

# Mã độc

## Chương 4. Kỹ thuật phân tích mã độc dựa trên gỡ rối

# Mục tiêu

- **Giới thiệu kỹ thuật gỡ rối**
- **Giới thiệu, hướng dẫn sinh viên sử dụng trình gỡ 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](https://samsclass.info/126/126_S17.shtml)**

# Nội dung

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

# Nội dung

## 1. Gỡ rối

2. Gỡ rối mức mã nguồn và gỡ rối mức mã Assembly

3. Gỡ rối chế độ nhân và chế độ người dùng

4. Sử dụng trình gỡ rối trong quá trình phân tích mã độc

5. Ngoại lệ

6. Chỉnh sửa ngoại lệ với trình gỡ 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 gỡ rối

□ Trình gỡ 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 gỡ rối

## ☐ Ollydbg

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

## ☐ Windbg

- Hỗ trợ gỡ rối mức nhân



# Nội dung

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

# Gỡ 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
- ☐ Dừng trong quá trình phân tích mã độc vì không có mã nguồn của mã độc

# Nội dung

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

# Gỡ rối chế độ người dùng

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

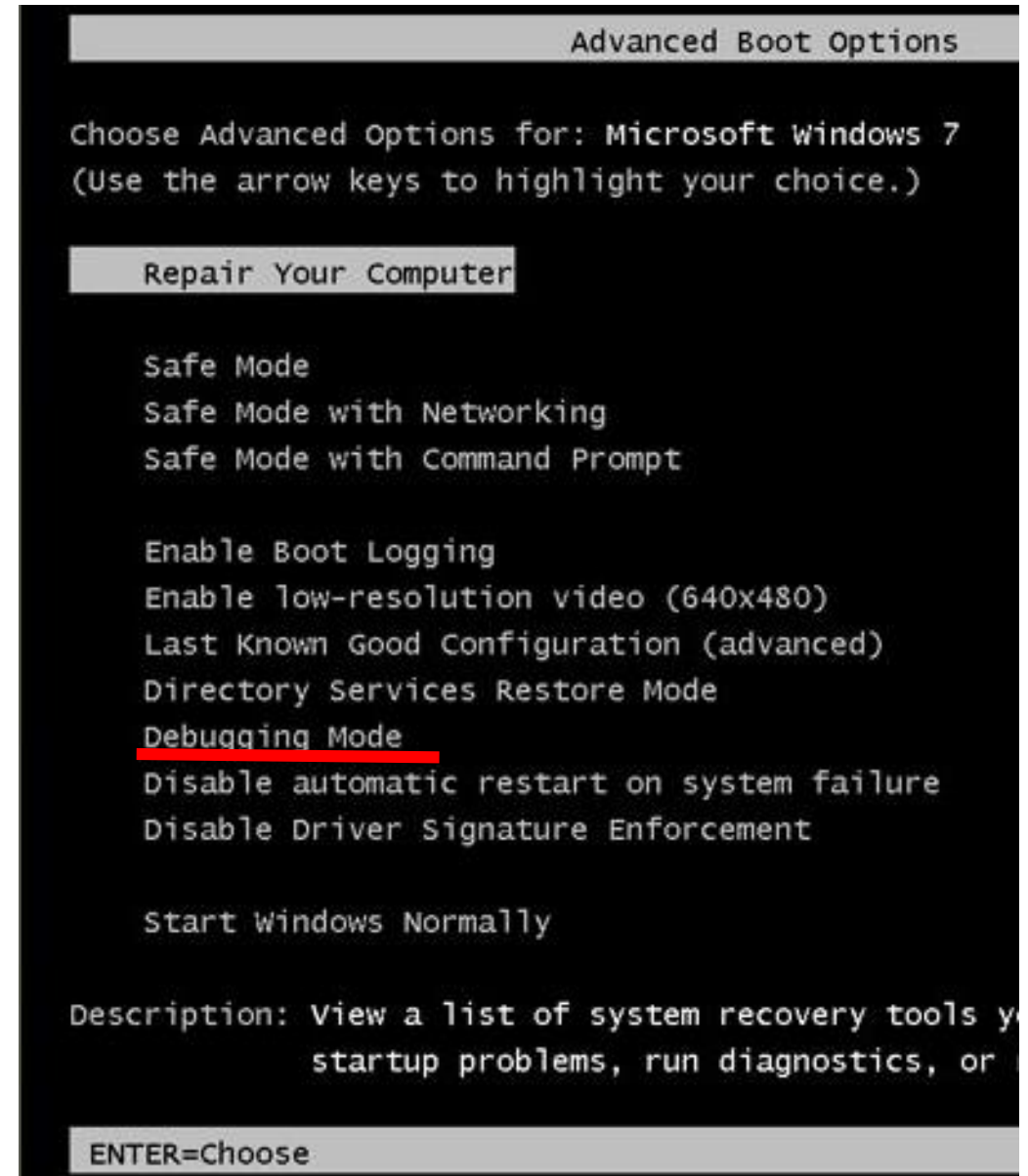
# Gỡ rối chế độ nhân

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



# Gỡ rối chế độ nhân

- ❑ Ấn F8 trong quá trình khởi động
- ❑ "Debugging Mode"



# Nội dung

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



# Sử dụng trình gỡ 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 gỡ 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

# Dừng 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

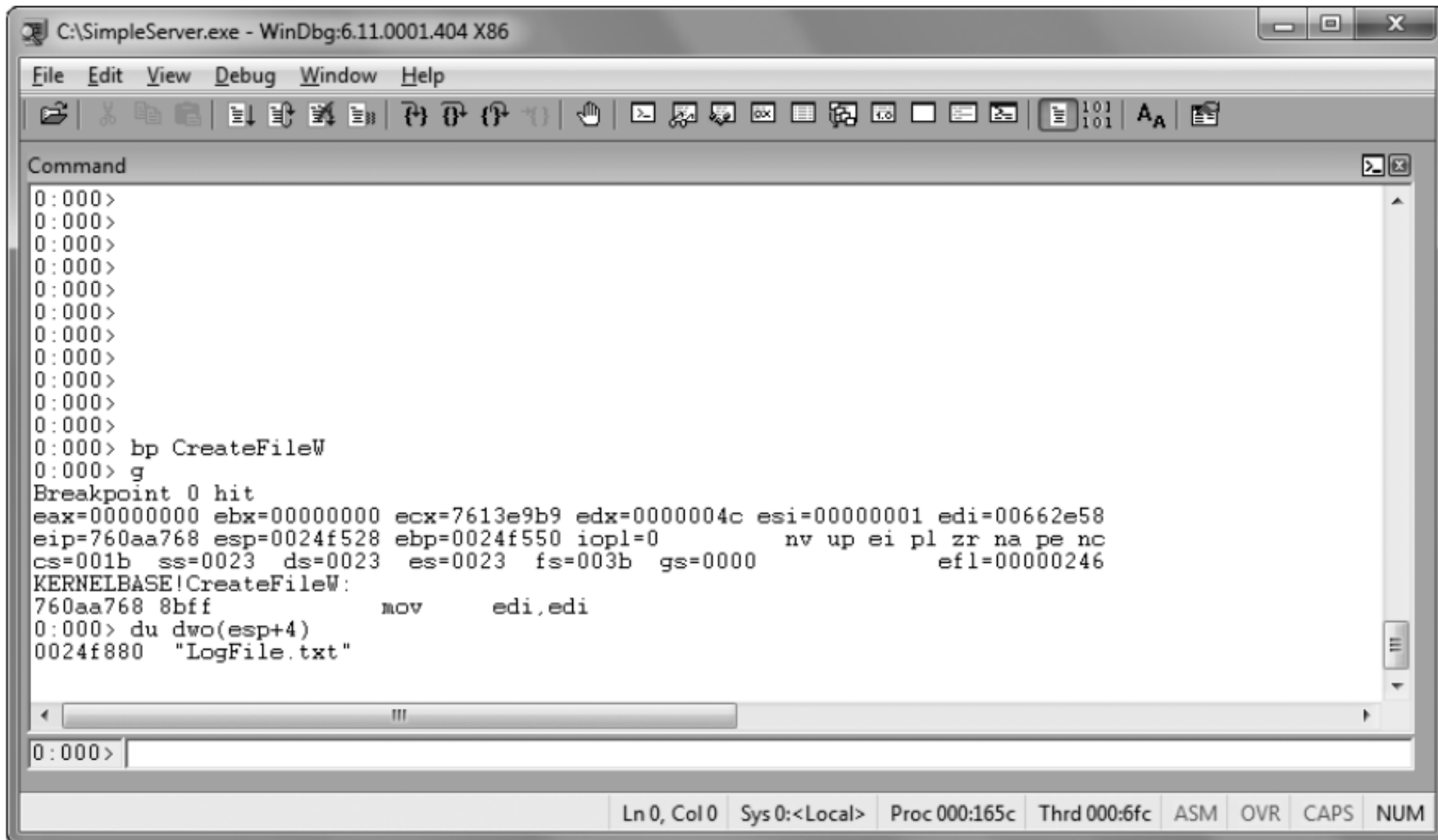
❑ Đoạn chương trình  
tín tên file, sau đó tạo  
file

