汇编语言与逆向技术

第10章 动态逆向分析技术

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2024-2025学年

本章知识点

- 动态逆向分析
- x64dbg加载程序
- x64dbg的界面
- Memory Map
  - 难点：重定位
- 执行指令
  - 难点：执行到返回、执行到用户代码
- 断点 Breakpoint
  - 难点：软件断点、硬件断点、内存断点、条件断点、消息断点

动态逆向分析

动态逆向分析

- 动态分析技术中，最重要的是调试器
  - 用户模式的调试器
  - 内核模式的调试器

用户模式调试器

- 调试用户模式的应用程序
- CPU Ring3级
- x64dbg

内核模式调试器

- 调试操作系统内核程序
- CPU Ring0级
- WinDbg

x64dbg

- x64dbg是一款具有可视化界面的用户模式调试器
- x64dbg结合了静态反汇编和动态调试
- x64dbg界面简洁优雅，显示的信息丰富而不杂乱
- x64dbg是最受欢迎的用户模式调适器之一
- x64dbg的外观与使用方法与经典的OllyDbg调试器基本一致
  - OllyDbg已经停止开发
  - x64dbg基本功能已经十分稳定，但仍时常有新功能加入

x64dbg

- 可识别C和Windows编程中常用的函数
- 能将参数注释出来。
- 自动分析函数过程、循环语句、代码中的字符串
- 提供开放的调试接口
- 大量的插件可以选择

单选题 10分

设置

x64dbg可以调试以下哪种级别的程序？

A Ring0

B Ring1

C Ring2

D Ring3

提交

南開大學 Nankai University

x64dbg的下载和安装

下载与安装x64dbg

- 下载地址https://snapshots.x64dbg.com/
  - 通常下载最新版即可
  - 亦可根据课程安排统一下载指定版本
- 下载后得到一个压缩包，将其解压到任意位置
- 运行snapshot_xxxxxxxxx\release文件夹内的x96dbg.exe
  - 在弹出的对话框中选择“是”
  - 该操作将x64dbg调试器注册为shell扩展
  - 这样在文件浏览器中右键单击一个.exe文件即可开始调试，无需预先打开x64dbg

x32dbg与x64dbg

- 目前最新的Windows操作系统一般位64位系统，既可以运行64位程序，也可以运行32位程序
- x64dbg其实包含两个可执行文件
  - x32dbg.exe用于调试32位程序，它位于release\x32文件夹
  - x64dbg.exe用于调试64位程序，它位于release\x64文件夹
- 本课程主要学习32位程序，所以用到的是x32dbg.exe
  - 但习惯上一般还是称其为x64dbg
- 在文件管理器内右键单击一个.exe文件，然后选择“Debug with x64dbg”，x64dbg会自动根据该文件是32位还是64位选择恰当的x32dbg/x64dbg

初次配置x64dbg

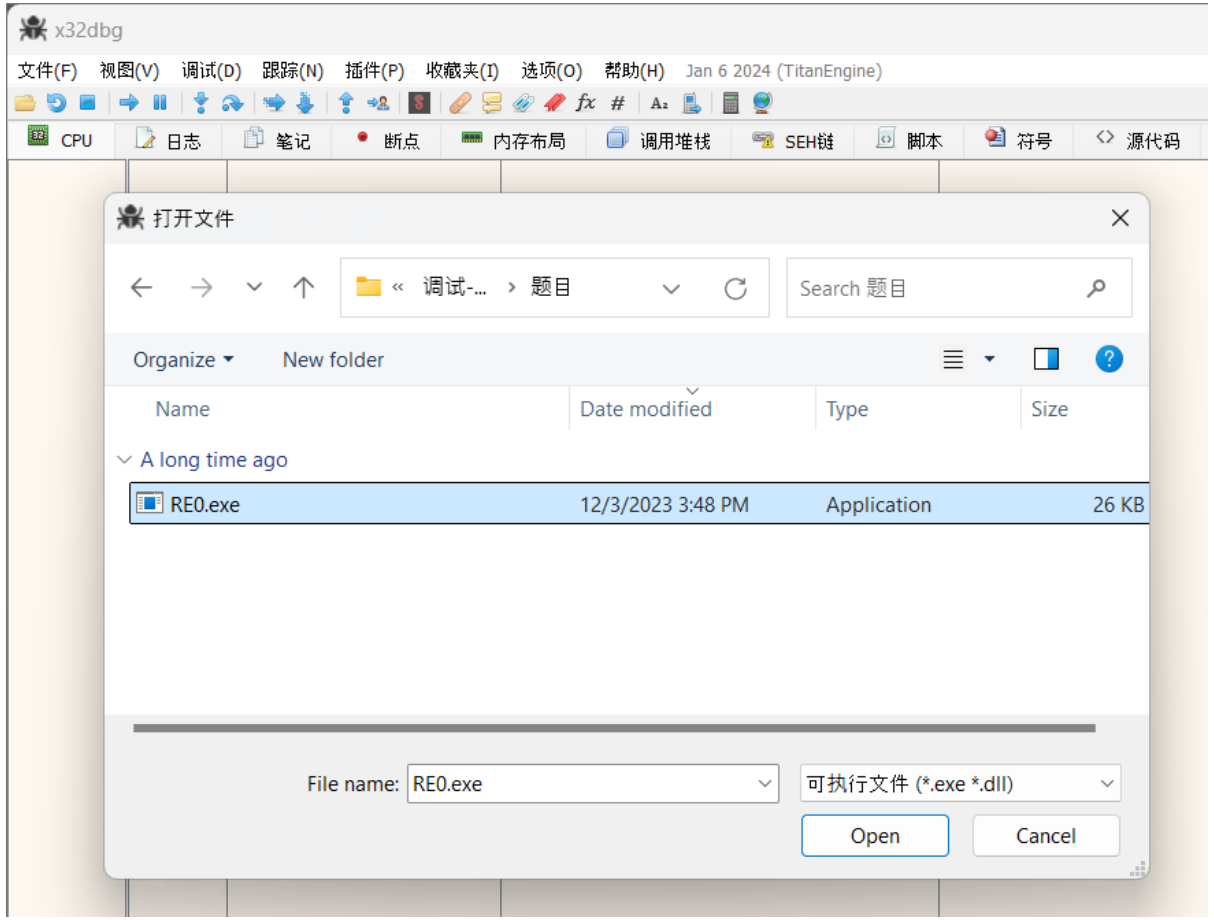
- 打开x64dbg，在Options-Languages菜单中将语言设置为中文，并重新打开x64dbg
- 菜单-选项-选项-引擎，然后在弹出的对话框中勾选“禁用ASLR”
  - 禁用ASLR可以保证每次EXE被加载的地址一致

x64dbg加载程序

x64dbg加载程序

- 直接加载
  - EXE可执行文件
  - DLL动态链接库程序
- 如果程序已经在系统上运行，可以通过附加进程的方式调试（Attach）

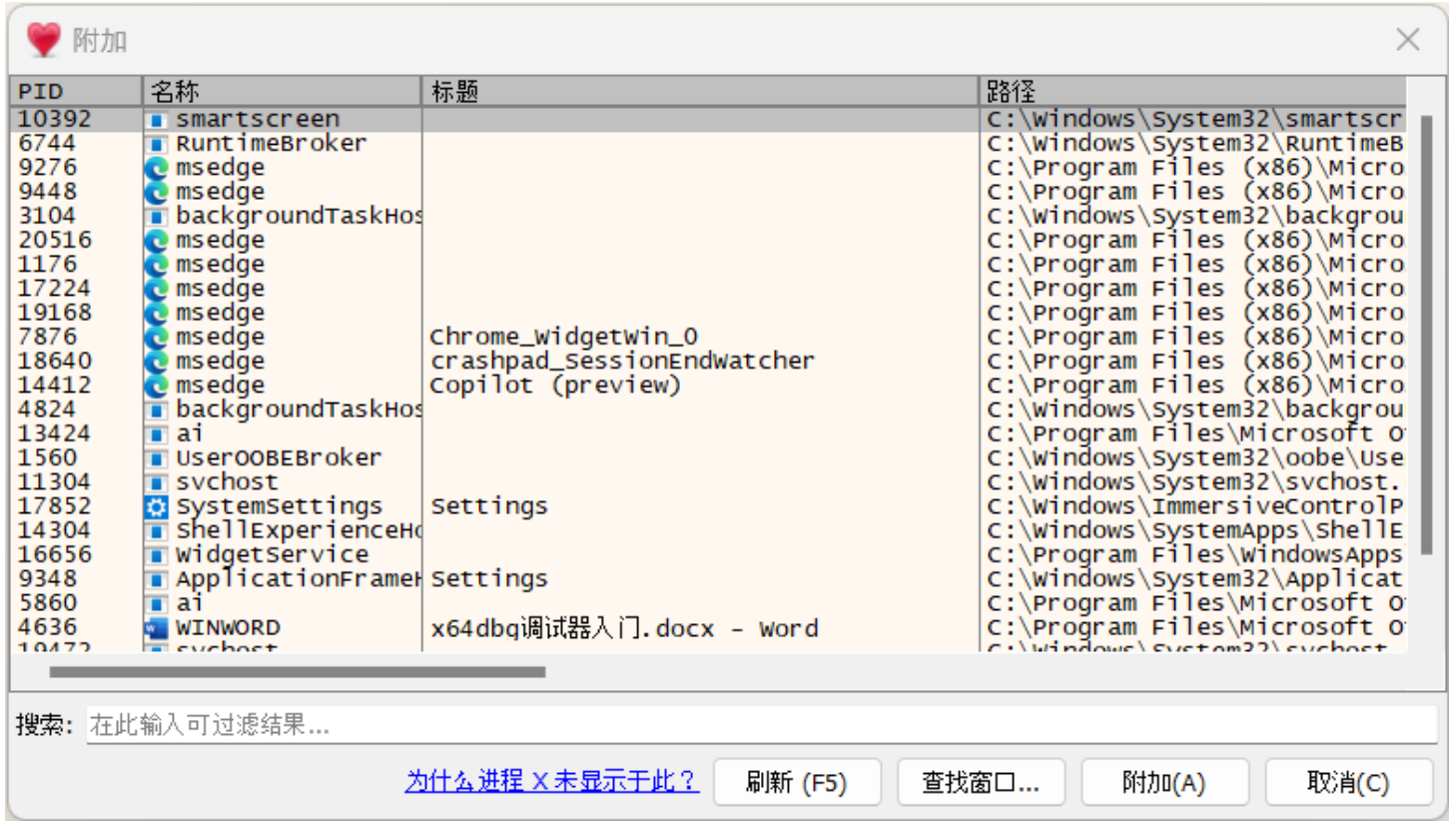
直接加载程序


x32dbg
文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Jan 6 2024 (TitanEngine)
CPU 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号 源代码
打开文件
« 调试-... › 题目
Search 题目
Organize
New folder
Name
Date modified
Type
Size
A long time ago
RE0.exe
12/3/2023 3:48 PM
Application
26 KB
File name: RE0.exe
可执行文件 (*.exe *.dll)
Open
Cancel

打开一个可执行文件

- x64dbg会在软件PE头部提供的入口点（Entry Point）处中断。
- 如果想要调试main函数的代码，需要在main函数所在的地址添加一个断点并恢复执行

附加进程


附加
PID 名称 标题 路径
10392 smartscreen C:\Windows\System32\smartscr
6744 RuntimeBroker C:\Windows\System32\RuntimeB
9276 msedge C:\Program Files (x86)\Micro
9448 msedge C:\Program Files (x86)\Micro
3104 backgroundTaskHos C:\Windows\System32\backgrou
20516 msedge C:\Program Files (x86)\Micro
1176 msedge C:\Program Files (x86)\Micro
17224 msedge C:\Program Files (x86)\Micro
19168 msedge C:\Program Files (x86)\Micro
7876 msedge Chrome_WidgetWin_0 C:\Program Files (x86)\Micro
18640 msedge crashpad_SessionEndWatcher C:\Program Files (x86)\Micro
14412 msedge Copilot (preview) C:\Program Files (x86)\Micro
4824 backgroundTaskHos C:\Windows\System32\backgrou
13424 ai C:\Program Files\Microsoft O
1560 UserOOBEBroker C:\Windows\System32\oobe\Use
11304 svchost C:\Windows\System32\svchost.
17852 SystemSettings Settings C:\Windows\ImmersiveControlP
14304 ShellExperienceHo C:\Windows\SystemApps\ShellE
16656 WidgetService C:\Program Files\WindowsApps
9348 ApplicationFrameH Settings C:\Windows\System32\Applicat
5860 ai C:\Program Files\Microsoft O
4636 WINWORD x64dbg调试器入门.docx - Word C:\Program Files\Microsoft O
搜索: 在此输入可过滤结果...
为什么进程 X 未显示于此？
刷新 (F5)
查找窗口...
附加(A)
取消(C)

