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

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

7

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

8

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

9

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

10

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

11

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

12

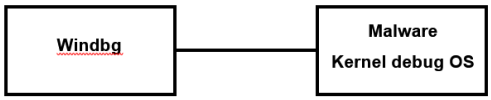
Gỡ rối chế độ người dùng

- 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

13

Gỡ rối chế độ nhân

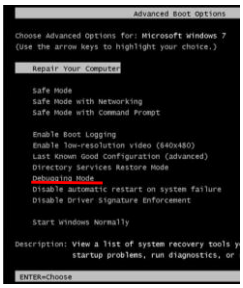
- Yêu cầu hai máy tính có kết nối mạng với nhau, một máy chạy đoạn mã được gỡ rối, máy khác chạy trình gỡ rối
- HĐH cần được cấu hình cho phép gỡ rối ở chế độ nhân



14

Gỡ rối chế độ nhân

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



15

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

16

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

17

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

18

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.

19

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:   F8FDF3D0
```

20

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

21

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
```

22

Dừng thực thi với Breakpoints

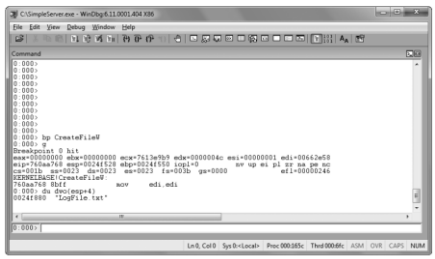
- Đoạn chương trình tín tên file, sau đó tạo file
- Đặt một breakpoint tại CreateFileW và xem trong stack tên file

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  call    CreateFileW ; CreateFileW(x,x,x,x,x,x,x)
```

23

Dừng thực thi với Breakpoints



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.

24

Các loại breakpoints

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

25

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

26

Software Breakpoints

- Hữu dụng cho string decoders

A string decoding breakpoint

```
push offset "4NNpTNHLKIXoPm7iBhUAjvRKNaUVBlr"
call String_Decoder
...
push offset "ugKLdNlLT6emldCeZi72mUjieuBqdfZ"
call String_Decoder
...
```

27

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

28

Conditional Breakpoints

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

29

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
```

30

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

31

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

32

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

33

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

34

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: - 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.

35

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

36

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

37

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

38

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

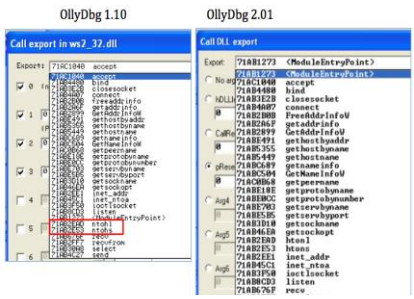
39

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

40

OllyDbg



41

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

42

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

43

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

44

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

45

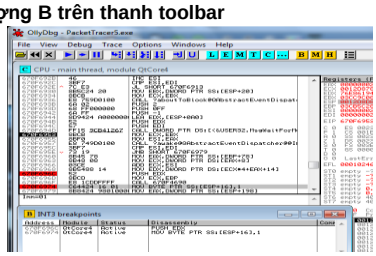
Tải một tiến trình đang chạy

- ❑ 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.

46

Xem các breakpoints đang có

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



47

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	

48

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.

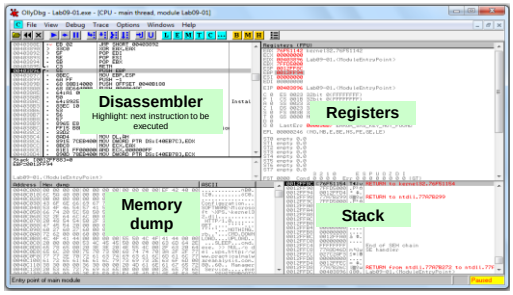
49

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

50

Giao diện OllyDbg



51

Chỉnh sửa dữ liệu

- Cửa sổ Disassembler
  - Nhấn phích 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"

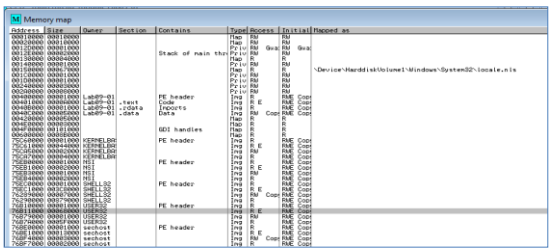
52

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

53

Bản đồ bộ nhớ



54

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)

55

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

56

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        mov 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.