❑ 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, 0FFFFFFFh
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 call    CreateFileW ; CreateFileW(x,x,x,x,x,x,x,x)
```

# Dừng thực thi với Breakpoints



The screenshot shows the WinDbg interface with the following content:

- Window title: C:\SimpleServer.exe - WinDbg:6.11.0001.404 X86
- Menu bar: File, Edit, View, Debug, Window, Help
- Toolbar: Standard debugging tools including Run, Step Over, Step Into, etc.
- Command window:

```
0:000>
0:000>
0:000>
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"
```
- Status bar: Ln 0, Col 0 | Sys 0:<Local> | 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 gỡ rối sẽ ghi đè lệnh 0xCC (INT 3)
- ❑ Thường là loại breakpoint mặc định của các trình gỡ rối

Disassembly and Memory Dump of a Function with a Breakpoint Set

Disassembly view				Memory dump
00401130	55	❶ push	ebp	00401130 ❷ CC 8B EC 83
00401131	8B 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

00C3FDB0	0095007C	CALL to VirtualAlloc from 00950079
00C3FDB4	00000000	Address = NULL
00C3FDB8	00000029	Size = 29 (41.)
00C3FDBC	00001000	AllocationType = MEM_COMMIT
00C3FDC0	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. Gỡ rối
2. Gỡ rối mức mã nguồn và gỡ rối mức mã Assembly
3. Gỡ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gỡ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gỡ rối
7. Phân tích mã độc với OllyDbg



