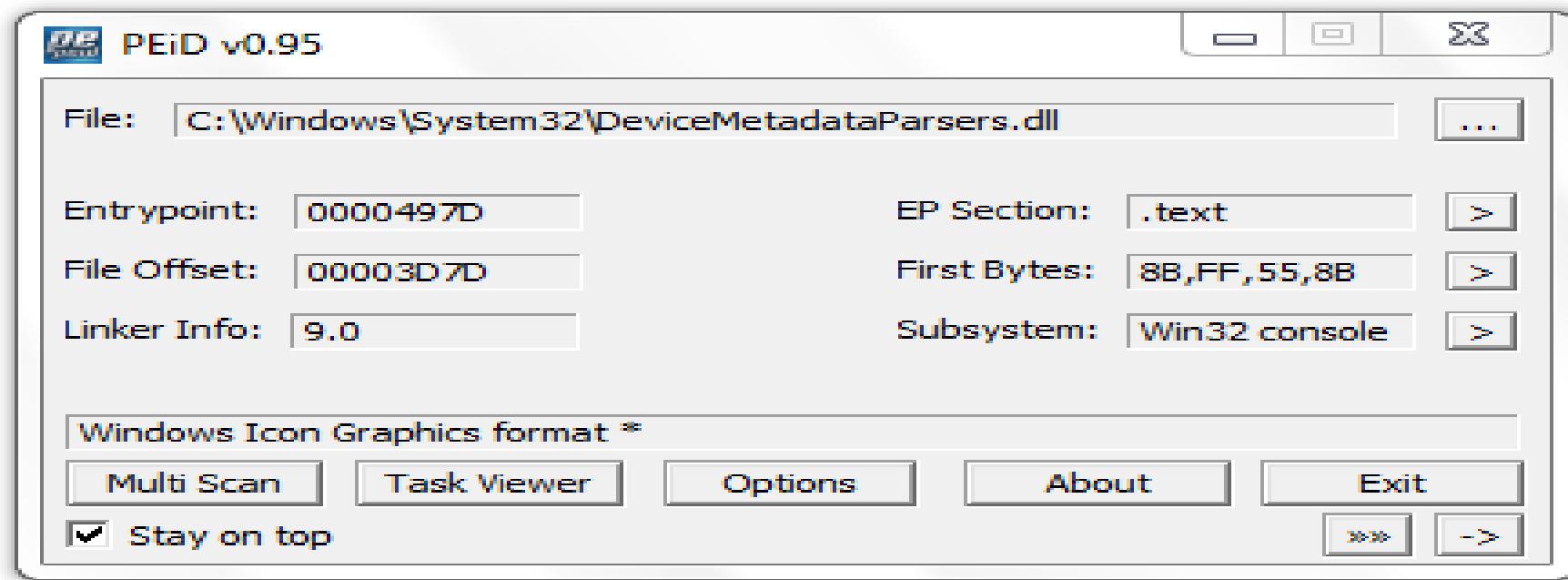
00401203	mov eax, [ebp+var_8]
00401206	cmp [ebp+var_4], 0
0040120a	jnz loc_0040120
0040120c	1mov eax, dword_40CF60

- 3 lệnh đầu tiên sẽ hoạt động tốt nếu được cấp phát lại vì chúng sử dụng các địa chỉ tương đối
- Lệnh cuối cùng có địa chỉ tuyệt đối, nó sẽ sai nếu được cấp phát lại.

Fix-up Locations

- Hầu hết các DLLs có một danh sách các fix-up locations trong đoạn .reloc của PE File.
 - Đây là những lệnh thay đổi khi mã nguồn được cấp phát lại
- DLLs được nạp sau các EXEs và theo thứ tự bất kỳ
- Không thể đoán trước được vị trí của các DLLs trong bộ nhớ khi nó được rebased.

Fix-up Locations



The screenshot shows the "Section Viewer" window. The title bar reads "Section Viewer". The window displays a table of memory section details:

Name	V. Offset	V. Size	R. Offset	R. Size	Flags
.text	00001000	00004DE3	00000400	00004E00	60000020
.data	00006000	000003E4	00005200	00000200	C0000040
.rsrc	00007000	00000438	00005400	00000600	40000040
.reloc	00008000	00000518	00005A00	00000600	42000040

At the bottom of the window is a "Close" button.

DLL Rebasing

- Các DLL có thể xóa bỏ đoạn .reloc của chúng
 - Không thể relocate một DLL như vậy
 - Phải nạp theo địa chỉ cơ sở ưu tiên của nó
- Relocating các DLL sẽ không tốt cho hiệu suất
 - Thêm thời gian nạp
 - Những chương trình được lập trình tốt thì sẽ không dễ mặc định địa chỉ cơ sở khi biên dịch các DLL

DLL Rebasing Olly Memory Map

- DLL-A và DLL-B ưu tiên vị trí 0x100000000

00340000	00001000	DLL-B	.text	PE header code	Imag	R	RWE
00341000	00009000	DLL-B	.rdata	imports, exp.	Imag	R	RWE
0034A000	00002000	DLL-B	.data	data	Imag	R	RWE
0034C000	00003000	DLL-B	.rsrc	resources	Imag	R	RWE
0034F000	00001000	DLL-B	.reloc	relocations	Imag	R	RWE
00400000	00001000	EXE-1		PE header	Imag	R	RWE
00401000	00010000	EXE-1	.textbss	code	Imag	R	RWE
00411000	00004000	EXE-1	.text	SFX	Imag	R	RWE
00415000	00002000	EXE-1	.rdata		Imag	R	RWE
00417000	00001000	EXE-1	.data	data	Imag	R	RWE
00418000	00001000	EXE-1	.idata	imports	Imag	R	RWE
00419000	00001000	EXE-1	.rsrc	resources	Imag	R	RWE
10000000	00001000	DLL-A		PE header	Imag	R	RWE
10001000	00009000	DLL-A	.text	code	Imag	R	RWE
1000A000	00002000	DLL-A	.rdata	imports, exp.	Imag	R	RWE
1000C000	00003000	DLL-A	.data	data	Imag	R	RWE
1000F000	00001000	DLL-A	.rsrc	resources	Imag	R	RWE
10010000	00001000	DLL-A	.reloc	relocations	Imag	R	RWE

DLL-B is relocated into a different memory address from its requested location

DLL Rebasing

- IDA Pro không thể attached một tiến trình đang chạy như Ollydbg.
- Nó không biết về cơ chế Rebasing
- Nếu cùng sử dụng Ollydbg và IDA Pro thì có thể sẽ cho kết quả khác
 - Để khắc phục được điều này, hãy sử dụng tùy chọn “Manual Load” trong IDA Pro
 - Chỉ định địa chỉ ảo cơ sở một cách thủ công

Xem các Thread và Stack

☐ Thao tác: View, Threads

☐ Right-click a thread to "Open in CPU", kill it, etc..

The screenshot shows a Windows-style application window titled 'Threads'. The window contains a table with ten columns: Ord, Ident, Window's title, Last error, Entry, TIB, Suspend, Priority, User time, and System time. There are seven rows of data, each representing a thread from the 'Cisco Packet Tracer' process. The 'Ident' column shows thread IDs like 00000F34, 00000488, etc. The 'Window's title' column shows 'Cisco Packet Tracer'. The 'Last error' column shows 'ERROR_SUCCESS (00)'. The 'Entry' column shows memory addresses like 7029345E, 76E6EB16, etc. The 'TIB' column shows memory addresses like 7FFDF000, 7FFDE000, etc. The 'Suspend' column shows values like 0., 0., etc. The 'Priority' column shows 'Normal' for most threads and 'High' for one. The 'User time' and 'System time' columns show time values like 1.1544 s, 0.2964 s, etc.

Ord	Ident	Window's title	Last error	Entry	TIB	Suspend	Priority	User time	System time
Main	00000F34	Cisco Packet Tracer	ERROR_SUCCESS (00)	7FFDF000	7FFDF000	0.	Normal	1.1544 s	0.2964 s
2.	00000488		ERROR_SUCCESS (00)	7029345E	7FFDE000	0.	Normal	0.0000 s	0.0000 s
3.	000007C4		ERROR_SUCCESS (00)	76E6EB16	7FFD0000	0.	Normal	0.0000 s	0.0000 s
4.	00000414		ERROR_SUCCESS (00)	76E6D34E	7FFDC000	0.	Normal	0.0000 s	0.0000 s
5.	00000A80		ERROR_SUCCESS (00)	76E6D34E	7FFDB000	0.	Normal	0.0000 s	0.0000 s
6.	0000093C		ERROR_SUCCESS (00)	7680C890	7FFDA000	0.	Normal	0.0000 s	0.0000 s
7.	000008C8		ERROR_SUCCESS (00)	749E6F14	7FFD9000	0.	High	0.0000 s	0.0000 s

Xem các Thread và Stack

☐ Mỗi Thread đều có ngăn xếp riêng của nó

M	Memory map	Address	Size	Owner	Section	Contains	Type	Access	Initial
		05050000	00800000				Priv	RW	RW
		05850000	00A80000				Priv	RW	RW
		06820000	003FC000				Map	R	R
		06D10000	00002000				Priv	RW	Guar:
		06D1F000	00001000			Stack of thread 2. (00000488)	Priv	RW	Guar:
		06E10000	00002000				Priv	RW	Guar:
		06E1F000	00001000			Stack of thread 3. (000007C4)	Priv	RW	Guar:
		06F10000	00B80000				Priv	RW	RW
		07AD0000	006B5000				Priv	RW	RW
		08280000	00002000				Priv	RW	Guar:
		0828F000	00001000			Stack of thread 4. (00000414)	Priv	RW	RW
		08380000	00002000				Priv	RW	Guar:
		0838F000	00001000			Stack of thread 5. (00000A80)	Priv	RW	RW
		0848C000	00002000				Priv	RW	Guar:
		0848E000	00002000			Stack of thread 6. (0000098C)	Priv	RW	Guar:
		08580000	00002000				Priv	RW	Guar:
		0858F000	00001000			Stack of thread 7. (000008C8)	Priv	RW	RW
		08630000	00019000				Priv	RW	RW
		08670000	0021F000				Map	RW	RW
		088B0000	01C57000				Priv	RW	RW
		0D510000	001FA000				Priv	RW	RW

Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Chạy tiến trình

OllyDbg Code-Execution Options

Function	Menu	Hotkey	Button
Run/Play	Debug ▶ Run	F9	
Pause	Debug ▶ Pause	F12	
Run to selection	Breakpoint ▶ Run to Selection	F4	
Run until return	Debug ▶ Execute till Return	CTRL-F9	
Run until user code	Debug ▶ Execute till User Code	ALT-F9	
Single-step/step-into	Debug ▶ Step Into	F7	
Step-over	Debug ▶ Step Over	F8	

Run and Pause

- ❑ Có thể chạy một chương trình (Run) và bấm Pause bất cứ lúc nào
- ❑ Đặt breakpoints cho kết quả tốt hơn

Run and Run to Selection

- Lệnh Run cho phép tiếp tục thực thi chương trình sau một breakpoint
- Lệnh Run to Selection thực thi chương trình đến trước vị trí được chọn.

Duyệt mã thực thi

- F7 - Single-step: chạy từng lệnh một và quan sát mọi thứ diễn ra trong một chương trình
- F8 -Step-over: Chạy step by step, không thực thi từng lệnh trong nội dung hàm, thực thi toàn hàm và nhận giá trị trả về. Có thể bỏ qua một số đoạn code quan trọng.

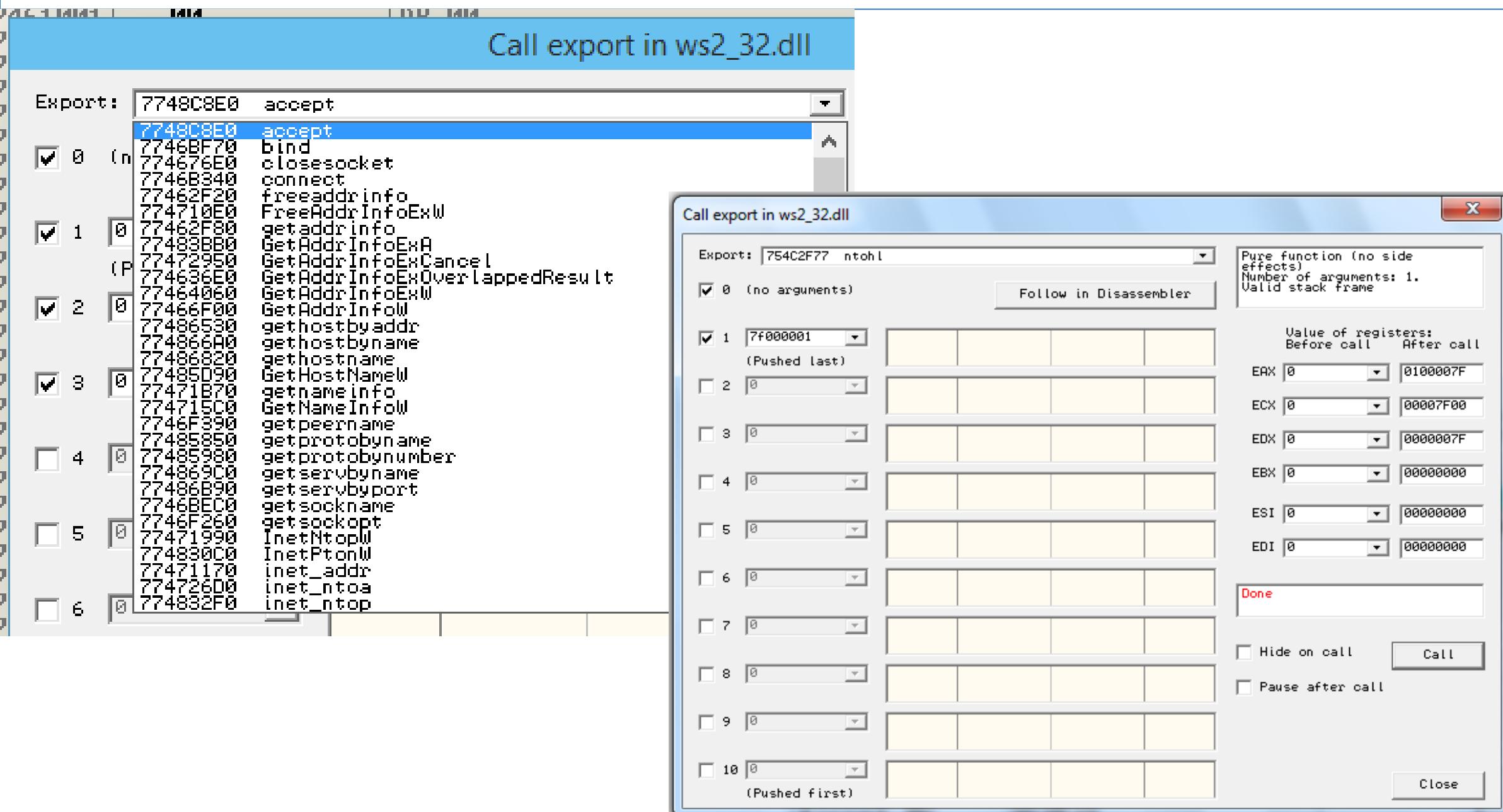
Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Tải các DLL

- ❑ DLLs không thể chạy trực tiếp
- ❑ OllyDbg sử dụng loaddll.exe để tải các dll

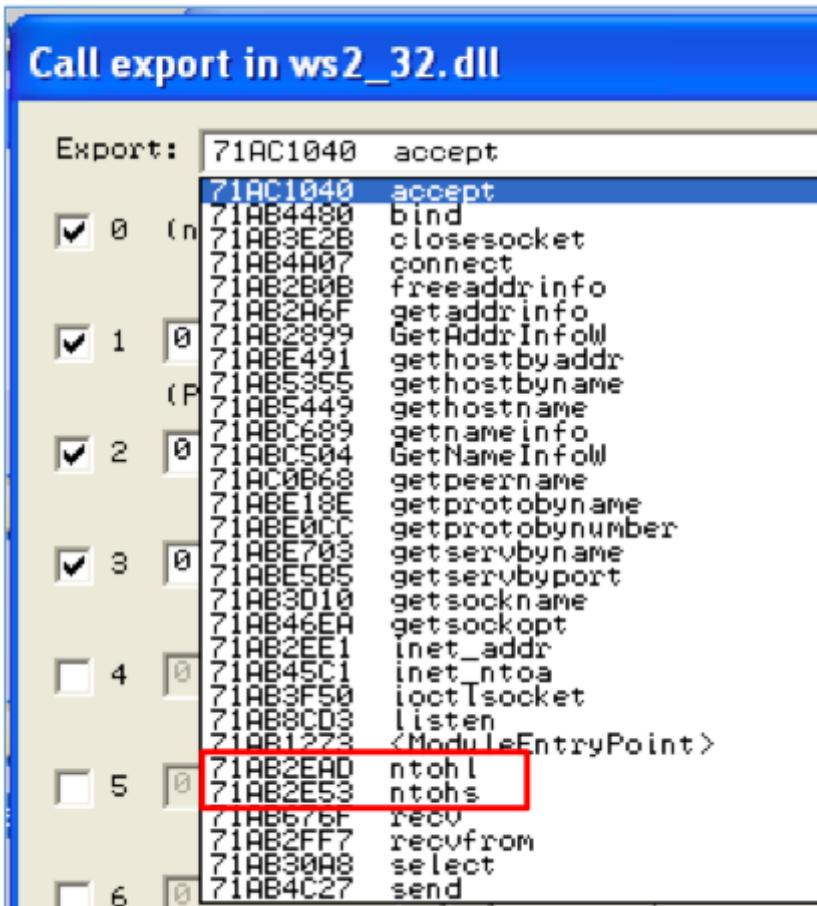
Tải các DLL



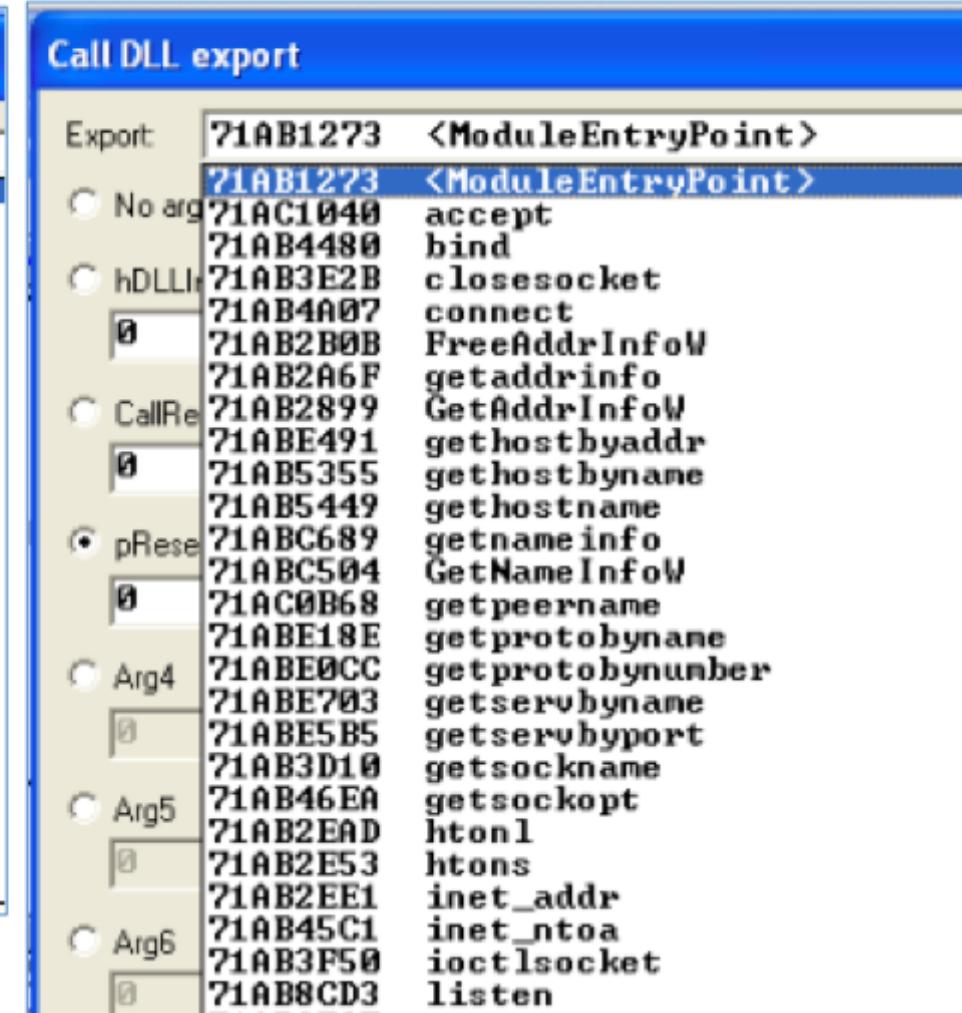
Tải các DLL

☐ Không sử dụng OllyDbg 2

OllyDbg 1.10



OllyDbg 2.01



Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Tracing

- ❑ Là một kỹ thuật Debugging mạnh mẽ
- ❑ Ghi lại các thông tin quá trình thực thi
- ❑ Các loại Tracing: Standard Back Trace, Call Stack Trace, Run Trace

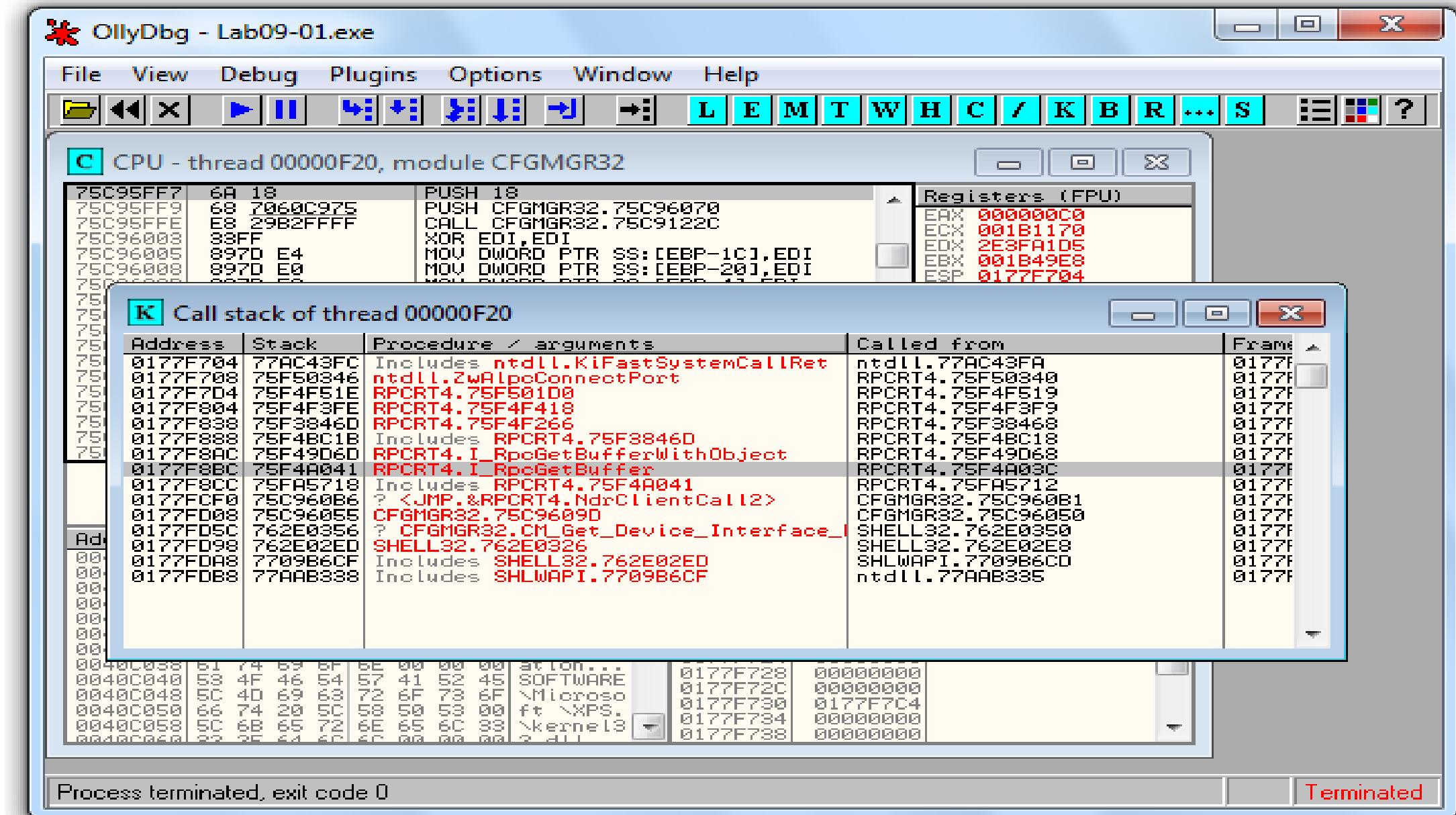
Standard Back Trace

- ❑ Chuyển qua Disassembler với các nút Step Into và Step Over
- ❑ Ollydbg sẽ ghi lại các thao tác đã thực hiện
- ❑ Sử dụng phím minus để xem hướng dẫn trước.
 - Nhưng sẽ không thấy được các giá trị của thanh ghi trước đó
- ❑ Phím Plus sẽ giúp chuyển tiếp
 - Nếu đã sử dụng Step Over thì sẽ không thể quay trở lại và quyết định Step into.