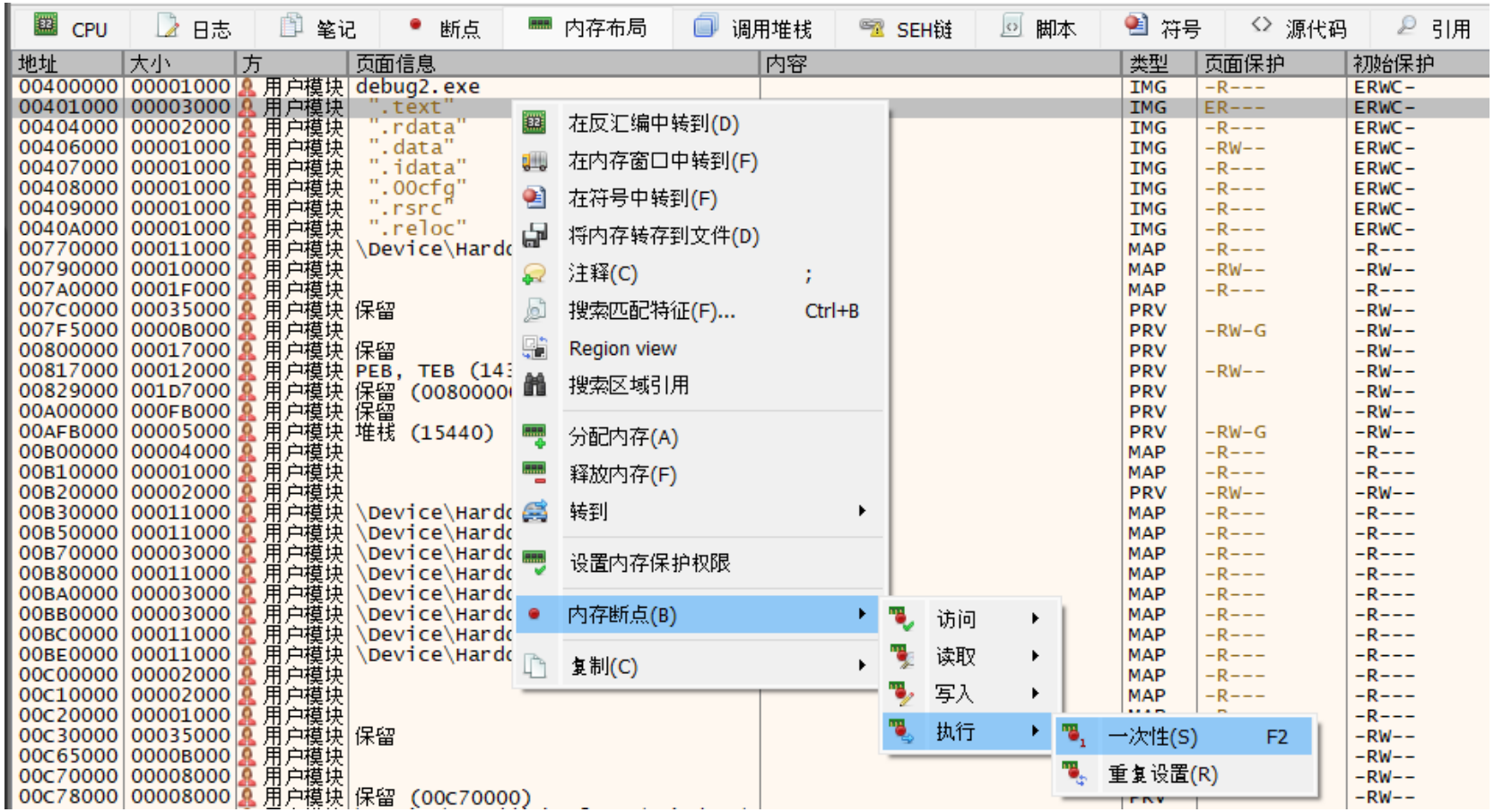
附加进程

- x64dbg会立即暂停程序以及它所有的线程；
- 被附加的程序会暂停在Ntdll.dll的DbgBreakPoint处；

附加进程

- 不希望调试Windows库，回到主代码最简单的方法就是在整个代码段中设置一个内存访问断点。
- 打开内存窗口，查看程序的内存空间，设置断点。

内存访问断点


CPU
日志
笔记
断点
内存布局
调用堆栈
SEH链
脚本
符号
源代码
引用
地址 大小 方 页面信息 内容 类型 页面保护 初始保护
00400000 00001000 用户模块 debug2.exe IMG -R--- ERWC-
00401000 00003000 用户模块 ".text" IMG ER--- ERWC-
00404000 00002000 用户模块 ".rdata" IMG -R--- ERWC-
00406000 00001000 用户模块 ".data" IMG -RW-- ERWC-
00407000 00001000 用户模块 ".idata" IMG -R--- ERWC-
00408000 00001000 用户模块 ".00cfg" IMG -R--- ERWC-
00409000 00001000 用户模块 ".rsrc" IMG -R--- ERWC-
0040A000 00001000 用户模块 ".reloc" IMG -R--- ERWC-
00770000 00011000 用户模块 \Device\Hard MAP -R--- -R---
00790000 00010000 用户模块 MAP -RW-- -RW--
007A0000 0001F000 用户模块 MAP -R--- -R---
007C0000 00035000 用户模块 保留 PRV -RW--
007F5000 0000B000 用户模块 PRV -RW-G -RW--
00800000 00017000 用户模块 保留 PRV -RW--
00817000 00012000 用户模块 PEB, TEB (14 PRV -RW-- -RW--
00829000 001D7000 用户模块 保留 (0080000 PRV -RW--
00A00000 000FB000 用户模块 保留 PRV -RW--
00AFB000 00005000 用户模块 堆栈 (15440) PRV -RW-G -RW--
00B00000 00004000 用户模块 MAP -R--- -R---
00B10000 00001000 用户模块 MAP -R--- -R---
00B20000 00002000 用户模块 PRV -RW-- -RW--
00B30000 00011000 用户模块 \Device\Hard MAP -R--- -R---
00B50000 00011000 用户模块 \Device\Hard MAP -R--- -R---
00B70000 00003000 用户模块 \Device\Hard MAP -R--- -R---
00B80000 00011000 用户模块 \Device\Hard MAP -R--- -R---
00BA0000 00003000 用户模块 \Device\Hard MAP -R--- -R---
00BB0000 00003000 用户模块 \Device\Hard MAP -R--- -R---
00BC0000 00011000 用户模块 \Device\Hard MAP -R--- -R---
00BE0000 00011000 用户模块 \Device\Hard MAP -R--- -R---
00C00000 00002000 用户模块 MAP -R--- -R---
00C10000 00002000 用户模块 MAP -R--- -R---
00C20000 00001000 用户模块 -R---
00C30000 00035000 用户模块 保留 -RW--
00C65000 0000B000 用户模块 -RW--
00C70000 00008000 用户模块 -RW--
00C78000 00008000 用户模块 保留 (00C70000) -RW--
在反汇编中转到(D)
在内存窗口中转到(F)
在符号中转到(F)
将内存转存到文件(D)
注释(C) ;
搜索匹配特征(F)... Ctrl+B
Region view
搜索区域引用
分配内存(A)
释放内存(F)
转到
设置内存保护权限
内存断点(B)
复制(C)
访问
读取
写入
执行
一次性(S) F2
重复设置(R)

隐藏进程

- 系统的隐藏进程不能使用上述方法进行附加；
  - PC Hunter、GMER等工具获得隐藏进程的pid；
  - x64dbg的-p参数，通过隐藏进程的pid进行附加。