# Ngoại lệ (Exception)

- ❑ Sử dụng bởi trình gỡ 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"><li>- Instruction address breakpoint fault</li><li>- Data address breakpoint trap</li><li>- General detect fault</li><li>- Single-step trap</li><li>- Task-switch breakpoint trap</li></ul>
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. Gỡ rối
2. Gỡ rối mức mã nguồn và gỡ rối mức mã Assembly
3. Gỡ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gỡ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gỡ rối
7. Phân tích mã độc với OllyDbg

# Bỏ qua một hàm

- ❑ Có thể thay đổi cờ 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. Gỡ rối
2. Gỡ rối mức mã nguồn và gỡ rối mức mã Assembly
3. Gỡ rối chế độ nhân và chế độ người dùng
4. Sử dụng trình gỡ rối trong quá trình phân tích mã độc
5. Ngoại lệ
6. Chỉnh sửa ngoại lệ với trình gỡ rối
7. Phân tích mã độc với OllyDbg

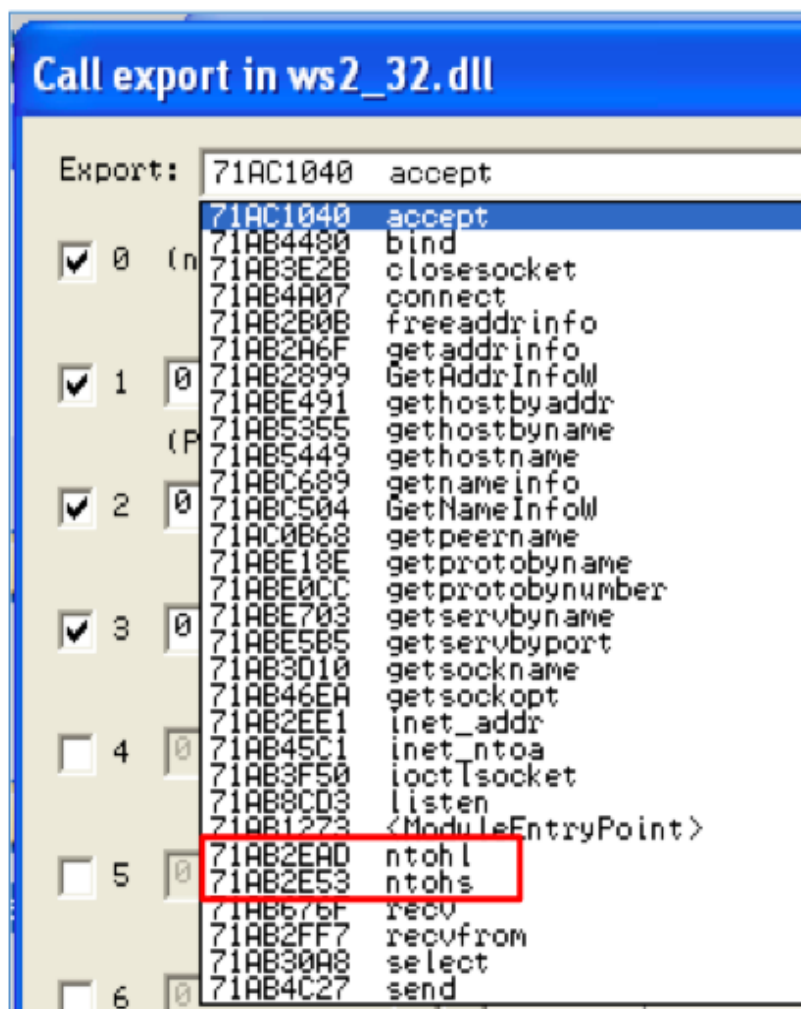
# OllyDbg

- ☐ 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

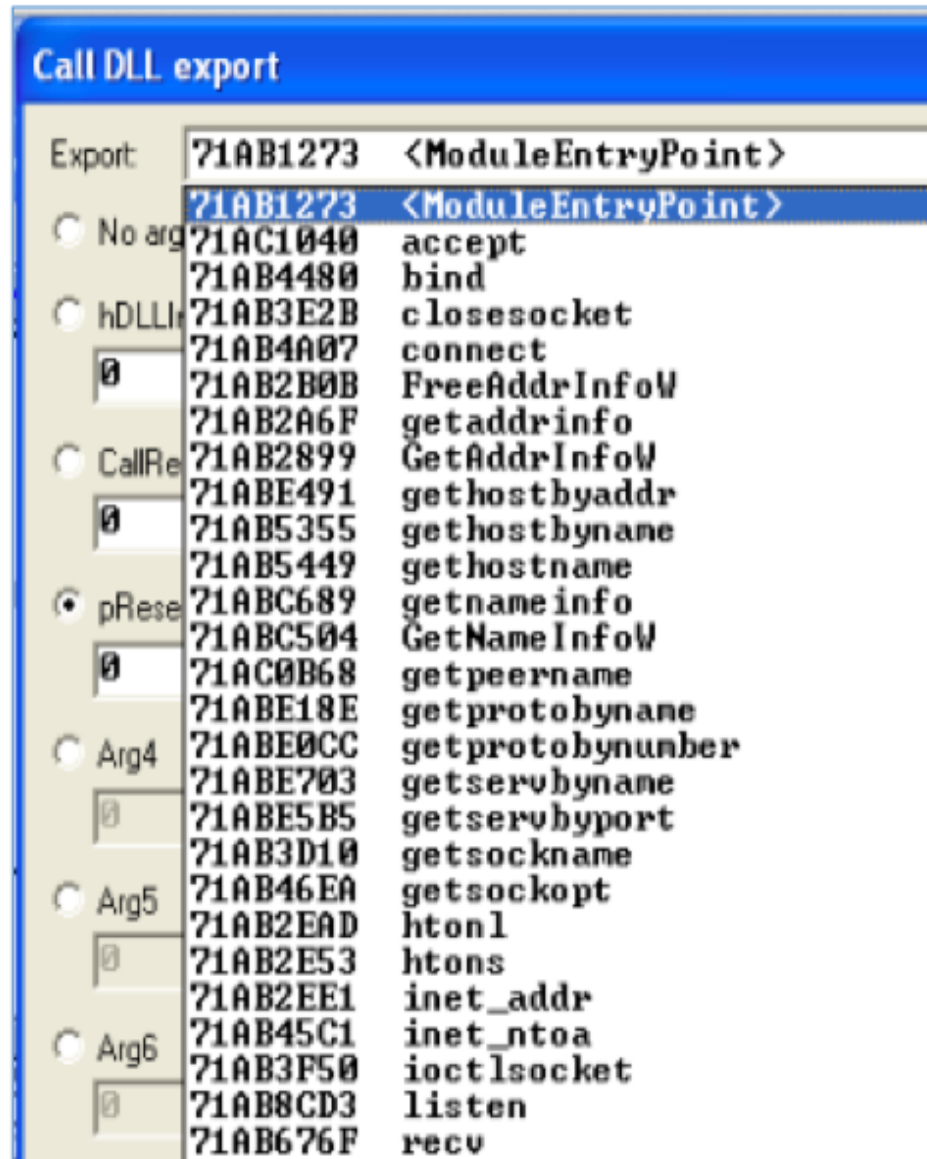


# OllyDbg

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

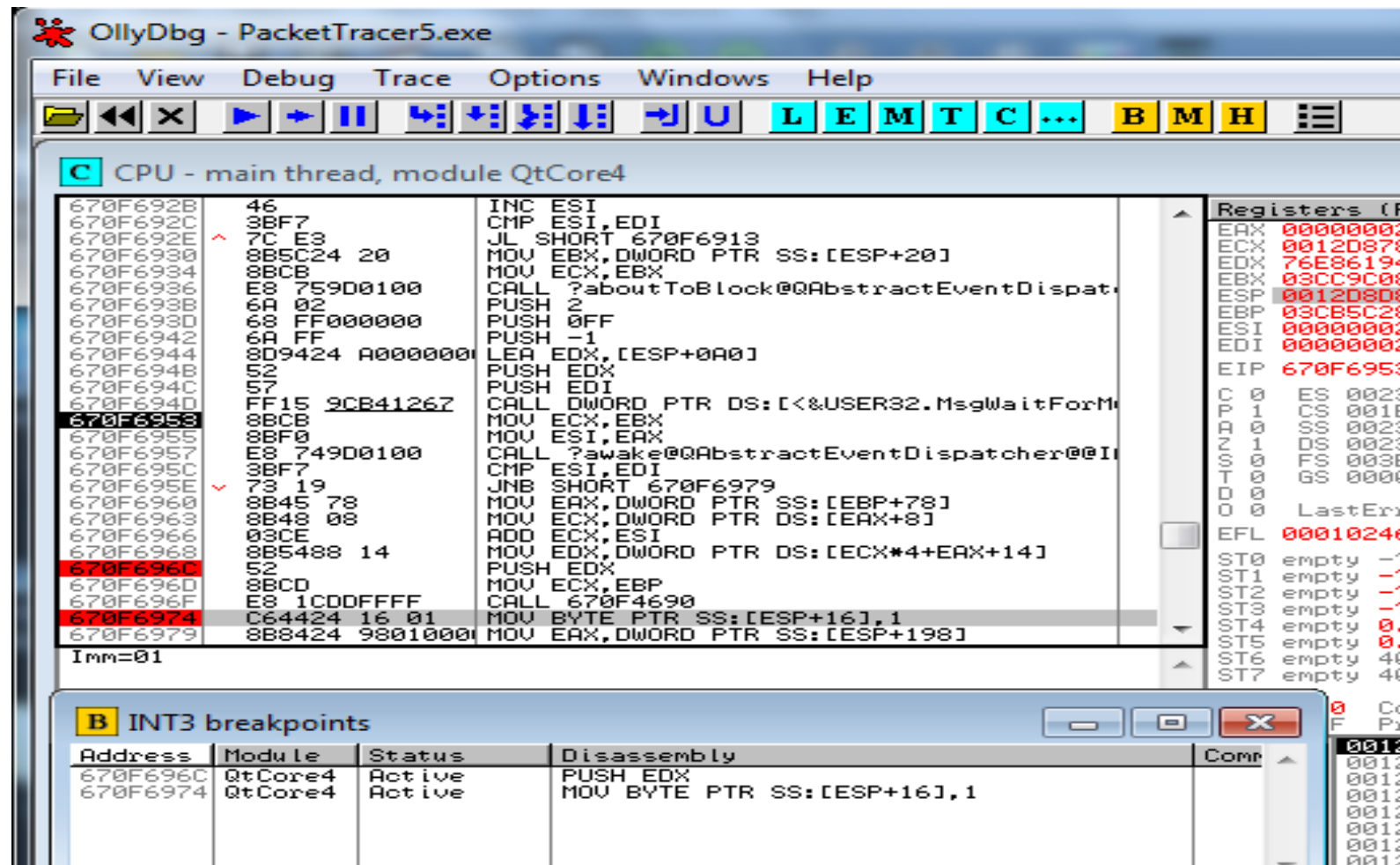
- ❑ Thao tác mở một binary: File > Open
- ❑ Thêm các đối số dòng lệnh nếu cần thiết
- ❑ Ollydbg sẽ dừng lại ở Entry Point, WinMain.. Nếu nó có thể xác định được
- ❑ Nếu không nó sẽ break tại điểm vào của chương trình được xác định trong PE Header
  - Cấu hình tại Option > Debugging Options

# 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

OllyDbg - Lab09-01.exe - [CPU - main thread, module Lab09-01]

File View Debug Trace Options Windows Help

Disassembler  
Highlight: next instruction to be executed

Registers (FPU)