Call Stack Trace

- Xem những đường dẫn thực thi đến một hàm nhất định
- Click View, Call Stack
- Hiển thị chuỗi các gọi hàm để tiếp cận vị trí hiện tại

Call Stack Trace



Run Trace

- ❑ Code runs và Ollydbg lưu lại tất cả các lệnh thực thi và các thay đổi vào thanh ghi và các thanh ghi cờ
- ❑ Highlight những đoạn mã, Right-click, Run Trace, Add Selection
- ❑ Sau khi thực thi những đoạn code, View, Run Trace
 - Để xem các lệnh đã thực thi
 - Phím + và – dùng để tiến hoặc lùi

Run Trace

OllyDbg - Lab09-01.exe

File View Debug Plugins Options Window Help

CPU - main thread, module Lab09-01

Address	Command
004038C2	XOR EDX,EDX
004038C4	MOU DL,AH
004038C6	MOU DWORD PTR DS:[40EB7C],EDX
004038CC	MOU ECX,EAX
004038CE	AND ECX,0FF
004038D4	MOU DWORD PTR DS:[40EB78],ECX
004038DA	SHL ECX,8
004038DD	ADD ECX,EDX
004038DF	MOU DWORD PTR DS:[40EB74],ECX
004038E5	SHR EAX,10
004038E8	MOU DWORD PTR DS:[40EB70],EAX
004038ED	PUSH 0
004038EF	CALL Lab09-01.00406355
004038F4	POP ECX
004038F5	TEST EAX,EAX
004038F7	JNZ SHORT Lab09-01.00403901
004038F9	PUSH 1C
004038FB	CALL Lab09-01.0040399A
00403900	POP ECX

Run trace

Back	Thread	Module	Address	Command	Modified registers
9.	Main	Lab09-01	004038C4	MOU DL,AH	EDX=00000002
8.	Main	Lab09-01	004038C6	MOU DWORD PTR DS:[40EB7C],EDX	ECX=23F00206
7.	Main	Lab09-01	004038CC	MOU ECX,EAX	ECX=00000006
6.	Main	Lab09-01	004038CE	AND ECX,0FF	ECX=000000600
5.	Main	Lab09-01	004038D4	MOU DWORD PTR DS:[40EB78],ECX	ECX=000000602
4.	Main	Lab09-01	004038DA	SHL ECX,8	EAX=000023F0
3.	Main	Lab09-01	004038DD	ADD ECX,EDX	
2.	Main	Lab09-01	004038DF	MOU DWORD PTR DS:[40EB74],ECX	
1.	Main	Lab09-01	004038E5	SHR EAX,10	
0.	Main	Lab09-01	004038E8	MOU DWORD PTR DS:[40EB70],EAX	

Run Trace

- Code runs và Ollydbg lưu lại tất cả các lệnh thực thi và các thay đổi vào thanh ghi và các thanh ghi cờ
- Highlight những đoạn mã, Right-click, Run Trace, Add Selection
- Sau khi thực thi những đoạn code, View, Run Trace
 - Để xem các lệnh đã thực thi
 - Phím + và – dùng để tiến hoặc lùi

Run Trace

- Tự động Step Into / Step Over
- Dễ sử dụng hơn Add Sections
- Nếu không đặt các breakpoint, Ollydbg sẽ cố gắng theo dõi toàn bộ chương trình, có thể mất nhiều thời gian và bộ nhớ

Run Trace

☐ Đặt điều kiện

- Trace cho đến khi gặp một điều kiện
- Điều kiện này bắt một Poison Ivy shellcode, nó được cấp phát trên bộ nhớ ở dưới 0x400000

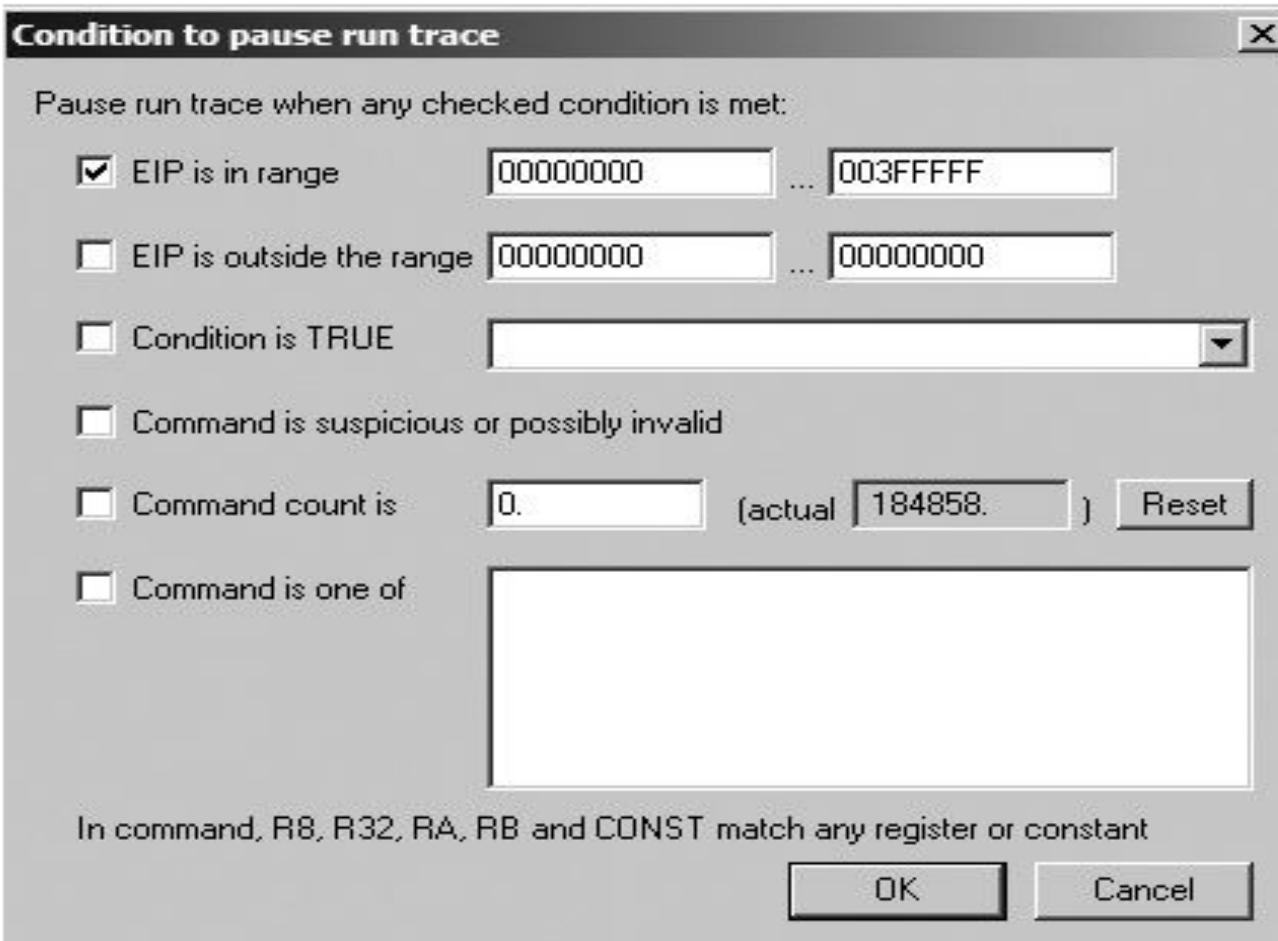


Figure 10-11. Conditional tracing

Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Ngoại lệ

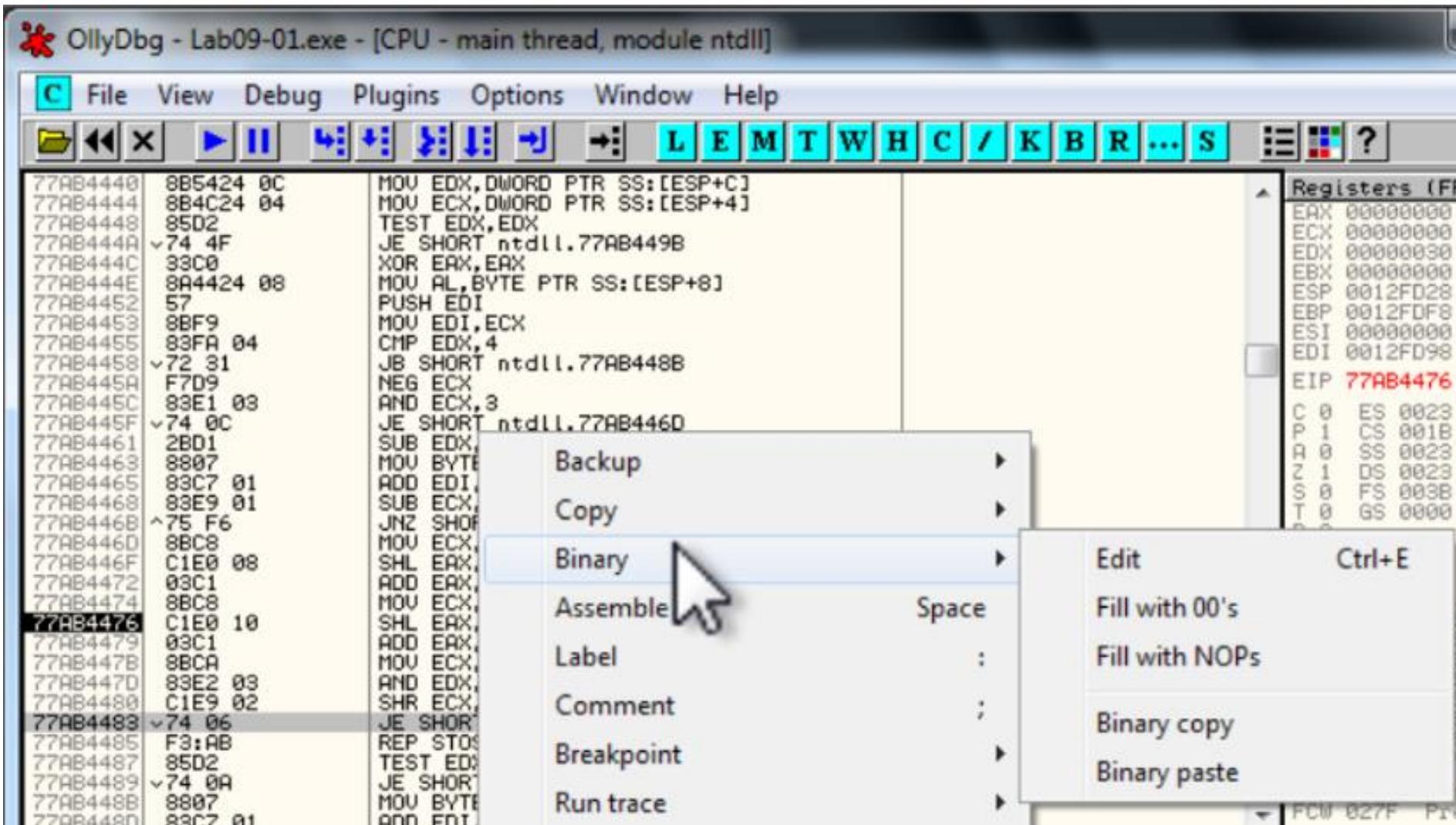
- Ollydbg sẽ dùng chương trình
- Có các tùy chọn để bỏ qua ngoại lệ:
 - Shift+F7 Step into exception – Nhảy vào bên trong ngoại lệ
 - Shift+F8: Step over exception – Nhảy qua ngoại lệ
 - Shift+F9: Run exception handler – chạy trình xử lý ngoại lệ
- Thường thì chỉ cần bỏ qua tất cả các ngoại lệ trong phân tích mã độc

Phân tích mã độc với OllyDbg

- Tải mã độc
- Giao diện OllyDbg
- Bản đồ bộ nhớ
- Chạy tiến trình
- Tải các DLL
- Tracing
- Ngoại lệ
- Patching

Patching

□ Binary edit



Nội dung

1. Gõ rối
2. Gõ rối mức mã nguồn và gõ rối mức mã Assembly
3. Gõ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gõ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gõ rối
7. Phân tích mã độc với OllyDbg

Mã độc

**Chương 5. Phân tích các chương
trình độc hại trên Window**

Mục tiêu

- Nhắc lại một số kiến thức về hệ điều hành Window
- Giới thiệu một số hành vi của mã độc chạy trên hệ điều hành Window

Tài liệu tham khảo

- [1] Michael Sikorski, Andrew Honig, 2012, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, (ISBN: 978-1593272906).
- [2] Sam Bowne, Slides for a college course at City College San Francisco,
https://samsclass.info/126/126_S17.shtml

Nội dung

- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

Nội dung

- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

Windows API

- Quản lý cách các chương trình tương tác với các thư viện của Microsoft
 - Handles
 - File System Functions
 - Special Files

Common API Types

Kiểu (Tiền tố)

- WORD (w) – 16 bits giá trị không âm
- DWORD (dw) – 32 bits giá trị không âm
- Handle (H) – Tham chiếu đến một đối tượng (Object)
- Long Pointer (LP) – Points to another type

Handles

- Handle được tạo bởi hệ điều hành
 - Giống như: Windows, Process, Menu, File,...
- Handles giống như con trỏ tới các đối tượng
- Điều duy nhất có thể làm với một handle là lưu trữ và sử dụng nó sau khi gọi hàm để tham chiếu đến một đối tượng

Handles

- Hàm **CreateWindowEx** trả về một **HWND**, một handle cho một cửa sổ
- Với handle đó có thể làm bất cứ thứ gì với cửa sổ mà nó đã tạo như: **DestroyWindow...**

File System Functions

- CreateFile, ReadFile, WriteFile**
 - Nhập/xuất với file thông thường
- CreateFileMapping, MapViewOfFile**
 - Thường được sử dụng bởi mã độc, tải tệp vào RAM
 - Có thể được sử dụng để thực thi một tệp tin mà không cần thông qua Windows loader

Special Files

- Các tệp được chia sẻ như \\server\share
 - Hoặc \\?\server\share
 - Ngừng phân tích cú pháp chuỗi, cho phép tên tập tin dài hơn
- Không gian tên (Namespace)
 - Các thư mục đặc biệt trên hệ thống tệp tin của windows
 - \ : Thư mục gốc chứa mọi thư
 - \\.\ : Thiết bị được lưu trữ sử dụng cho input/output
 - Sâu Witty đã viết vào \\.\PhysicalDisk1 để làm hỏng đĩa

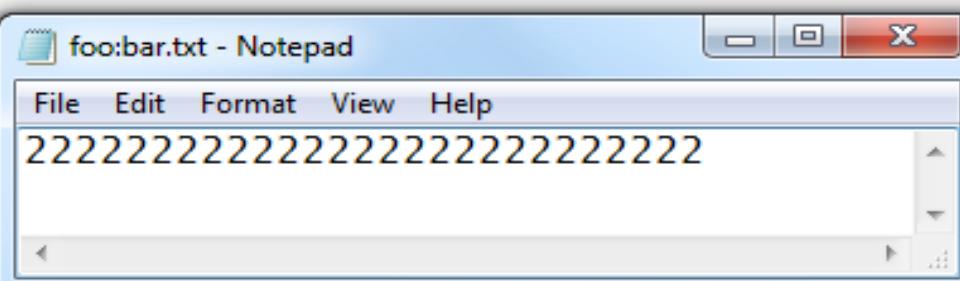
Special Files

☐ Điều khiển luồng

dữ liệu

☐ Dữ liệu được đẩy

vào files

```
Administrator: Command Prompt  
C:\Users\sam\ads>echo 1 > foo  
  
C:\Users\sam\ads>dir foo  
Volume in drive C is Win7  
Volume Serial Number is 80F8-F717  
  
Directory of C:\Users\sam\ads  
09/23/2013  05:31 PM              4 foo  
                   1 File(s)           4 bytes  
                   0 Dir(s)  78,679,588,864 bytes free  
  
C:\Users\sam\ads>echo 22222222222222222222222222222222 > foo:bar.txt  
  
C:\Users\sam\ads>dir foo  
Volume in drive C is Win7  
Volume Serial Number is 80F8-F717  
  
Directory of C:\Users\sam\ads  
09/23/2013  05:31 PM              4 foo  
                   1 File(s)           4 bytes  
                   0 Dir(s)  78,679,588,864 bytes free  
  
C:\Users\sam\ads>notepad foo:bar.txt  
  
C:\Users\sam\ads>  
  

```

Nội dung

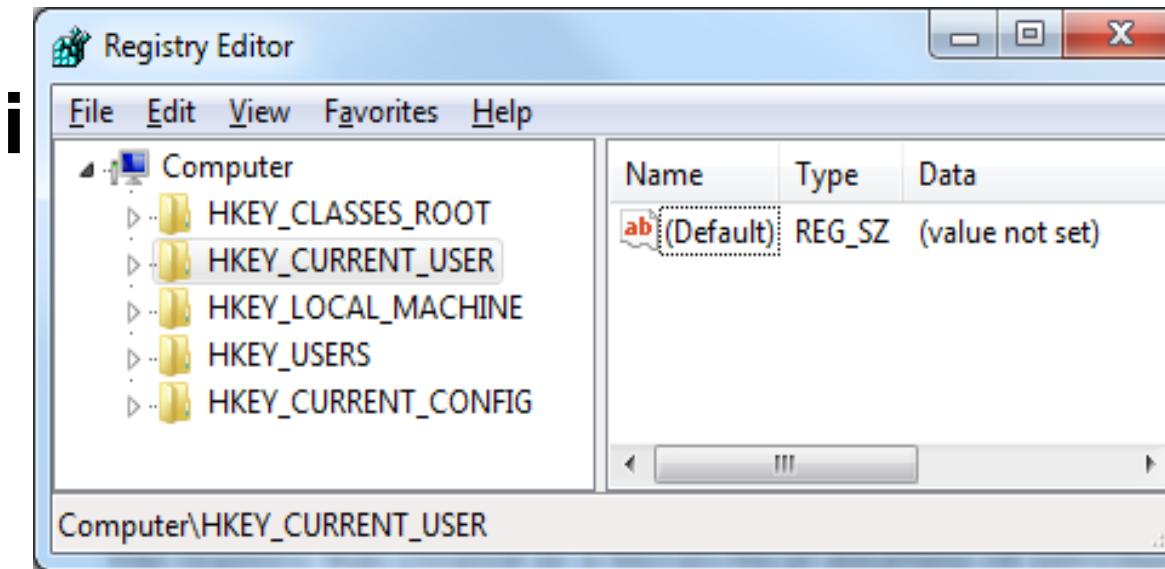
- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

Registry là gì

- Lưu trữ hệ điều hành và những cài đặt cấu hình của chương trình
 - Desktop background, mouse preferences, etc.
- Mã độc thường sử dụng Registry để duy trì sự tồn tại của nó trên hệ thống
 - Làm cho mã độc tự khởi động cùng hệ thống

Registry

- **ROOT KEYS** – Có 5 khóa chính
- **SUBKEY** – Mỗi khóa chính lại có các khóa con bên trong
- **KEY** – Thư mục, có thể chứa các thư mục khác hoặc giá trị
- **VALUE ENTRY** – Gồm hai phần: Name và Data
- **VALUE** hoặc **DATA** – Dữ liệu được lưu trữ trong Registry entry



Root Keys

HKEY_LOCAL_MACHINE (HKLM) Stores settings that are global to the local machine

HKEY_CURRENT_USER (HKCU) Stores settings specific to the current user

HKEY_CLASSES_ROOT Stores information defining types

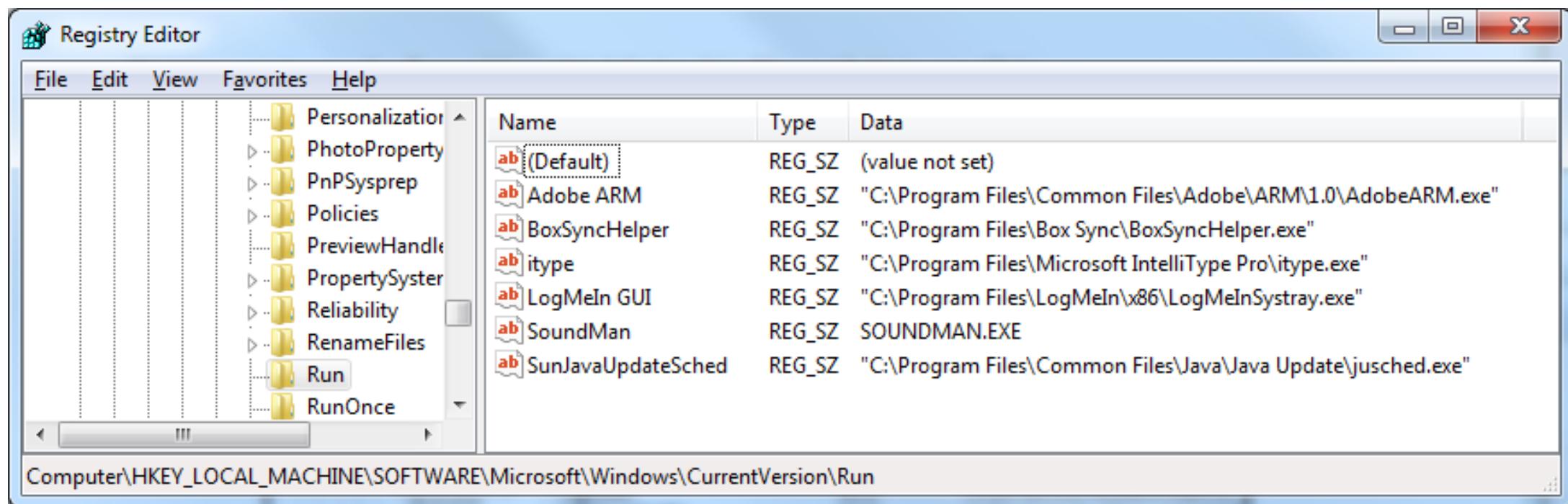
HKEY_CURRENT_CONFIG Stores settings about the current hardware configuration, specifically differences between the current and the standard configuration

HKEY_USERS Defines settings for the default user, new users, and current users

Run Key

□ HKLM\SOFTWARE\Microsoft\Windows\ \CurrentVersion\Run

- Các chương trình thực thi được khởi chạy khi người dùng đăng nhập vào hệ thống



Common Registry Functions

❑ RegOpenKeyEx

- Mở một Registry key để chỉnh sửa và truy vấn

❑ RegSetValueEx

- Thêm một giá trị mới vào Registry và set dữ liệu cho nó

❑ RegGetValue

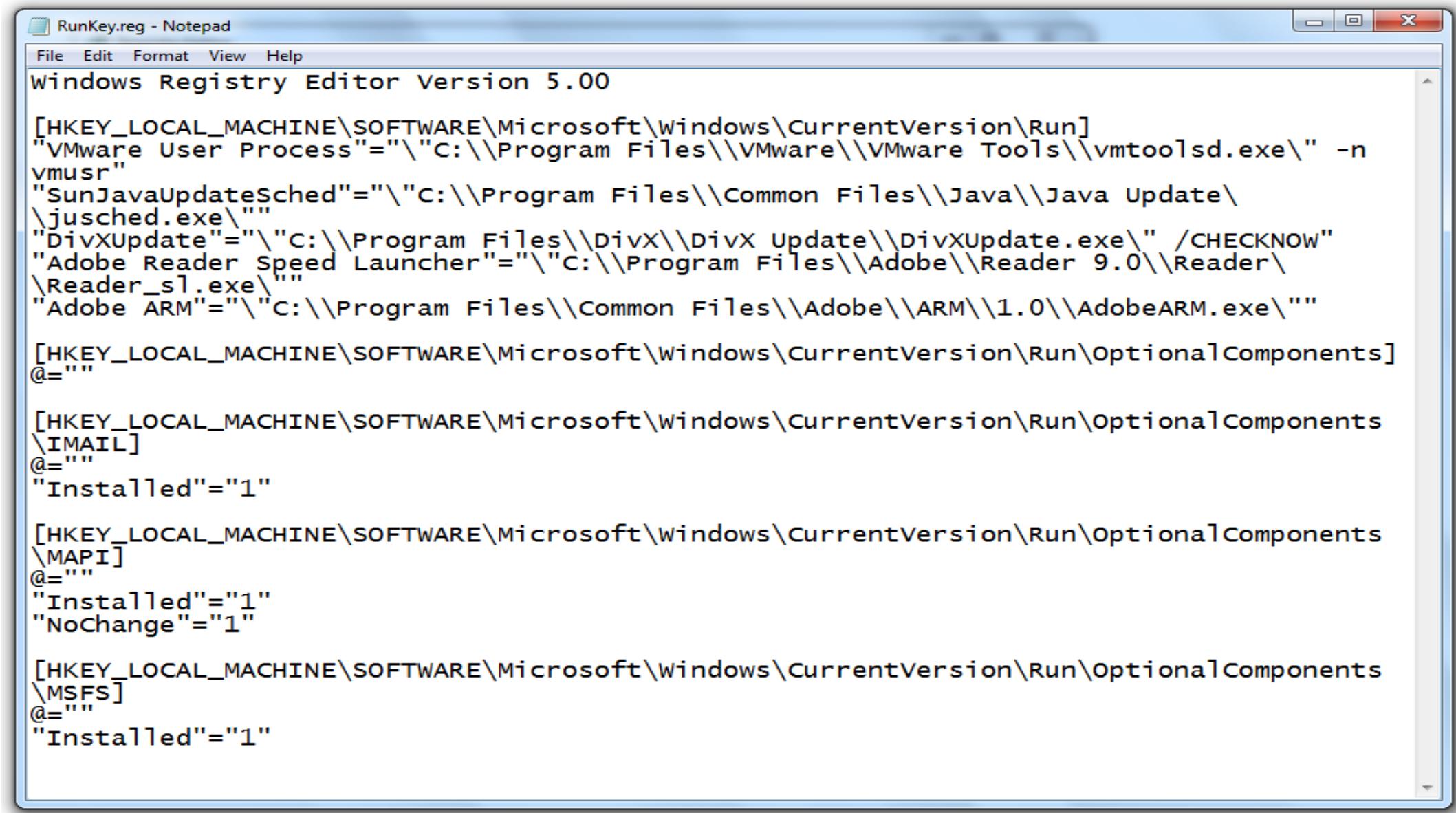
- Trả về dữ liệu cho một entry trong Registry

Registry Code

Code that modifies registry settings

```
0040286F    push    2          ; samDesired  
00402871    push    eax        ; ulOptions  
00402872    push    offset SubKey ;  
"Software\\Microsoft\\Windows\\CurrentVersion\\Run"  
00402877    push    HKEY_LOCAL_MACHINE ; hKey
```

.REG Files



The screenshot shows a Windows Notepad window titled "RunKey.reg - Notepad". The window contains the content of a .REG file, which is a Windows Registry Editor file. The file includes registry keys for the Run and OptionalComponents sections of the Windows registry.

```
RunKey.reg - Notepad
File Edit Format View Help
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"VMware User Process"="\"C:\\Program Files\\VMware\\VMware Tools\\vmtoolsd.exe\" -n
vmusr"
"SunJavaUpdateSched"="\"C:\\Program Files\\Common Files\\Java\\Java Update\\
\\jusched.exe\""
"DivXUpdate"="\"C:\\Program Files\\DivX\\DivX Update\\DivXUpdate.exe\" /CHECKNOW"
"Adobe Reader Speed Launcher"="\"C:\\Program Files\\Adobe\\Reader 9.0\\Reader\\
\\Reader_s1.exe\""
"Adobe ARM"="\"C:\\Program Files\\Common Files\\Adobe\\ARM\\1.0\\AdobeARM.exe\""

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents]
@=""

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\optionalComponents
\IMAIL]
@=""
"Installed"="1"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\optionalComponents
\MAPI]
@=""
"Installed"="1"
"NoChange"="1"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\optionalComponents
\MSFS]
@=""
"Installed"="1"
```

Nội dung

- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

Các API xử lý kết nối mạng

- ❑ Berkeley Compatible Sockets
- ❑ The WinINet API

Các API xử lý kết nối mạng

- ❑ Berkeley Compatible Sockets
- ❑ The WinINet API

Berkeley Compatible Sockets

- Thư viện Winsock, chủ yếu là ws2_32.dll
- Hầu như là giống nhau trên cả Windows và Unix

Berkeley Compatible Sockets

Function Description

`socket` Creates a socket

`bind` Attaches a socket to a particular port, prior to the `accept` call

`listen` Indicates that a socket will be listening for incoming connections

`accept` Opens a connection to a remote socket and accepts the connection

`connect` Opens a connection to a remote socket; the remote socket must be waiting for the connection

`recv` Receives data from the remote socket

`send` Sends data to the remote socket

Server and Client Sides

□ Server side

- Duy trì và luôn mở socket đợi kết nối từ client
- Các hàm thường được gọi theo thứ tự: **socket, bind, listen, accept**
- Sau đó **send** và **recv** khi cần thiết

□ Client side

- Kết nối vào một socket đang chờ
- Gọi các hàm theo thứ tự: **socket, connect**
- Sau đó **send** và **recv** khi cần thiết

Simplified Server Program

```
00401041 push    ecx          ; lpWSAData
00401042 push    202h         ; wVersionRequested
00401047 mov     word ptr [esp+250h+name.sa_data], ax
0040104C call    ds:WSAStartup
00401052 push    0             ; protocol
00401054 push    1             ; type
00401056 push    2             ; af
00401058 call    ds:socket
0040105E push    10h          ; namelen
00401060 lea     edx, [esp+24Ch+name]
00401064 mov     ebx, eax
00401066 push    edx          ; name
00401067 push    ebx          ; s
00401068 call    ds:bind
0040106E mov     esi, ds:listen
00401074 push    5             ; backlog
00401076 push    ebx          ; s
00401077 call    esi ; listen
00401079 lea     eax, [esp+248h+addrlen]
0040107D push    eax          ; addrlen
0040107E lea     ecx, [esp+24Ch+hostshort]
00401082 push    ecx          ; addr
00401083 push    ebx          ; s
00401084 call    ds:accept
```

Các API xử lý kết nối mạng

- ❑ Berkeley Compatible Sockets
- ❑ The WinINet API

The WinINet API

- ❑ Thuộc dạng API mức cao hơn Winsock
- ❑ Các hàm trong Wininet.dll sử dụng với các chương trình ở mức ứng dụng: HTTP, FTP, SMTP, POP,...
- ❑ Một số hàm
 - InternetOpen – Mở kết nối internet
 - InternetOpenURL – Kết nối đến một đường dẫn
 - InternetReadFile - Đọc dữ liệu từ tập tin đã tải

Nội dung

- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

Phân tích mã độc trên Windows

Việc thực thi code được chuyển sang cho một đối tượng khác xử lý, có những cách để chuyển việc thực thi đó như:

- DLLs
- Processes
- Threads
- Mutexes
- Services

DLLs (Dynamic Link Libraries)

- Chia sẻ và dùng chung những đoạn code giữa nhiều ứng dụng
- DLLs export code có thể được sử dụng bởi các ứng dụng
- Static libraries được sử dụng trước DLLs
 - Chúng vẫn tồn tại nhưng ít phổ biến hơn
 - Static libraries thì không chia sẻ bộ nhớ giữa các tiến trình
 - Các Static libraries sử dụng nhiều RAM hơn các DLLs

Ưu điểm của DLL

- Sử dụng DLL có sẵn trong windows giúp cho kích thước của chương trình được nhỏ hơn
- Các công ty về phần mềm cũng có thể tạo ra những DLL tùy chỉnh

Cách mã độc sử dụng DLLs

- Lưu trữ những đoạn mã độc hại trong DLL
 - Đôi khi những DLL độc hại được nạp vào các tiến trình
- Sử dụng Windows DLLs
 - Các mã độc hầu hết đều sử dụng những DLL cơ bản
- Sử dụng DLL của bên thứ 3
 - Sử dụng FireFox để kết nối tới server thay vì Windows API