重新装载

- Ctrl+F2可以重新装载当前调试的进程
- F2断点，记录之前的调试位置

多选题 10分

设置

x64dbg可以调试以下哪种类型的程序？

A exe程序

B dll程序

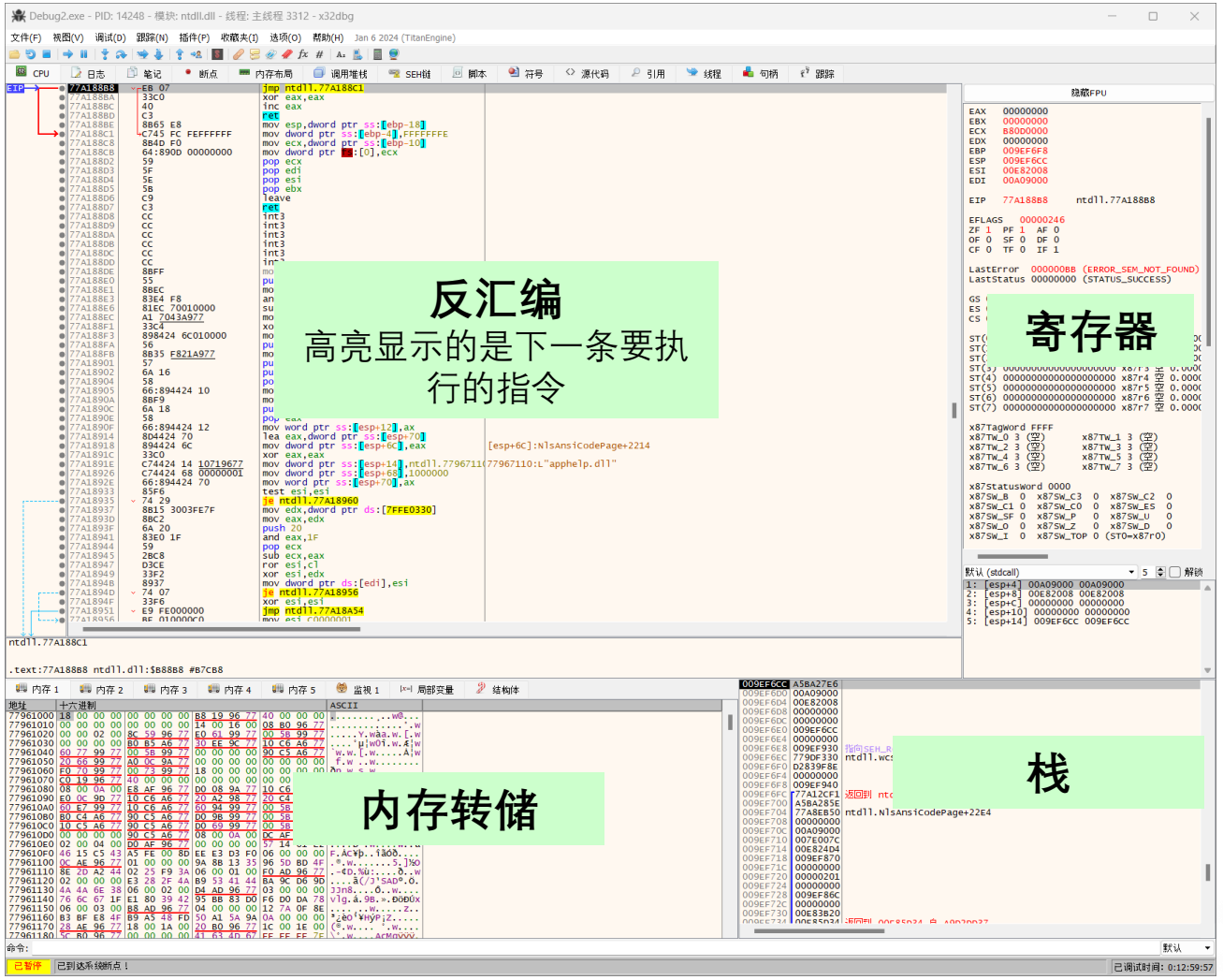
C 驱动程序

D 正在执行的程序

提交

x64dbg的界面

x64dbg界面

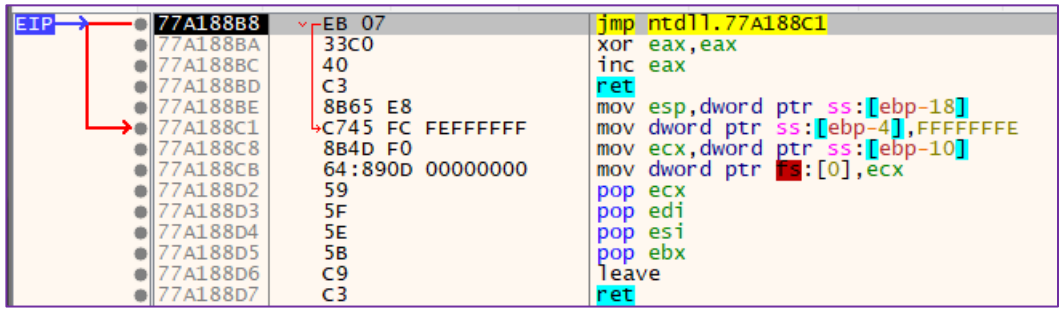


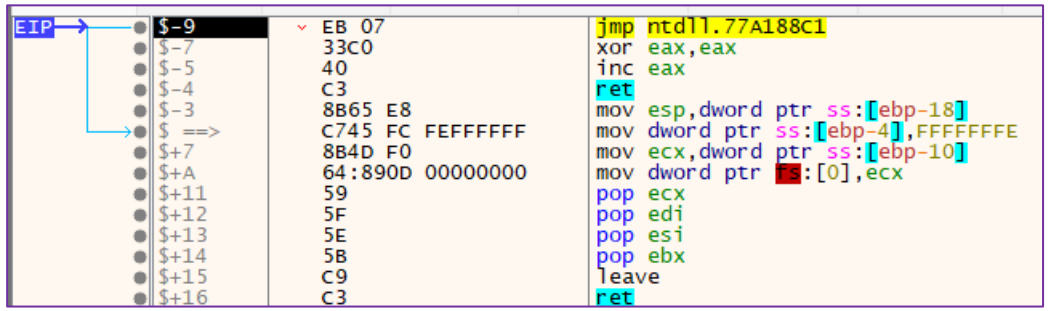
反汇编窗口

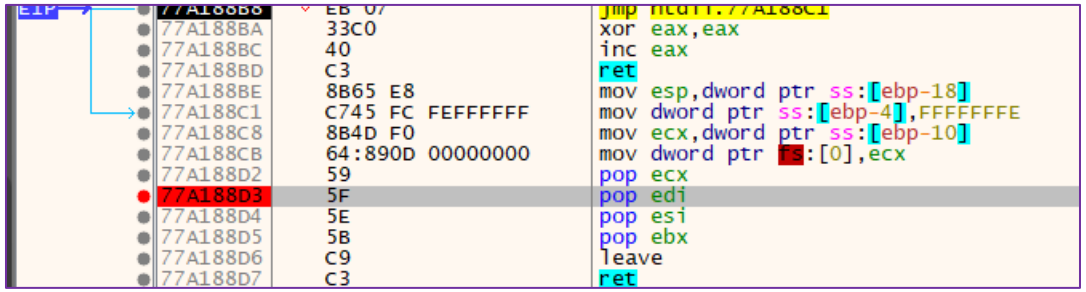
- 反汇编面板窗口(Disassembler window)显示被调试程序的代码
  - 地址(Address)
  - 十六进制的机器码(Hex dump)
  - 反汇编代码(Disassembly)
  - 注释(Comment)，相关API参数

反汇编窗口的双击操作

- Address列:显示相对地址，再次双击返回标准地址模式。
- Hex dump列:设置或取消无条件断点，对应的快捷键是“F2"键。

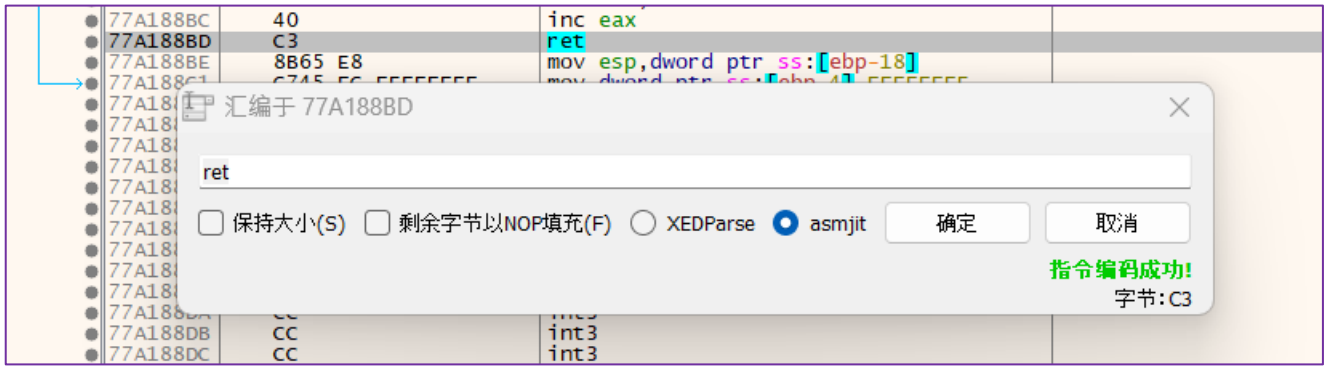






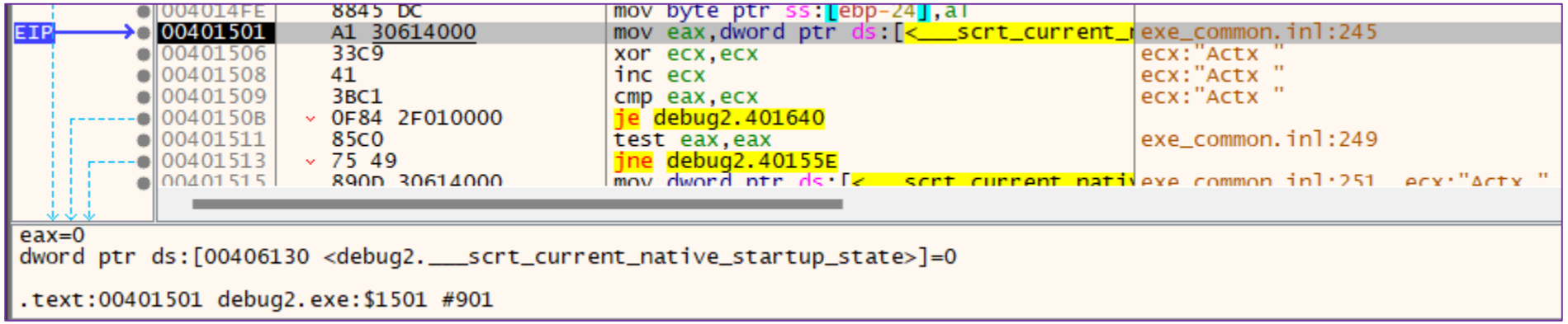
反汇编窗口的双击操作

- Disassembly列:修改汇编代码，对应的快捷键是空格键。
- Comment列:增加或编辑注释，对应的快捷键是“;”键。



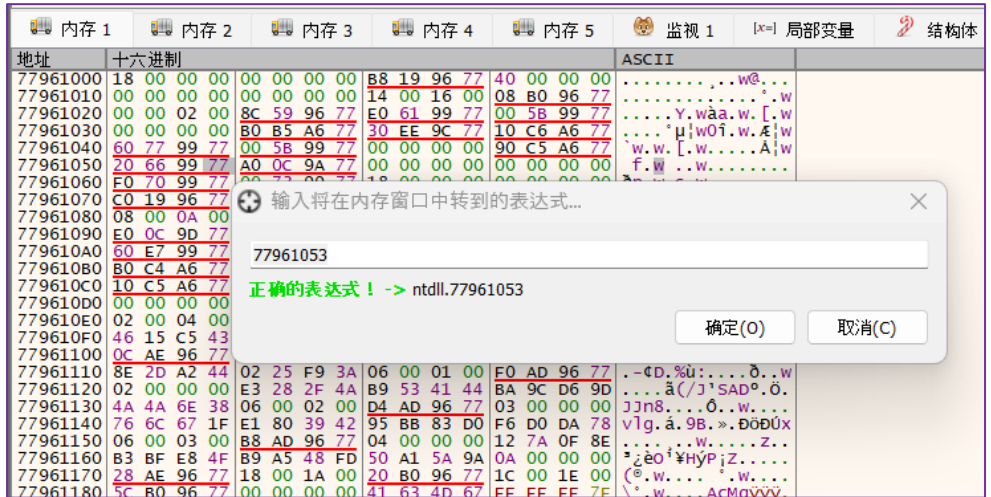
信息窗口

- 在进行动态跟踪时，信息窗口(Information window)显示与指令相关的各寄存器的值、API函数调用提示和跳转提示等信息。

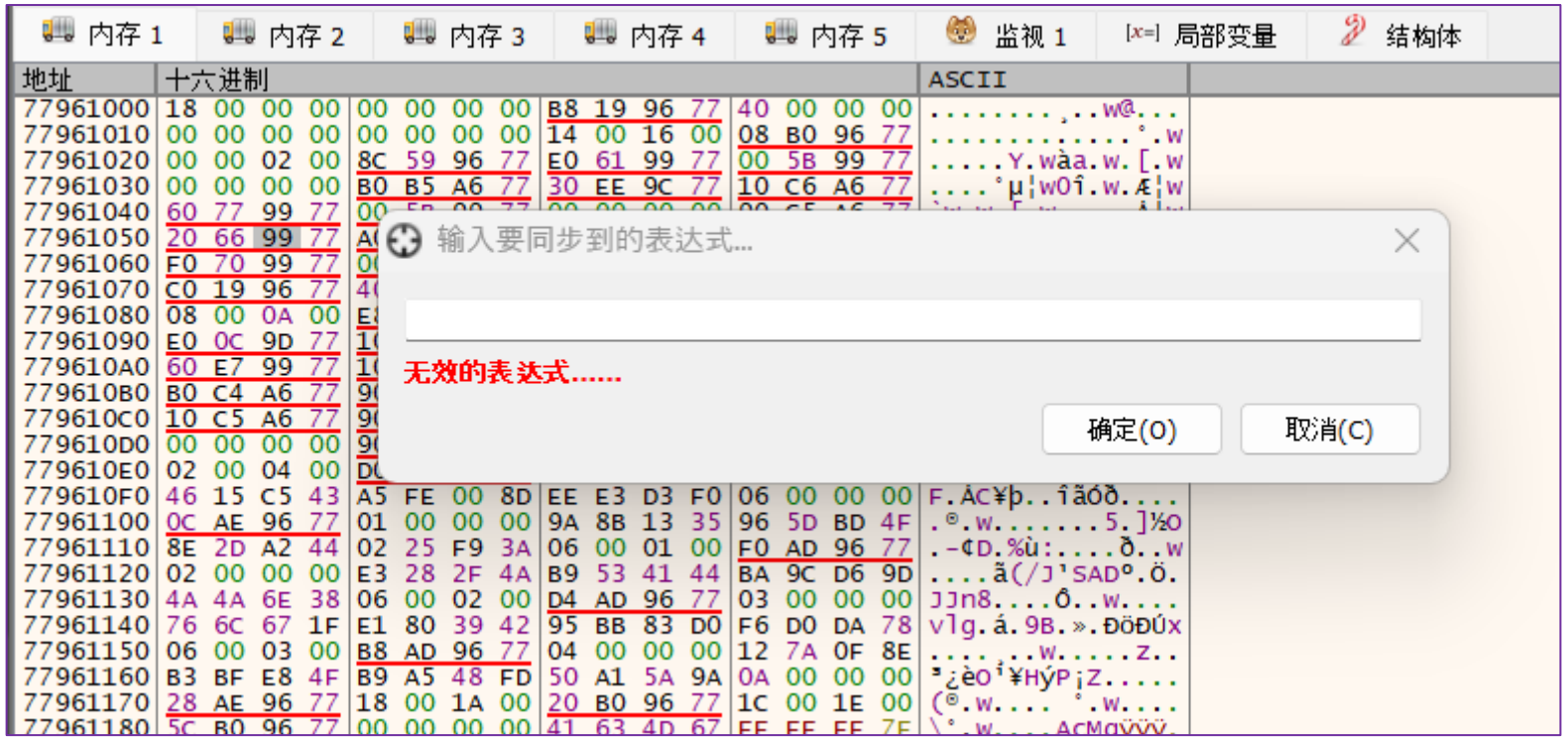


数据窗口

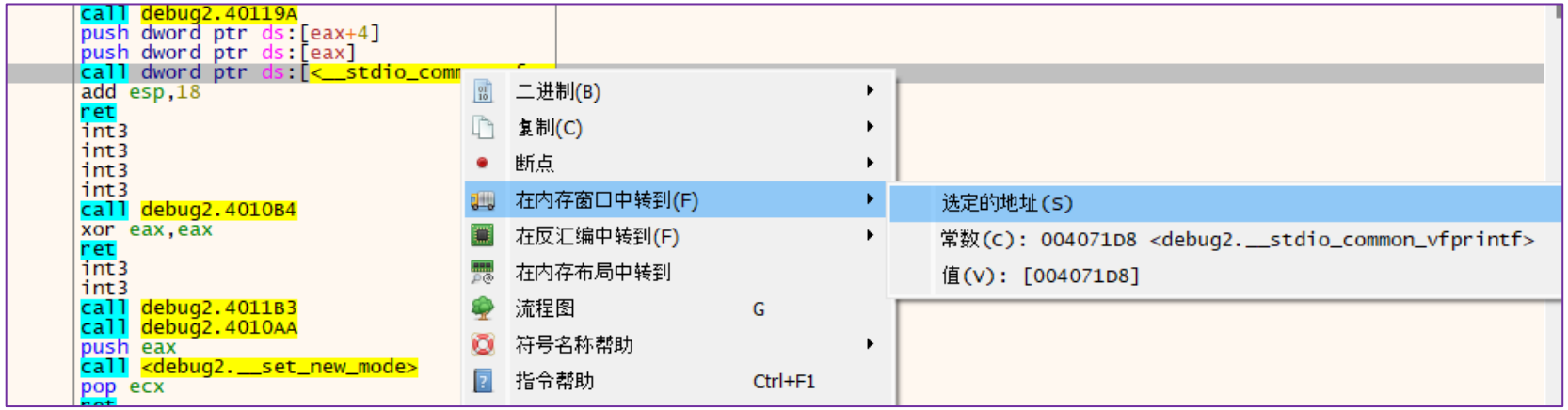
- 数据窗口〔Dump window)以十六进制和字符方式显示可执行文件内存中的数据
- 要显示指定内存地址的数据，可单击右键快捷菜单中的“Go to expression”命令或按“Ctrl十G"快捷键，打开地址窗口，输人地址