Registers

Stack dump

Stack

Entry point of main module

Paused

The screenshot displays the OllyDbg interface for Lab09-01.exe. The main window is divided into several panes. The top pane shows the disassembly of the main thread, with the next instruction to be executed highlighted. The right pane shows the registers (FPU) and the stack. The bottom pane shows the memory dump. The status bar at the bottom indicates the entry point of the main module and the current state (Paused).

Address	Hex dump	ASCII
0040C000	00 00 00 00 00 00 00 00 00 00 00 00 EF 42 40 00	.....nB@.
0040C010	6C 5A 40 00 00 00 00 00 00 00 00 00 12 00 00 00	12@.....c@.
0040C020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0040C030	43 6F 6E 66 69 67 77 77 77 77 77 77 77 77 77 77	Configuration...
0040C040	53 4F 46 54 57 41 55 55 55 55 55 55 55 55 55 55	SOFTWARE\Microso
0040C050	66 74 20 5C 58 50 55 55 55 55 55 55 55 55 55 55	ft \XPS.\kernel3
0040C060	32 2E 64 6C 6C 6C 6C 6C 6C 6C 6C 6C 6C 6C 6C 6C	2.dll.....
0040C070	20 48 54 54 50 2F 30 30 30 30 30 30 30 30 30 30	HTTP/1.0.....
0040C080	47 45 54 20 00 00 00 00 00 00 00 00 00 00 00 00	GET .....
0040C090	60 27 60 27 60 00 00 00 00 00 00 00 00 00 00 00	.....NOTHING.
0040C0A0	72 62 00 00 60 00 00 00 00 00 00 00 00 00 00 00	rb.....CMD. DOWN
0040C0B0	4C 4F 41 44 00 00 00 00 55 50 4C 4F 41 44 00 00	LOAD....UPLOAD..
0040C0C0	20 00 00 00 53 4C 45 45 50 00 00 00 63 6D 64 2E	...SLEEP...cmd.
0040C0D0	65 78 65 00 20 3E 30 20 4E 55 4C 00 2F 63 20 64	exe. >> NUL./c d
0040C0E0	65 6C 20 00 75 70 73 00 68 74 74 70 3A 2F 2F 77	el .ups.http://w