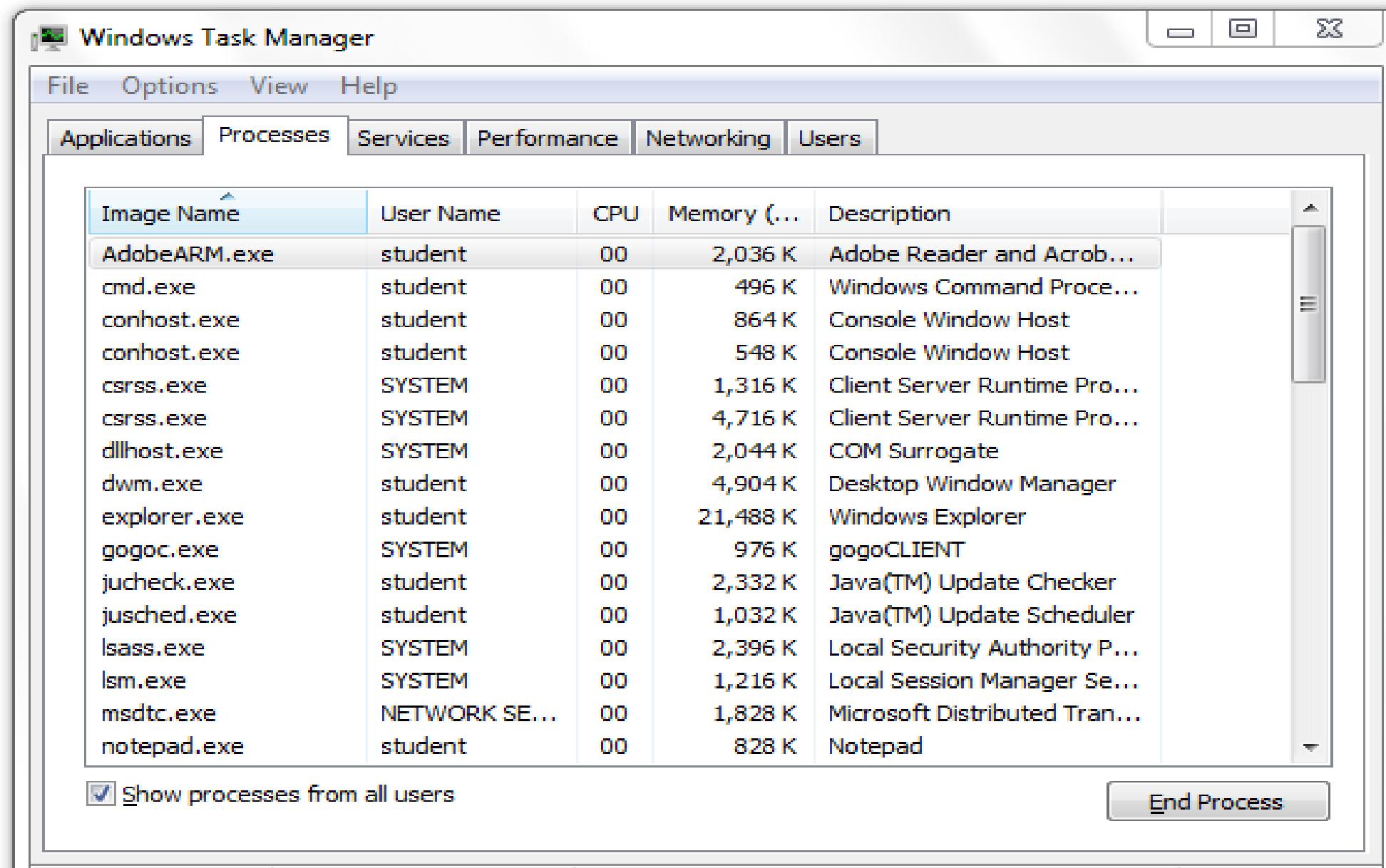
Cấu trúc DLL cơ bản

- Cấu trúc của DLL gần giống với EXEs
- Định dạng file thực thi – PE File
- Một cờ chỉ ra rằng nó là DLL chứ không phải EXE
- DLL có nhiều exports và ít exports
- DLLmain là một hàm chính, không export nhưng được chỉ định là Entry point trong PE Header
 - Được gọi khi một hàm nạp hoặc gỡ bỏ thư viện

Processes

- ❑ Tiến trình là những chương trình đang chạy bởi windows
- ❑ Mỗi tiến trình đều có các tài nguyên riêng (Handles, bộ nhớ,...)
- ❑ Mỗi tiến trình có một hoặc nhiều luồng xử lý
- ❑ Những mã độc cũ thường chạy như một tiến trình độc lập
- ❑ Những mã độc mới thực thi mã của nó như là một phần của tiến trình

Processes



Quản lý bộ nhớ

- Mỗi tiến trình đều sử dụng tài nguyên hệ thống như: CPU, File System và bộ nhớ,...
- Hệ điều hành sẽ cấp phát vùng nhớ cho mỗi tiến trình
- Hai tiến trình cùng truy cập vào cùng địa chỉ bộ nhớ nhưng thực sự là truy cập vào vị trí khác nhau trên RAM
 - Virtual address space

Tạo một Process mới

- Tạo ra một remote shell với một lời gọi hàm
- Tham số STARTUPINFO chưa các handle cho nhập/xuất chuẩn và standard error streams.
 - Có thể được thiết lập một socket, tạo một remote shell

Tạo một Shell

Sample code using the CreateProcess call

```
004010DA  mov      eax, dword ptr [esp+58h+SocketHandle]
004010DE  lea      edx, [esp+58h+StartupInfo]
004010E2  push     ecx          ; lpProcessInformation
004010E3  push     edx          ; lpStartupInfo
004010E4  1mov    [esp+60h+StartupInfo.hStdError], eax
004010E8  2mov    [esp+60h+StartupInfo.hStdOutput], eax
004010EC  3mov    [esp+60h+StartupInfo.hStdInput], eax
004010F0  4mov    eax, dword_403098
004010F5  push     0           ; lpCurrentDirectory
004010F7  push     0           ; lpEnvironment
004010F9  push     0           ; dwCreationFlags
004010FB  mov      dword ptr [esp+6Ch+CommandLine], eax
```

Nạp socket handle, StdError, StdOutput và StdInput thành
lpProcessInformation.

Tạo một Shell

```
004010FF push    1          ; bInheritHandles  
00401101 push    0          ; lpThreadAttributes  
00401103 lea     eax, [esp+74h+CommandLine]  
00401107 push    0          ; lpProcessAttributes  
00401109 5push   eax        ; lpCommandLine  
0040110A push    0          ; lpApplicationName  
0040110C mov     [esp+80h+StartupInfo.dwFlags], 101h  
00401114 6call   ds>CreateProcessA
```

- **CommandLine contains**
- **Nó được thực thi khi hàm CreateProcess được gọi**

Threads

❑ Các tiến trình chính là các containers

- Mỗi tiến trình chứa một hoặc nhiều Thread

❑ Thread là những gì Windows thực sự thực thi

❑ Thread (luồng)

- Chuỗi các lệnh độc lập
- Được thực thi bởi CPU mà không cần phải đợi các thread khác
- Các thread trong cùng một tiến trình sẽ dùng chung vùng nhớ
- Mỗi thread có thanh ghi và Stack riêng

Thread Context

- ❑ Khi một thread đang chạy nó có toàn quyền kiểm soát CPU
- ❑ Các thread khác không làm ảnh hưởng đến trạng thái của CPU
- ❑ Khi một thread thay đổi một thanh ghi, nó không ảnh hưởng đến thread khác
- ❑ Khi hệ điều hành chuyển qua thread khác, nó lưu lại tất cả các giá trị của CPU trong một cấu trúc (struct) được gọi là thread context

Tạo một Thread

- CreateThread
- Hàm gọi đã chỉ định một địa chỉ bắt đầu, còn được gọi là một hàm bắt đầu

Cách mã độc sử dụng Thread

- Sử dụng CreateThread để tải một DLL độc hại vào một tiến trình
- Tạo hai thread cho input và output
 - Sử dụng để giao tiếp với một ứng dụng đang chạy

Mutexes

- ❑ Mutexes là các đối tượng toàn cục, điều phối các tiến trình và thread
- ❑ Ở kernel, chúng được gọi là mutants
- ❑ Mutexes thường sử dụng hard-coded để xác định mã độc

Functions for Mutexes

□ WaitForSingleObject

- Cung cấp một thread truy cập vào mutex
- Bất kỳ các thread con khác truy cập vào nó phải đợi

□ ReleaseMutex

- Được gọi khi một thread hoàn tất sử dụng mutex

□ CreateMutex

□ OpenMutex

- Get một handle để xử lý những tiến trình mutex khác

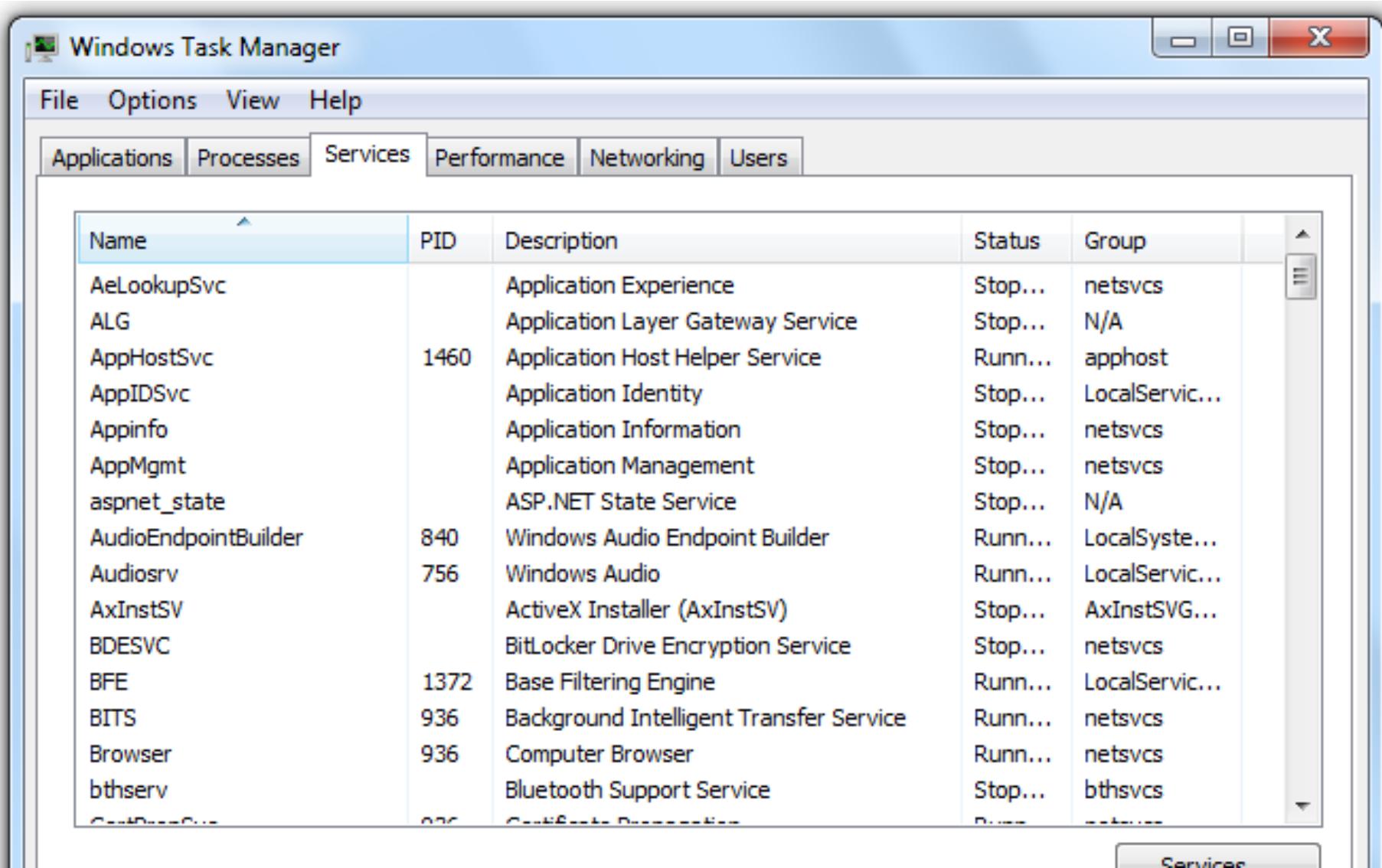
Kiểm tra tiến trình đang chạy

- OpenMutex kiểm tra có HGL345 tồn tại hay không
- Nếu không, nó sẽ được tạo ra với CreateMutex
- test eax, eax – Set cờ Z nếu eax bằng 0

```
00401007  push  1F0001h          ; dwDesiredAccess
0040100C  1call  ds:_imp_OpenMutexW@12 ;
OpenMutexW(x,x,x)
00401012  2test  eax, eax
00401014  3jz    short loc_40101E
00401016  push   0              ; int
00401018  4call  ds:_imp_exit
0040101E  push   offset Name    ; "HGL345"
00401023  push   0              ; bInitialOwner
00401025  push   0              ; lpMutexAttributes
00401027  5call  ds:_imp_CreateMutexW@12 ;
CreateMutexW(x,x,x)
```

Services

- ☐ Dịch vụ chạy nền mà không có đầu vào của người dùng



The screenshot shows the Windows Task Manager window with the 'Services' tab selected. The window title is 'Windows Task Manager'. The menu bar includes File, Options, View, and Help. Below the menu is a tab bar with Applications, Processes, Services (which is highlighted), Performance, Networking, and Users. The main area is a table listing various system services:

Name	PID	Description	Status	Group
AeLookupSvc		Application Experience	Stop...	netsvcs
ALG		Application Layer Gateway Service	Stop...	N/A
AppHostSvc	1460	Application Host Helper Service	Runn...	apphost
AppIDSvc		Application Identity	Stop...	LocalServic...
Appinfo		Application Information	Stop...	netsvcs
AppMgmt		Application Management	Stop...	netsvcs
aspnet_state		ASP.NET State Service	Stop...	N/A
AudioEndpointBuilder	840	Windows Audio Endpoint Builder	Runn...	LocalSyste...
Audiosrv	756	Windows Audio	Runn...	LocalServic...
AxInstSV		ActiveX Installer (AxInstSV)	Stop...	AxInstSVG...
BDESVC		BitLocker Drive Encryption Service	Stop...	netsvcs
BFE	1372	Base Filtering Engine	Runn...	LocalServic...
BITS	936	Background Intelligent Transfer Service	Runn...	netsvcs
Browser	936	Computer Browser	Runn...	netsvcs
bthserv		Bluetooth Support Service	Stop...	bthsvcs
cryptsvc	820	Cryptographic Service	Runn...	...

SYSTEM Account

- Dịch vụ thường chạy ở mức hệ thống, thậm chí còn mạnh hơn cả Administrator
- Dịch vụ có thể tự khởi chạy khi windows khởi động
 - Đây là một cơ chế để mã độc có thể tận dụng để duy trì sự tồn tại của nó trên hệ thống
 - Mã độc vẫn tồn tại khi hệ thống khởi động lại

Service API Functions

□ OpenSCManager

- Trả về một handle cho Service Control Manager

□ CreateService

- Thêm một service mới vào Service Control Manager
- Có thể xác định dịch vụ sẽ tự khởi động khi khởi động

□ StartService

- Chỉ được sử dụng nếu service được thiết lập khởi động
một cách thủ công

Svchost.exe

WIN32_SHARE_PROCESS

- ❑ Là một loại service phổ biến nhất, mục tiêu của mã độc
- ❑ Lưu trữ code cho service trong một DLL
- ❑ Kết hợp một số service vào một tiến trình chia sẻ duy nhất có tên svchost.exe

Svchost.exe

Process Explorer - Sysinternals: www.sysinternals.com [W7\student]

File	Options	View	Process	Find	DLL	Users	Help
Process		PID	CPU	Private Bytes	Working Set	Description	
System Idle Process		0	97.61	0 K	24 K		
System		4	0.15	44 K	672 K		
Interrupts		n/a	0.42	0 K	0 K	Hardware Inter	
smss.exe		260		224 K	792 K	Windows Sess	
csrss.exe		352		2,472 K	4,160 K	Client Server R	
wininit.exe		404		892 K	3,360 K	Windows Start	
services.exe		508		4,312 K	6,512 K	Services and C	
svchost.exe		640		2,904 K	7,208 K	Host Process f	
WmiPrvSE.exe		3736		1,768 K	4,752 K	WMI Provider I	
svchost.exe		708		3,196 K	6,716 K	Host Process f	
svchost.exe		756		14,268 K	14,420 K	Host Process f	
audiodg.exe		1680		15,016 K	14,024 K	Windows Audio	
svchost.exe		840	< 0.01	44,436 K	50,672 K	Host Process f	
dwm.exe		2848	0.20	88,212 K	34,328 K	Desktop Windo	
svchost.exe						fi	
svchost.exe						fi	
svchost.exe						fi	
spoolsv.exe						fi	
svchost.exe						fi	
svchost.exe						fi	
gogoc.exe						fi	
salwriter.exe						fi	

Command Line:
C:\Windows\System32\svchost.exe +k LocalSystemNetworkRestricted

Path:
C:\Windows\System32\svchost.exe (LocalSystemNetworkRestricted)

Services:

- Desktop Window Manager Session Manager [UxSms]
- Distributed Link Tracking Client [TrkWks]
- Network Connections [Netman]
- Offline Files [CscService]
- Program Compatibility Assistant Service [PcaSvc]
- Remote Desktop Services UserMode Port Redirector [UmRdpService]
- Superfetch [SysMain]
- Windows Audio Endpoint Builder [AudioEndpointBuilder]
- Windows Driver Foundation - User-mode Driver Framework [wudfsvc]

Một số service khác

WIN32_OWN_PROCESS

- Chạy như một EXEs trong một tiến trình độc lập

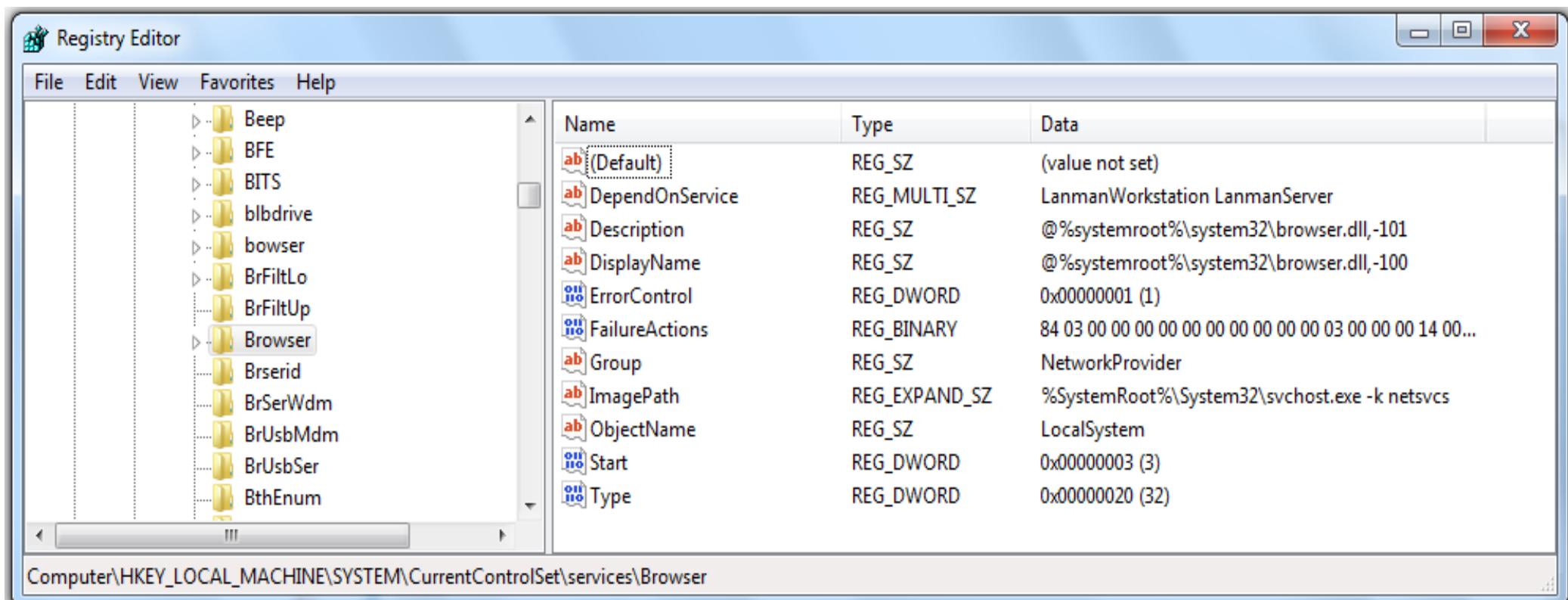
KERNEL_DRIVER

- Được sử dụng để load code vào kernel

Thông tin về Service trong Registry

HKLM\System\CurrentControlSet\Services

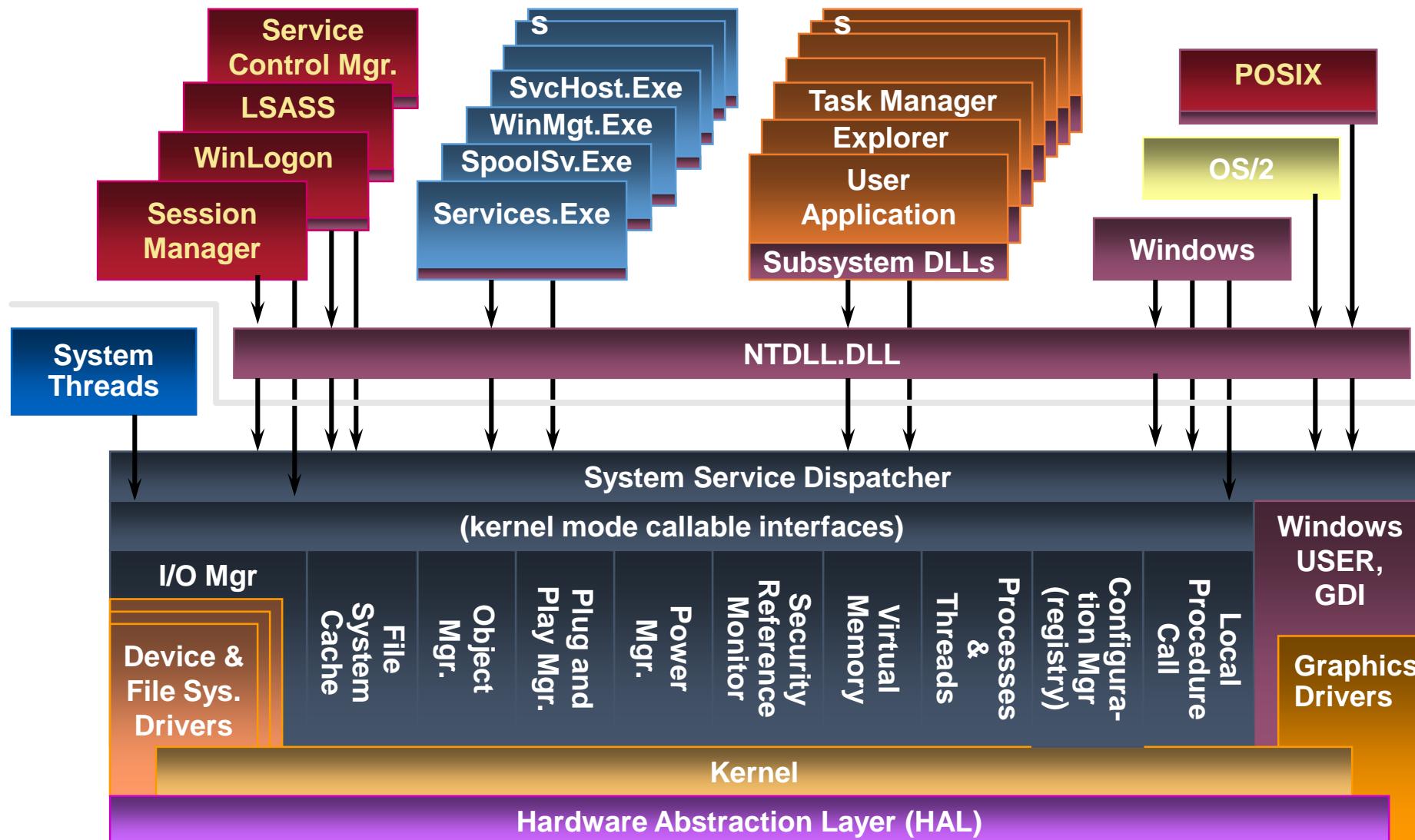
- Bắt đầu giá trị = 0x03 cho “Load on Demand”
- Type = 0x20 cho WIN32_SHARE_PROCESS



Nội dung

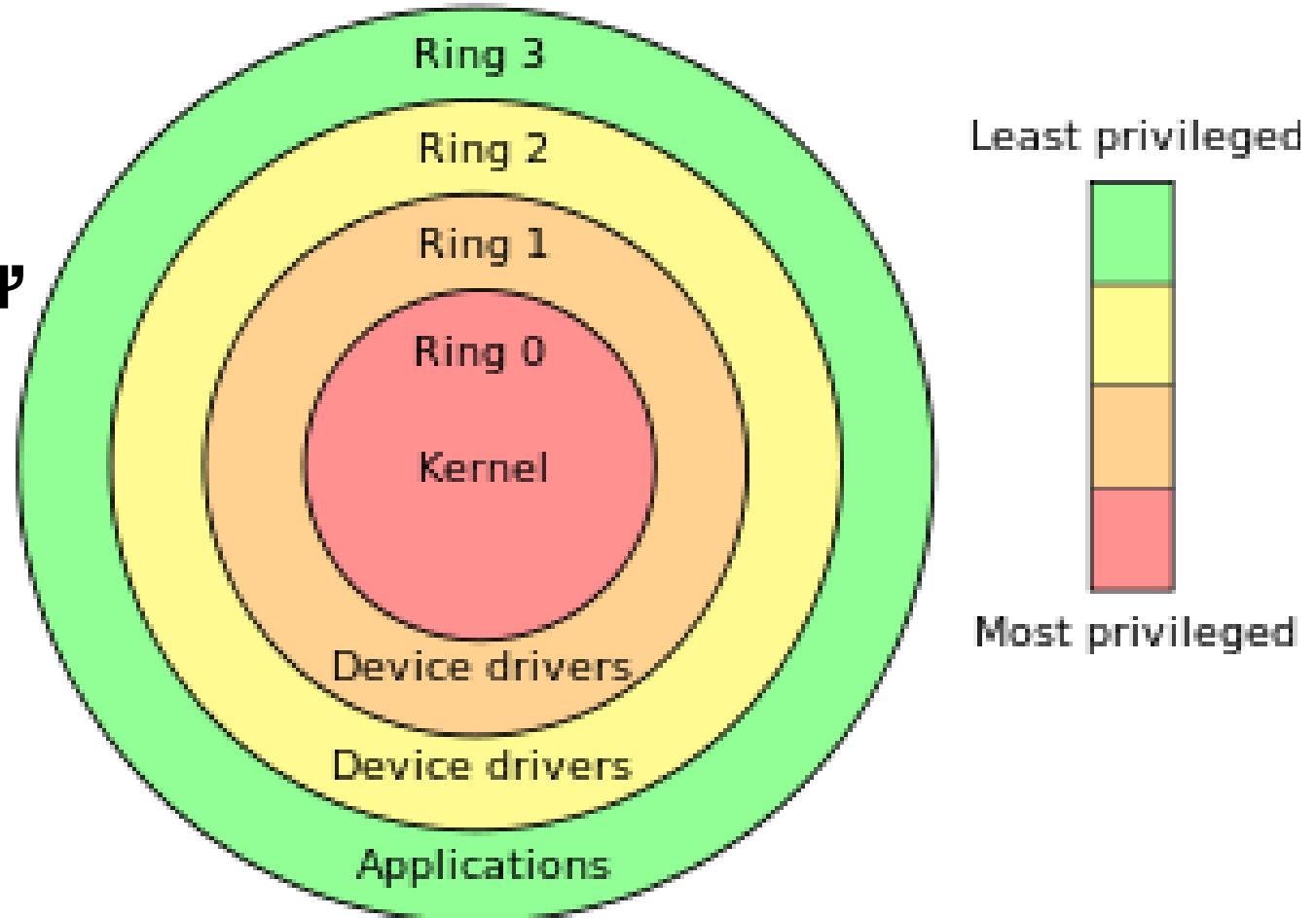
- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

System Architecture



Privilege Levels

- Ring 0: Kernel Mode
- Ring 3: User Mode
- Ring 1 & 2: không sử dụng bởi Windows



User mode

- Hầu như tất cả code đều chạy ở User-mode
 - Ngoại trừ hệ điều hành và các driver phần cứng, chúng chạy ở Kernel-mode
- Chế độ User-mode không thể truy cập phần cứng trực tiếp
- Ở User-mode một số lệnh CPU bị hạn chế
- Chỉ có thể thao tác với phần cứng không qua Windows API.

User mode processes

- ❑ Mỗi tiến trình có vùng nhớ của riêng nó, quyền hạn nhất định và tài nguyên riêng
- ❑ Các tiến trình ở User-mode không can thiệp được vào vùng nhớ của nhau
- ❑ Nếu một chương trình ở User-mode thực thi những lệnh không hợp lệ và gây Crash chương trình thì hệ điều hành Windows có thể khôi phục lại các tài nguyên và chấm dứt chương trình.