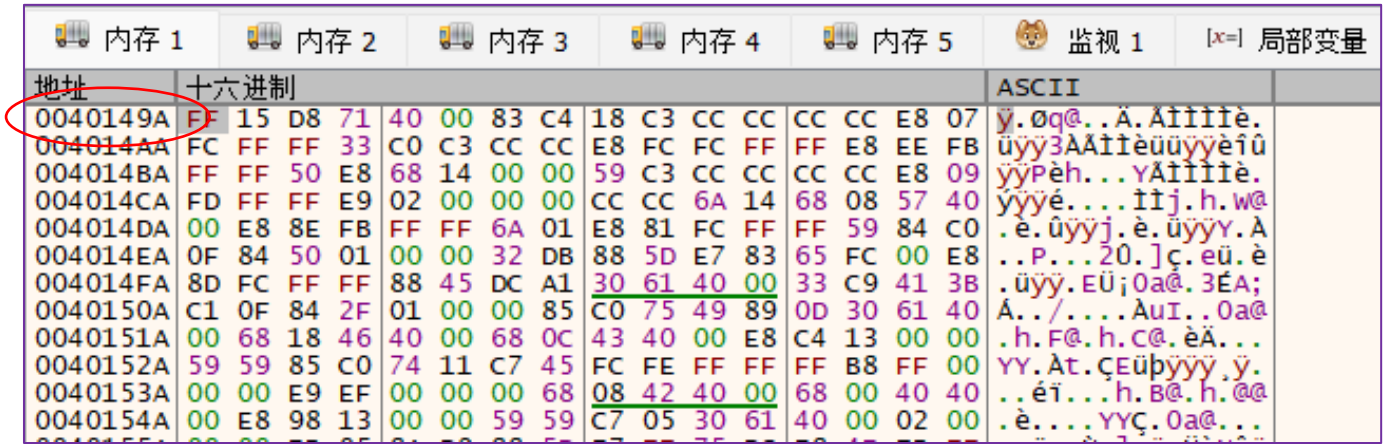


数据窗口


内存 1
内存 2
内存 3
内存 4
内存 5
监视 1
局部变量
结构体
地址
十六进制
ASCII
77961000 18 00 00 00 00 00 00 00 B8 19 96 77 40 00 00 00
77961010 00 00 00 00 00 00 00 00 14 00 16 00 08 B0 96 77
77961020 00 00 02 00 8C 59 96 77 E0 61 99 77 00 5B 99 77
77961030 00 00 00 00 B0 B5 A6 77 30 EE 9C 77 10 C6 A6 77
77961040 60 77 99 77
77961050 20 66 99 77
77961060 F0 70 99 77
77961070 C0 19 96 77
77961080 08 00 0A 00
77961090 E0 0C 9D 77
779610A0 60 E7 99 77
779610B0 B0 C4 A6 77
779610C0 10 C5 A6 77
779610D0 00 00 00 00
779610E0 02 00 04 00
779610F0 46 15 C5 43 A5 FE 00 8D EE E3 D3 F0 06 00 00 00
77961100 0C AE 96 77 01 00 00 00 9A 8B 13 35 96 5D BD 4F
77961110 8E 2D A2 44 02 25 F9 3A 06 00 01 00 F0 AD 96 77
77961120 02 00 00 00 E3 28 2F 4A B9 53 41 44 BA 9C D6 9D
77961130 4A 4A 6E 38 06 00 02 00 D4 AD 96 77 03 00 00 00
77961140 76 6C 67 1F E1 80 39 42 95 BB 83 D0 F6 D0 DA 78
77961150 06 00 03 00 B8 AD 96 77 04 00 00 00 12 7A 0F 8E
77961160 B3 BF E8 4F B9 A5 48 FD 50 A1 5A 9A 0A 00 00 00
77961170 28 AE 96 77 18 00 1A 00 20 B0 96 77 1C 00 1E 00
77961180 5C B0 96 77 00 00 00 00 41 63 4D 67 FF FF FF 7F
输入要同步到的表达式...
无效的表达式.....
确定(O)
取消(C)

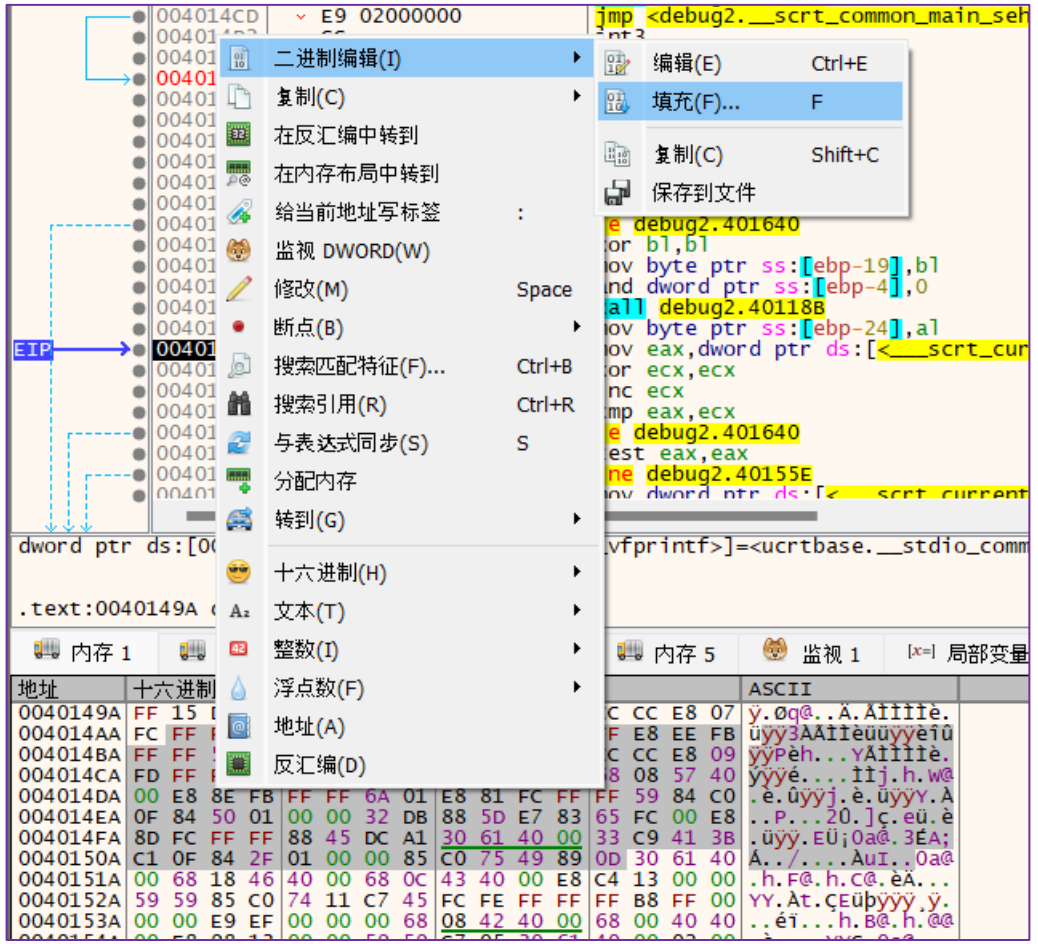
在数据窗口中跟踪





南開大學 Nankai University

数据编辑

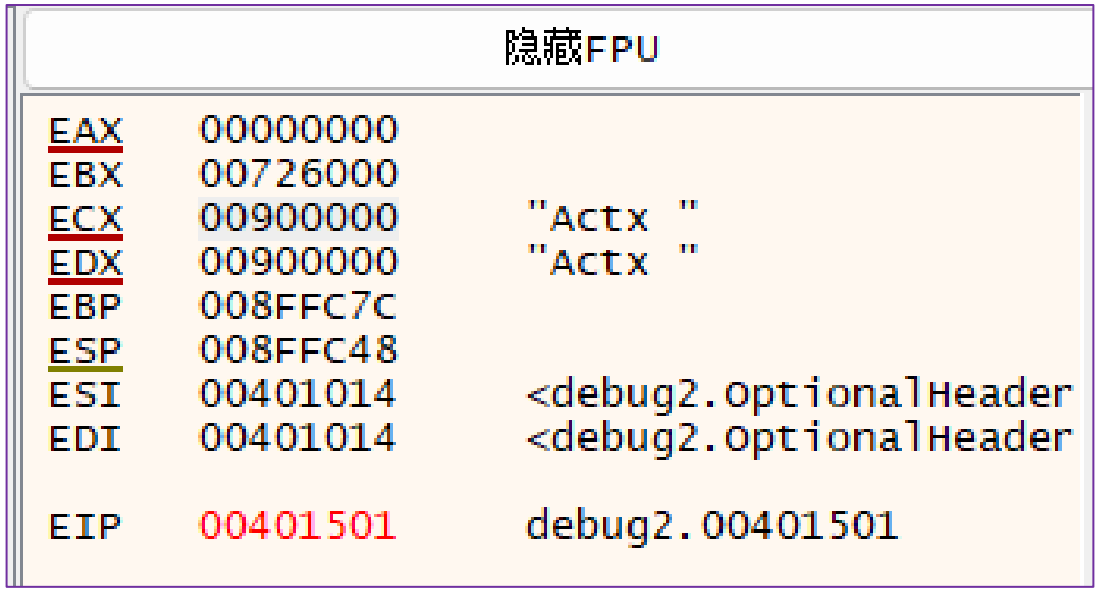

二进制编辑(I)
复制(C)
在反汇编中转到
在内存布局中转到
给当前地址写标签 :
监视 DWORD(W)
修改(M) Space
断点(B)
搜索匹配特征(F)... Ctrl+B
搜索引用(R) Ctrl+R
与表达式同步(S) S
分配内存
转到(G)
十六进制(H)
文本(T)
整数(I)
浮点数(F)
地址(A)
反汇编(D)
编辑(E) Ctrl+E
填充(F)... F
复制(C) Shift+C
保存到文件
EIP
内存 1
内存 5
监视 1
局部变量
地址
ASCII

寄存器窗口

- 寄存器面板窗口(Registers window)显示CPU各寄存器的值，支持浮点、MMX寄存器。
- 红色用来高亮显示，上一条CPU指令执行之后，数值发生变化的寄存器

寄存器窗口

- 通过双击一个寄存器值，修改该寄存器的数值



栈窗口

- 栈窗口(Stack window)显示栈的内容，即ESP指向内存地址的内容。
- 栈中有函数的参数、局部变量、函数的返回地址等重要信息

多选题 10分

设置

x64dbg界面有哪几个窗口

A 反汇编窗口(Disassembler window)

B 信息窗口(Information Window)

C 寄存器窗口(Register Window)

D 数据窗口(Dump Window)