0040C0F0	77 77 2E 70 72 61 63 74 69 63 61 6C 60 61 6C 77	ww.practicalmalw
0040C100	61 72 65 61 6E 61 6C 79 73 69 73 2E 63 6F 6D 00	areanalysis.com.
0040C110	38 30 00 00 36 30 00 00 20 4D 61 6E 61 67 65 72	80..60.. Manager
0040C120	20 53 65 72 76 69 63 65 00 00 00 00 2E 65 78 65	Service....exe
0040C130	00 00 00 00 2F 52 53 53 54 4F 4D 53 4F 4F 54 2F	%SYSTEMROOT%

# 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	
0012D000	00001000				Priv	RW	Guar	
0012E000	00002000			Stack of main thr	Priv	RW	RW	
00130000	00004000				Map	R	R	
00140000	00001000				Priv	RW	RW	
00150000	00067000				Map	R	R	
001C0000	00001000				Priv	RW	RW	
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
00420000	00005000				Map	R	R	
004E0000	00003000				Map	R	R	
004F0000	00101000			GDI handles	Map	R	R	
00600000	00088000				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
75EC0000	00001000	SHELL32		PE header	Img	R	RWE	Cop
75EC1000	003C8000	SHELL32			Img	R E	RWE	Cop
76289000	00007000	SHELL32			Img	RW	Cop	RWE
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
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*

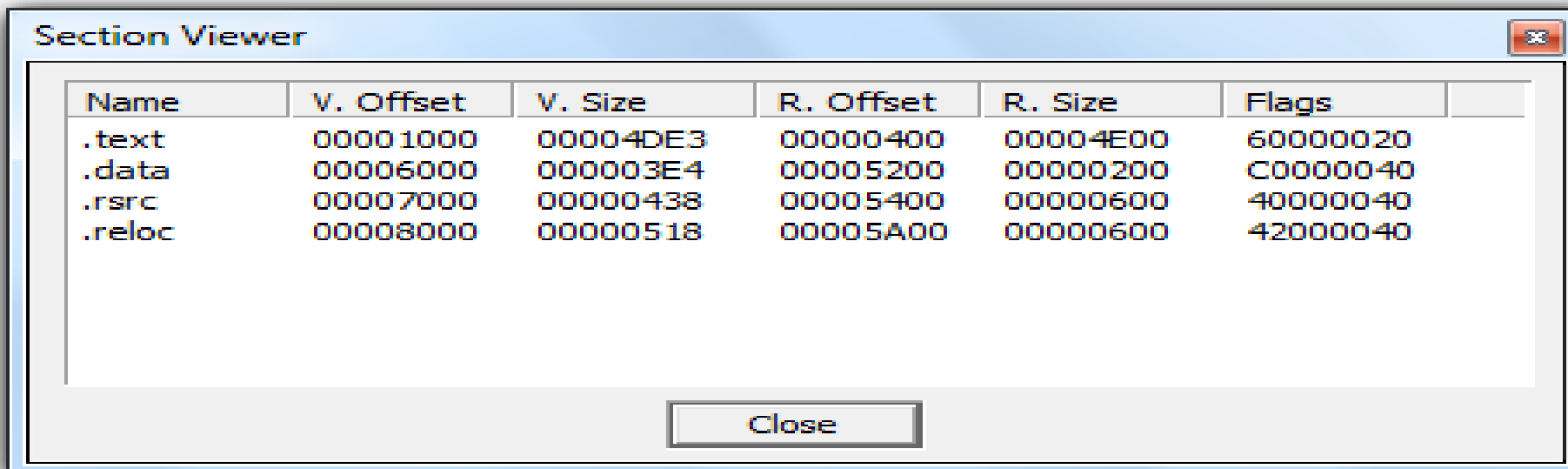
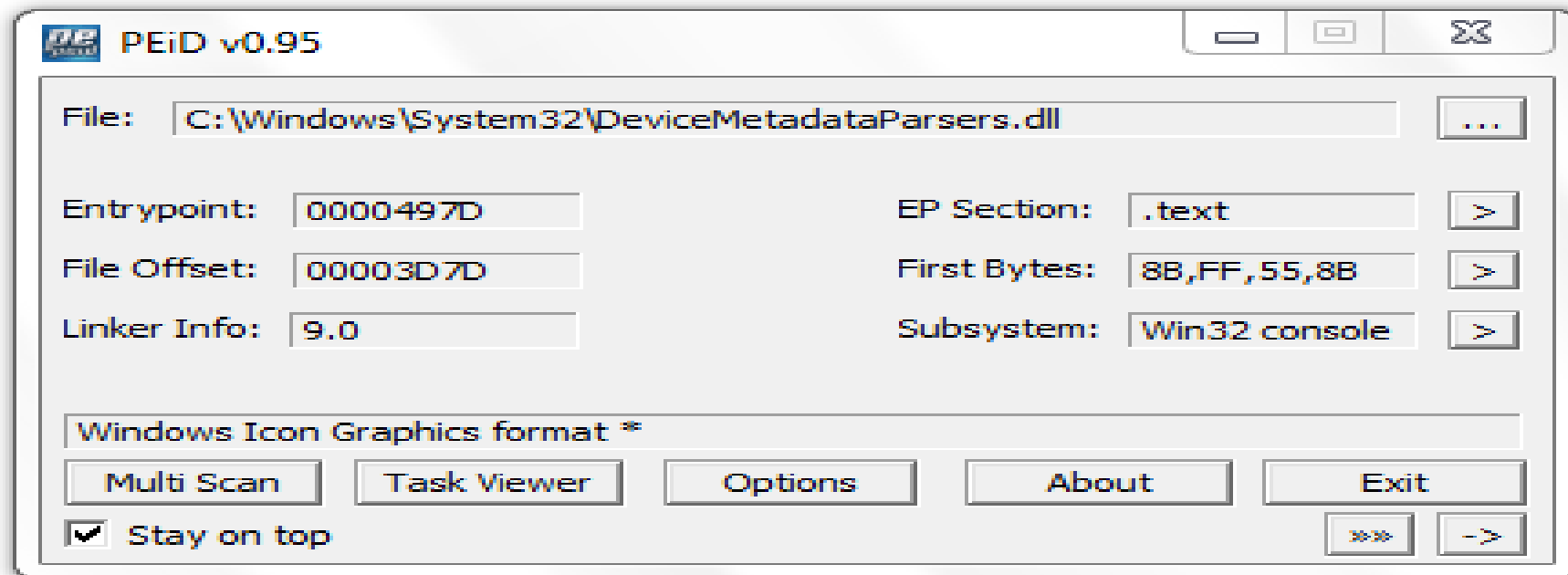
```
00401203      mov eax, [ebp+var_8]
00401206      cmp [ebp+var_4], 0
0040120a      jnz loc_0040120
0040120c      1mov eax, dword_40CF60
```

- ❑ 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



# DLL Rebasing

- ❑ Các DLL có thể xóa bỏ đoạn .reloc của chúng
  - Không thể relocated 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		PE header	Imag	R	RWE