Calling the Kernel

- ❑ Không thể nhảy trực tiếp từ User-mode sang Kernel
- ❑ SYSENTER, SYSCALL hoặc lệnh INT 0x2E sử dụng bảng tra cứu để xác định trước các hàm.

Kernel Processes

- ❑ Tất cả các tiến trình ở Kernel-mode đều chia sẻ tài nguyên và địa chỉ bộ nhớ cho nhau
- ❑ Các tiến trình ở kernel-mode có quyền cao và được ưu tiên hơn các tiến trình ở user-mode
- ❑ Kiểm tra bảo mật ít hơn

Kernel Processes

- ❑ Ở Kernel-mode nếu thực thi một lệnh không hợp lệ, hệ điều hành sẽ gặp sự cố với màn hình xanh chết chóc
- ❑ Các phần mềm anti-virus và Firewall chạy ở chế độ Kernel-mode

Malware in Kernel Mode

- ❑ Loại mã độc này nguy hiểm hơn mã độc chạy ở chế độ User-mode
- ❑ Auditing không áp dụng được với kernel
- ❑ Hầu hết các Rootkits đều sử dụng kernel code
- ❑ Tuy nhiên phần lớn mã độc không bắt gặp nhiều ở kernel mode mà chủ yếu gắp user mode

Nội dung

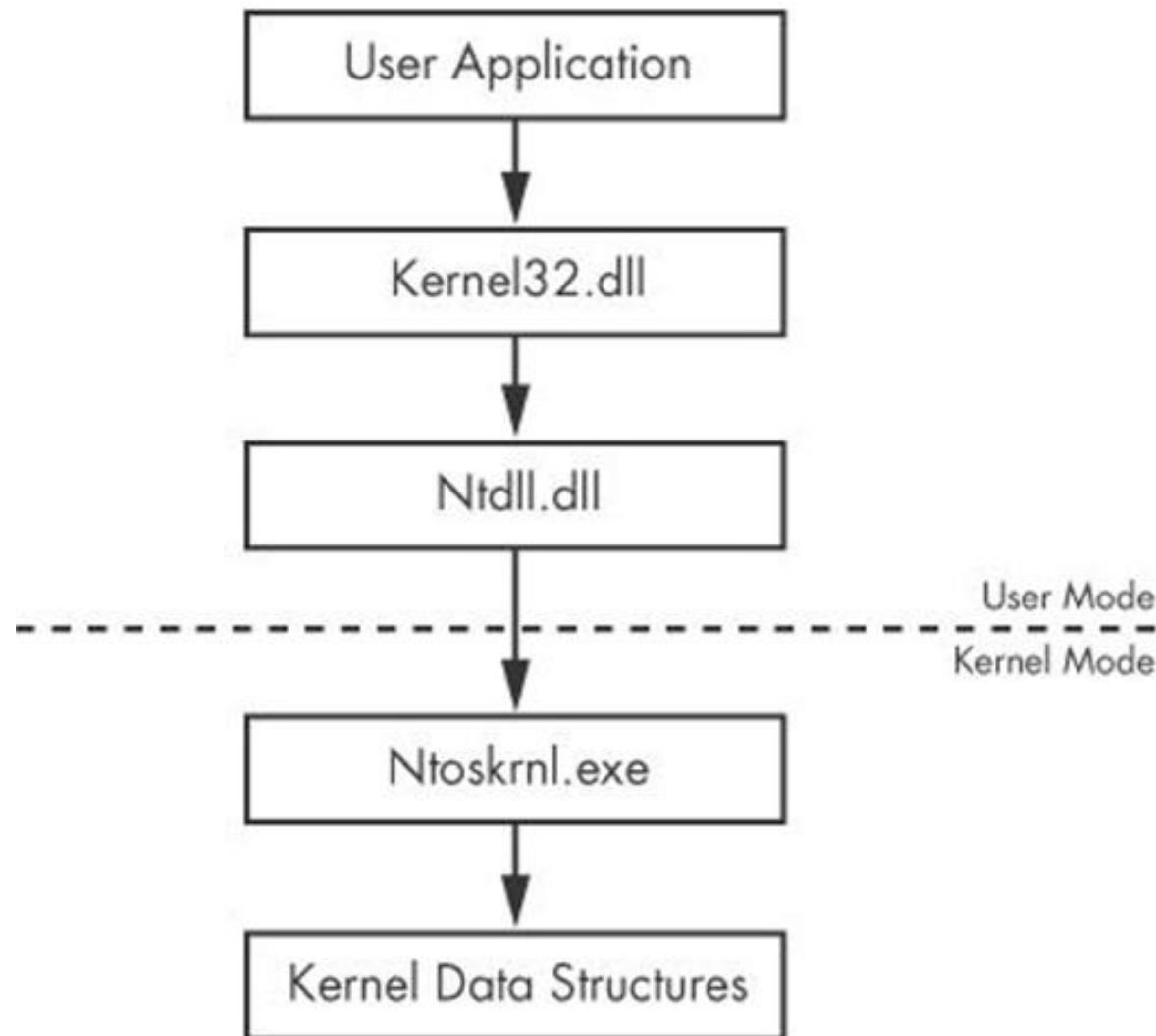
- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

Các native API

- Là những API mức thấp để tương tác với windows
- mức sâu: tiến trình, bộ nhớ,...
- Các chương trình thông thường thì hiếm khi sử dụng
- Thường gặp phổ biến ở mã độc

Ntdll.dll

- ❑ Ntdll.dll quản lý tương tác giữa user-mode và kernel-mode
- ❑ Các hàm của Ntdll tạo thành các Native API

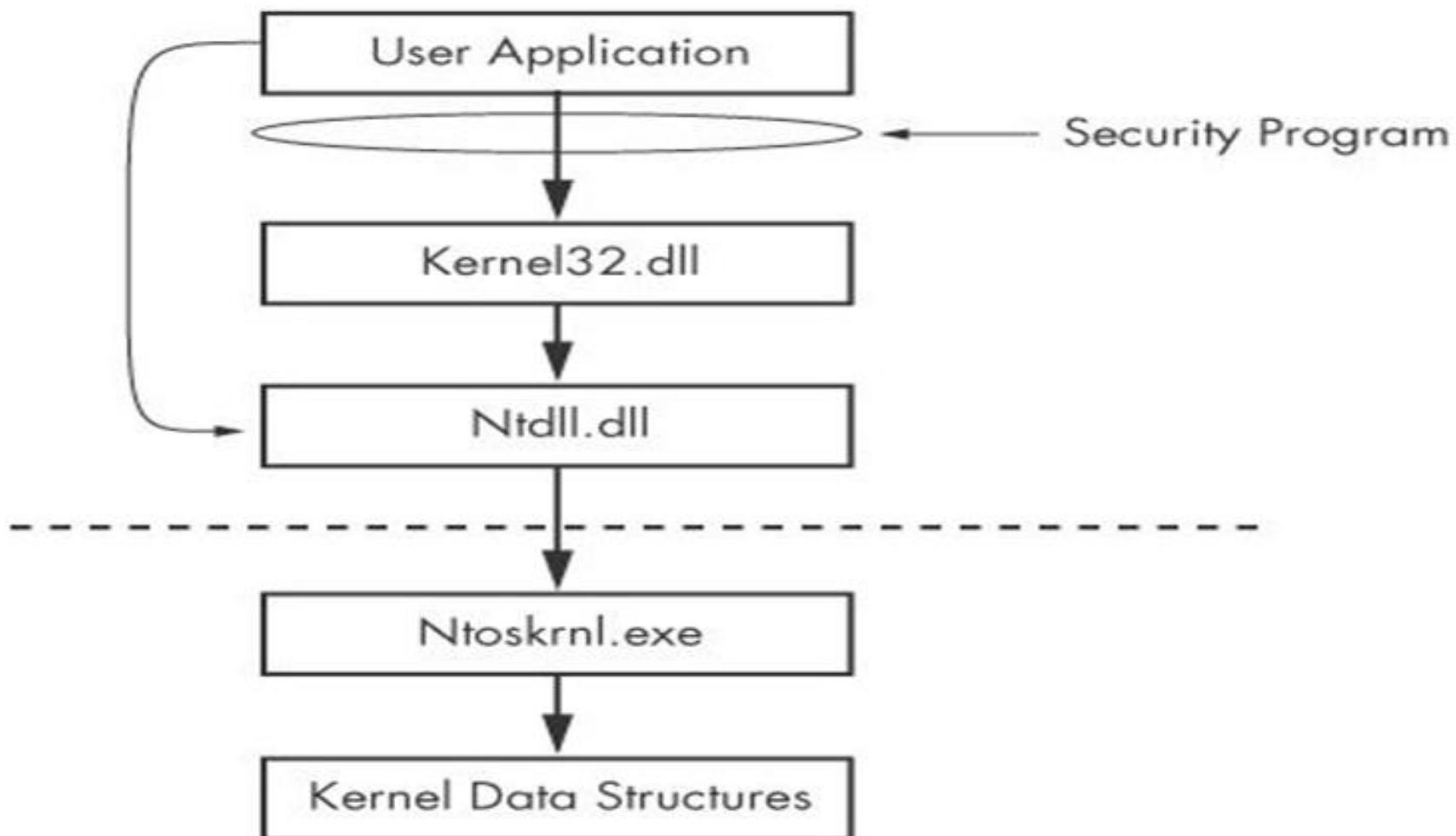


User mode and kernel mode

Các native API

- Không có tài liệu đề cập đến dạng API này
- Được dành cho Windows sử dụng nội bộ
- Native API có thể “mạnh” hơn và được gọi trong các mã độc.

Sử dụng native API



Using the Native API to avoid detection

Các native API thường có trong mã độc

- NtQuerySystemInformation
- NtQueryInformationProcess
- NtQueryInformationThread
- NtQueryInformationFile
- NtQueryInformationKey

Các native API thường có trong mã độc

NtContinue

- Trả về một ngoại lệ
- Có thể được dùng để chuyển thực thi theo những cách phức tạp
- Được sử dụng để gây sự nhầm lẫn cho các nhà phân tích và làm cho một chương trình khó debug hơn.

Nội dung

- 1. Windows API**
- 2. Windows Registry**
- 3. Các API xử lý kết nối mạng**
- 4. Phân tích mã độc trên Windows**
- 5. Chế độ nhân và chế độ người dùng**
- 6. Các native API**

Mã độc

**Chương 6. Phân tích một số cơ chế
và hành vi thông thường của mã độc**

Mục tiêu

- Giới thiệu một số cơ chế hoạt động thường gặp của mã độc
- Phân tích một số cơ chế hoạt động thường gặp của mã độc

Tài liệu tham khảo

- [1] Michael Sikorski, Andrew Honig, 2012, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, (ISBN: 978-1593272906).
- [2] Sam Bowne, Slides for a college course at City College San Francisco,
https://samsclass.info/126/126_S17.shtml

Nội dung

- 1. Downloader**
- 2. Backdoor**
- 3. Công cụ đánh cắp thông tin**
- 4. Các cơ chế duy trì hiện diện**
- 5. Leo thang đặc quyền**
- 6. Các kỹ thuật trong Rootkit**

Nội dung

- 7. Launcher**
- 8. Tiêm vào tiến trình**
- 9. Thay thế tiến trình**
- 10. Tiêm vào hook**
- 11. Detour**
- 12. Tiêm vào APC**

Nội dung

- 1. Downloader**
- 2. Backdoor**
- 3. Công cụ đánh cắp thông tin**
- 4. Các cơ chế duy trì hiện diện**
- 5. Leo thang đặc quyền**
- 6. Các kỹ thuật trong Rootkit**

Downloader

☐ Tải một chương trình độc hại khác về.

☐ Che giấu nó trước các anti virus.

☐ Thường sử dụng windows API

URLDownloadToFileA theo sau đó là một lời gọi đến

WinExec

Loaders

Chuẩn bị một phần mềm độc hại khác và thực thi nó
một cách bí mật

- Có thể thực thi ngay lập tức hoặc lúc nào đó
- Lưu trữ mã độc ở những nơi không mong muốn,
chẳng hạn như .rsrc section của PE file

Nội dung

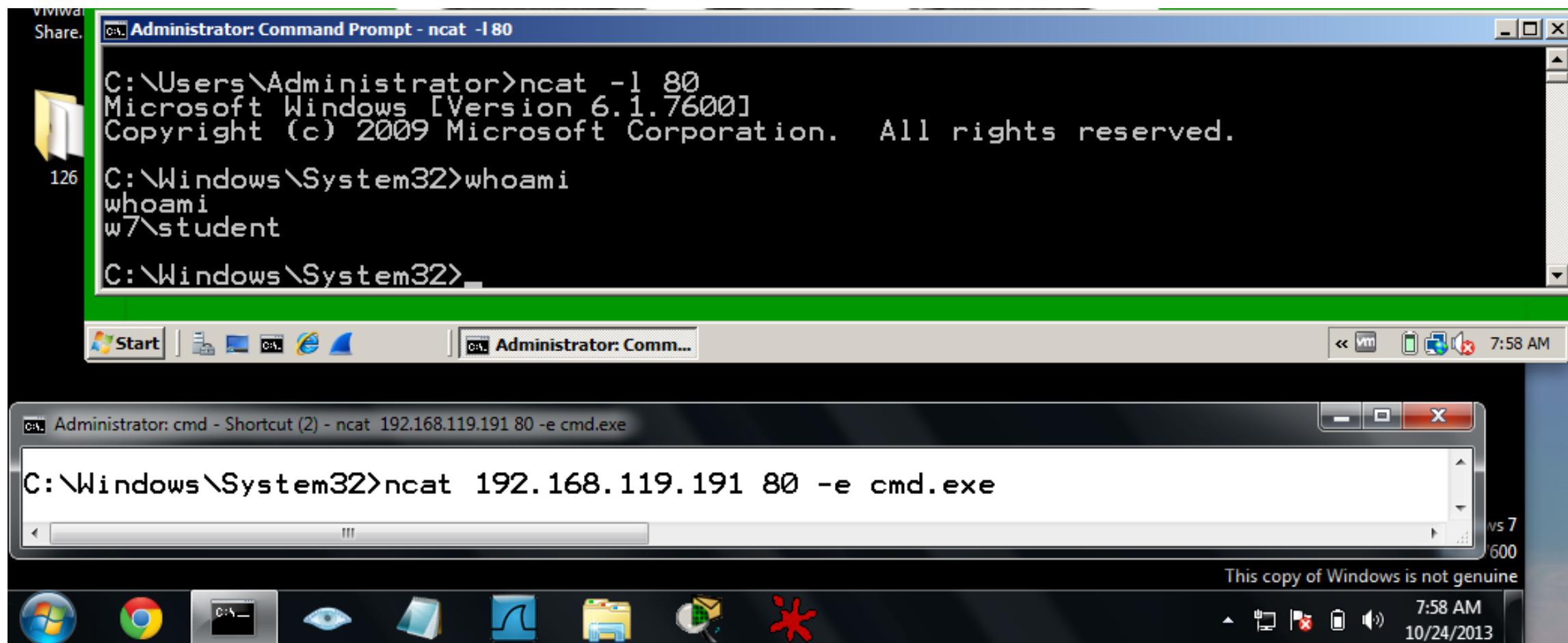
- 1. Downloader**
- 2. Backdoor**
- 3. Công cụ đánh cắp thông tin**
- 4. Các cơ chế duy trì hiện diện**
- 5. Leo thang đặc quyền**
- 6. Các kỹ thuật trong Rootkit**

Backdoor

- ❑ Tạo một truy cập từ xa đến máy nạn nhân
- ❑ Đây là loại mã độc phổ biến nhất
- ❑ Thông thường mã độc này nhắm đến cổng 80
- ❑ Khả năng chung: Thao tác trên Registry, liệt kê các cửa sổ hiển thị, tạo các thư mục, tìm kiếm file,...

Reverse Shell

- ☐ Máy tính bị lây nhiễm sẽ gọi kẻ tấn công từ bên ngoài, yêu cầu thực thi lệnh



Windows Reverse Shells

Hành động cơ bản

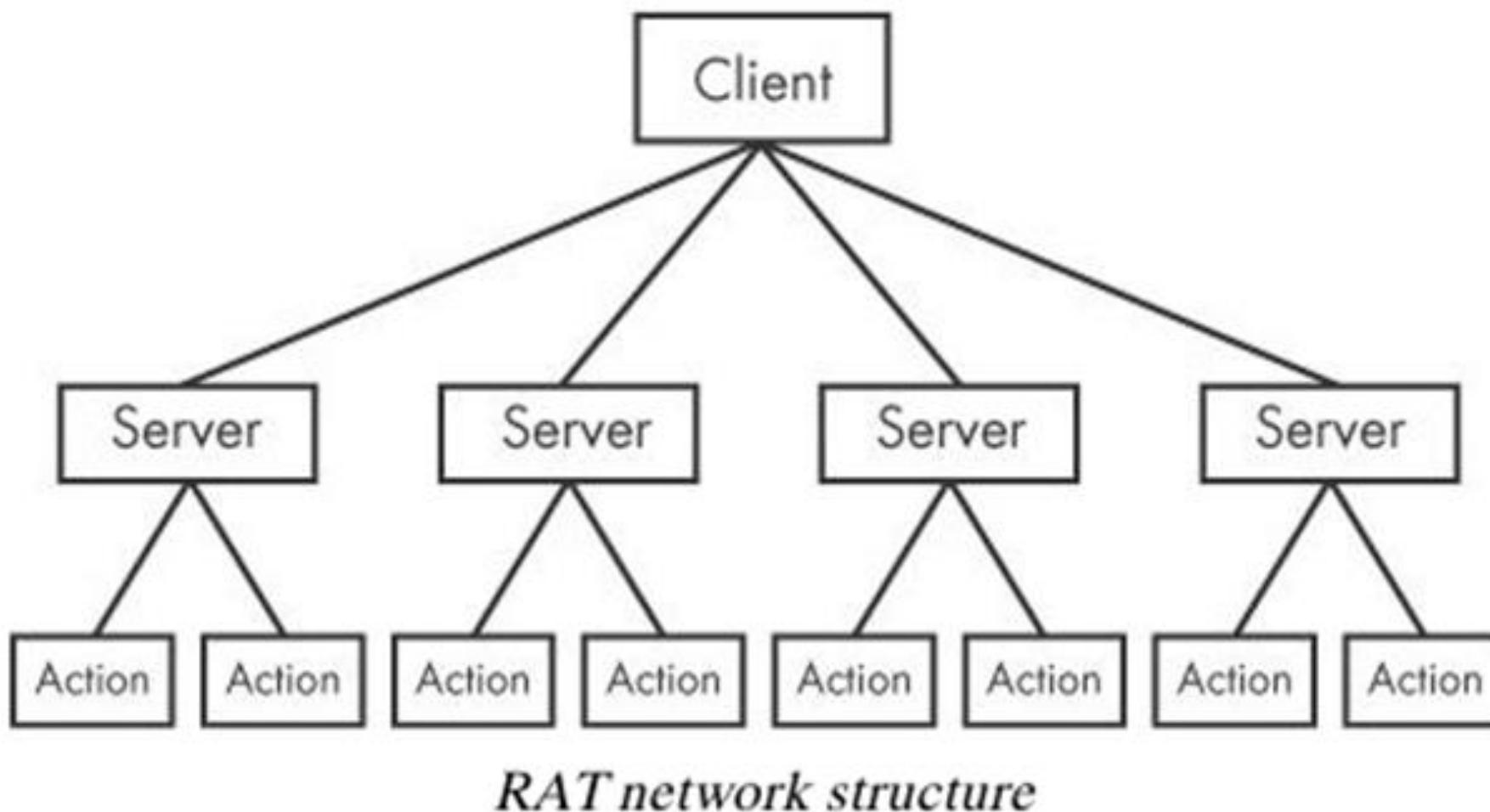
- Gọi **CreateProcess** và thao tác với cấu trúc **STARTUPINFO**
- Tạo **socket** cho điều khiển máy từ xa
- Sau đó gắn socket với **standard input, output, và error** cho **cmd.exe**
- **CreateProcess** chạy cmd.exe với cửa sổ bị chặn lại để ẩn nó

Windows Reverse Shells

- ❑ Đa luồng – Multithreaded
- ❑ Tạo một socket, hai pipe và hai thread
- ❑ Tìm các lời gọi API đến CreateThread và CreatePipe
- ❑ Một thread cho stdin, một cho stdout

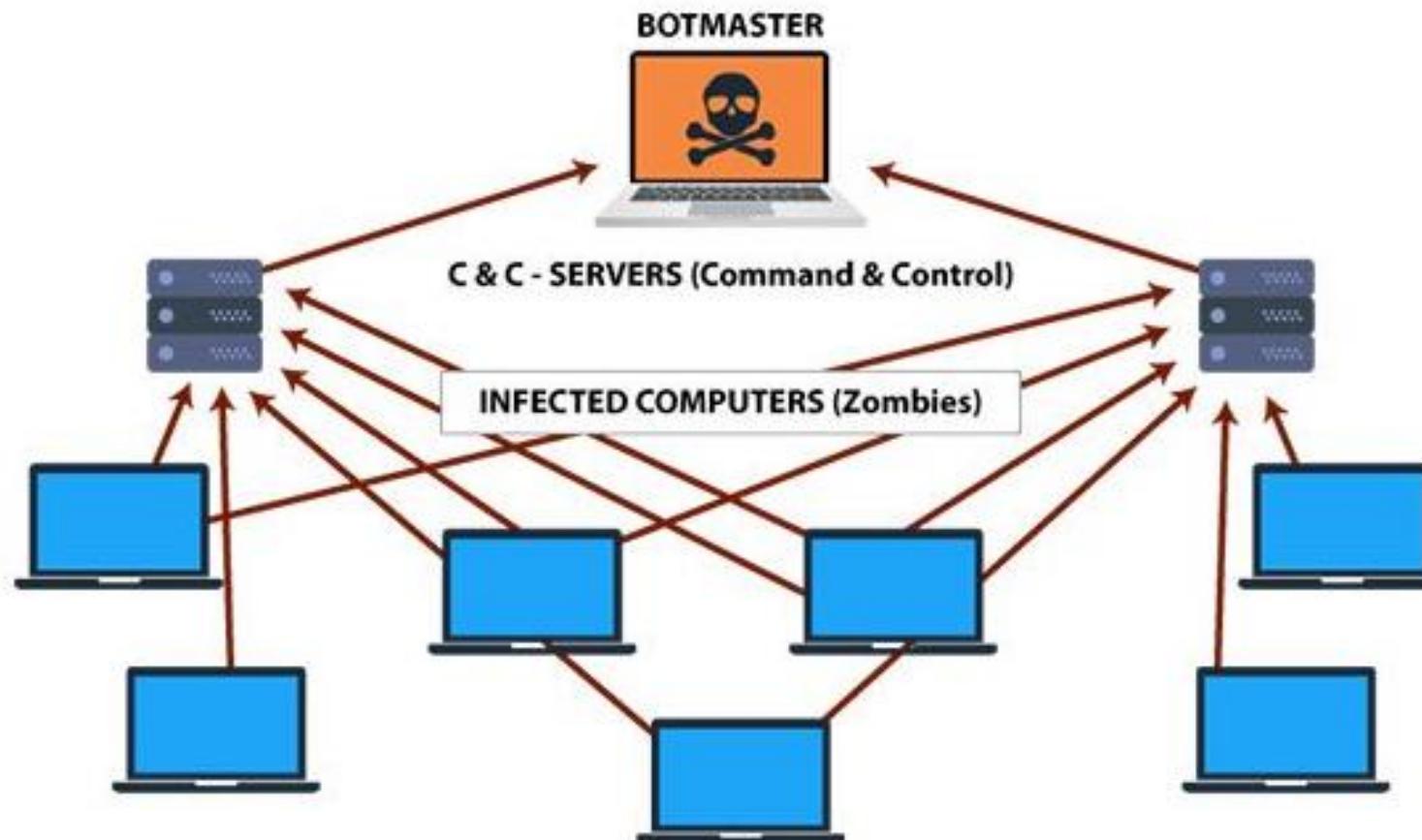
RATs (Remote Administration Tools)

□ Ví dụ Poison Ivy



Botnets

- Một tập hợp các máy bị nhiễm mã độc
 - Được gọi là bots hoặc zombies



Botnets v. RATs

- Botnet gồm nhiều máy, RATs kiểm soát ít máy hơn
- Tất cả bots/zombies đều được kiểm soát cùng một lúc; RATs điều khiển từng nạn nhân một
- RATs dành cho các cuộc tấn công có mục tiêu rõ ràng; Botnets được dùng cho các cuộc tấn công nhắm đến nhiều đối tượng chung: DDOS, Spam,...

Nội dung

1. Downloader
2. Backdoor
- 3. Công cụ đánh cắp thông tin**
4. Các cơ chế duy trì hiện diện
5. Leo thang đặc quyền
6. Các kỹ thuật trong Rootkit

Đánh cắp thông tin

Ba cách:

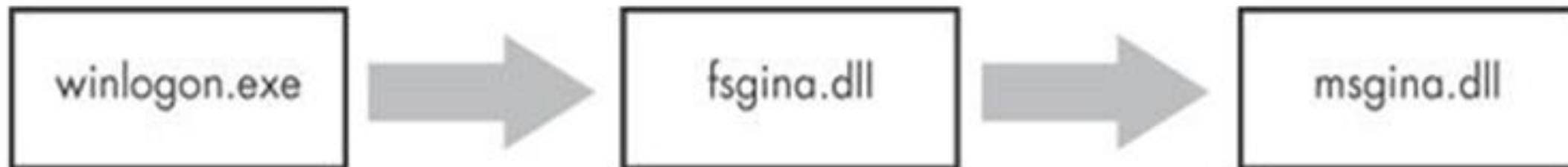
- ❑ Tấn công vào cơ chế đăng nhập và lấy cắp thông tin
- ❑ Dump dữ liệu lưu trữ, chẳng hạn như Passwprd hashes
- ❑ Lưu lại các thao tác gõ phím của victim (Keylogger)

GINA Interception

- Nhận dạng và xác thực của bên thứ ba (GINA)**
 - Cho phép bên thứ ba tùy chỉnh tiến trình đăng nhập cho RFID hoặc thẻ thông minh
 - Mã độc có thể chặn bắt thông tin gửi đến tiến trình xác thực để đánh cắp thông tin
- GINA có trong msgina.dll**
 - Được load bởi Winlogon thực thi trong quá trình đăng nhập
- WinLogon cũng load những tùy chỉnh của bên thứ ba trong DLLs load giữa WinLogon và GINA.**

GINA Registry Key

- ❑ HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL
- ❑ Chứa các DLL bên thứ ba được WinLogon nạp



Malicious fsgina.dll sits in between the Windows system files to capture data.