E 栈窗口（Stack Window）

提交

多选题 10分

设置

x64dbg的栈窗口可以看到下面哪些信息

A 函数返回地址

B 函数的参数

C 函数的局部变量

D 全局变量

提交

多选题 10分

设置

x64dbg中可以修改以下哪些内容？

A 内存数据

B CPU寄存器

C 栈上的数据

D CPU指令

提交

Memory Map

Memory Map

CPU | 日志 | 笔记 | 断点 | 内存布局 | 调用堆栈 | SEH链 | 脚本 | 符号 | 源代码 | 引用

地址	大小	方	页面信息	内容	类型	页面保护	初始保护
00400000	00001000	用户模块	debug2.exe		IMG	-R---	ERWC-
00401000	00003000	用户模块	".text"		IMG	ER---	ERWC-
00404000	00002000	用户模块	".rdata"		IMG	-R---	ERWC-
00406000	00001000	用户模块	".data"		IMG	-RW--	ERWC-
00407000	00001000	用户模块	".idata"		IMG	-R---	ERWC-
00408000	00001000	用户模块	".00cfg"		IMG	-R---	ERWC-
00409000	00001000	用户模块	".rsrc"		IMG	-R---	ERWC-
0040A000	00001000	用户模块	".reloc"		IMG	-R---	ERWC-
00570000	00011000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
00590000	00010000	用户模块			MAP	-RW--	-RW--
005A0000	0001F000	用户模块			MAP	-R---	-R---
005C0000	00035000	用户模块	保留		PRV		-RW--
005F5000	0000B000	用户模块			PRV	-RW-G	-RW--
00600000	00125000	用户模块	保留		PRV		-RW--
00725000	00016000	用户模块	PEB, TEB (14660), WOW64 TEB (146		PRV	-RW--	-RW--
0073B000	000C5000	用户模块	保留 (00600000)		PRV		-RW--
00800000	000FB000	用户模块	保留		PRV		-RW--
008FB000	00005000	用户模块	堆栈 (14660)		PRV	-RW-G	-RW--
00900000	00004000	用户模块			MAP	-R---	-R---
00910000	00001000	用户模块			MAP	-R---	-R---
00920000	00002000	用户模块			PRV	-RW--	-RW--
00930000	00011000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
00950000	00011000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
00970000	00003000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
00980000	00011000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
009A0000	00003000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
009B0000	00003000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
009C0000	000CE000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
00A90000	00008000	用户模块			PRV	-RW--	-RW--
00A98000	00008000	用户模块	保留 (00A90000)		PRV		-RW--
00AA0000	00011000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
00AC0000	00011000	用户模块	\Device\HarddiskVolume3\Windows\S		MAP	-R---	-R---
00AE0000	00002000	用户模块			MAP	-R---	-R---
00AF0000	00002000	用户模块			MAP	-R---	-R---
00B00000	00001000	用户模块			MAP	-R---	-R---
00B10000	00035000	用户模块	保留		PRV		-RW--
00B45000	0000B000	用户模块			PRV	-RW-G	-RW--
00B50000	00035000	用户模块	保留		PRV		-RW--
00B85000	0000B000	用户模块			PRV	-RW-G	-RW--

Memory Map

- 内存中的exe和dll程序的位置被x64dbg识别
- 双击有可执行属性的行，会跳转到反汇编窗口
- 双击没有可执行属性的行，会跳转到数据窗口

重定位（Rebasing）

- 重定位发生在PE文件装载时，没有装载到指定的Image Base地址时
- PE文件头中Image Base
  - exe文件会装载到Image Base
  - 大部分exe的Image Base是0x00400000
  - ASLR（Address Space Layout Randomization）

dll重定位

- dll经常发生重定位
  - 一个应用程序会装载多个dll
  - Windows的dll文件通常会使用不同的Image Base
    - kernel32.dll 6B800000， user32.dll 69E00000
  - 第三方的dll通常都是用相同的Image Base

绝对地址对相对地址

Example 10-1. 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
```

- 相对地址（Relative Address）不受重定向影响
- 绝对地址（Absolute Address）受影响

多选题 10分

设置

选出图中相对地址？

Example 10-1. 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
```

A [ebp + var_8]

B [ebp + var_4]

C jnz loc_0040120

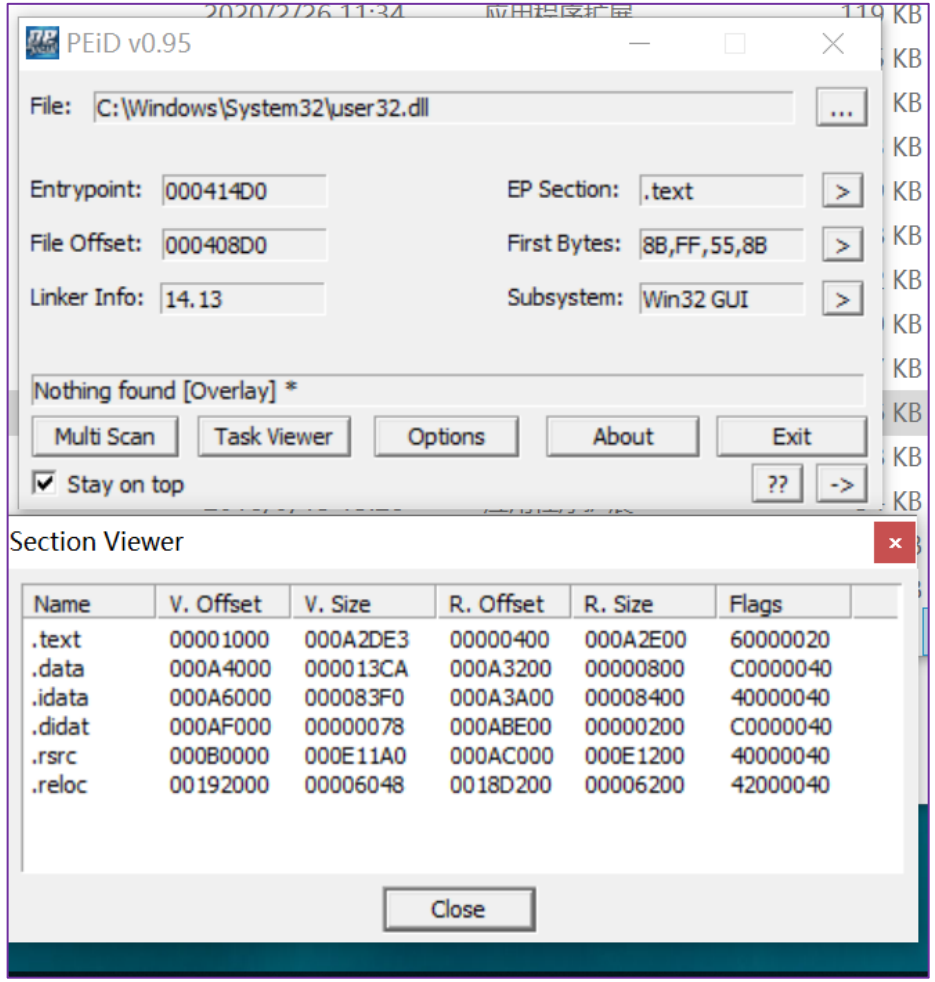
D dword_40CF60

提交

修改绝对地址

- 大部分dll文件，都有一个.reloc节，用于重定位发生时的修复
- dll文件的装载顺序是不固定的
- 重定位的位置也是不可预测的

.reloc节


PEiD v0.95
File: C:\Windows\System32\user32.dll
Entrypoint: 000414D0
EP Section: .text
File Offset: 000408D0
First Bytes: 8B,FF,55,8B
Linker Info: 14.13
Subsystem: Win32 GUI
Nothing found [Overlay] *
Multi Scan
Task Viewer
Options
About
Exit
Stay on top
Section Viewer
Name V. Offset V. Size R. Offset R. Size Flags
.text 00001000 000A2DE3 00000400 000A2E00 60000020
.data 000A4000 000013CA 000A3200 00000800 C0000040
.idata 000A6000 000083F0 000A3A00 00008400 40000040
.didat 000AF000 00000078 000ABE00 00000200 C0000040
.rsrc 000B0000 000E11A0 000AC000 000E1200 40000040
.reloc 00192000 00006048 0018D200 00006200 42000040
Close

dll重定位

- 如果删除dll的.reloc节
  - dll不能进行重定位
  - 必须装载到PE文件头指定的Image Base地址
- 重定位会影响程序的装载时间
  - Good programmer specifies non-default base image for dlls。

Binary Ninja 和 x64dbg

- 重定位
  - Binary Ninja静态反汇编是会将可执行文件加载到其默认地址
  - x64dbg的动态调试，是重定位之后的结果
  - Binary Ninja 和x64dbg的反汇编地址有可能不一致
    - 可以在Binary Ninja中进行rebase以匹配调试过程中的实际加载地址
    - 或者可以在x64dbg中关闭ASLR

线程和栈

- Memory Map窗口，除了exe和dll，还有栈的信息
- 每个线程都有一个自己私有的栈

地址	大小	方	页面信息	内容	类型	页面保护	初始保护
00400000	00001000	用户模块	debug2.exe		IMG	-R---	ERWC-
00401000	00003000	用户模块	".text"		IMG	ER---	ERWC-
00404000	00002000	用户模块	".rdata"		IMG	-R---	ERWC-
00406000	00001000	用户模块	".data"		IMG	-RW--	ERWC-
00407000	00001000	用户模块	".idata"		IMG	-R---	ERWC-
00408000	00001000	用户模块	".00cfg"		IMG	-R---	ERWC-
00409000	00001000	用户模块	".rsrc"		IMG	-R---	ERWC-
0040A000	00001000	用户模块	".reloc"		IMG	-R---	ERWC-
00570000	00011000	用户模块	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
00590000	00010000	用户模块			MAP	-RW--	-RW--
005A0000	0001F000	用户模块			MAP	-R---	-R---
005C0000	00035000	用户模块	保留		PRV		-RW--
005F5000	0000B000	用户模块			PRV	-RW-G	-RW--
00600000	00125000	用户模块	保留		PRV		-RW--
00725000	00016000	用户模块	PEB, TEB (14660), WoW64 TEB (146		PRV	-RW--	-RW--
0073B000	000C5000	用户模块	保留 (00600000)		PRV		-RW--
00800000	000FB000	用户模块	保留		PRV		-RW--
008FB000	00005000	用户模块	堆栈 (14660)		PRV	-RW-G	-RW--
00900000	00004000	用户模块			MAP	-R---	-R---
00910000	00001000	用户模块			MAP	-R---	-R---
00920000	00002000	用户模块			PRV	-RW--	-RW--