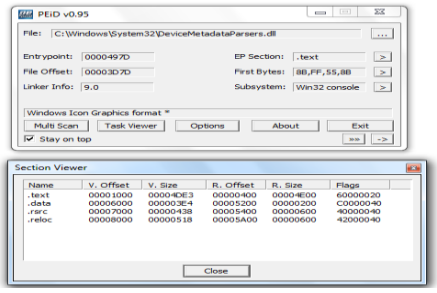
57

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.

58

Fix-up Locations



59

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 để mặc định địa chỉ cơ sở khi biên dịch các DLL

60

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
10009000	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

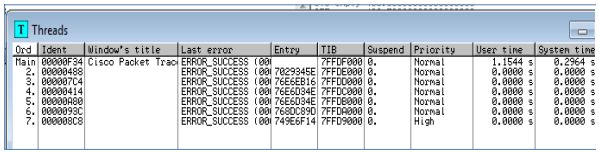
DLL-B is relocated into a different memory address from its requested location

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

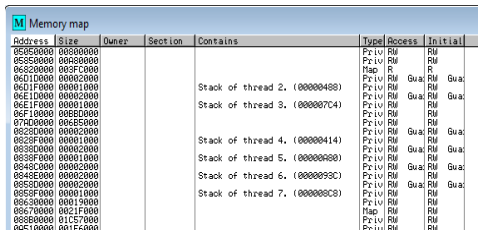
Xem các Thread và Stack

- Thao tác: View, Threads
- Right-click a thread to "Open in CPU", kill it, etc..



Xem các Thread và Stack

- Mỗi Thread đều có ngăn xếp riêng của nó



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

67

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.

68

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.

69

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

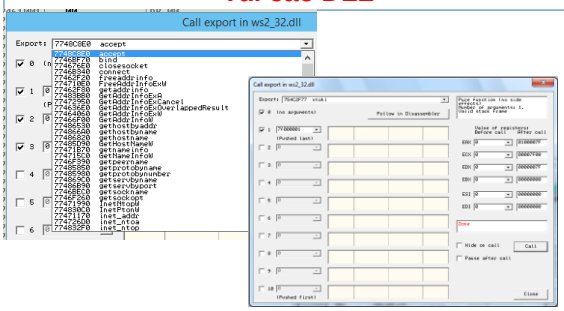
70

Tải các DLL

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

71

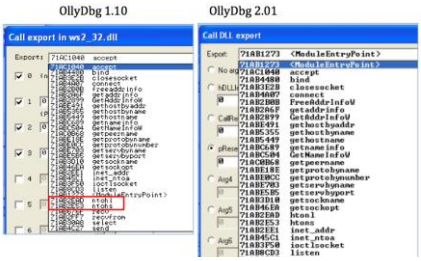
Tải các DLL



72

Tải các DLL

- Không sử dụng OllyDbg 2



73

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

74

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

75

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.

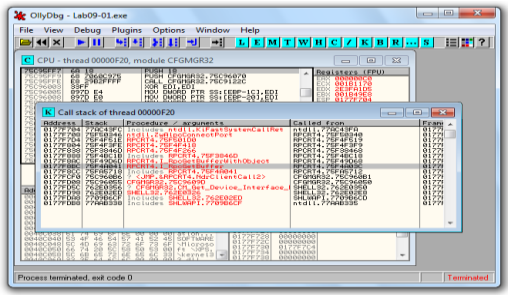
76

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

77

Call Stack Trace



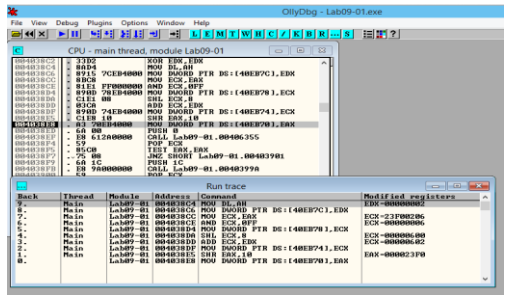
78

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

79

Run Trace



80

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

81

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

82

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

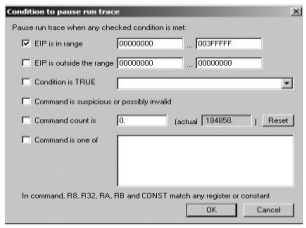


Figure 10-11. Conditional tracing

83

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

84

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

85

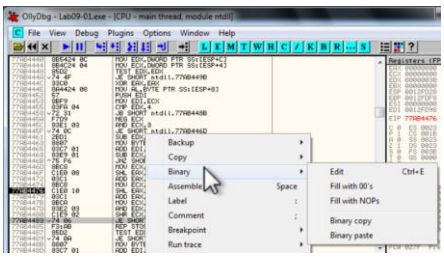
Phân tích mã độc với OllyDbg

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

86

Patching

- Binary edit



87

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

88