MITM Attack

Mã độc phải export tất cả các hàm trong msgina.dll,
hoạt động như một MITM

- Có hơn 15 hàm, hầu hết đều bắt đầu với wlx
- Mã độc export rất nhiều hàm wlx, có thể để chặn
bắt
- Là dấu hiệu nhận biết

WlxLoggedOutSAS

- ☐ Hầu hết các export chỉ đơn giản là gọi qua các hàm trong msgina.dll
- ☐ Tại (2) mã độc logs lại các thông tin vào:**%SystemRoot%\system32\drivers\tcpudp.sys**

*GINA DLL WlxLoggedOutSAS export function for logging
stolen credentials*

```
100014A0 WlxLoggedOutSAS
100014A0      push    esi
100014A1      push    edi
100014A2      push    offset aWlxloggedout_0 ; "WlxLoggedOutSAS"
100014A7      call    Call_msgina_dll_function 1
...
100014FB      push    eax ; Args
100014FC      push    offset aUSDSPSOpS ;"U: %s D: %s P: %s OP: %s"
10001501      push    offset aDRIVERS ; "drivers\tcpudp.sys"
10001503      call    Log_To_File 2
```

Hash Dumping

☐ Mật khẩu đăng nhập được lưu trữ dưới dạng LM hoặc NTLM hashes

- Hashes có thể được sử dụng trực tiếp để xác thực (pass-the-hash-attack)
- Hoặc cracked offline để tìm password

☐ Pwdump and Pass-the-Hash Toolkit

- Công cụ hacking miễn phí cung cấp việc hash dumping
- Mã nguồn mở
- Được sử dụng lại trong nhiều mã độc, sửa đổi để bypass qua anti-virus

Pwdump

- Injects một DLL vào LSASS (Local Security Authority Subsystem Service)
- Get hashes từ SAM (Security Account Manager)
- Inject DLL chạy bên trong những tiến trình khác
- Lấy tất cả các quyền của tiến trình
- LSASS là mục tiêu phổ biến
 - Đặc quyền cao
 - Truy cập vào nhiều API hữu ích

Pwdump

- Injects Isaext.dll vào lsass.exe
 - Gọi hàm GetHash, export của Isaext.dll
 - Hash extraction, Sử dụng những hàm không cung cấp tài liệu của windows
- Kẻ tấn công có thể thay đổi tên của hàm GetHash

Pwdump

□ Sử dụng các thư viện:

- **samsrv.dll** để truy cập vào SAM
- **advapi32.dll** để truy cập vào các hàm chưa được imported vào lsass.exe
- Hashes trích xuất bởi SamlGetPrivateData
- Giải mã với SystemFunction025 và SystemFunction027

□ Tất cả các hàm đều không có tài liệu nào nói về nó

Pwdump

Unique API calls used by a pwdump variant's export function GrabHash

1000123F	push	offset LibFileName ; "samsrv.dll" 1
10001244	call	esi ; LoadLibraryA
10001248	push	offset aAdvapi32_dll_0 ; "advapi32.dll" 2
...		
10001251	call	esi ; LoadLibraryA
...		
1000125B	push	offset ProcName ; "SamIConnect"
10001260	push	ebx ; hModule
10001265	call	esi ; GetProcAddress
...		
10001281	push	offset aSamrqu ; "SamrQueryInformationUser"
10001286	push	ebx ; hModule
1000128C	call	esi ; GetProcAddress
...		
100012C2	push	offset aSamigetpriv ; "SamIGetPrivateData"
100012C7	push	ebx ; hModule
100012CD	call	esi ; GetProcAddress
...		
100012CF	push	offset aSystemfuncti ; "SystemFunction025" 3
100012D4	push	edi ; hModule
100012DA	call	esi ; GetProcAddress
100012DC	push	offset aSystemfuni_0 ; "SystemFunction027" 4
100012E1	push	edi ; hModule
100012E7	call	esi ; GetProcAddress

Pwdump

Unique API calls used by a whosthere-alt variant's export function TestDump

```
10001119      push    offset LibFileName ; "secur32.dll"
1000111E      call    ds:LoadLibraryA
10001130      push    offset ProcName ; "LsaEnumerateLogonSessions"
10001135      push    esi                ; hModule
10001136      call    ds:GetProcAddress 1
...
10001670      call    ds:GetSystemDirectoryA
10001676      mov     edi, offset aMsv1_0_dll ; \\msv1_0.dll
...
100016A6      push    eax                ; path to msv1_0.dll
100016A9      call    ds:GetModuleHandleA 2
```

Keystroke Logging

Kernel-Based Keyloggers

- ❑ Khó phát hiện với những ứng dụng ở user-mode
- ❑ Thường là một phần của Rootkits
- ❑ Hoạt động như drivers của bàn phím
- ❑ Vượt qua các chương trình bảo vệ người dùng ở user-space

Keystroke Logging

□ User-Space Keyloggers

- Sử dụng Windows API
- Thực hiện với các hooking hoặc polling

□ Hooking

- Sử dụng hàm SetWindowsHookEx để thông báo cho mã độc mỗi lần nhấn phím

□ Polling

- Dùng hàm GetAsyncKeyState và hàm GetForegroundWindow để liên tục thăm dò trạng thái của các phím

Polling Keyloggers

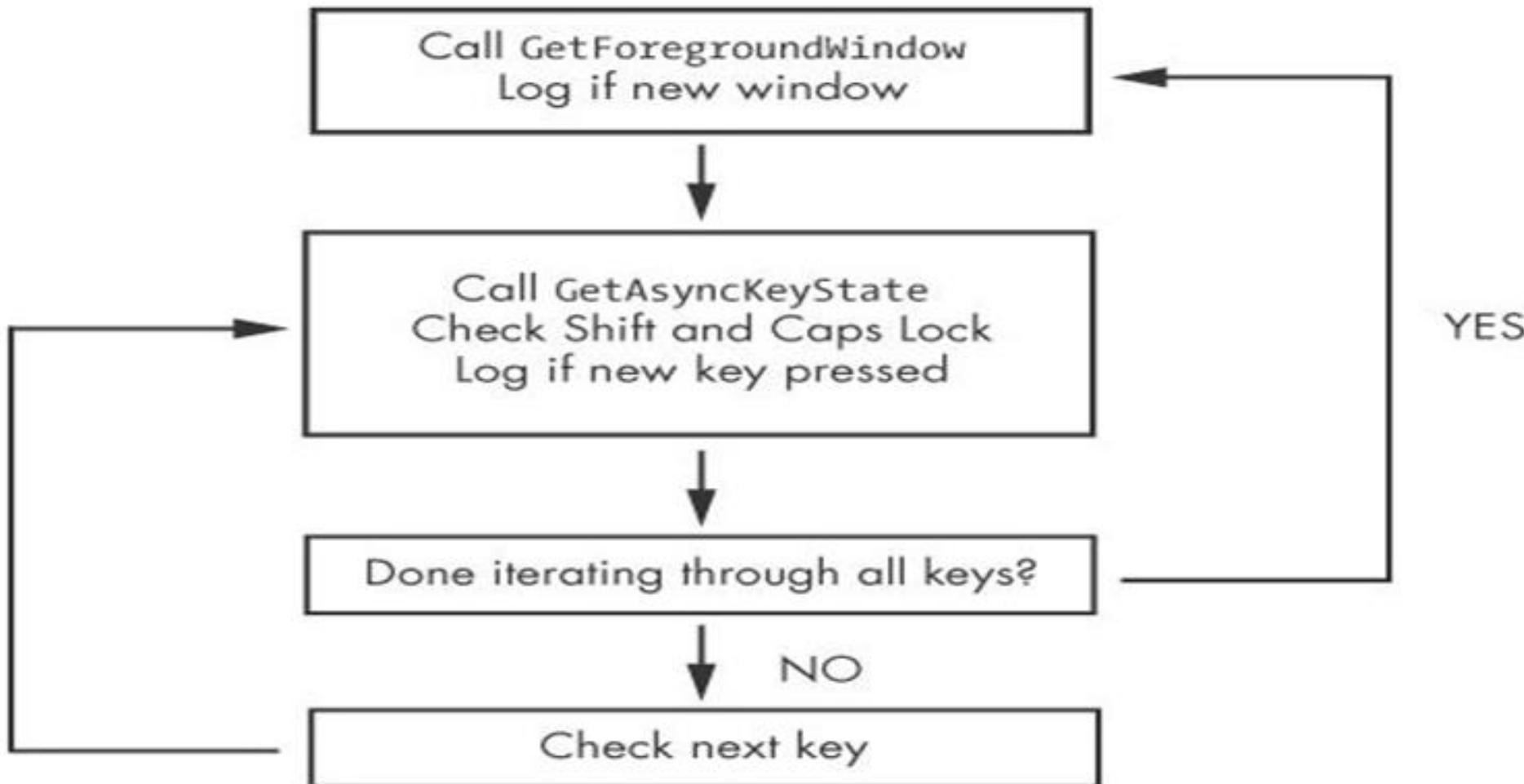
GetAsyncKeyState

- Xác định xem một phím đã được nhấn hay không

GetForegroundWindow

- Xác định cửa sổ foreground

Polling Keyloggers



Loop structure of GetAsyncKeyState and GetForegroundWindow keylogger

Polling Keyloggers

[Up]
[Num Lock]
[Down]
[Right]
[UP]
[Left]
[PageDown]

Nội dung

1. Downloader
2. Backdoor
3. Công cụ đánh cắp thông tin
4. Các cơ chế duy trì hiện diện
5. Leo thang đặc quyền
6. Các kỹ thuật trong Rootkit

Các cơ chế duy trì hiện diện

- Sửa đổi Registry: Run key
- Các Registry entries quan trọng
 - ApplInit_DLLs
 - Winlogon Notify
 - SvcHost DLLs

Sửa đổi Registry

Run key

- HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\

Windows\ CurrentVersion\ Run

- Nhiều thứ khác, với Autoruns

Công cụ *Process Monitor* hiển thị các sửa đổi Registry (Sửa những giá trị trong registry, thêm key...)

APPINIT DLLS

AppInit_DLLs được nạp vào mọi tiến trình có sử dụng User32.dll

- HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Windows
- Registry này chứa một danh sách các DLL
- Có nhiều tiến trình load chúng
- Mã độc sẽ gọi DLLMain để kiểm tra tiến trình trước khi nó khởi chạy payload

Winlogon Notify

Giá trị của Notify có trong

- HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows
- Những DLLs handles sẽ xử lý các sự kiện của winlogon.exe
- Mã độc thường gắn với các sự kiện như đăng nhập, khởi động cùng hệ thống, khóa màn hình,...
- Nó thậm chí có thể khởi chạy ở cả trong chế độ Safe mode của windows.

SvcHost DLLs

- Svchost là một tiến trình chung cho các service khác nhau
- Nhiều tiến trình Svchost có thể chạy cùng một lúc
- Các Group được xác định tại
 - HKEY_LOCAL_MACHINE\ SOFTWARE\ Microsoft\ Windows NT\ CurrentVersion\ Svchost
- Các Services được xác định tại
 - HKEY_LOCAL_MACHINE\ System\ CurrentControlSet\ Services\ ServiceName

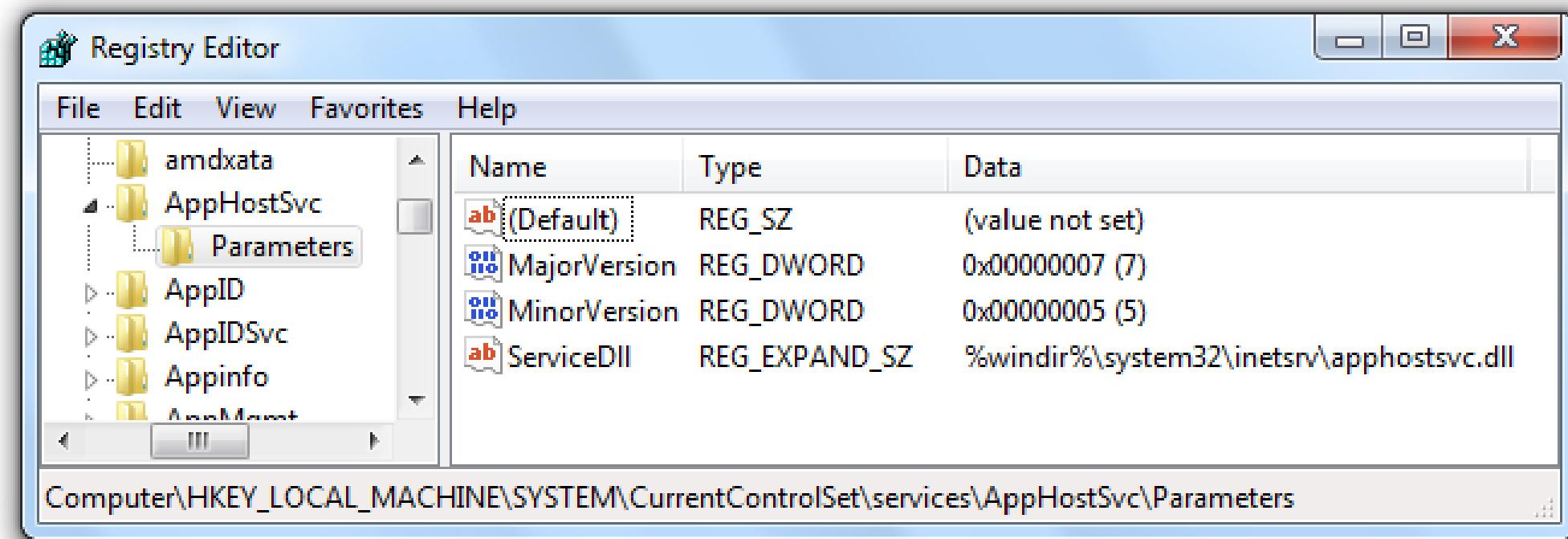
Process Explorer

Name	PID	Processor	Memory
svchost.exe	636	0.03	3,000 K
WmiPrvSE.exe	372	0.03	17,428 K
WmiPrvSE.exe	1580	0.03	3,968 K
WmiPrvSE.exe	2820	0.09	5,044 K
svchost.exe	716	0.01	3,524 K
svchost.exe	756	0.01	14,184 K
audiogd.exe	2180		14,988 K
svchost.exe	844		51,092 K
dwm.exe	2968	0.15	103,948 K
svchost.exe	940	0.25	27,900 K
svchost.exe	1100	0.01	5,652 K
svchost.exe			
spoolsv.exe			
svchost.exe			
svchost.exe			
gogoc.exe			
sqlwriter.exe			
TeamViewer			
vmtoolsd.exe			
svchost.exe			
wradvs.exe			

Command Line:
C:\Windows\system32\svchost.exe -k netsvcs
Path:
C:\Windows\System32\svchost.exe (netsvcs)
Services:
Background Intelligent Transfer Service [BITS]
Certificate Propagation [CertPropSvc]
Group Policy Client [gpsvc]
IP Helper [iphlpsvc]
IKE and AuthIP IPsec Keying Modules [IKEEXT]
Multimedia Class Scheduler [MMCSS]
Remote Desktop Configuration [SessionEnv]
Shell Hardware Detection [ShellHWDetection]
System Event Notification Service [SENS]
Server [LanmanServer]
Task Scheduler [Schedule]
Themes [Themes]
User Profile Service [ProfSvc]
Windows Update [wuauserv]
Windows Management Instrumentation [Winmgmt]

ServiceDLL

- ☐ Tất cả các svchost.exe DLL chứa một tham số key với một giá trị ServiceDLL
- ☐ Mã độc sẽ set ServiceDLL đến vị trí của các DLL độc hại



Groups

- Mã độc thường tự add nó vào một Group đang tồn tại
 - Hoặc ghi đè lên một nonvital service
 - Thường thì một service rất hiếm khi được sử dụng bởi nhóm netsvcs
- Phát hiện điều này với phân tích động và theo dõi Registry
- Hoặc tìm các hàm của service giống như CreateServiceA trong lúc Disassembly

Trojanized System Binaries

- Mã độc tiến hành Patch các byte của một binary trên hệ thống để buộc hệ thống thực thi mã độc
- Một khi bị nhiễm thì những lần sau hệ thống đều load mã độc
- Các DLLs thường là mục tiêu phổ biến
- Thông thường thì các entry function được sửa đổi
- Nhảy tới đoạn mã được chèn vào một phần trống của binary
- Sau đó thực thi DLL như bình thường

Trojanized System Binaries

rtutils.dll's DLL Entry Point Before and After Trojanization

Original code

```
DllEntryPoint(HINSTANCE hinstDLL,  
    DWORD fdwReason, LPVOID  
    lpReserved)
```

```
mov    edi, edi  
push   ebp  
mov    ebp, esp  
push   ebx  
mov    ebx, [ebp+8]  
push   esi  
mov    esi, [ebp+0Ch]
```

Trojanized code

```
DllEntryPoint(HINSTANCE hinstDLL,  
    DWORD fdwReason, LPVOID  
    lpReserved)
```

```
jmp    DllEntryPoint_0
```

DLL Load-Order Hijacking

The default search order for loading DLLs on Windows XP is as follows:

1. The directory from which the application loaded
2. The current directory
3. The system directory (the `GetSystemDirectory` function is used to get the path, such as .../*Windows/System32*/)
4. The 16-bit system directory (such as .../*Windows/System*/)
5. The Windows directory (the `GetWindowsDirectory` function is used to get the path, such as .../*Windows*/)
6. The directories listed in the PATH environment variable

KnownDLLs Registry Key

- Chứa danh sách các vị trí DLL cụ thể
- Ghi đè trình tự tìm kiếm các DLL đã được liệt kê
- Thay đổi thứ tự nạp các DLL,
 - Chiếm quyền ưu tiên nạp các DLL của thư mục System32
 - Không được bảo vệ bởi KnownDLLs

explorer.exe

- Nằm tại /Windows**
- Load ntshru.dll từ System32**
- Không biết ntshru.dll nằm ở đâu -> thực hiện việc tìm kiếm**
- Một ntshru.dll độc hại trong /Windows sẽ được nạp thay thế**

Vulnerable DLLs

- **Bất kỳ những binary không được tìm thấy trong /System32 thì đều có thể gây ra sự nguy hiểm**
- **explorer.exe có khoảng 50 DLL dễ gây nguy hiểm**
- **Các DLL biết đến thì hầu hết không được bảo vệ đầy đủ, vì:**
 - **Nhiều DLL lại load các DLL khác**
 - **Import đệ quy tuân theo trình tự tìm kiếm mặc định**

Nội dung

1. Downloader
2. Backdoor
3. Công cụ đánh cắp thông tin
4. Các cơ chế duy trì hiện diện
- 5. Leo thang đặc quyền**
6. Các kỹ thuật trong Rootkit

Leo thang đặc quyền

- Các tiến trình chạy bởi người dùng không thể làm được mọi thứ
- Những hàm giống như TerminateProcess hoặc CreateRemoteThread yêu cầu quyền của hệ thống (trên quản trị viên)
- SeDebugPrivilege là quyền dùng để debug
- Cho phép các tài khoản quản trị cục bộ có thể leo thang lên quyền của hệ thống (System privileges)

Leo thang đặc quyền

Setting the access token to SeDebugPrivilege

```
00401003  lea      eax, [esp+1Ch+TokenHandle]
00401006  push     eax                      ; TokenHandle
00401007  push     (TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY)
; DesiredAccess
00401009  call    ds:GetCurrentProcess
0040100F  push     eax                      ; ProcessHandle
00401010  call    ds:OpenProcessToken 1
00401016  test    eax, eax
00401018  jz     short loc_401080
0040101A  lea     ecx, [esp+1Ch+Luid]
0040101E  push     ecx                      ; lpLuid
0040101F  push     offset Name            ; "SeDebugPrivilege"
00401024  push     0                        ; lpSystemName
00401026  call    ds:LookupPrivilegeValueA
0040102C  test    eax, eax
0040102E  jnz     short loc_40103E
```

Leo thang đặc quyền

```
...
0040103E  mov      eax, [esp+1Ch+Luid.LowPart]
00401042  mov      ecx, [esp+1Ch+Luid.HighPart]
00401046  push     0          ; ReturnLength
00401048  push     0          ; PreviousState
0040104A  push     10h        ; BufferLength
0040104C  lea      edx, [esp+28h+NewState]
00401050  push     edx        ; NewState
00401051  mov      [esp+2Ch+NewState.Privileges.Luid.LowPt], eax 3
00401055  mov      eax, [esp+2Ch+TokenHandle]
00401059  push     0          ; DisableAllPrivileges
0040105B  push     eax        ; TokenHandle
0040105C  mov      [esp+34h+NewState.PrivilegeCount], 1
00401064  mov      [esp+34h+NewState.Privileges.Luid.HighPt], ecx 4
00401068  mov      [esp+34h+NewState.Privileges.Attributes],
SE_PRIVILEGE_ENABLED 5
00401070  call    ds:AdjustTokenPrivileges 2
```

Nội dung

1. Downloader
2. Backdoor
3. Công cụ đánh cắp thông tin
4. Các cơ chế duy trì hiện diện
5. Leo thang đặc quyền
6. Các kỹ thuật trong Rootkit

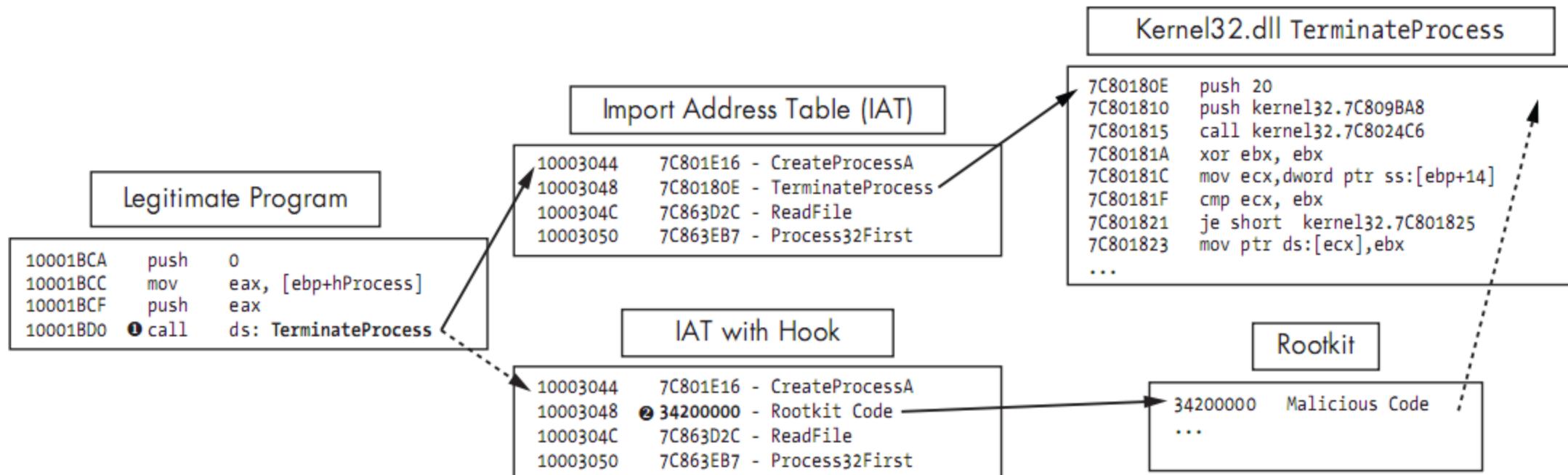
User-Mode Rootkits

- ❑ Sửa đổi các hàm cục bộ của hệ điều hành
- ❑ Ân tập tin, kết nối mạng, tiến trình,...

IAT (Import Address Table) Hooking

- Có thể sửa đổi
 - IAT (Import Address Table)
 - EAT (Export Address Tables)
- Các thành phần của PE File
- Làm đầy trong bộ Loader

IAT Hooking



IAT hooking of `TerminateProcess`. The top path is the normal flow, and the bottom path is the flow with a rootkit.

Inline Hooking

- ❑ Ghi đè các API
- ❑ Chứa trong các Import DLLs
- ❑ Thay đổi những mã trong actual function, không phải con trỏ

Các kỹ thuật khởi chạy

Nội dung

7. Launcher

8. Tiêm vào tiến trình

9. Thay thế tiến trình

10. Tiêm vào hook

11. Detour

12. Tiêm vào APC

Launcher

- Sets chính bản thân nó là một phần của mã độc
- Có thể thực thi ngay tức thì hoặc một thời điểm nào đó
- Nó che giấu những hành vi của mã độc trước người dùng
- Thường chứa mã độc mà họ đang load lên
 - Một file thực thi hoặc DLL trong phần rsrc của nó
- Các items thông thường có bên trong .rsrc
 - Icon, images, menus, strings,...

Mã hóa hoặc giải mã

- Phần .rsrc có thể được mã hóa hoặc nén
- Extraction phần rsrc sẽ sử dụng các API như:
 - FindResource
 - LoadResource
 - SizeofResource
- Thường sẽ có những đoạn code để leo thang đặc quyền

Nội dung

7. Launcher

8. Tiêm vào tiến trình

9. Thay thế tiến trình

10. Tiêm vào hook

11. Detour

12. Tiêm vào APC

Tiêm vào tiến trình

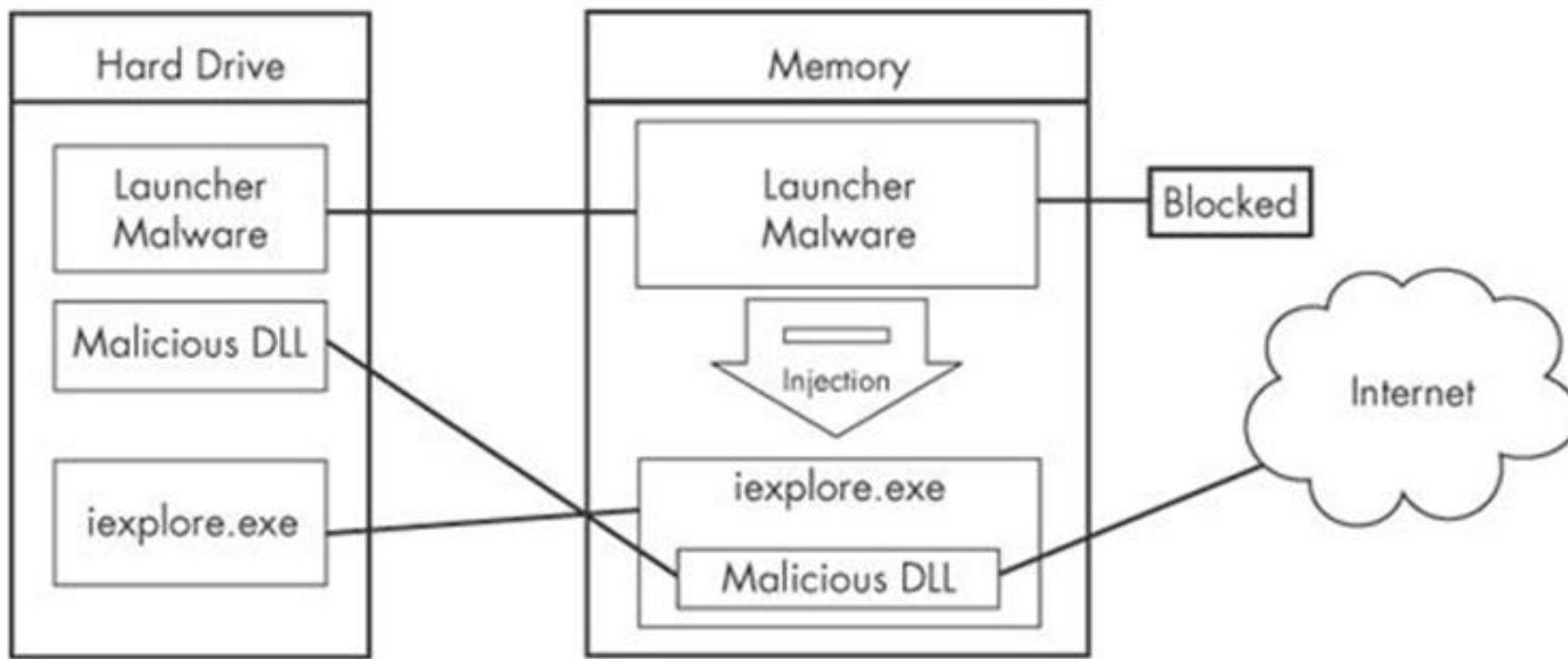
- ❑ Là một trong những kỹ thuật khởi chạy của mã độc phổ biến nhất
- ❑ Injects code vào một chương trình đang chạy
- ❑ Che giấu những hành vi nguy hiểm
- ❑ Có thể vượt qua tường lửa và các cơ chế bảo vệ
- ❑ Các API thường được gọi:
 - VirtualAllocEx để cấp phát vùng nhớ
 - WriteProcessMemory để ghi lên nó

Tiêm vào tiến trình

- ❑ Tiêm code vào một tiến trình từ xa với việc gọi **LoadLibrary**
- ❑ Khi load, hệ điều hành tự động gọi **DLLMain** chứa những đoạn mã độc hại

Tiêm vào tiến trình

- Code của mã độc có được đặc quyền giống như mã được tiêm vào



DLL injection—the launcher malware cannot access the Internet until it injects into iexplore.exe.

Tiêm vào tiến trình

CreateRemoteThread sử dụng ba tham số

- Tiến trình xử lý hProcess
- Điểm bắt đầu lpStartAddress (LoadLibrary)
- Tham số lpParameter tên của DLL độc hại

C Pseudocode for DLL injection

```
hVictimProcess = OpenProcess(PROCESS_ALL_ACCESS, 0, victimProcessID ①);

pNameInVictimProcess = VirtualAllocEx(hVictimProcess,...,sizeof(maliciousLibraryName),....,...);
WriteProcessMemory(hVictimProcess,...,maliciousLibraryName, sizeof(maliciousLibraryName),...);
GetModuleHandle("Kernel32.dll");
GetProcAddress(...,"LoadLibraryA");
② CreateRemoteThread(hVictimProcess,...,...,LoadLibraryAddress,pNameInVictimProcess,...,...);
```

Direct Injection

- Tiêm mã trực tiếp vào tiến trình từ xa
- Không sử dụng DLL
- Linh hoạt hơn so với tiêm vào DLL
- Yêu cầu nhiều về tùy chỉnh mã
- Đảm bảo chạy được mà không ảnh hưởng đến tiến trình
- Khó thực hiện

Nội dung

7. Launcher

8. Tiêm vào tiến trình

9. Thay thế tiến trình

10. Tiêm vào hook

11. Detour

12. Tiêm vào APC

Thay thế tiến trình

- ❑ Mã độc ghi đè lên vùng nhớ của một đối tượng đang chạy
- ❑ Che giấu mã độc và làm nó giống như một tiến trình hợp lệ
- ❑ Tránh nguy cơ bị crash giữa một tiến trình với tiến trình tiêm
- ❑ Mã độc nhận được các quyền của tiến trình thay thế
- ❑ *svchost.exe* thường là mục tiêu.

Suspended State

- ❑ Trong trạng thái suspended, tiến trình vẫn được nạp vào bộ nhớ nhưng primary Thread của nó thì cũng bị suspended theo.
- ❑ Mã độc có thể ghi đè mã của nó lên đó trước khi tiến trình thoát khỏi trạng thái suspended và chạy trở lại.
- ❑ Sử dụng giá trị CREATE_SUSPENDED trong tham số dwCreationFlags trong khi gọi đến các hàm CreateProcess .