多选题 10分

设置

如何解决和避免dll的重定位问题？

A 使用不同的Image Base地址

B 使用编译器默认的Image Base地址

C .reloc节记录需要修改的信息

D 指定dll的装载顺序

提交

南開大學 Nankai University

多选题 10分

设置

以下哪些是线程私有的？

A 栈

B 寄存器

C 内存空间

D 代码

提交

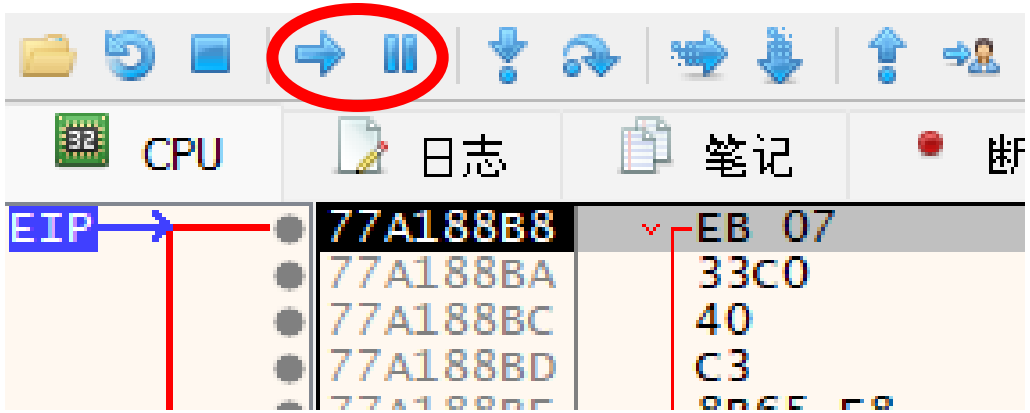
南開大學 Nankai University

执行指令

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	

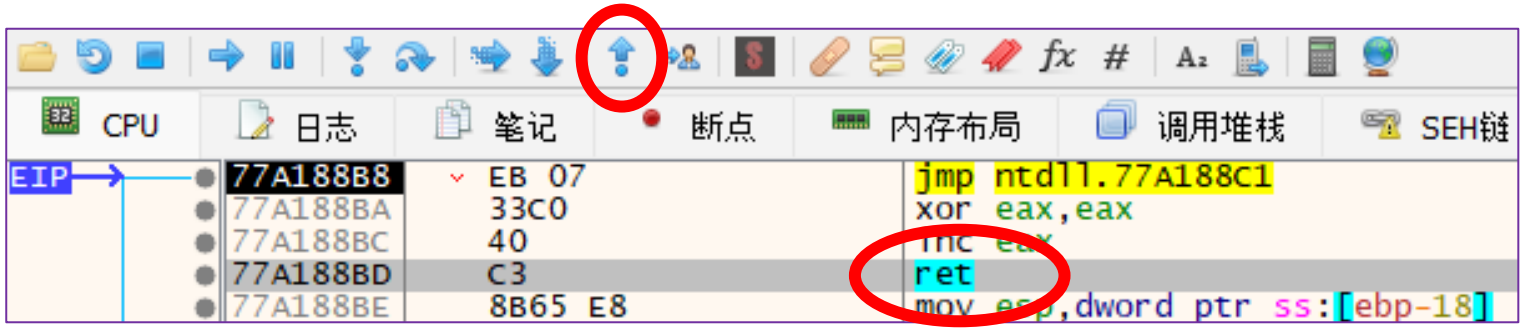
运行和暂停

- 运行调试应用程序（F9）
- 暂停调试程序（F12）



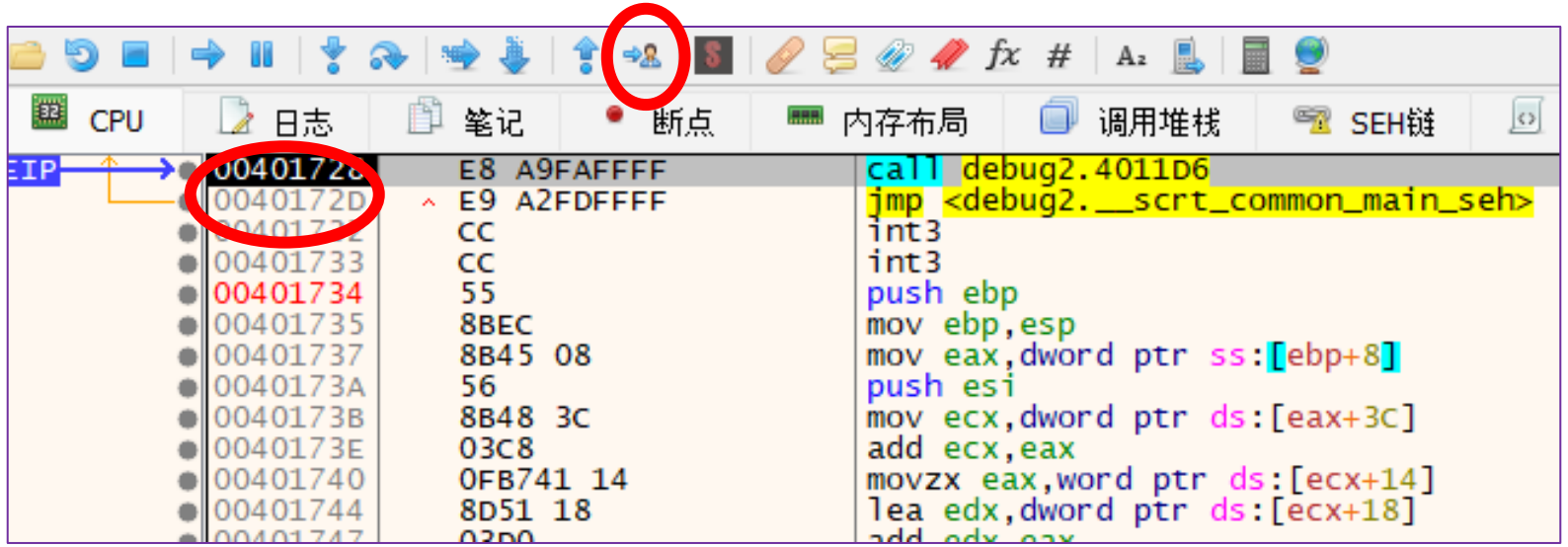
执行直到返回

- “执行直到返回”当前函数执行到返回指令之前的指令。
- 结束当前调试的函数



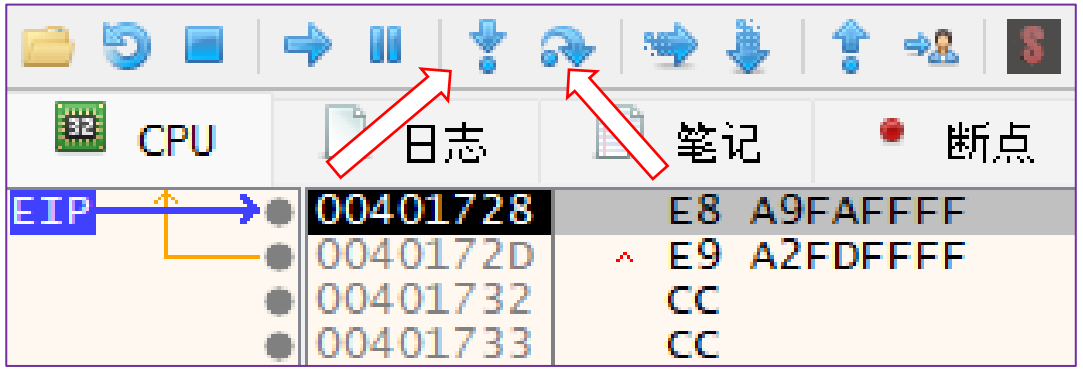
执行到用户代码

- “执行到用户代码”会从库函数中返回到程序的代码节，例如.text节的代码



按步执行

- 步入（F7），也叫做单步执行
- 步过（F8），对于函数，可以使用步过，执行完整个函数后暂停。



多选题 10分

设置

x64dbg支持哪些代码执行操作？

A 执行和暂停

B 执行直到返回

C 执行到用户代码

D 按步执行

提交

单选题 10分

设置

x64dbg调试中，如果想快速执行完当前函数，需要选择哪种执行方式？

A 执行和暂停

B 执行直到返回

C 执行到用户代码

D 按步执行

提交

南開大學 Nankai University

单选题 10分

设置

x64dbg调试中，如果想快速从dll库代码空间回到用户代码空间，需要选择哪种执行方式？

A 执行和暂停

B 执行直到返回

C 执行到用户代码

D 按步执行

提交

南開大學 Nankai University

断点

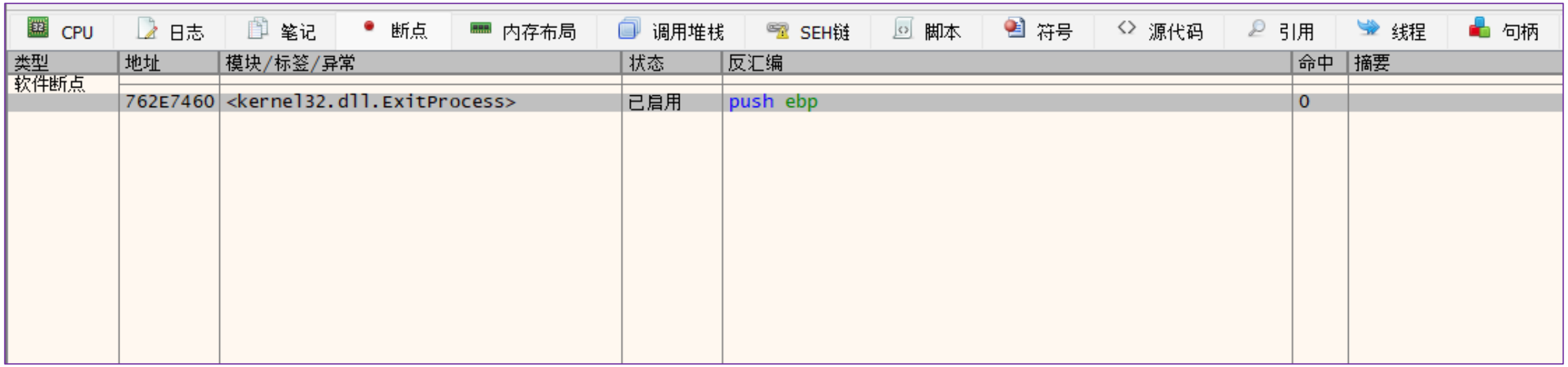
断点（Breakpoint）

- 断点（Breakpoint）是调试器的重要功能，使执行的程序中断在指定的地方，从而方便对其进行分析。
  - 查看间接跳转的位置
  - 查看函数的参数
  - 快速执行循环结构体

断点类型

- 软件断点
- 硬件断点
- 内存断点
- 消息断点
- 条件断点

断点窗口


CPU
日志
笔记
断点
内存布局
调用堆栈
SEH链
脚本
符号
源代码
引用
线程
句柄
类型
地址
模块/标签/异常
状态
反汇编
命中
摘要
软件断点
762E7460
<kernel32.dll.ExitProcess>
已启用
push ebp
0

保存断点

- 关闭x64dbg之后， x64dbg会自动保存调试时设置的断点信息。
- 再次打开x64dbg调试相同的程序，断点信息还在。

软件断点

- 在x64dbg中可以使用bp命令或者“F2”快捷键来设置/消断点。
  - INT 3指令替换原始CPU指令，操作码是0xCC
  - INT 3指令产生异常，停止程序执行
  - x64dbg捕获异常
  - 恢复原始的CPU指令

软件断点

- 优点
  - 可以设置无数个断点
- 缺点
  - 修改了原始程序的机器码
  - 容易被软件检测到
    - 检测API函数入口点是否为0xCC

软件断点

- 一些软件会检测API的首地址是否为0xCC来检测INT 3断点。方法是取得检测函数的地址，然后读取它的第1个字节，判断它是否等于“CC"

```
FARPROC Uaddr ;
BYTE Mark = 0;
(FARPROC&) Uaddr =GetProcAddress ( LoadLibrary("user32.dll"), "MessageBoxA");
Mark = *((BYTE*)Uaddr);          //取 MessageBoxA 函数的第 1 个字节
if(Mark ==0xCC)                  //如果该字节为“CC”，则认为 MessageBoxA 函数被下断了
    return TRUE                  //发现断点
```

软件断点

- 将软件断点设在函数内部或末尾，例如将断点设在函数入口的下一行
- 使用硬件断点

多选题 10分

设置

软件断点同时可以设置多少个？

A 1

B 2

C 4

D 任意多个

提交

南開大學 Nankai University

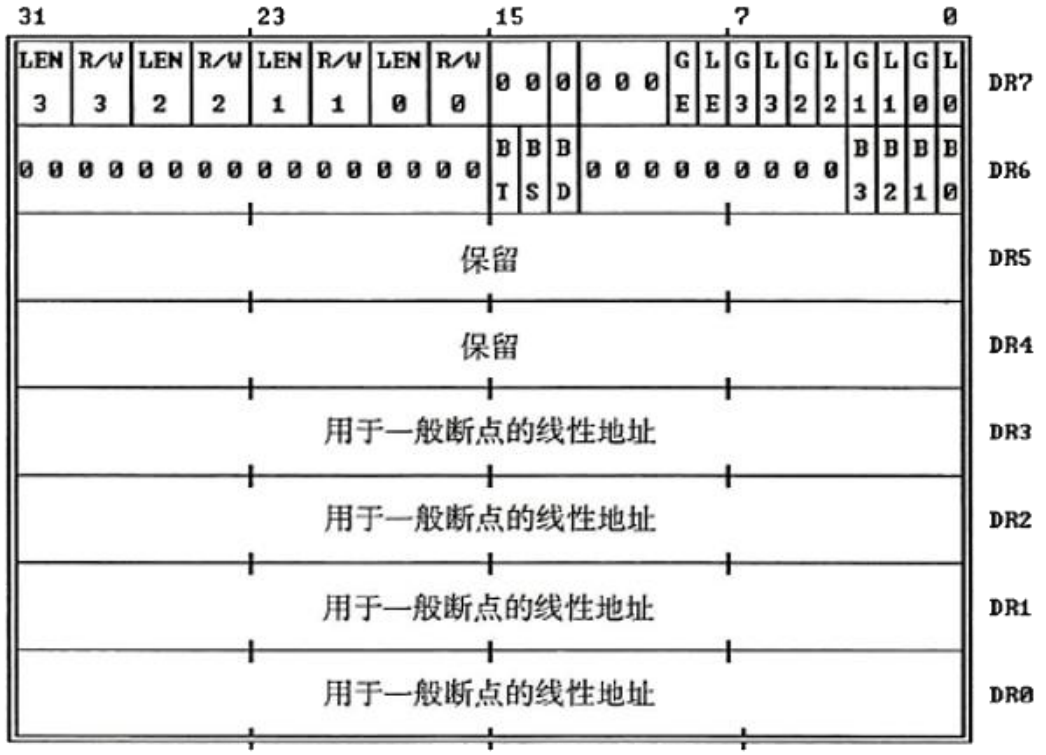
硬件断点

- 不需要将CPU指令首字节修改为"CC"
- 基于CPU中断DRx寄存器设置
- 硬件断点分为三类：
  - 硬件执行断点
  - 硬件读取断点
  - 硬件写入断点