00341000	00009000	DLL-B	.text	code	Imag	R	RWE
0034A000	00002000	DLL-B	.rdata	imports, exp	Imag	R	RWE
0034C000	00003000	DLL-B	.data	data	Imag	R	RWE
0034F000	00001000	DLL-B	.rsrc	resources	Imag	R	RWE
00350000	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*

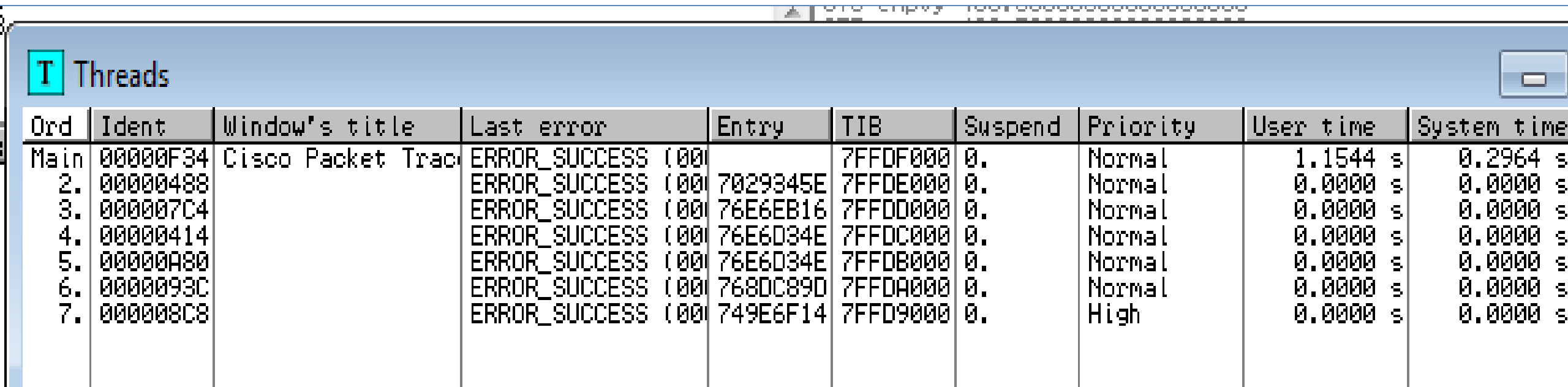
# DLL Rebasing

- ❑ 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..



Ord	Ident	Window's title	Last error	Entry	TIB	Suspend	Priority	User time	System time
Main	00000F34	Cisco Packet Trac	ERROR_SUCCESS (00000000)		7FFDF000	0.	Normal	1.1544 s	0.2964 s
2.	00000488		ERROR_SUCCESS (00000000)	7029345E	7FFDE000	0.	Normal	0.0000 s	0.0000 s
3.	000007C4		ERROR_SUCCESS (00000000)	76E6EB16	7FFDD000	0.	Normal	0.0000 s	0.0000 s
4.	00000414		ERROR_SUCCESS (00000000)	76E6D34E	7FFDC000	0.	Normal	0.0000 s	0.0000 s
5.	00000A80		ERROR_SUCCESS (00000000)	76E6D34E	7FFDB000	0.	Normal	0.0000 s	0.0000 s
6.	0000093C		ERROR_SUCCESS (00000000)	768DC89D	7FFDA000	0.	Normal	0.0000 s	0.0000 s
7.	000008C8		ERROR_SUCCESS (00000000)	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	Gua	RW
06D1F000	00001000			Stack of thread 2. (00000488)	Priv	RW	RW	Gua
06E10000	00002000				Priv	RW	Gua	RW
06E1F000	00001000			Stack of thread 3. (000007C4)	Priv	RW	RW	Gua
06F10000	00BB0000				Priv	RW	RW	
07A00000	006B5000				Priv	RW	RW	
08280000	00002000				Priv	RW	Gua	RW
0828F000	00001000			Stack of thread 4. (00000414)	Priv	RW	RW	Gua
08380000	00002000				Priv	RW	Gua	RW
0838F000	00001000			Stack of thread 5. (00000A80)	Priv	RW	RW	Gua