Thay thế tiến trình

Assembly code showing process replacement

00401535	push	edi	; lpProcessInformation
00401536	push	ecx	; lpStartupInfo
00401537	push	ebx	; lpCurrentDirectory
00401538	push	ebx	; lpEnvironment
00401539	push	CREATE_SUSPENDED	; dwCreationFlags
0040153B	push	ebx	; bInheritHandles
0040153C	push	ebx	; lpThreadAttributes
0040153D	lea	edx, [esp+94h+CommandLine]	
00401541	push	ebx	; lpProcessAttributes
00401542	push	edx	; lpCommandLine
00401543	push	ebx	; lpApplicationName
00401544	mov	[esp+0A0h+StartupInfo.dwFlags]	, 101h
0040154F	mov	[esp+0A0h+StartupInfo.wShowWindow]	, bx
00401557	call	ds:	CreateProcessA

Thay thế tiến trình

C pseudocode for process replacement

```
CreateProcess(...,"svchost.exe",...,CREATE_SUSPEND,...);
ZwUnmapViewOfSection(...);
VirtualAllocEx(...,ImageBase,SizeOfImage,...);
WriteProcessMemory(...,headers,...);
for (i=0; i < NumberOfSections; i++) {
    1 WriteProcessMemory(...,section,...);
}
SetThreadContext();
...
ResumeThread();
```

- **ZwUnmapViewOfSection** giải phóng tất cả bộ nhớ được trả bởi một section
- **VirtualAllocEx** – Cấp phát lại bộ nhớ
- **WriteProcessMemory** – Đẩy/Ghi mã độc vào nó

Thay thế tiến trình

C pseudocode for process replacement

```
CreateProcess(...,"svchost.exe",...,CREATE_SUSPEND,...);
ZwUnmapViewOfSection(...);
VirtualAllocEx(...,ImageBase,SizeOfImage,...);
WriteProcessMemory(...,headers,...);
for (i=0; i < NumberOfSections; i++) {
    1 WriteProcessMemory(...,section,...);
}
SetThreadContext();
...
ResumeThread();
```

- **SetThreadContext** – Khôi phục lại môi trường của tiến trình nạn nhân
- **ResumeThread** – Thực thi mã độc hại

Nội dung

7. Launcher

8. Tiêm vào tiến trình

9. Thay thế tiến trình

10. Tiêm vào hook

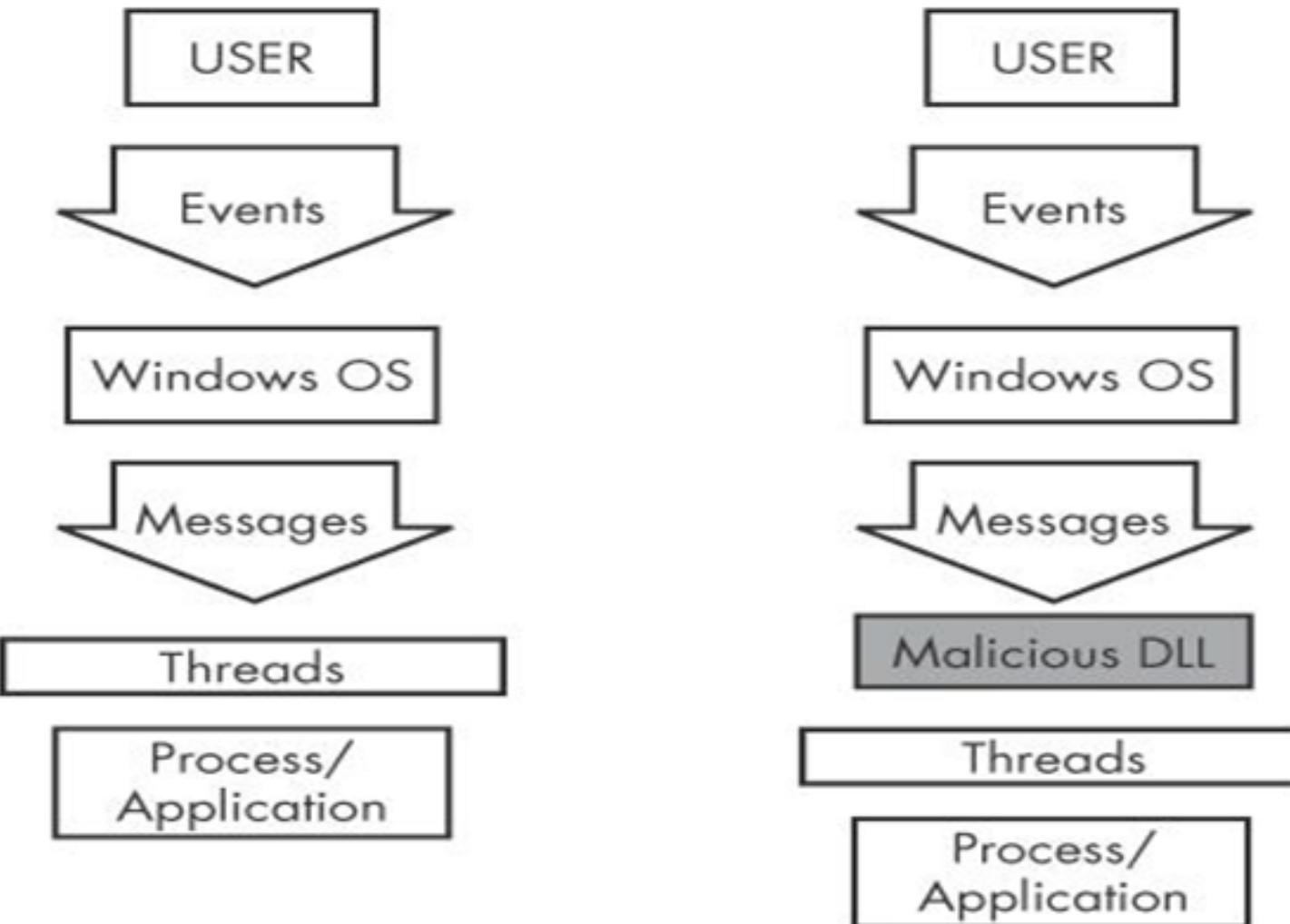
11. Detour

12. Tiêm vào APC

Hooks

- Windows Hook sẽ chặn bắt các thông điệp trao đổi giữa các ứng dụng
- Malicious hooks
 - Đảm bảo rằng mã độc sẽ chạy bất cứ khi nào mà và một thông điệp trao đổi có thể bị chặn
 - Đảm bảo rằng một DLL sẽ được nào vào vùng nhớ của tiến trình phía nạn nhân

Tiêm vào hook



Event and message flow in Windows with and without hook injection

Local and Remote Hooks

- *Local Hook*: Sẽ chặn bắt và sửa đổi các thông điệp bên trong một tiến trình cục bộ
- *Remote Hook*: Sẽ chặn bắt và sửa đổi các thông điệp được trao đổi giữa các tiến trình ở xa với nhau

High-Level and Low-Level Remote Hooks

□ High-level remote hooks

- Yêu cầu các thủ tục hook exported các hàm có trong DLL
- Mapped bởi hệ điều hành vào không gian tiến trình của một hook thread hoặc tất cả các thread

□ Low-level remote hooks

- Yêu cầu các thủ tục hook được chứa trong quá trình cài đặt các hook

Keyloggers dù dụng Hooks

- ☐ Các thao tác trên bàn phím sẽ được chụp lại bởi các hook mức cao hoặc mức thấp, sử dụng các thủ tục WH_KEYBOARD hoặc WH_KEYBOARD_LL

Keyloggers sử dụng *SetWindowsHookEx*

☐ Các tham số

- idHook – Loại hook cài đặt
- Lpfn - points to hook procedure
- hMod – Xử lý các DLL hoặc các module cục bộ, thủ tục Lpfn được xác định
- dwThreadId – Thread liên kết các hook với nhau. Zero = tất cả các thread

☐ Thủ tục hook phải gọi *CallNextHookEx* để truyền đến các hook tiếp theo

Thread Targeting

- ❑ Load tất cả các thread có thể làm giảm hiệu suất của hệ thống
- ❑ Cũng có thể kích hoạt một IPS
- ❑ Keyloggers load tất cả các thread để lấy được tất cả các hành động từ bàn phím
- ❑ Có những mã độc nhầm vào thread đơn
- ❑ Thông thường một mục tiêu Message của windows hiếm khi được sử dụng, chẳng hạn như WH_CBT (a computer-based training message)

Keylogger

Hook injection, assembly code

00401100	push	esi	
00401101	push	edi	
00401102	push	offset LibFileName ; "hook.dll"	
00401107	call	LoadLibraryA	
0040110D	mov	esi, eax	
0040110F	push	offset ProcName ; "MalwareProc"	
00401114	push	esi	; hModule
00401115	call	GetProcAddress	
0040111B	mov	edi, eax	
0040111D	call	GetNotepadThreadId	
00401122	push	eax	; dwThreadId
00401123	push	esi	; hmod
00401124	push	edi	; lpfn
00401125	push	WH_CBT ; idHook	
00401127	call	SetWindowsHookExA	

Keylogger

- Malicious DLL sẽ nạp hook.dll
- Thu được địa chỉ của các Malicious hook
- Thủ tục hook chỉ gọi CallNextHookEx
- Một thông điệp WH_CBT được gửi đến Notepad thread
- Buộc hook.dll được load bởi Notepad
- Nó sẽ bắt đầu hook các thông điệp gõ phím từ chương trình Notepad khi mà Notepad bắt đầu chạy

Nội dung

7. Launcher

8. Tiêm vào tiến trình

9. Thay thế tiến trình

10. Tiêm vào hook

11. Detour

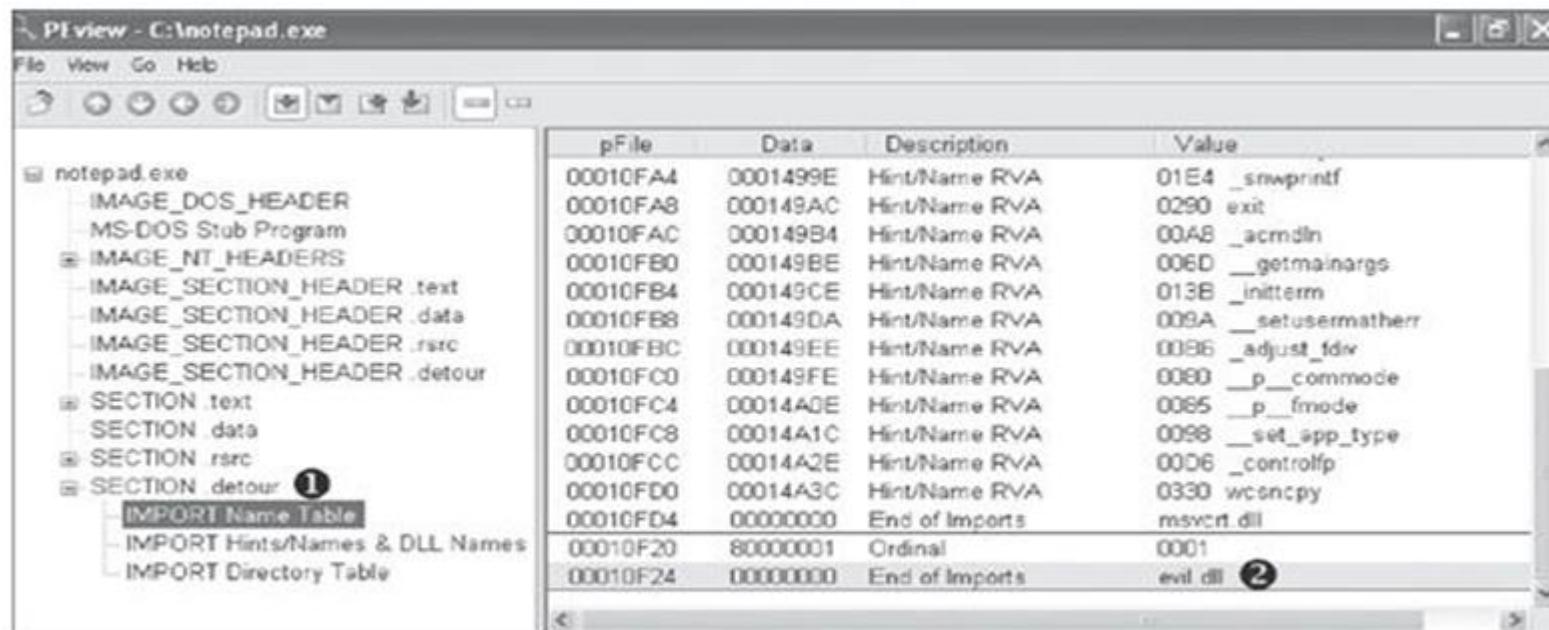
12. Tiêm vào APC

Detour

- Detours giúp các nhà phát triển ứng dụng dễ dàng sửa đổi ứng dụng trên hệ điều hành
- Các mã độc sử dụng nó để thêm các DLL mới vào các binary có trên đĩa
- Sửa đổi cấu trúc PE để tạo một **.detour** section
- Chứa PE header gốc với một bảng địa chỉ được Import

Detour

- **setdll** là một công cụ của Microsoft được sử dụng để trỏ PE vào bảng địa chỉ import mới
- Có nhiều cách khác nhau để thêm một .detour section



Nội dung

7. Launcher

8. Tiêm vào tiến trình

9. Thay thế tiến trình

10. Tiêm vào hook

11. Detour

12. Tiêm vào APC

Asynchronous Procedure Call (APC)

- Trực tiếp một thread được thực thi code khác trước khi thực thi theo đường dẫn thông thường của nó
- Mỗi thread đều có hàng đợi APC đi kèm với nó
- Chúng được xử lý khi thread trong trạng thái có thể thay đổi, chẳng hạn như các hàm này được gọi:
 - **WaitForSingleObjectEx**
 - **WaitForMultipleObjectsEx**
 - **Sleep**

Các loại APC

❑ Kernel-Mode APC

- Tạo cho hệ thống hoặc driver

❑ User-Mode APC

- Tạo cho một ứng dụng

❑ APC Injection được sử dụng trong cả hai trường hợp

Chèn APC từ chế độ người dùng

- ❑ Sử dụng API **QueueUserAPC**
- ❑ Thread phải ở trạng thái có thể thay đổi
- ❑ **WaitForSingleObjectEx** là một trong những hàm được gọi phổ biến trong Windows API
- ❑ Nhiều Thread ở trạng thái có thể thay đổi

Chèn APC từ chế độ người dùng

- Mở một trình xử lý Thread
- QueueUserAPC được gọi với pfnAPC từ LoadLibraryA (Load một DLL)
- dwData chứa tên của DLL (*dbnet.dll*)
- Svchost.exe thường là mục tiêu của APC injection

APC injection from a user-mode application

00401DA9	push	[esp+4+dwThreadId]	; dwThreadId
00401DAD	push	0	; bInheritHandle
00401DAF	push	10h	; dwDesiredAccess
00401DB1	call	ds:OpenThread 1	
00401DB7	mov	esi, eax	
00401DB9	test	esi, esi	
00401DBB	jz	short loc_401DCE	
00401DBD	push	[esp+4+dwData]	; dwData = dbnet.dll
00401DC1	push	esi	; hThread
00401DC2	push	ds:LoadLibraryA 2	; pfnAPC
00401DC8	call	ds:QueueUserAPC	

Chèn APC từ nhân

- Các driver độc hại và Rootkits thường muốn thực thi code ở không gian người dùng
- Điều này là khó thực hiện
- Một phương pháp là APC injection để có được không gian phía người dùng
- Mục tiêu hay được sử dụng nhất là *svchost.exe*
- Các hàm sử dụng
 - *KInitializeApc*
 - *KInsertQueueApc*

Chèn APC từ nhân

User-mode APC injection from kernel space

000119BD	push	ebx
000119BE	push	1 1
000119C0	push	[ebp+arg_4] 2
000119C3	push	ebx
000119C4	push	offset sub_11964
000119C9	push	2
000119CB	push	[ebp+arg_0] 3
000119CE	push	esi
000119CF	call	ds:KeInitializeApc
000119D5	cmp	edi, ebx
000119D7	jz	short loc_119EA
000119D9	push	ebx
000119DA	push	[ebp+arg_C]
000119DD	push	[ebp+arg_8]
000119E0	push	esi
000119E1	call	; KeInsertQueueApc

Nội dung

- 1. Downloader**
- 2. Backdoor**
- 3. Công cụ đánh cắp thông tin**
- 4. Các cơ chế duy trì hiện diện**
- 5. Leo thang đặc quyền**
- 6. Các kỹ thuật trong Rootkit**

Nội dung

7. Launcher

8. Tiêm vào tiến trình

9. Thay thế tiến trình

10. Tiêm vào hook

11. Detour

12. Tiêm vào APC

Mã độc

Chương 7. Các kỹ thuật chống phân tích

Mục tiêu

- Giới thiệu, phân tích một số kỹ thuật chống phân tích thường gặp trong mã độc

Tài liệu tham khảo

- [1] Michael Sikorski, Andrew Honig, 2012, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, (ISBN: 978-1593272906).
- [2] Sam Bowne, Slides for a college course at City College San Francisco,
https://samsclass.info/126/126_S17.shtml

Nội dung

1. Nén
2. Mã hóa
3. Làm rối mã
4. Một số cơ chế khác

Nội dung

- 1. Nén**
- 2. Mã hóa**
- 3. Làm rối mã**
- 4. Một số cơ chế khác**

Nén

- ❑ Packer (nén) là một kiểu chương trình nén hoặc che giấu file thực thi (executable file).
- ❑ Giảm kích thước của file, làm cho việc tải file nhanh hơn.

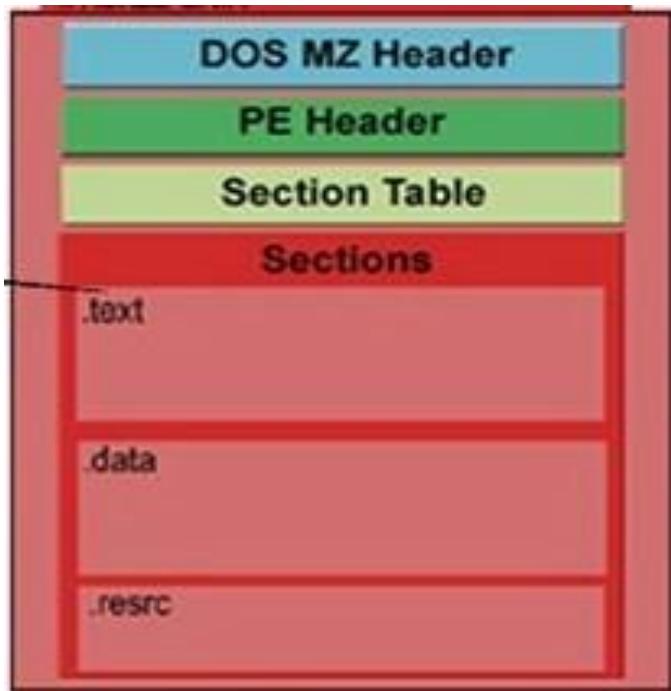
Nén

- ❑ Packer nén, mã hóa, lưu hoặc dấu code gốc của chương trình, tự động bổ sung một hoặc nhiều section, sau đó sẽ thêm đoạn code Unpacking Stub và chuyển hướng Entry Point (EP) tới vùng code này.
- ❑ Một file không đóng gói (Nonpacked) sẽ được tải bởi OS.

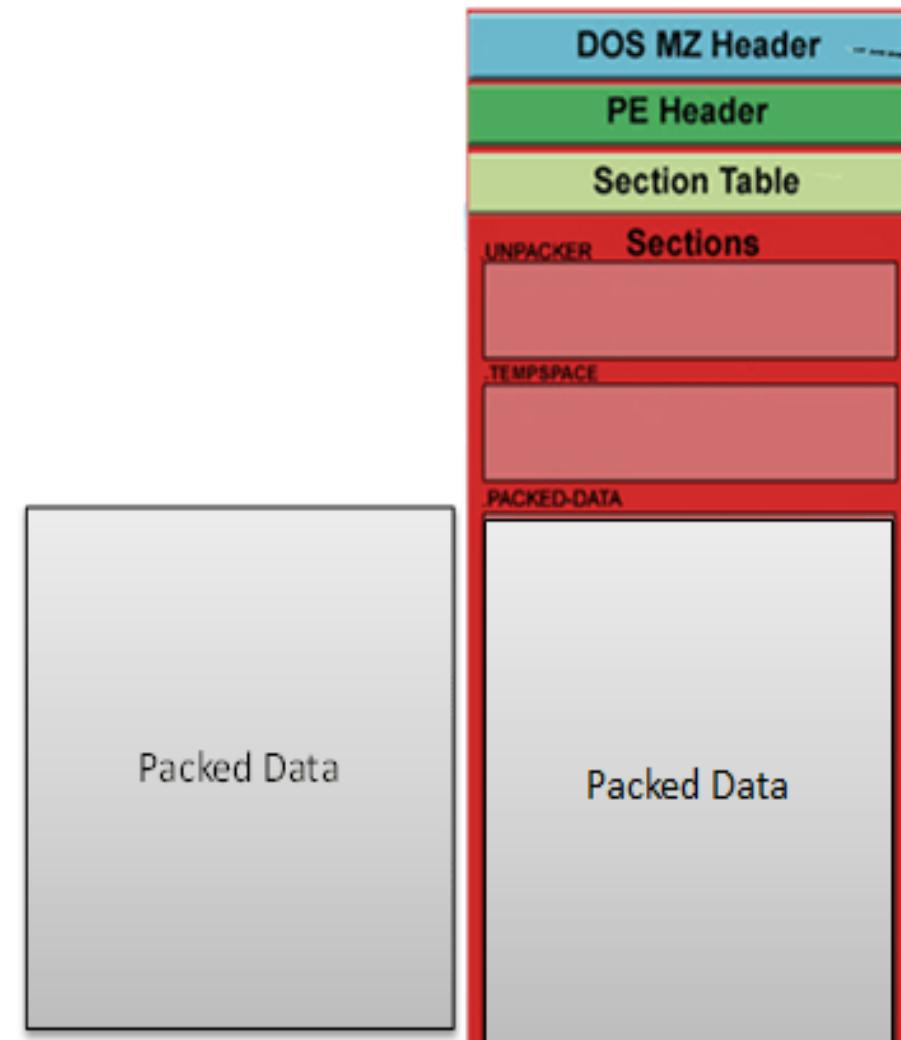
Nén

- ❑ File đã đóng gói: Unpacking Stub sẽ được tải bởi OS, sau đó chương trình gốc sẽ tải Unpacking Stub.
- ❑ Code EP của file thực thi sẽ trả tới Unpacking Stub thay vì trả vào mã gốc.
- ❑ Trong môi trường DOS, chương trình sẽ kiểm tra DOS header và nếu hợp lệ sẽ thực thi DOS Stub. Bình thường file chạy trên Windows thì 2 trường này có thể bỏ qua.