硬件执行断点

- 实现方式
  - 将断点的内存地址存储到CPU的DRx寄存器
  - CPU执行过程中会判断下一条指令的地址与DRx寄存器中的地址是否一致
  - 如果一致，CPU产生异常，中止程序执行，将控制权转移给调试器

硬件执行断点



注意：0 或 1 保留，未定义

图 2.24　Intel 调试寄存器示意图

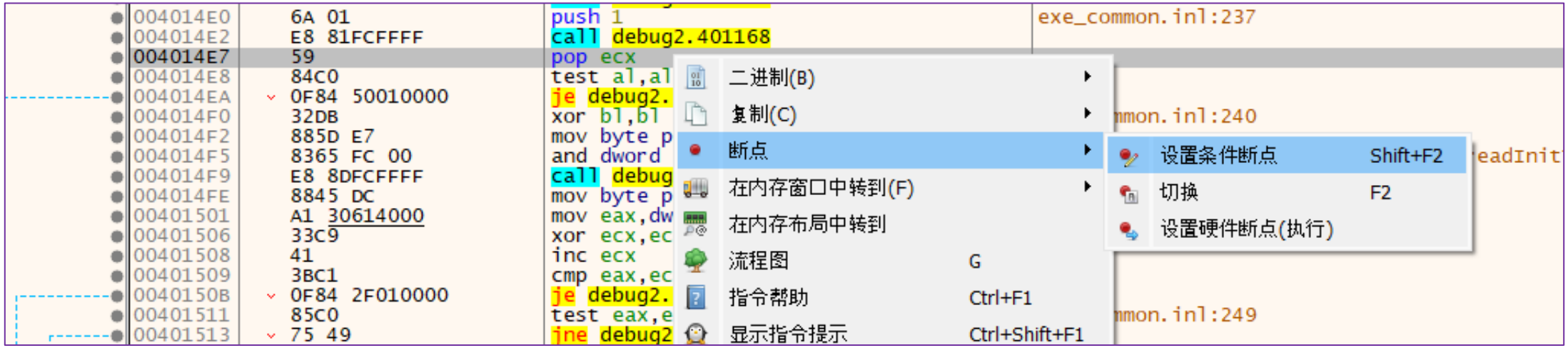
硬件执行断点

- 优点
  - 隐蔽性强，不修改CPU指令，更难检测到
  - 速度快，在CPU内部进行判断
- 缺点
  - 只有4个CPU寄存器存储断点地址，DR0到DR3
  - 所有硬件断点共享4个DRx寄存器，包括后面介绍的硬件读取/写入断点

硬件执行断点设置

- 在指定的代码行单击右键，执行快捷菜单中的“断点”→“硬件断点”
- 在命令行中，输入命令“he 断点地址”

硬件执行断点


004014E0 6A 01 push 1
004014E2 E8 81FCFFFF call debug2.401168
004014E7 59 pop ecx
004014E8 84C0 test al,al
004014EA 0F84 50010000 je debug2.
004014F0 32DB xor bl,bl
004014F2 885D E7 mov byte p
004014F5 8365 FC 00 and dword
004014F9 E8 8DFCFFFF call debug
004014FE 8845 DC mov byte p
00401501 A1 30614000 mov eax,dw
00401506 33C9 xor ecx,ec
00401508 41 inc ecx
00401509 3BC1 cmp eax,ec
0040150B 0F84 2F010000 je debug2.
00401511 85C0 test eax,e
00401513 75 49 jne debug2
exe_common.inl:237
mon.inl:240
eadInit
mon.inl:249
二进制(B)
复制(C)
断点
在内存窗口中转到(F)
在内存布局中转到
流程图 G
指令帮助 Ctrl+F1
显示指令提示 Ctrl+Shift+F1
设置条件断点 Shift+F2
切换 F2
设置硬件断点(执行)

查看和管理硬件断点

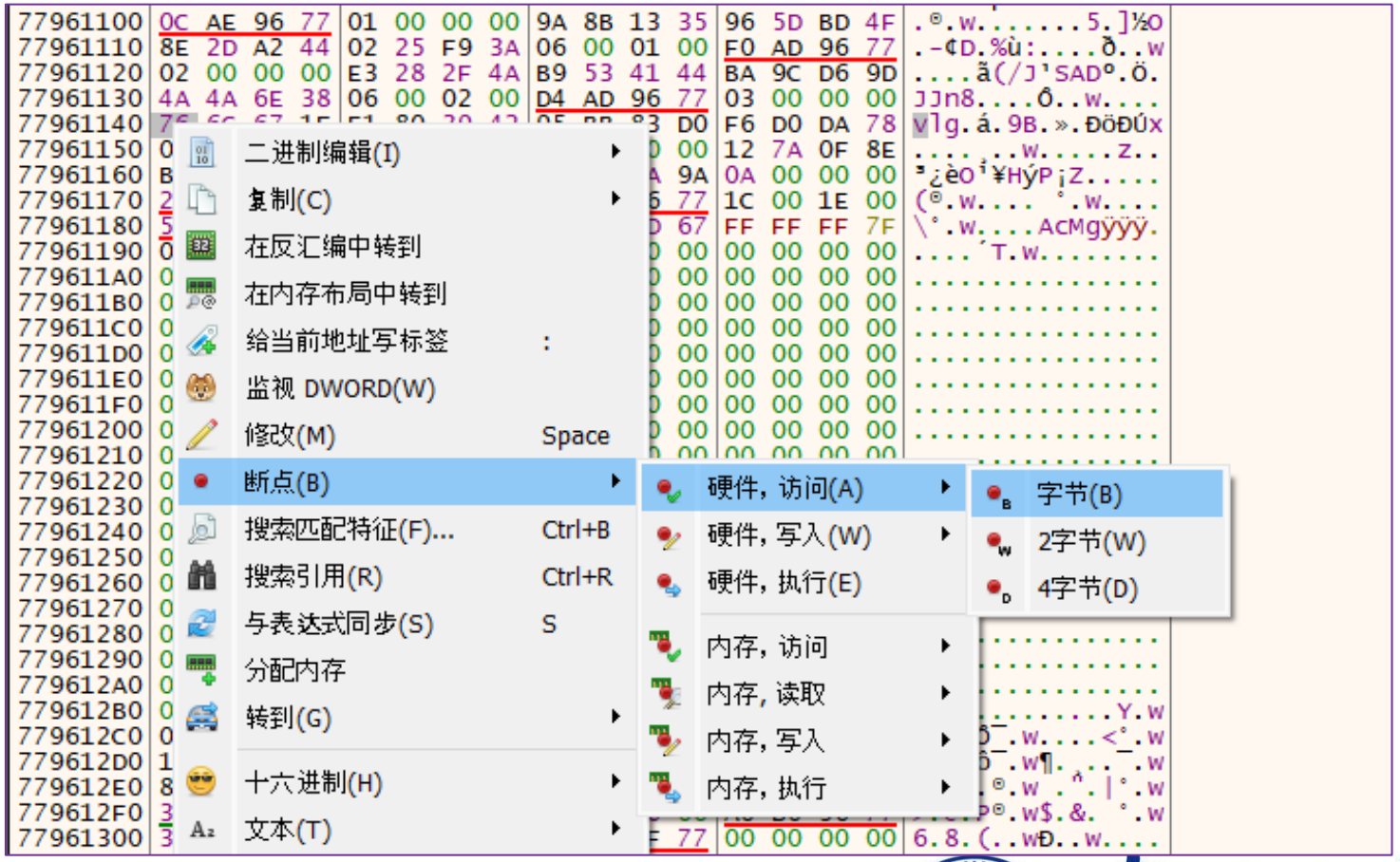
- 单击菜单项“查看”→“硬件断点"，打开硬件断点面板
- 单击“Delete"按钮删除相应的硬件断点

硬件写入/读取断点

- 硬件断点除了可以在特定指令被执行时触发，还可以在特定字节被读取时触发
- 可以用于发现内存中的数据何时被修改
- 这样的断点我们成为硬件写入断点或者硬件读取断点

设置硬件读取/写入断点

- 硬件写入/读取断点可以设置触发条件为读取/写入/执行
- 硬件写入/读取断点可以设置大小位1/2/4字节



单选题 10分

设置

硬件断点同时最多可以设置几个？

A 1

B 2

C 4

D 任意多个

提交

内存断点

- 内存页的属性
  - 可读 readable
  - 可写 writable
  - 可执行 executable

```
mov   dword ptr  [405528], edx     ;对[405528]处的内存进行写入
mov   dword ptr  edx,[405528]      ;对[405528]处的内存进行读取
```

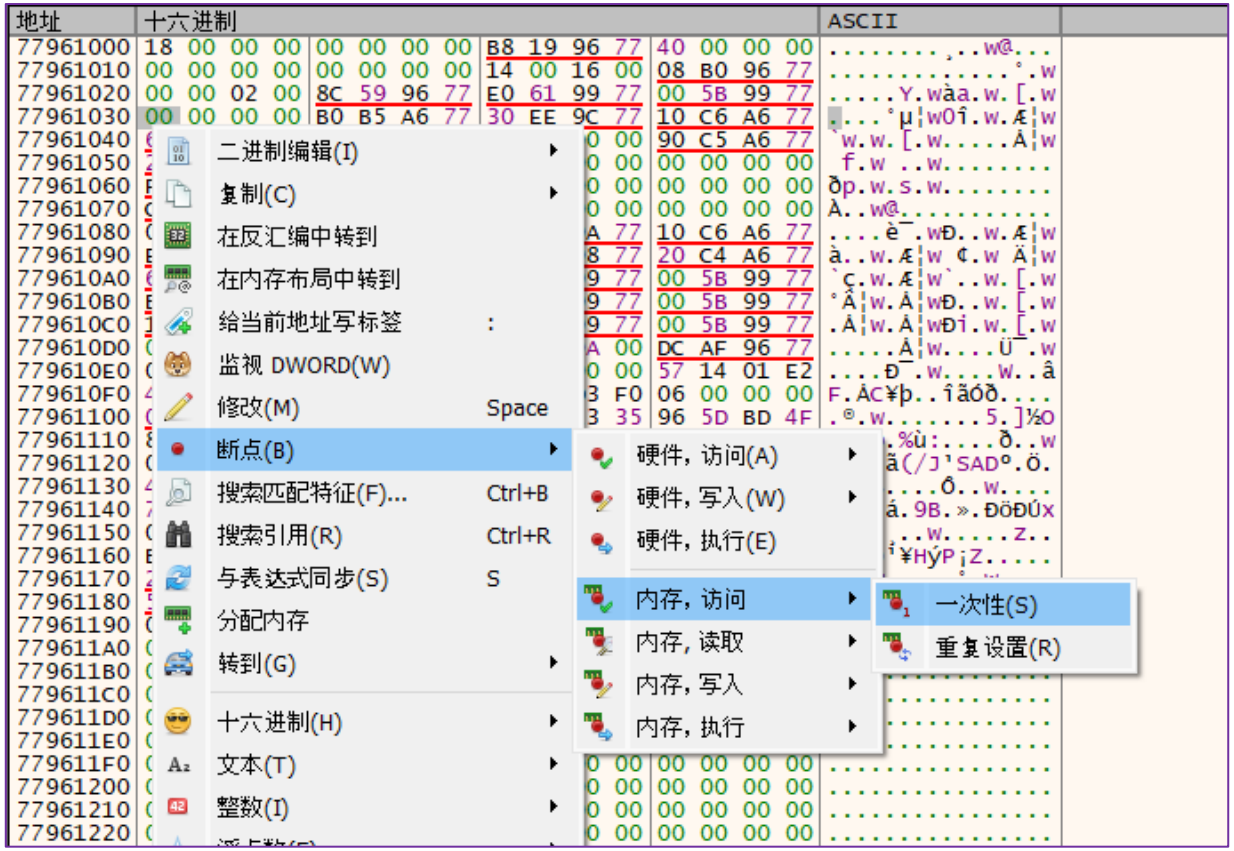
内存断点原理

- 对所设的内存地址页设置不可访问或者不可写属性
- 当被调试程序访问或者写入内存页时，产生异常，中断执行
- CPU将控制权交给x64dbg

内存断点原理

- x64dbg比较异常地址是不是断点地址
- 如果是断点地址，就继续中断程序执行
- 如果不是断点地址，恢复程序的执行

设置内存断点


地址
十六进制
ASCII
77961000 18 00 00 00 00 00 00 00 B8 19 96 77 40 00 00 00
77961010 00 00 00 00 00 00 00 00 14 00 16 00 08 B0 96 77
77961020 00 00 02 00 8C 59 96 77 E0 61 99 77 00 5B 99 77
77961030 00 00 00 00 B0 B5 A6 77 30 EE 9C 77 10 C6 A6 77
二进制编辑(I)
复制(C)
在反汇编中转到
在内存布局中转到
给当前地址写标签 :
监视 DWORD(W)
修改(M) Space
断点(B)
搜索匹配特征(F)... Ctrl+B
搜索引用(R) Ctrl+R
与表达式同步(S) S
分配内存
转到(G)
十六进制(H)
文本(T)
整数(I)
硬件，访问(A)
硬件，写入(W)
硬件，执行(E)
内存，访问
内存，读取
内存，写入
内存，执行
一次性(S)
重复设置(R)