0848C000	00002000				Priv	RW	Gua	RW
0848E000	00002000			Stack of thread 6. (0000093C)	Priv	RW	RW	Gua
0858D000	00002000				Priv	RW	Gua	RW
0858F000	00001000			Stack of thread 7. (000008C8)	Priv	RW	RW	Gua
08630000	00019000				Priv	RW	RW	
08670000	0021F000				Map	RW	RW	
088B0000	01C57000				Priv	RW	RW	
00E10000	001F6000				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.

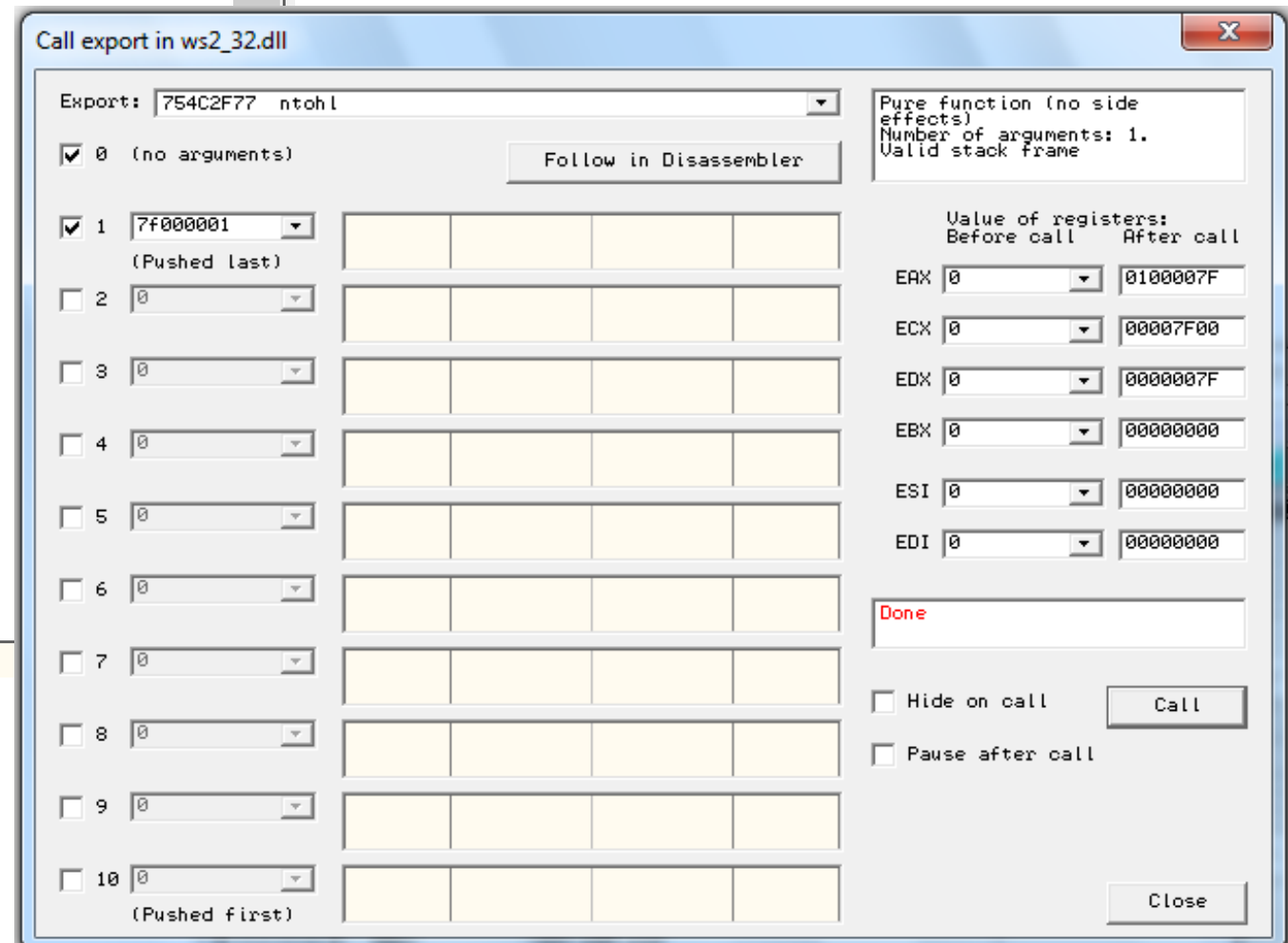
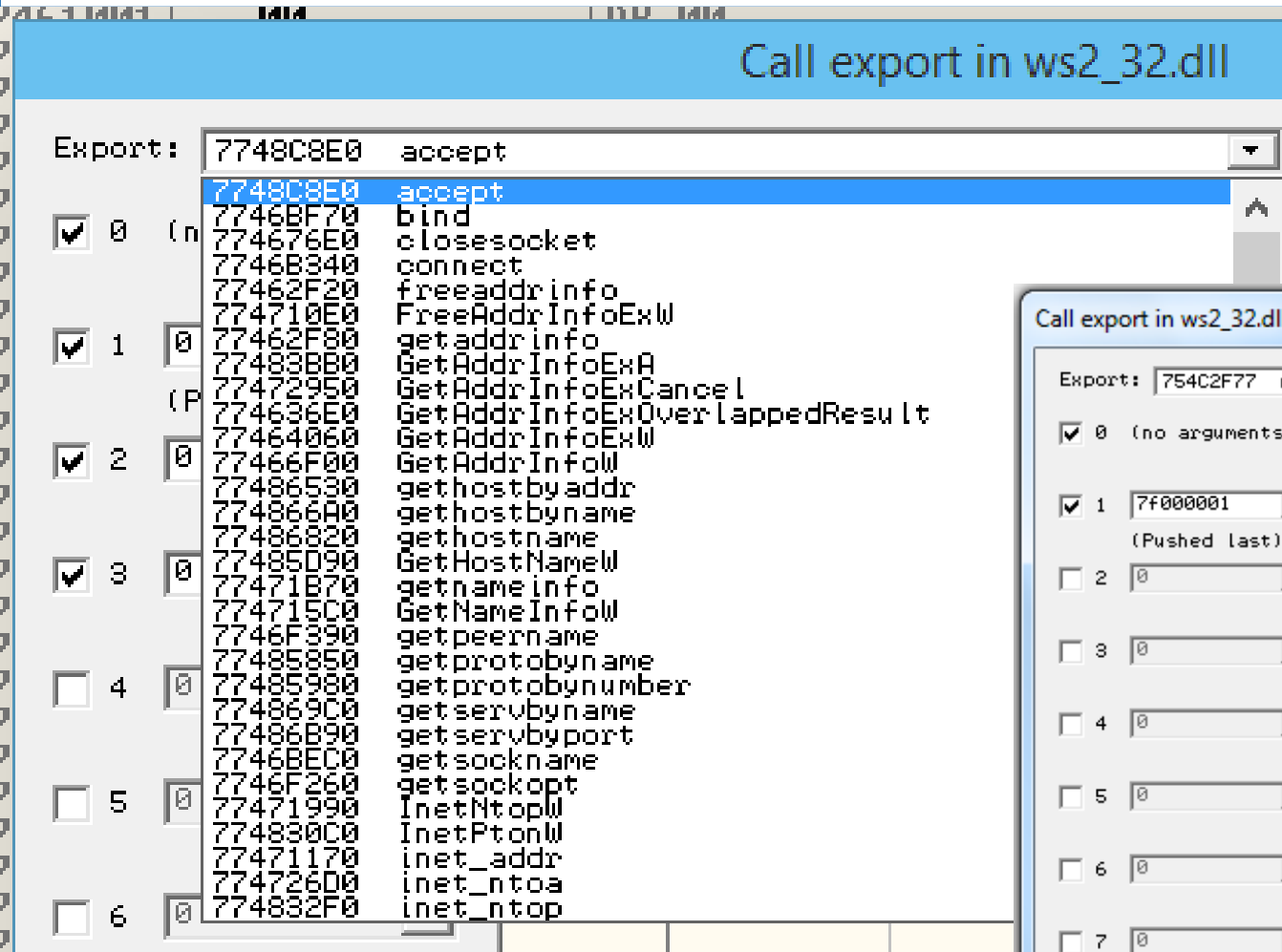
# Phân tích mã độc với OllyDbg

- ☐ 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 loadll.exe để tải các dll

# Tải các DLL

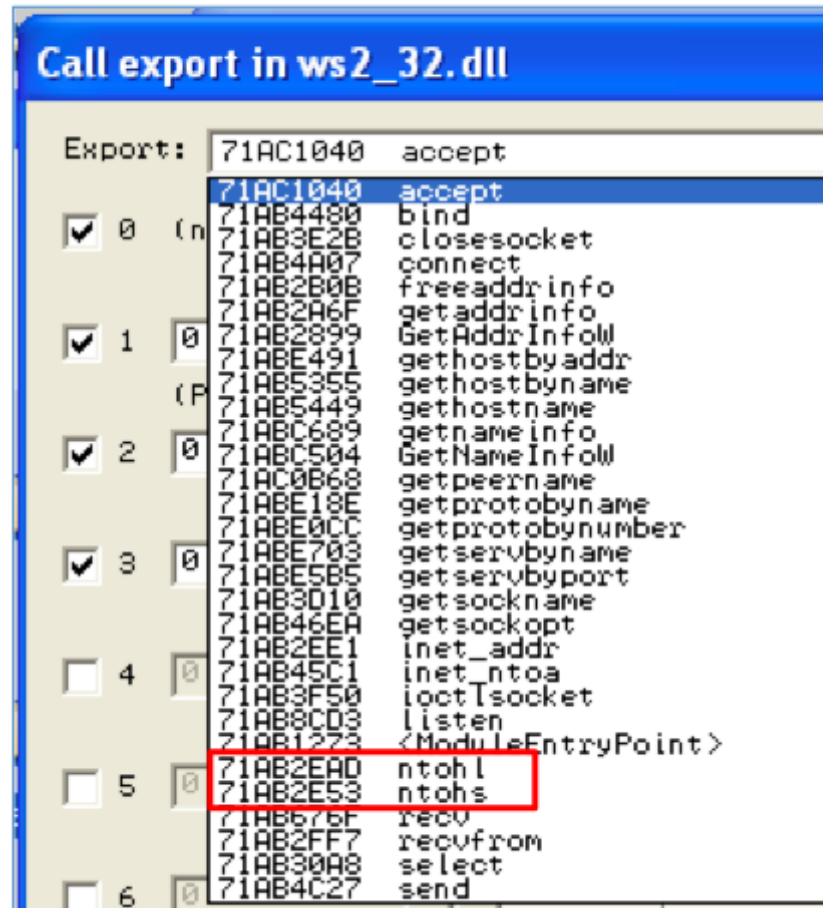




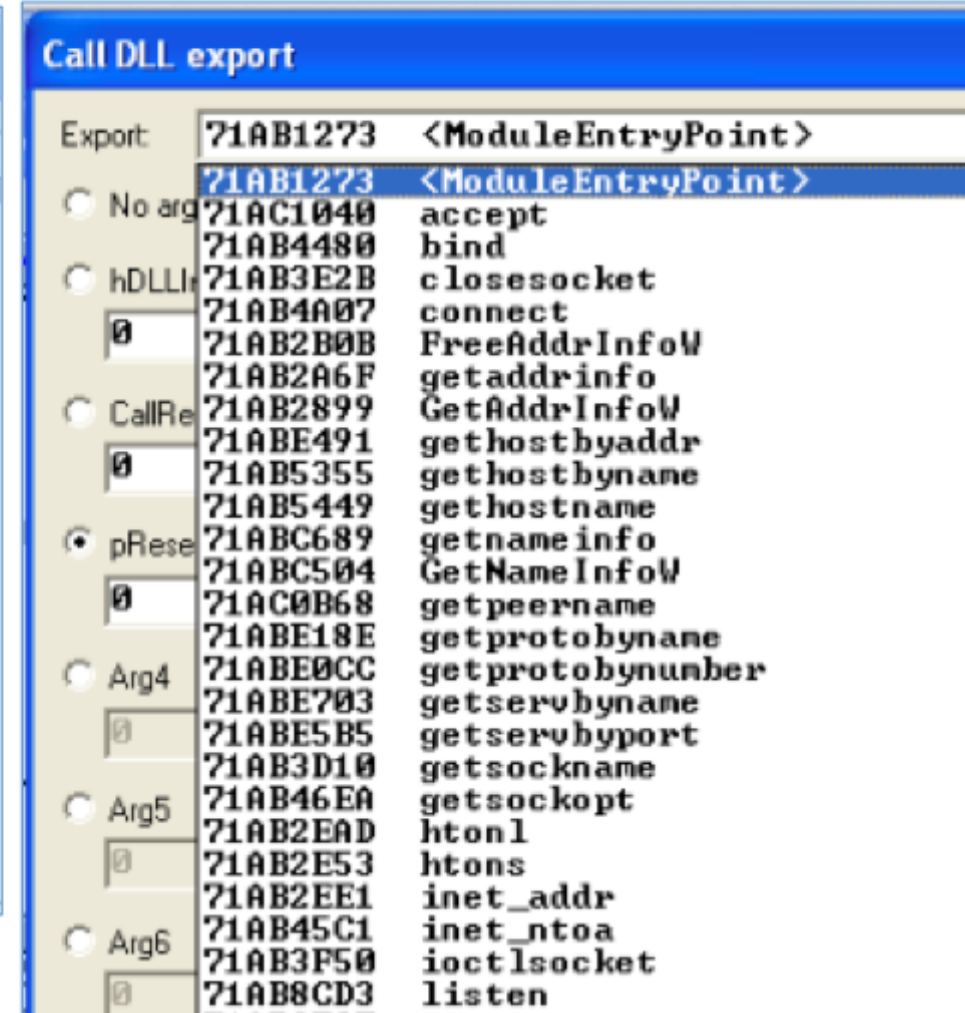
# Tải các DLL

## ❑ 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

OllyDbg - Lab09-01.exe

File View Debug Plugins Options Window Help

CPU - thread 00000F20, module CFGMGR32

Registers (FPU)

EAX	000000C0
ECX	001B1170
EDX	2E3FA1D5
EBX	001B49E8
ESP	0177F704

Call stack of thread 00000F20

Address	Stack	Procedure / arguments	Called from	Frame
0177F704	77AC43FC	Includes ntdll.KiFastSystemCallRet	ntdll.77AC43FA	0177F704
0177F708	75F50346	ntdll.ZwAlpcConnectPort	RPCRT4.75F50340	0177F708
0177F7D4	75F4F51E	RPCRT4.75F501D0	RPCRT4.75F4F519	0177F7D4
0177F804	75F4F3FE	RPCRT4.75F4F418	RPCRT4.75F4F3F9	0177F804
0177F830	75F3846D	RPCRT4.75F4F266	RPCRT4.75F38468	0177F830
0177F888	75F4BC18	Includes RPCRT4.75F3846D	RPCRT4.75F4BC18	0177F888
0177F8AC	75F49D6D	RPCRT4.I_RpcGetBufferWithObject	RPCRT4.75F49D68	0177F8AC
0177F8BC	75F4A041	RPCRT4.I_RpcGetBuffer	RPCRT4.75F4A03C	0177F8BC