Nguyên lý hoạt động của Packer



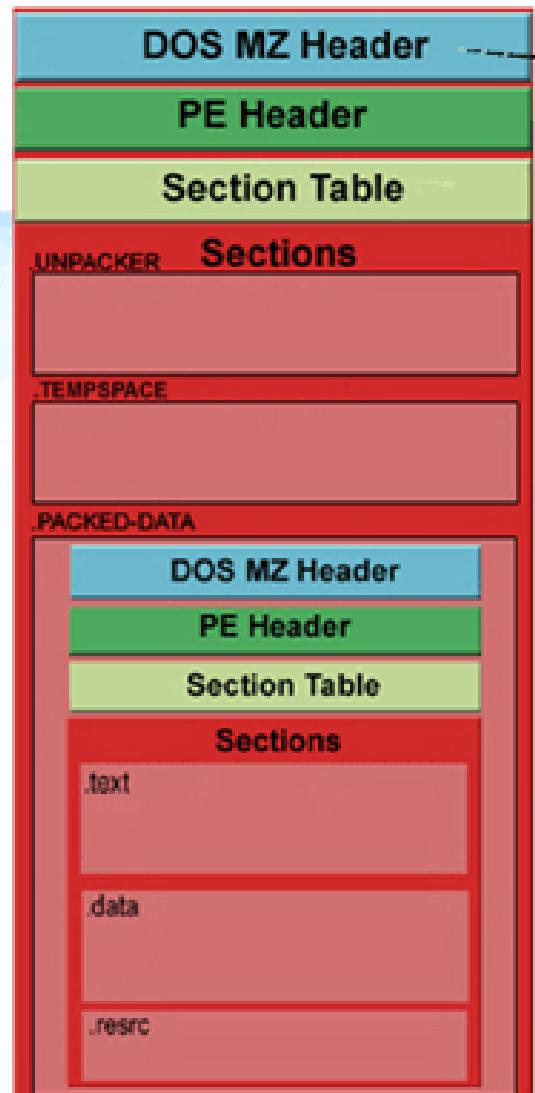
File ban đầu



Thêm vào một hay nhiều section
do packer tự tạo ra

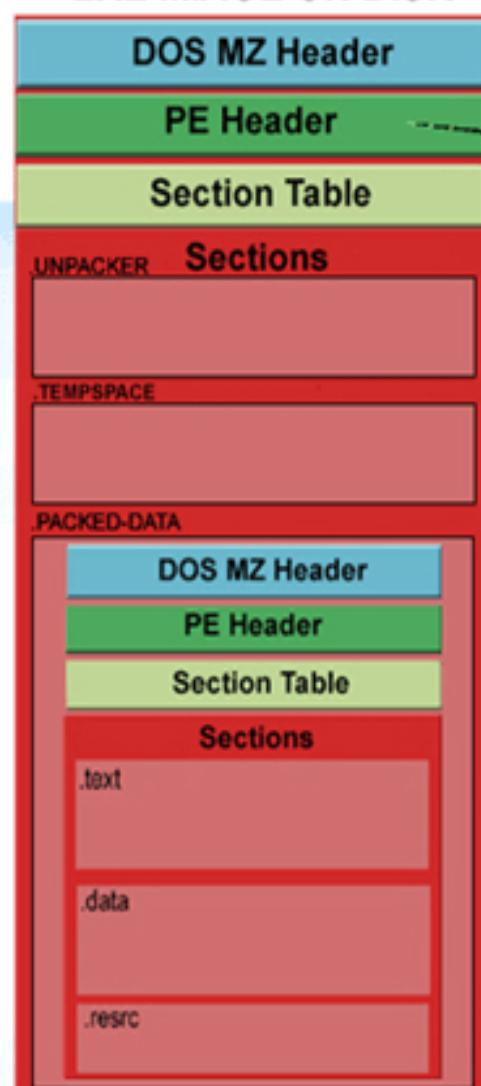
Thực thi tệp đã nén

EXE IMAGE ON DISK



Khi không chạy
trong môi trường
DOS thì phần này sẽ
bị bỏ qua

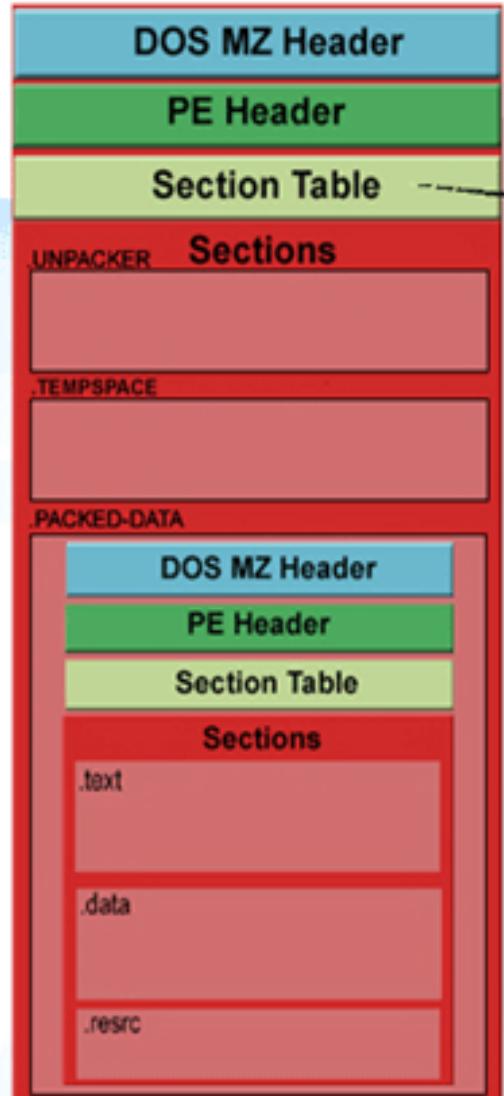
EXE IMAGE ON DISK



Tiếp theo chương trình sẽ
thực thi PE header

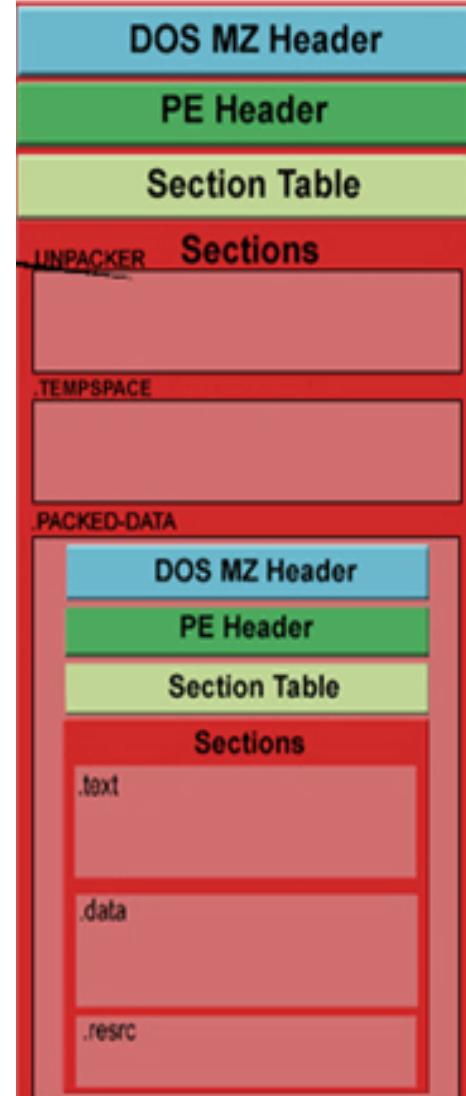
Thực thi tệp đã nén

EXE IMAGE ON DISK



Section table sẽ được
đọc để xem file có
những section nào

EXE IMAGE IN MEMORY

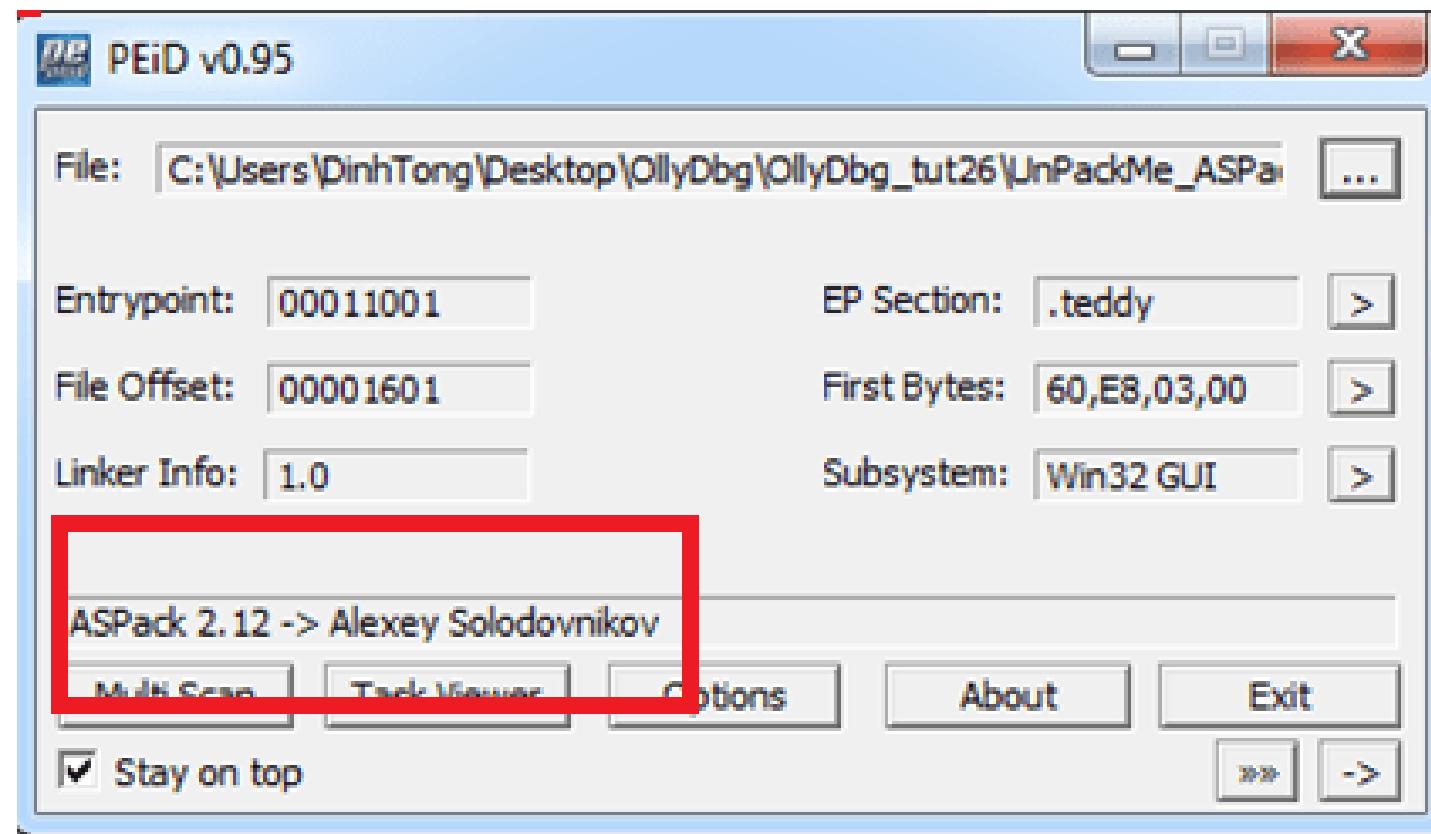


Chương trình sẽ nhảy
đến Entry Point của
section này và bắt đầu
thực thi

Thực thi tệp đã nén

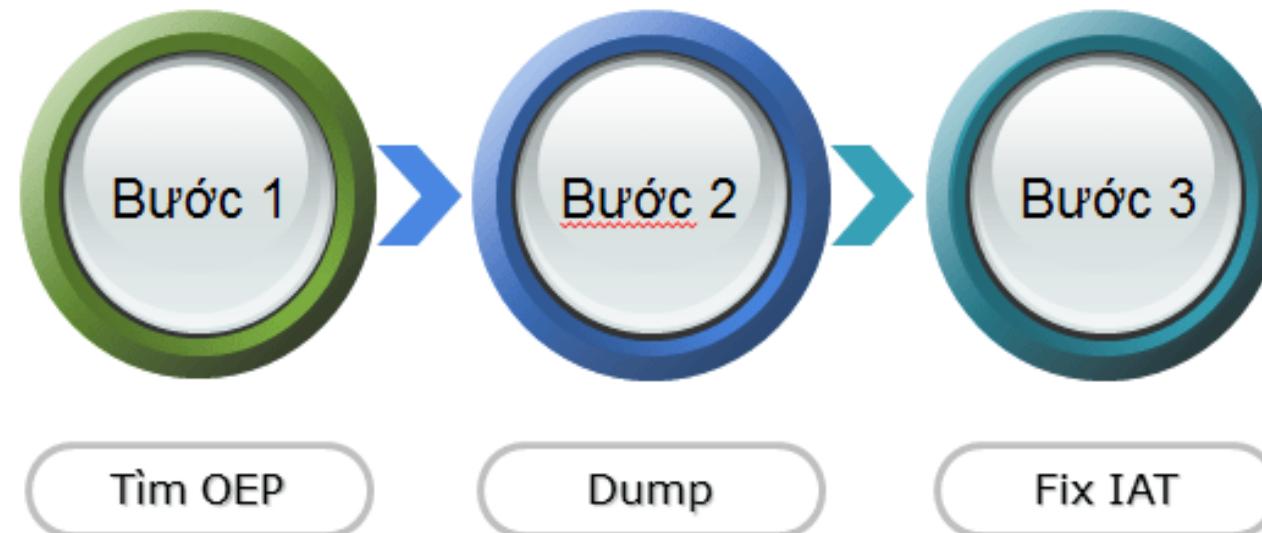
- Nhảy đến Entry Point của section UNPACKER, lưu lại toàn bộ giá trị của các thanh ghi bằng lệnh PUSHAD.
- Tính toán lại Import Table.
- Khôi phục lại giá trị các thanh ghi đã được lưu trong Stack bằng lệnh POPAD.
- Nhảy đến Origin EntryPoint và thực thi như file lúc chưa bị đóng gói.

Nhận biết một tệp tin bị nén



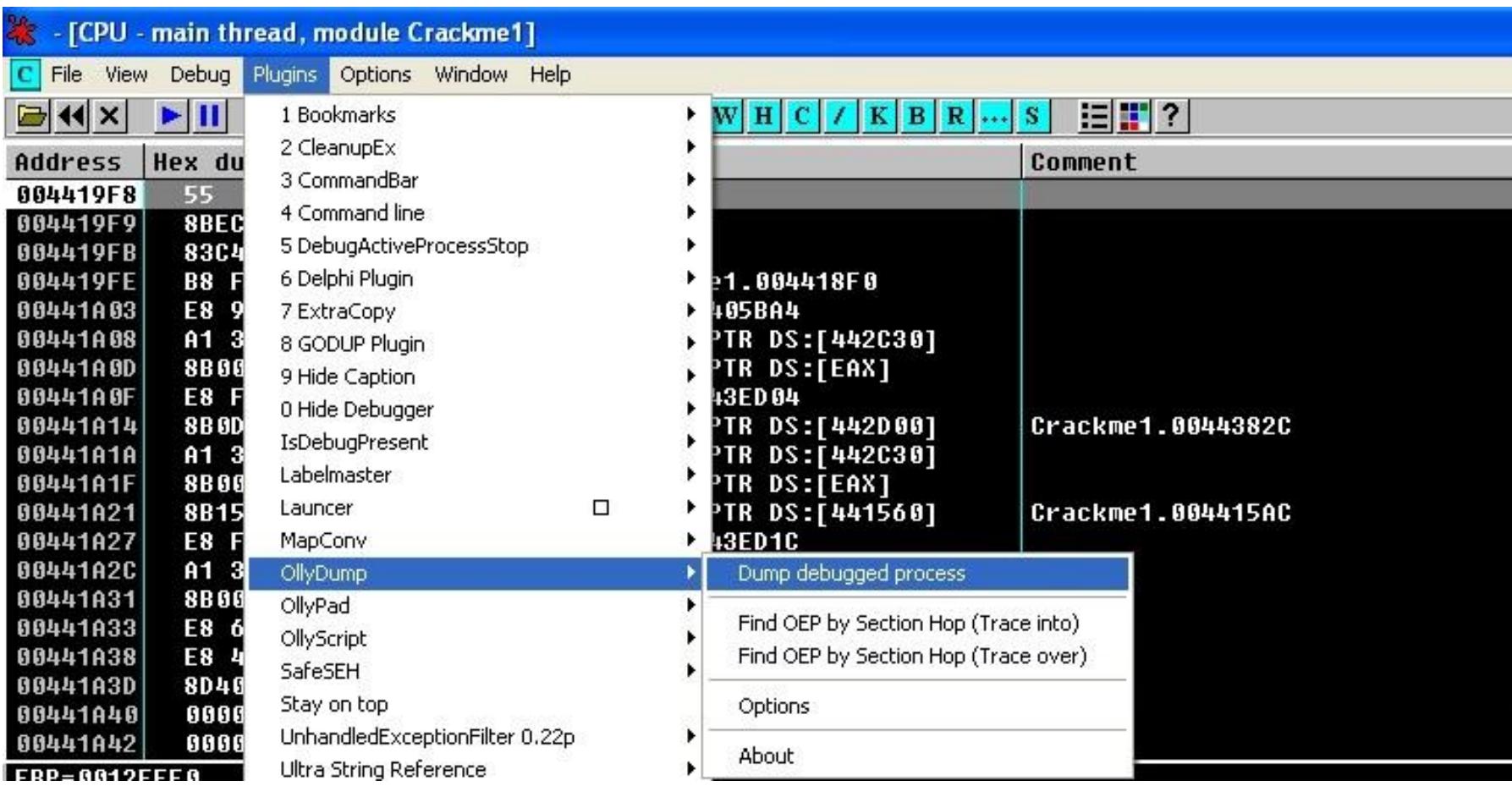
Giải nén

☐ Các bước cơ bản để thực hiện giải nén



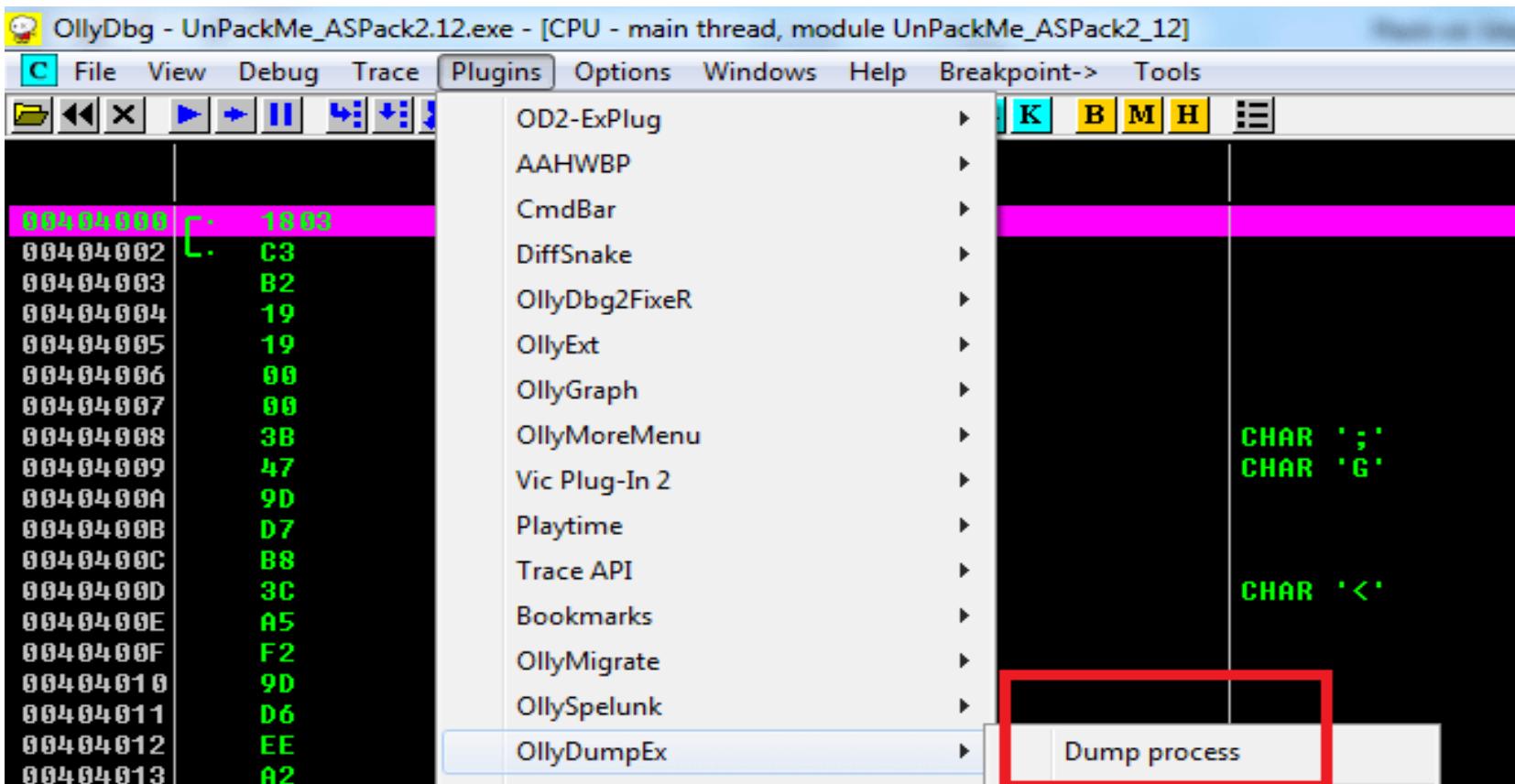
Tìm OEP

❑ Orginal entry point là nơi mà chương trình gốc thực sự bắt đầu thực thi.



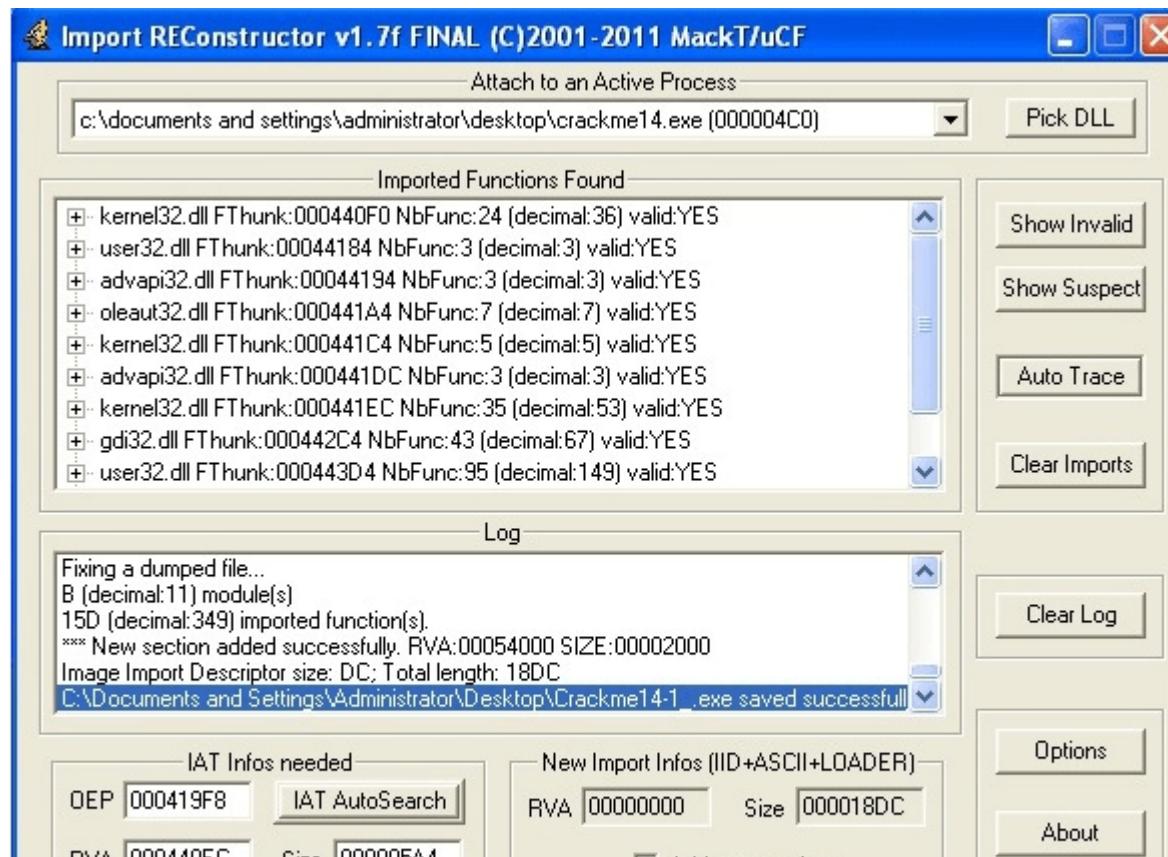
Dump file

□ Sau khi nhảy đến OEP ta sẽ tiến hành dump file. Mục đích của việc này là fix lại các section và import table như file ban đầu trước khi bị đóng gói.



Fix IAT

☐ Kiểm tra file xem còn các cơ chế Anti hoặc ngăn chặn việc thực thi hay không rồi tiến hành chỉnh sửa. Đảm bảo file sau unpack thực thi bình thường



Nội dung

- 1. Nén**
- 2. Mã hóa**
- 3. Làm rối mã**
- 4. Một số cơ chế khác**

Mã hóa

Lý do mã độc sử dụng các biện pháp mã hóa:

- Che giấu thông tin cấu hình, chẳng hạn như C&C Domain.**
- Lưu thông tin vào tệp tin trước khi đánh cắp**
- Che giấu mã độc bên trong một công cụ hợp pháp**
- Ẩn các chuỗi bị nghi ngờ**

Mã hóa

- Thuật toán mã hóa đơn giản.
- Thuật toán mã hóa mạnh
- Thuật toán mã hóa tùy chỉnh

Mã hóa

- Thuật toán mã hóa đơn giản
- Thuật toán mã hóa mạnh
- Thuật toán mã hóa tùy chỉnh

Thuật toán mã hóa đơn giản

- Kích thước nhỏ, phù hợp với các môi trường bị ràng buộc như khai thác bằng shellcode
- Ít ảnh hưởng đến hiệu năng
- Gây khó khăn trong việc phát hiện, tuy nhiên không thể qua mặt được các nhà phân tích có kỹ năng tốt

Mã Caesar

- Dịch chuyển từng chữ cái về trước 3 vị trí trong bảng chữ cái alphabet

ABCDEFGHIJKLMNOPQRSTUVWXYZ

DEFGHIJKLMNOPQRSTUVWXYZABC

- Ví dụ

ATTACK AT NOON

DWWDFN DW QRRQ

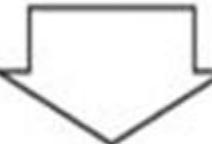
XOR

- ❑ Sử dụng một khóa để mã hóa
- ❑ Sử dụng một bit của dữ liệu và một bit của khóa trong một thời điểm
- ❑ Ví dụ: EncodeEncode HI với khóa 0x3C
HI = 0x48 0x49 (ASCII encoding)

Data:	0100 1000 0100 1001
Key:	0011 1100 0011 1100
Kết quả:	0111 0100 0111 0101

0 xor 0 = 0
0 xor 1 = 1
1 xor 0 = 1
1 xor 1 = 0

XOR

A	T	T	A	C	K		A	T		N	O	O	N
0x41	0x54	0x54	0x41	0x43	0x4B	0x20	0x41	0x54	0x20	0x4E	0x4F	0x4F	0x4E
													
}	h	h	}	DEL	W	FS	}	H	FS	r	s	s	r
0x7d	0x68	0x68	0x7d	0x7F	0x77	0x1C	0x7d	0x68	0x1C	0x72	0x71	0x71	0x72

The string *ATTACK AT NOON* encoded with an XOR of *0x3C*
(original string at the top; encoded strings at the bottom)

Xác định vòng lặp với lệnh XOR trong IDA Pro

- Các vòng lặp với lệnh XOR bên trong
- Bắt đầu với “IDA View” (xem code)
- Click Search, Text
- Nhập xor và tìm tất cả các lần xuất hiện

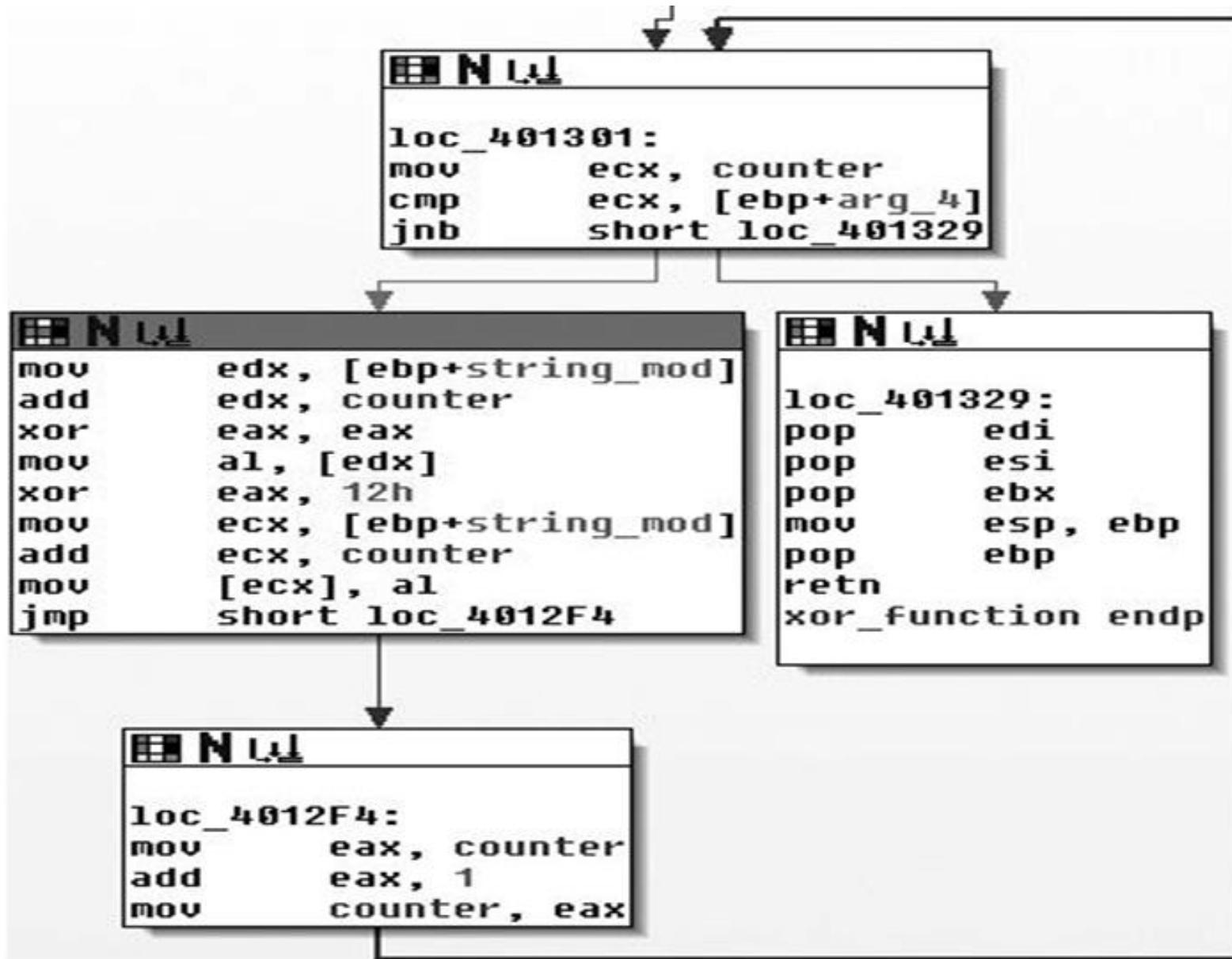
The screenshot shows a window titled "Occurrences of: xor". The window has two tabs at the top: "Edit" and "Search", with "Search" selected. The main area is a table with three columns: "Address", "Function", and "Instruction". The table lists several occurrences of the XOR instruction (opcode 33) across different memory locations and functions. The last row of the table indicates there are 31 total occurrences.