0177F8CC	75FA5718	Includes RPCRT4.75F4A041	RPCRT4.75FA5712	0177F8CC
0177FCF0	75C960B6	? <JMP.&RPCRT4.NdrClientCall2>	CFGMGR32.75C960B1	0177FCF0
0177FD08	75C96055	CFGMGR32.75C9609D	CFGMGR32.75C96050	0177FD08
0177FD5C	762E0356	? CFGMGR32.CM_Get_Device_Interface_	SHELL32.762E0350	0177FD5C
0177FD98	762E02ED	SHELL32.762E0326	SHELL32.762E02E8	0177FD98
0177FDA8	7709B6CF	Includes SHELL32.762E02ED	SHLWAPI.7709B6CD	0177FDA8
0177FDB8	77AAB338	Includes SHLWAPI.7709B6CF	ntdll.77AAB335	0177FDB8

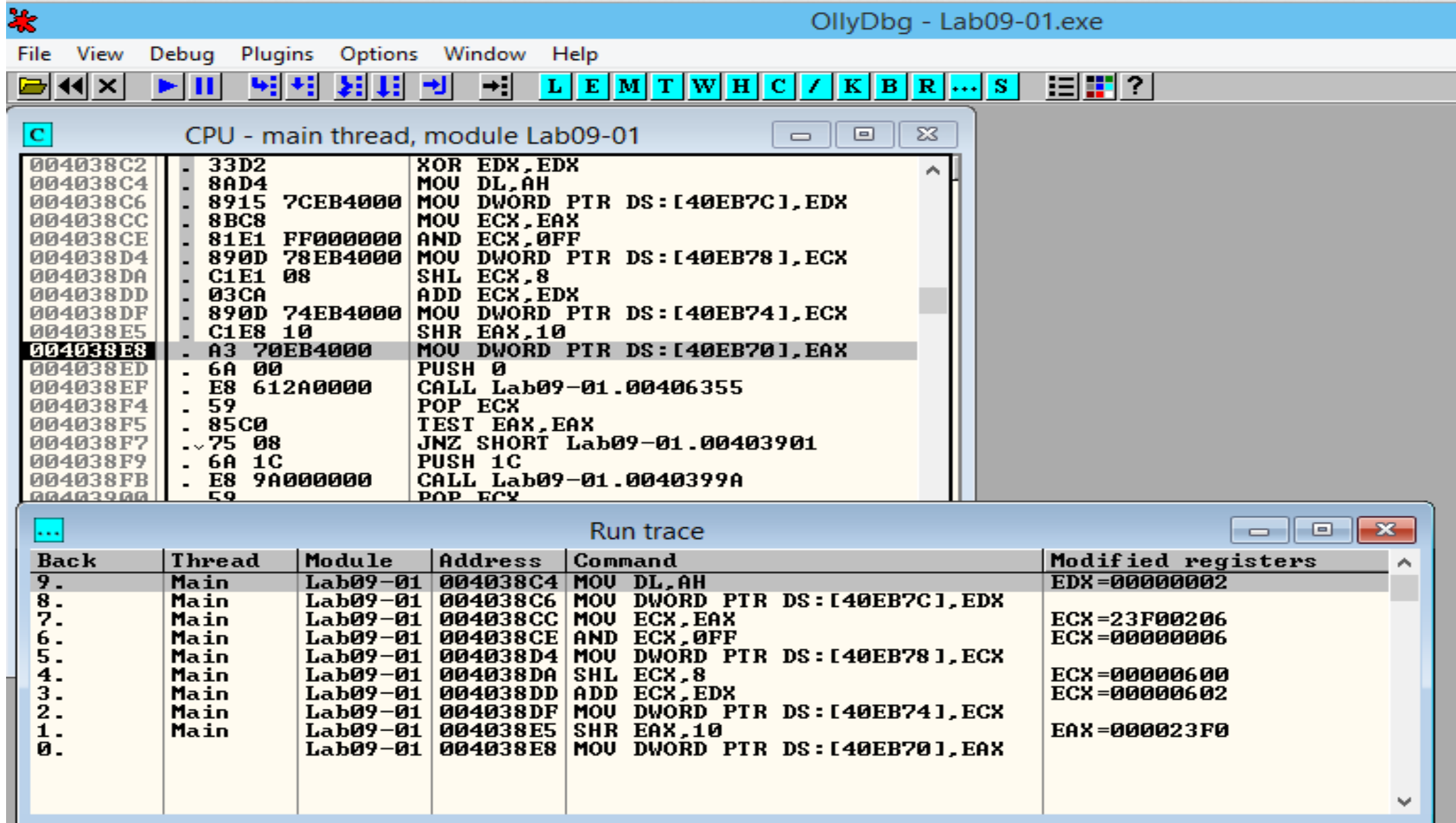
Process terminated, exit code 0

Terminated

# 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 còn
- ❑ 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





# 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

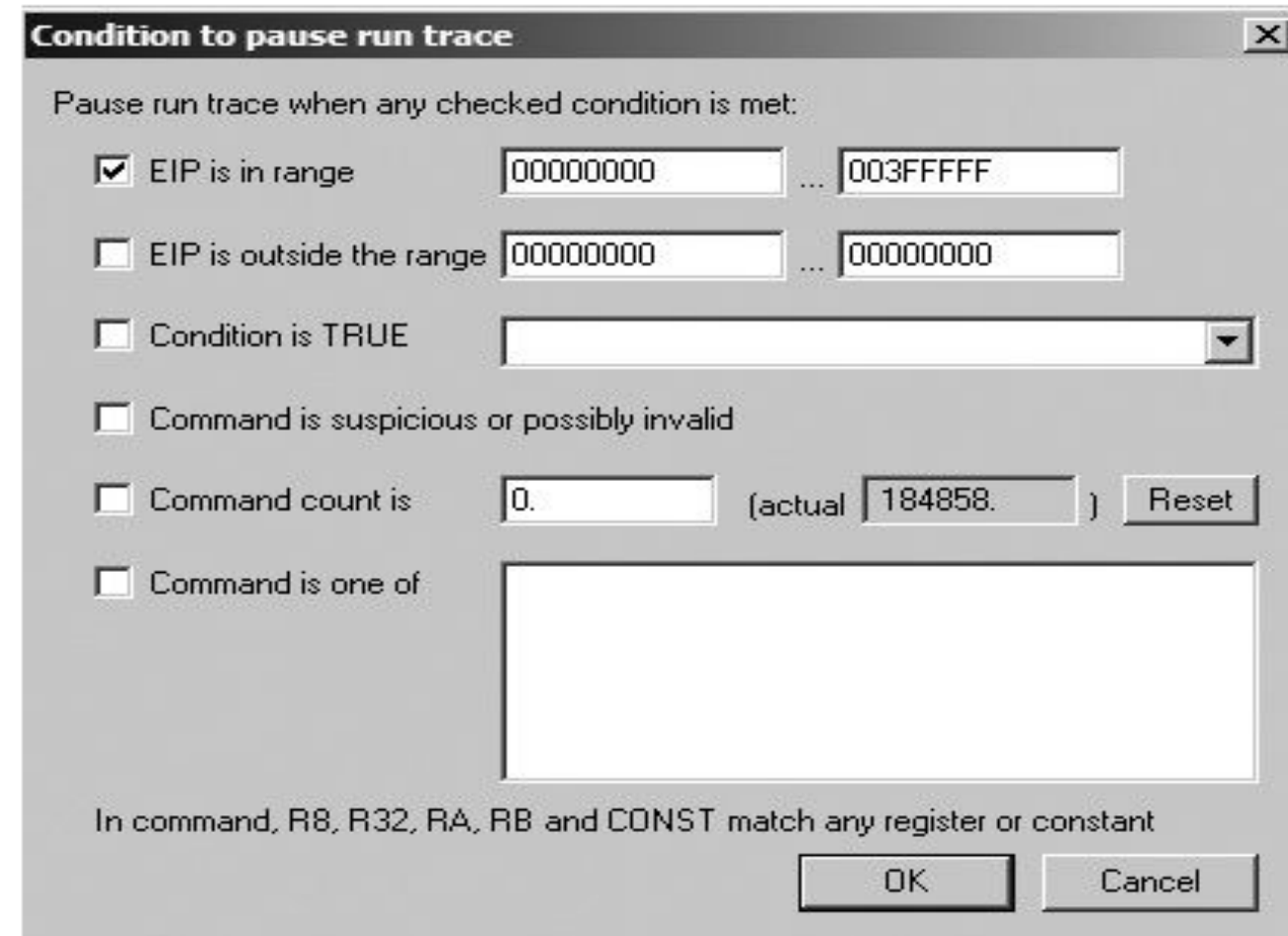


Figure 10-11. Conditional tracing

# 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

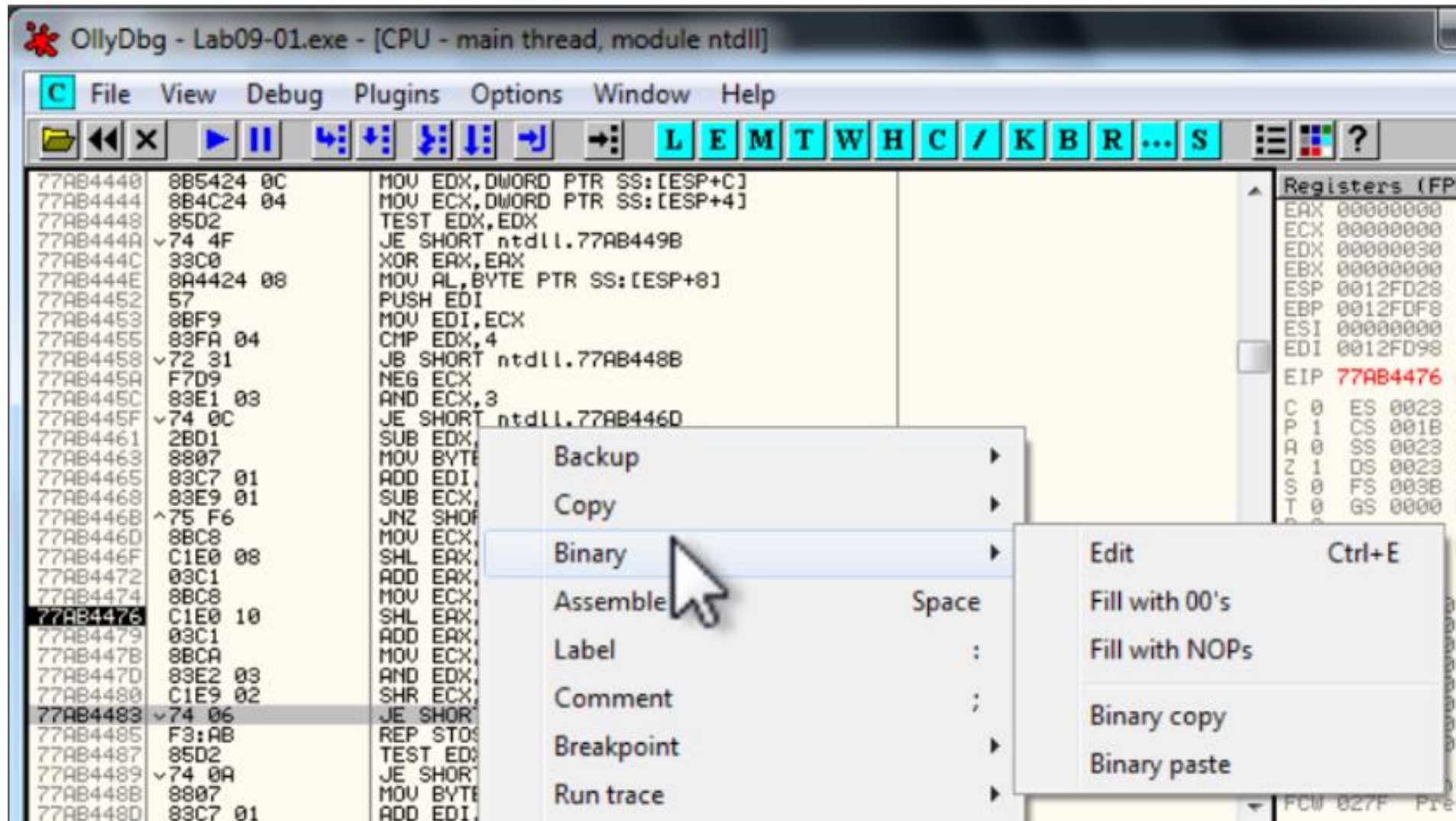
- ❑ Ollydbg sẽ dừng chương trình
- ❑ Có các tùy chọn để bỏ qua ngoại lệ:
  - Shift+F7 Step into exception – Nhảy vào bên trong ngoại lệ
  - Shift+F8: Step over exception – Nhảy qua ngoại lệ
  - Shift+F9: Run exception handler – chạy trình xử lý ngoại lệ
- ❑ Thường thì chỉ cần bỏ qua tất cả các ngoại 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. Gỡ rối**
- 2. Gỡ rối mức mã nguồn và gỡ rối mức mã Assembly**
- 3. Gỡ rối chế độ nhân và chế độ người dùng**
- 4. Sử dụng trình gỡ rối trong quá trình phân tích mã độc**
- 5. Ngoại lệ**
- 6. Chỉnh sửa ngoại lệ với trình gỡ rối**
- 7. Phân tích mã độc với OllyDbg**