Address	Function	Instruction
.text:00401230	sub_401200	33 D2 xor edx, edx
.text:00401269	sub_401200	33 C9 xor ecx, ecx
.text:00401277	sub_401200	33 C0 xor eax, eax
.text:00401312	s_x_func	83 F2 12 xor edx, 12h
.text:00401395		33 C0 xor eax, eax
.text:00401470		32 C0 xor al, al
.text:004014D6		32 C0 xor al, al
.text:0040151F		32 C0 xor al, al

Line 1 of 31

Searching for XOR in IDA Pro

Xác định vòng XOR trong IDA Pro



Base64

- Chuyển đổi 6 bits thành một ký tự trong bảng chữ cái 64 ký tự
- Sử dụng các khối 3 byte (24 bits)
- Chia thành 4 trường 6-bits
- Chuyển từng trường thành base64

Base64

Part of raw email message showing Base64 encoding

```
Content-Type: multipart/alternative;
  boundary="_002_4E36B98B966D7448815A3216ACF82AA201ED633ED1MBX3THNDRBIRD_"
MIME-Version: 1.0
--_002_4E36B98B966D7448815A3216ACF82AA201ED633ED1MBX3THNDRBIRD_
Content-Type: text/html; charset="utf-8"
Content-Transfer-Encoding: base64
```

```
SWYgeW91IGFyZSBzZWFKaW5nIHRoaXMsIHlvdSBwcm9iYWJseSBzaG91bGQganVzdCBza2lwIHRoaX
MgY2hhcHRlcjBhbmcQgZ28gdG8gdGhlIG5leHQgb25lljBEbyB5b3UgcmbhbGx5IGHdmUgdGhlIHRp
bWUgdG8gdHlwZSB0aGlzIHdob2xlIHN0cmLuZyBpbj8gWW91IGFyZSBvYnZpb3VzbHkgdGFsZW50ZW
QuIE1heWJlIHlvdSBzaG91bGQgY29udGFjdCB0aGUgYXV0aG9ycyBhbmcQgc2VlIGlmIH
```

Base64

GET /X29tbVEuYC8=/index.htm

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Host: www.practicalmalwareanalysis.com

Connection: Keep-Alive

Cookie: Ym90NTQxNjQ

GET /c2UsYi1kYWM0cnUjdFlvbiAjb21wbFU0YP==/index.htm

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Host: www.practicalmalwareanalysis.com

Connection: Keep-Alive

Cookie: Ym90NTQxNjQ

URL và Cookie được mã hóa bằng Base64

Giải mã các URLs

- Tùy chỉnh “indexing string:

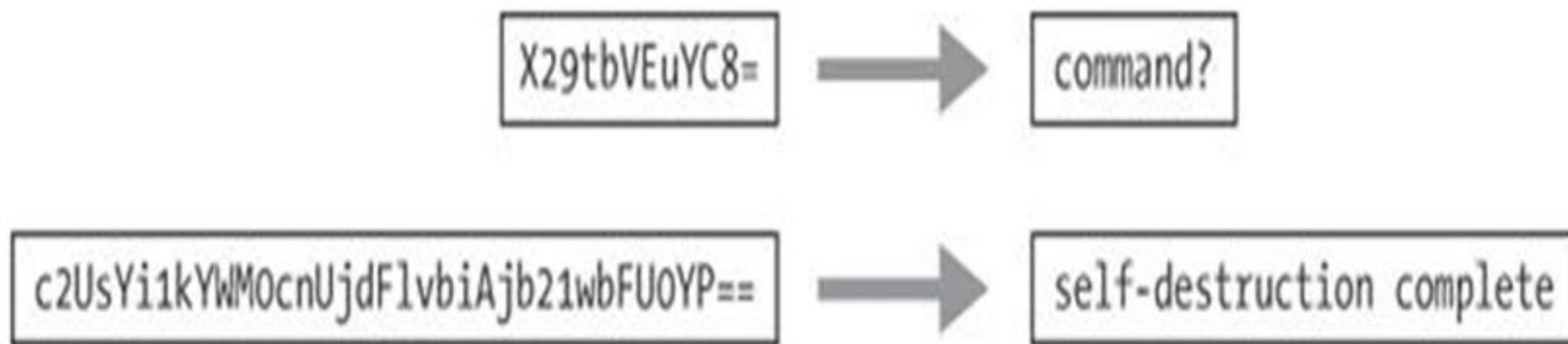
aABCDEF~~GHIJKL~~MNOPQRSTUVWXYZbcdefghijklmnopqrstuvwxyz0123456789+/

- Tìm ký tự = ở chuỗi đã được encoding



Unsuccessful attempt to decode Base64 string due to nonstandard indexing string

Giải mã các URLs



Successful decoding of Base64 string using custom indexing string

Mã hóa

- Thuật toán mã hóa đơn giản
- **Thuật toán mã hóa mạnh**
- Thuật toán mã hóa tùy chỉnh

Các thuật toán mã hóa mạnh

- AES, RSA,
- Chống lại các tấn công brute-force,
- Các thư viện mật mã chuẩn dễ bị phát hiện (các hàm được import, function matching, các hằng số mật mã),
- Yêu cầu nhiều tài nguyên tính toán

Các thuật toán mã hóa mạnh

Address	Ordinal	Name	Library
0408A068		RegEnumKeyExA	ADVAPI32
0408A0...		CryptAcquireContextA	ADVAPI32
0408A070		CryptCreateHash	ADVAPI32
0408A074		CryptHashData	ADVAPI32
0408A078		CryptDeriveKey	ADVAPI32
0408A0...		CryptDestroyHash	ADVAPI32
0408A080		CryptDecrypt	ADVAPI32
0408A084		CryptEncrypt	ADVAPI32
0408A088		RegOpenKeyExA	ADVAPI32

IDA Pro imports listing showing cryptographic functions

Mã hóa

- Thuật toán mã hóa đơn giản
- Thuật toán mã hóa mạnh
- **Thuật toán mã hóa tùy chỉnh**

Thuật toán mã hóa tùy chỉnh

- ❑ Do người dùng tự phát triển
- ❑ Tùy chỉnh của các thuật toán mã hóa được công bố
- ❑ Ví dụ: Một vòng XOR, sau đó Base64
- ❑ Gây khó khăn cho quá trình phân tích

Giải mã

- ❑ Lập trình các hàm giải mã
- ❑ Có thể sử dụng lại những hàm trong chính mã độc

Lập trình các hàm giải mã

☐ Một số hàm chuẩn sẵn có

Sample Python Base64 script

```
import string
import base64

example_string = 'VGhpcyBpcyBhIHRlc3Qgc3RyaW5n'
print base64.decodestring(example_string)
```

Sample Python NULL-preserving XOR script

```
def null_preserving_xor(input_char, key_char):
    if (input_char == key_char or input_char == chr(0x00)):
        return input_char
    else:
        return chr(ord(input_char) ^ ord(key_char))
```

Lập trình các hàm giải mã

☐ Một số hàm chuẩn sẵn có

Sample Python custom Base64 script

```
import string
import base64

s = ""
custom = "9ZABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345678+/"
Base64 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+"

ciphertext = 'TEgobxZobxZgGFPkb20='

for ch in ciphertext:
    if (ch in Base64):
        s = s + Base64[string.find(custom,str(ch))]

    elif (ch == '='):
        s += '='

result = base64.decodestring(s)
```

Lập trình các hàm giải mã

□ Một số thư viện mật mã (PyCrypto,...)

Sample Python DES script

```
from Crypto.Cipher import DES  
import sys  
  
obj = DES.new('password',DES.MODE_ECB)  
cfile = open('encrypted_file','r')  
cbuf = f.read()  
print obj.decrypt(cbuf)
```

Nội dung

- 1. Nén**
- 2. Mã hóa**
- 3. Làm rối mã**
- 4. Một số cơ chế khác**

Làm rối mã

- ❑ Mục tiêu: Ngăn chặn việc phân tích, phát hiện dựa vào dấu hiệu (signature) và chống dịch ngược
- ❑ Code được Obfuscation và mutation
- ❑ Phát hiện debuggers và máy ảo nó sẽ dừng và không thực thi hành vi của nó nữa

Obfuscation

- Đặt lại các lệnh, điều kiện bị đảo ngược, tên các thanh ghi khác nhau, trật tự khác nhau,...
- Các lệnh JUMP và NOP được chèn vào những vị trí ngẫu nhiên
- Bộ Garbage opcodes được chèn vào các khu vực không thể chèn mã
- Các lệnh được thay thế bằng những lệnh khác nhưng vẫn đảm bảo giống chức năng

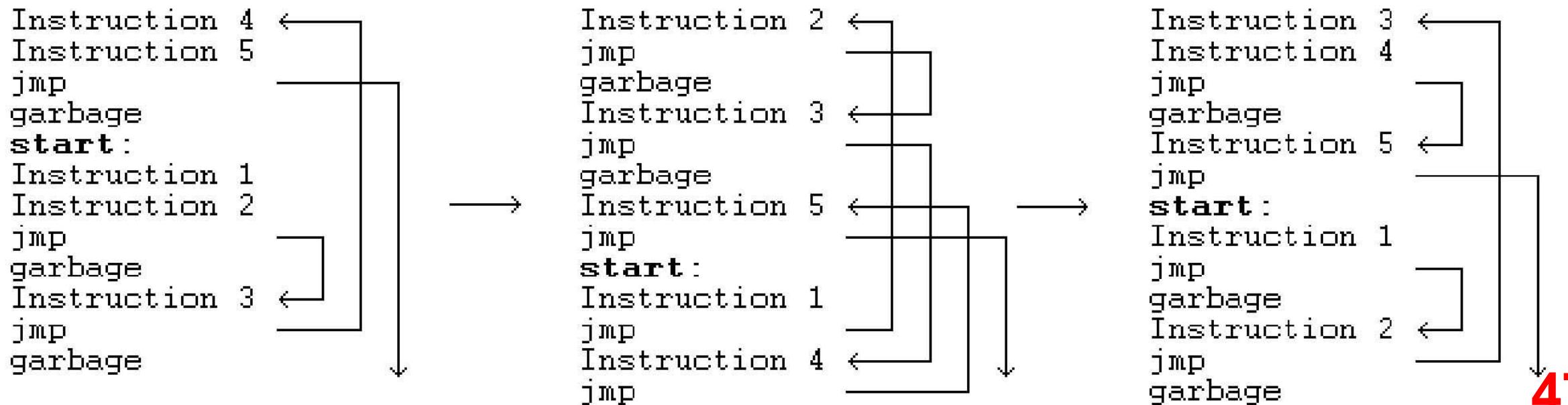
Polymorphic Viruses

- ❑ Mỗi bản sao tạo ra một bản mã ngẫu nhiên mới của cùng một virus
- ❑ Mã hóa > giải mã > Mã hóa > ...

Metamorphic Viruses

- ❑ Lai tạo và kết hợp nhiều kiểu đa hình khác nhau trong cùng một virus
- ❑ Tự động biến đổi, lai tạp, hình thành các thế hệ virus F1, F2, F3... Fn
- ❑ Các đoạn mã độc được rải rác và nằm ở nhiều nơi

- ☐ “Islands” của mã được tích hợp vào các vị trí ngẫu nhiên trong chương trình của máy lưu trữ và liên kết bằng các lệnh nhảy
- ☐ Virus tự ghép các phần của nó (tích hợp mã)



Nội dung

- 1. Nén**
- 2. Mã hóa**
- 3. Làm rối mã**
- 4. Một số cơ chế khác**

Một số cơ chế khác

❑ Anti debugging

(PMA chapter 16)

❑ Anti VM

(PMA chapter 17)

Nội dung

1. Nén
2. Mã hóa
3. Làm rối mã
4. Một số cơ chế khác

Mã độc

Chương 8. Phòng chống mã độc

Mục tiêu

- Giới thiệu một số biện pháp phòng chống mã độc

Tài liệu tham khảo

[1] TS. Lương Thế Dũng, KS. Hoàng Thanh Nam,
2013, Giáo trình Mã độc, Học viện kỹ thuật Mật mã

Nội dung

1. Xây dựng chính sách phòng chống mã độc
2. Nâng cao nhận thức
3. Quản lý các lỗ hổng
4. Triển khai các công nghệ phòng chống mã độc

Nội dung

- 1. Xây dựng chính sách phòng chống mã độc**
- 2. Nâng cao nhận thức**
- 3. Quản lý các lỗ hổng**
- 4. Triển khai các công nghệ phòng chống mã độc**

Chính sách phòng chống mã độc

- Yêu cầu dùng phần mềm quét các thiết bị lưu trữ của các đơn vị bên ngoài tổ chức trước khi sử dụng.
- Yêu cầu những tập tin đính kèm trong thư điện tử, bao gồm cả các tập tin nén như file .zip cần được lưu vào ổ đĩa và kiểm tra trước khi được mở ra.
- Cấm gửi hoặc nhận một số loại tập tin có các đuôi tập tin là exe qua thư điện tử.

Chính sách phòng chống mã độc

- Hạn chế hoặc cấm việc sử dụng phần mềm không cần thiết, ví dụ như các ứng dụng những dịch vụ không cần thiết hoặc các phần mềm được cung cấp bởi các tổ chức không rõ nguồn gốc.**
- Hạn chế cung cấp quyền quản trị cho người sử dụng**
- Yêu cầu luôn cập nhật phần mềm, các bản vá, cho hệ điều hành.**

Chính sách phòng chống mã độc

- Hạn chế sử dụng các thiết bị di động (ví dụ: đĩa mềm, đĩa CD, USB), đặc biệt là trên hệ thống có nguy cơ ảnh hưởng cao, như các điểm truy cập công cộng.**
- Yêu cầu nêu rõ các loại phần mềm phòng chống mã độc đối với từng hệ thống và các ứng dụng.**

Chính sách phòng chống mã độc

- Người dùng nếu muốn có quyền truy cập vào các mạng khác (bao gồm cả Internet) cần thông qua sự đồng ý của tổ chức.**
- Yêu cầu thay đổi cấu hình tường lửa để phù hợp với chính sách công ty**
- Hạn chế việc sử dụng các thiết bị di động trên các mạng tin cậy.**

Nội dung

1. Xây dựng chính sách phòng chống mã độc
2. Nâng cao nhận thức
3. Quản lý các lỗ hổng
4. Triển khai các công nghệ phòng chống mã độc

Nâng cao nhận thức

- Hướng dẫn cho các cán bộ, nhân viên cách phòng tránh sự cố liên quan đến mã độc hại, giảm thiểu mức độ nghiêm trọng của sự cố.
- Tất cả cán bộ, nhân viên trong tổ chức đều phải được đào tạo về các nguy cơ, cách thức phần mềm độc hại xâm nhập vào hệ thống, lây nhiễm, lây lan.

Nâng cao nhận thức

Không thực hiện một số công việc như sau:

- Không truy cập vào những trang web có khả năng chứa nội dung độc hại.
- Không mở các tập tin với phần mở rộng có khả năng kết hợp với phần mềm độc hại (Ví dụ: .bat, .exe, .pif, .vbs...).

Nâng cao nhận thức

Không thực hiện một số công việc như sau:

- Không mở những thư điện tử hoặc tập tin đính kèm từ những địa chỉ của người gửi không rõ ràng hoặc có dấu hiệu nghi ngờ.
- Không truy cập vào các popup trên trình duyệt mà cảm thấy nghi ngờ hoặc có dấu hiệu bất thường.

Nâng cao nhận thức

Một số khuyến cáo:

- Không trả lời các thư điện tử yêu cầu cung cấp các thông tin tài chính và thông tin cá nhân.
- Không cung cấp mật khẩu, mã PIN hoặc các loại mã truy cập khác để trả lời thư điện tử hay điền thông tin vào popup hiển thị không mong muốn. Chỉ nhập thông tin vào các trang web chính thống của tổ chức.

Nâng cao nhận thức

Một số khuyến cáo:

- Không mở các tập tin đính kèm đáng ngờ trong email, thậm chí nếu những email này đến từ những người gửi đã biết.
- Không trả lời bất kỳ email nào đáng ngờ hoặc không mong muốn.

Nội dung

1. Xây dựng chính sách phòng chống mã độc
2. Nâng cao nhận thức
3. Quản lý các lỗ hổng
4. Triển khai các công nghệ phòng chống mã độc

Nội dung

- Quản lý bản vá
- Đặc quyền tối thiểu
- Biện pháp hỗ trợ khác

Nội dung

- Quản lý bản vá
- Đặc quyền tối thiểu
- Biện pháp hỗ trợ khác

Biện pháp hỗ trợ khác

- Vô hiệu hóa, gỡ bỏ những dịch vụ không cần thiết.
- Loại bỏ những tập tin chia sẻ không đảm bảo.
- Sử dụng những tên đăng nhập và mật khẩu phức tạp phù hợp với chính sách của công ty.
- Yêu cầu xác thực trước khi cho phép truy cập vào dịch vụ mạng.
- Vô hiệu hóa cơ chế tự động thực thi các tệp tin nhị phân và các tệp tin scripts.

Nội dung

1. Xây dựng chính sách phòng chống mã độc
2. Nâng cao nhận thức
3. Quản lý các lỗ hổng
4. Triển khai các công nghệ phòng chống mã độc

Triển khai các công nghệ phòng chống mã độc

- Phần mềm chống virus
- Phát hiện phần mềm gián điệp
- Ngăn ngừa sự xâm nhập hệ thống (IDS)
- Tường lửa

Mã độc

Chương 9. Phát hiện và xử lý sự cố mã độc

Mục tiêu

- Giới thiệu quy trình xử lý sự cố mã độc

Tài liệu tham khảo

[1] TS. Lương Thế Dũng, KS. Hoàng Thanh Nam,
2013, Giáo trình Mã độc, Học viện kỹ thuật Mật mã

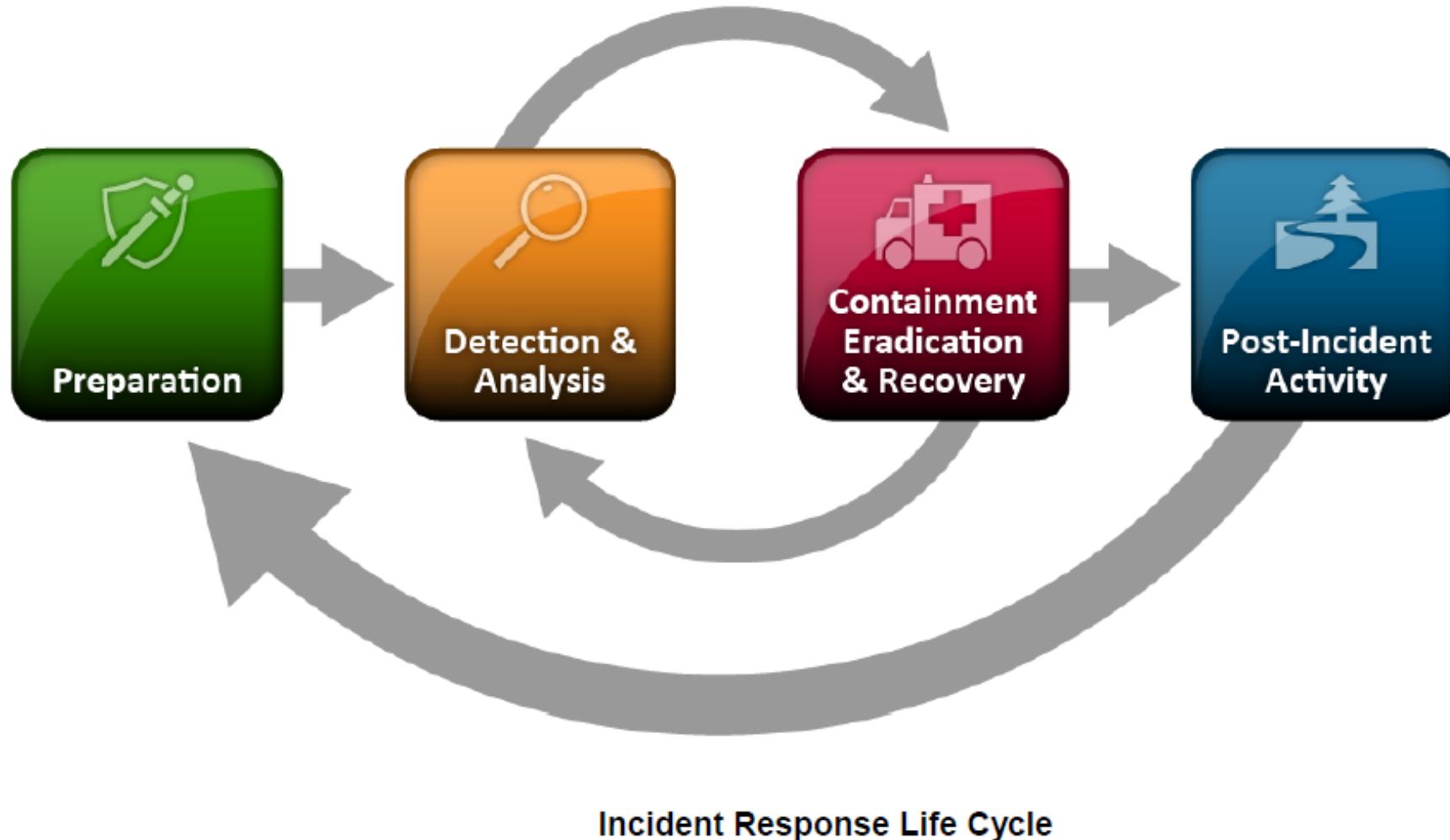
Nội dung

1. Quy trình xử lý sự cố mã độc
2. Chuẩn bị
3. Phát hiện và phân tích
4. Ngăn chặn
5. Loại bỏ
6. Phục hồi
7. Các hoạt động sau sự cố

Nội dung

- 1. Quy trình xử lý sự cố mã độc**
- 2. Chuẩn bị**
- 3. Phát hiện và phân tích**
- 4. Ngăn chặn**
- 5. Loại bỏ**
- 6. Phục hồi**
- 7. Các hoạt động sau sự cố**

Quy trình xử lý sự cố mã độc



Chuẩn bị

Giai đoạn ban đầu của phản ứng sự cố phần mềm độc hại bao gồm thực hiện các hoạt động chuẩn bị:

- Phát triển các quy trình xử lý sự cố dành riêng cho phần mềm độc hại**
- Chương trình đào tạo cho các đội ứng phó sự cố**
- Xây dựng bộ công cụ**

Phát hiện và phân tích

- Cảnh báo cho tổ chức bất cứ khi nào sự cố xảy ra
- Phát hiện sớm các sự cố phần mềm độc hại

Ngăn chặn, loại bỏ và hồi phục

- Giảm thiểu tác động của phần mềm độc hại**
- Tiêu diệt các tác hại của mã độc**
- Phục hồi sau sự cố**

Hoạt động sau sự cố

- Báo cáo chi tiết nguyên nhân và thiệt hại**
- Các bước cần thực hiện để ngăn ngừa sự cố trong tương lai**
- Chuẩn bị hiệu quả để xử lý các sự cố sẽ xảy ra trong tương lai**

Nội dung

1. Quy trình xử lý sự cố mã độc

2. Chuẩn bị

3. Phát hiện và phân tích

4. Ngăn chặn

5. Loại bỏ

6. Phục hồi

7. Các hoạt động sau sự cố

Chuẩn bị

Các tổ chức nên thực hiện các biện pháp chuẩn bị để đảm bảo rằng họ có khả năng ứng phó hiệu quả với các sự cố phần mềm độc hại:

- Chuẩn bị xử lý sự cố
- Ngăn ngừa sự cố

Chuẩn bị xử lý sự cố

Tài nguyên dùng để phân tích sự cố:

- Tài liệu
- Sơ đồ mạng và danh sách các tài sản quan trọng
- Đường cơ sở hiện tại của mạng, hệ thống và hoạt động ứng dụng dự kiến
- Giá trị hàm băm mật mã của các tập tin quan trọng

Ngăn ngừa sự cố

- Đánh giá rủi ro định kỳ của các hệ thống và ứng dụng**
- Ngăn chặn phần mềm độc hại: Phần mềm phát hiện và ngăn chặn phần mềm độc hại**
- Nhận thức và đào tạo người dùng**

Nội dung

- 1. Quy trình xử lý sự cố mã độc**
- 2. Chuẩn bị**
- 3. Phát hiện và phân tích**
- 4. Ngăn chặn**
- 5. Loại bỏ**
- 6. Phục hồi**
- 7. Các hoạt động sau sự cố**

Phát hiện và phân tích

- Dấu hiệu của sự cố
- Phân tích sự cố
- Tài liệu hóa sự cố, thông báo sự cố

Dấu hiệu của sự cố

Sự cố có thể được phát hiện thông qua nhiều phương tiện khác nhau: Khả năng phát hiện tự động bao gồm IDPS, dựa trên mạng và máy chủ lưu trữ, phần mềm chống vi-rút và máy phân tích nhật ký; vẫn đề được báo cáo bởi người dùng.

Source	Description
Alerts	
IDPSs	<p>IDPS products identify suspicious events and record pertinent data regarding them, including the date and time the attack was detected, the type of attack, the source and destination IP addresses, and the username (if applicable and known). Most IDPS products use attack signatures to identify malicious activity; the signatures must be kept up to date so that the newest attacks can be detected. IDPS software often produces <i>false positives</i>—alerts that indicate malicious activity is occurring, when in fact there has been none. Analysts should manually validate IDPS alerts either by closely reviewing the recorded supporting data or by getting related data from other sources.³¹</p>
SIEMs	<p>Security Information and Event Management (SIEM) products are similar to IDPS products, but they generate alerts based on analysis of log data (see below).</p>
Antivirus and antispam software	<p>Antivirus software detects various forms of malware, generates alerts, and prevents the malware from infecting hosts. Current antivirus products are effective at stopping many instances of malware if their signatures are kept up to date. Antispam software is used to detect spam and prevent it from reaching users' mailboxes. Spam may contain malware, phishing attacks, and other malicious content, so alerts from antispam software may indicate attack attempts.</p>
File integrity checking software	<p>File integrity checking software can detect changes made to important files during incidents. It uses a hashing algorithm to obtain a cryptographic checksum for each designated file. If the file is altered and the checksum is recalculated, an extremely high probability exists that the new checksum will not match the old checksum. By regularly recalculating checksums and comparing them with previous values, changes to files can be detected.</p>
Third-party monitoring services	<p>Third parties offer a variety of subscription-based and free monitoring services. An example is fraud detection services that will notify an organization if its IP addresses, domain names, etc. are associated with current incident activity involving other organizations. There are also free real-time blacklists with similar information. Another example of a third-party monitoring service is a CSIRC notification list; these lists are often available only to other incident response teams.</p>

Phân tích sự cố

Các khuyến nghị để làm cho phân tích sự cố dễ dàng và hiệu quả hơn:

- Hồ sơ mạng và hệ thống
- Hiểu các hành vi bình thường
- Tạo Chính sách lưu giữ nhật ký
- Thực hiện tương quan sự kiện
- Giữ đồng bộ tất cả máy chủ
- Thu thập dữ liệu bổ sung, lọc dữ liệu
- Sử dụng công cụ tìm kiếm Internet để nghiên cứu

Tài liệu hóa sự cố, thông báo sự cố

Các yếu tố liên quan đến sự cố:

- Tác động chức năng của sự cố
- Tác động thông tin của sự cố
- Khả năng phục hồi từ sự cố

Thông báo sự cố: CIO, Trưởng phòng an ninh thông tin, nhân viên an ninh thông tin

Nội dung

1. Quy trình xử lý sự cố mã độc
2. Chuẩn bị
3. Phát hiện và phân tích
4. Ngăn chặn
5. Loại bỏ
6. Phục hồi
7. Các hoạt động sau sự cố

Ngăn chặn

- Chọn chiến lược ngăn chặn
- Thu thập và xử lý bằng chứng
- Xác định máy chủ tấn công

Chọn chiến lược ngăn chặn

Tiêu chí để xác định chiến lược phù hợp:

- Thiệt hại tiềm tàng và trộm cắp tài nguyên
- Cần bảo quản bằng chứng
- Tính khả dụng của dịch vụ (ví dụ: kết nối mạng, dịch vụ được cung cấp cho bên ngoài)

Chọn chiến lược ngăn chặn

Tiêu chí để xác định chiến lược phù hợp:

- Thời gian và nguồn lực cần thiết để thực hiện chiến lược
- Hiệu quả của chiến lược (ví dụ: ngăn chặn một phần, ngăn chặn hoàn toàn)
- Thời gian

Thu thập và xử lý bằng chứng

Một bản ghi chi tiết nên được lưu giữ cho tất cả các bằng chứng:

- Xác định thông tin (vị trí, số sê-ri, số kiểu máy, tên máy chủ, địa chỉ MAC và địa chỉ IP của máy tính)
- Tên, tiêu đề và số điện thoại của từng cá nhân đã thu thập hoặc xử lý bằng chứng trong quá trình điều tra
- Thời gian của mỗi lần xử lý bằng chứng
- Vị trí lưu trữ chứng cứ

Xác định máy chủ tấn công

Các hoạt động được thực hiện phổ biến nhất để tấn công nhận dạng máy chủ:

- Xác thực địa chỉ IP tấn công máy chủ tấn công
- Nghiên cứu máy chủ tấn công thông qua công cụ tìm kiếm
- Sử dụng cơ sở dữ liệu sự cố
- Giám sát các kênh truyền thông của kẻ tấn công có thể

Nội dung

1. Quy trình xử lý sự cố mã độc
2. Chuẩn bị
3. Phát hiện và phân tích
4. Ngăn chặn
5. Loại bỏ
6. Phục hồi
7. Các hoạt động sau sự cố

Loại bỏ

- Mục tiêu chính của là loại bỏ phần mềm độc hại khỏi các máy chủ bị nhiễm .
- Các tình huống khác nhau đòi hỏi sự kết hợp khác nhau của các kỹ thuật, các công cụ phổ biến nhất để diệt trừ: phần mềm phòng chống mã độc, công nghệ quản lý lỗ hổng, phần mềm kiểm soát truy cập mạng...

Nội dung

1. Quy trình xử lý sự cố mã độc
2. Chuẩn bị
3. Phát hiện và phân tích
4. Ngăn chặn
5. Loại bỏ
6. Phục hồi
7. Các hoạt động sau sự cố

Phục hồi

Phục hồi có thể bằng các hành động như:

- Khôi phục hệ thống từ bản sao lưu sạch,
- Xây dựng lại hệ thống từ đầu,
- Thay thế các tập tin bị xâm nhập bằng các phiên bản sạch,
- Cài đặt các bản vá, thay đổi mật khẩu
- Thắt chặt an ninh mạng vành đai

Nội dung

- 1. Quy trình xử lý sự cố mã độc**
- 2. Chuẩn bị**
- 3. Phát hiện và phân tích**
- 4. Ngăn chặn**
- 5. Loại bỏ**
- 6. Phục hồi**
- 7. Các hoạt động sau sự cố**

Các hoạt động sau sự cố

Sau một sự cố liên quan đến mã độc các đơn vị tổ chức cần thực hiện những công việc:

- Sửa đổi chính sách an ninh, chính sách bảo mật có thể ngăn ngừa sự cố tương tự,
- Đào tạo nâng cao nhận thức bảo mật cho người sử dụng,

Các hoạt động sau sự cố

- Cấu hình lại các phần mềm, hệ điều hành hoặc cấu hình ứng dụng để hỗ trợ chính sách bảo mật,
- Triển khai, cấu hình lại phần mềm phát hiện mã độc.