内存断点

- 程序中断在访问内存的CPU指令
  - 还没有访问或者写入内存
- 硬件断点，程序中断在访问内存CPU指令的下一条指令
  - 内存的访问或者写操作已经完成

内存断点

- 内存中的数据可以设置内存断点，内存中的指令也可以设置内存断点
  - CPU指令存储在内存中
  - 当CPU指令被执行的时候，CPU会访问内存读取下一条指令
  - 触发内存访问异常

内存断点

- 优点：
  - 不需要修改CPU指令
  - 当INT 3断点和硬件断点无效的时候可以使用内存断点

内存断点

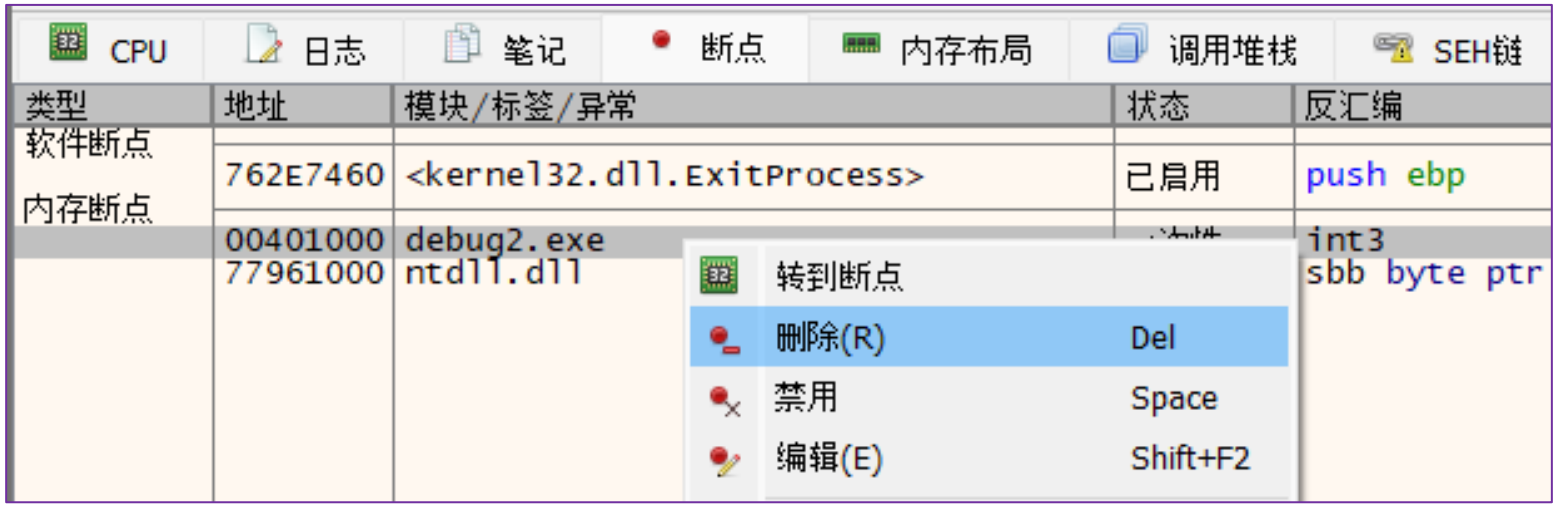
- 缺点：
  - 执行速度慢
  - 每次出现内存异常时，都要通过比较来确定是否应该中断程序执行
  - 内存断点会降低x64dbg的执行速度
  - x64dbg同一时间，同一内存页上，只能设置1个内存断点

内存断点

CPU | 日志 | 笔记 | 断点 | 内存布局 | 调用堆栈 | SEH链 | 脚本 | 符号 | 源代码 | 引

地址	大小	方	页面信息	内容	类型	页面保护	初始保护
00400000	00001000	用户模块	debug2.exe		IMG	-R---	ERWC-
00401000	00003000	用户模块	".text"		IMG	ER--G	ERWC-
00404000	00002000	用户模块	".rdata"		IMG	-R---	ERWC-
00406000	00001000	用户模块	".data"		IMG	-RW--	ERWC-
00407000	00001000	用户模块	".idata"		IMG	-R---	ERWC-
00408000	00001000	用户模块	".00cfg"		IMG	-R---	ERWC-
00409000	00001000	用户模块	".rsrc"		IMG	-R---	ERWC-
0040A000	00001000	用户模块	".reloc"		IMG	-R---	ERWC-
00480000	00011000	用户模块	\Device\HarddiskVolume3\Windows\		MAP	-R---	-R---
004A0000	00010000	用户模块			MAP	-RW--	-RW--
004B0000	0001F000	用户模块			MAP	-R---	-R---
004D0000	00035000	用户模块	保留		PRV		-RW--
00505000	0000B000	用户模块			PRV	-RW-G	-RW--

内存断点的删除


CPU
日志
笔记
断点
内存布局
调用堆栈
SEH链
类型
地址
模块/标签/异常
状态
反汇编
软件断点
762E7460
<kernel32.dll.ExitProcess>
已启用
push ebp
内存断点
00401000
debug2.exe
int3
77961000
ntdll.dll
sbb byte ptr
转到断点
删除(R)
Del
禁用
Space
编辑(E)
Shift+F2

多选题 10分

设置

x64dbg支持哪些内存访问行为的断点

A 内存读断点

B 内存写断点

C 内存执行断点

D 内存读、写、执行组合的断点

提交

单选题 10分

设置

对于同一地址，内存断点同时可以设置几个？

A 1

B 2

C 4

D 任意多个

提交

消息断点

- Windows是由消息驱动的
- 消息断点：当某个特定窗口函数接收到某个特定消息时，消息断点将使程序中断

消息断点

- 消息断点与INT 3断点的区别
  - INT 3断点可以在程序启动之前设置
  - 消息断点只有在窗口被创建之后才能被设置并拦截消息。

Windows消息

- 当用户单击一个按钮、移动光标时，都会产生消息发送给当前窗体。
- Windows消息有4个参数
  - 1个窗口句柄(hwnd)
  - 1个消息编号(msg)
  - 2个32位长(long)的参数

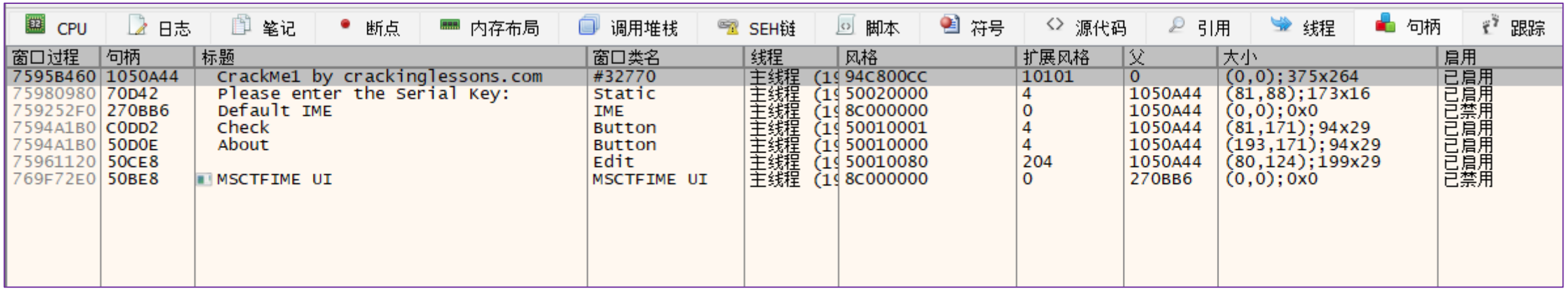
Windows消息

- Windows通过句柄来标识它所代表的对象。
- 在单击某个按钮时，Windows通过句柄来判断单击了哪一个按钮，然后发送相应的消息来通知程序

Windows消息

- 在句柄页签中，单击右键-刷新，即可查看被调试进程所有窗口的信息
- 因为更新该信息会占用较多资源，所以x64dbg不会自动刷新，需要手动刷新才能看到最新的信息

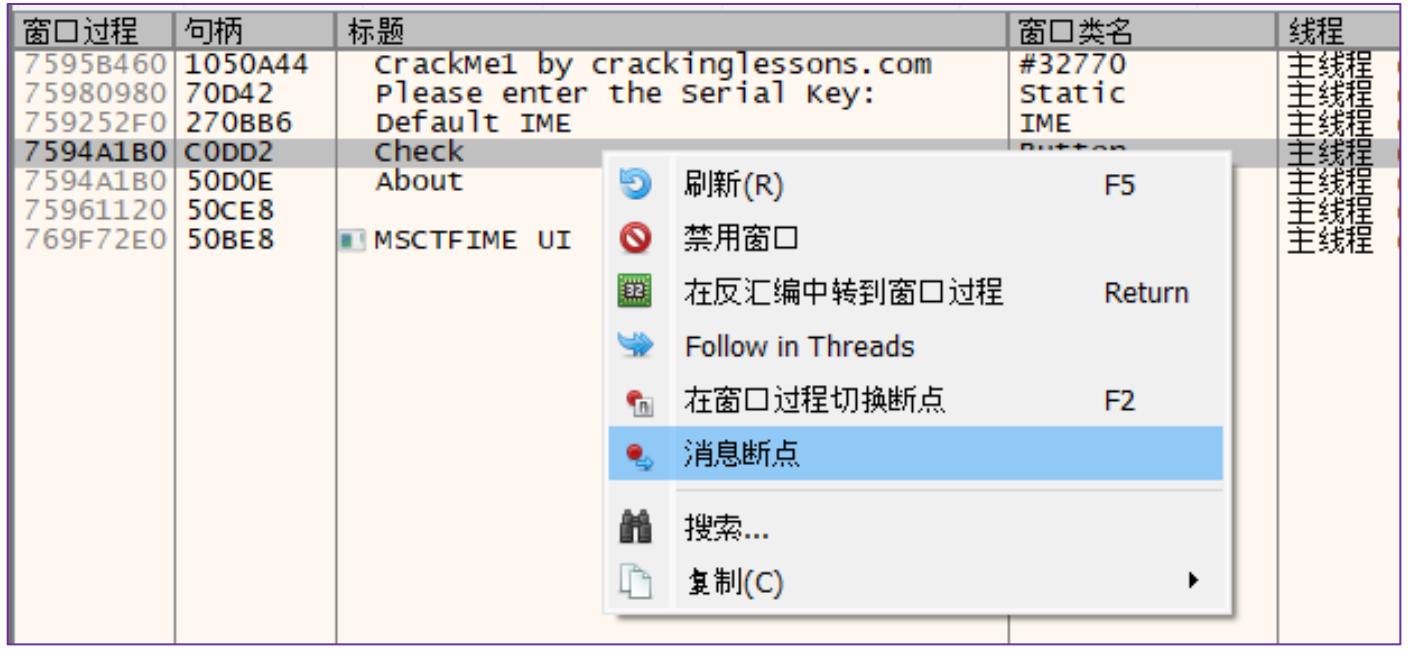
Windows消息



窗口过程	句柄	标题	窗口类名	线程	风格	扩展风格	父	大小	启用
7595B460	1050A44	CrackMe1 by crackinglessons.com	#32770	主线程 (19	94C800CC	10101	0	(0,0);375x264	已启用
75980980	70D42	Please enter the Serial Key:	Static	主线程 (19	50020000	4	1050A44	(81,88);173x16	已启用
759252F0	270BB6	Default IME	IME	主线程 (19	8C000000	0	1050A44	(0,0);0x0	已禁用
7594A1B0	C0DD2	Check	Button	主线程 (19	50010001	4	1050A44	(81,171);94x29	已启用
7594A1B0	50D0E	About	Button	主线程 (19	50010000	4	1050A44	(193,171);94x29	已启用
75961120	50CE8		Edit	主线程 (19	50010080	204	1050A44	(80,124);199x29	已启用
769F72E0	50BE8	MSCTFIME UI	MSCTFIME UI	主线程 (19	8C000000	0	270BB6	(0,0);0x0	已禁用

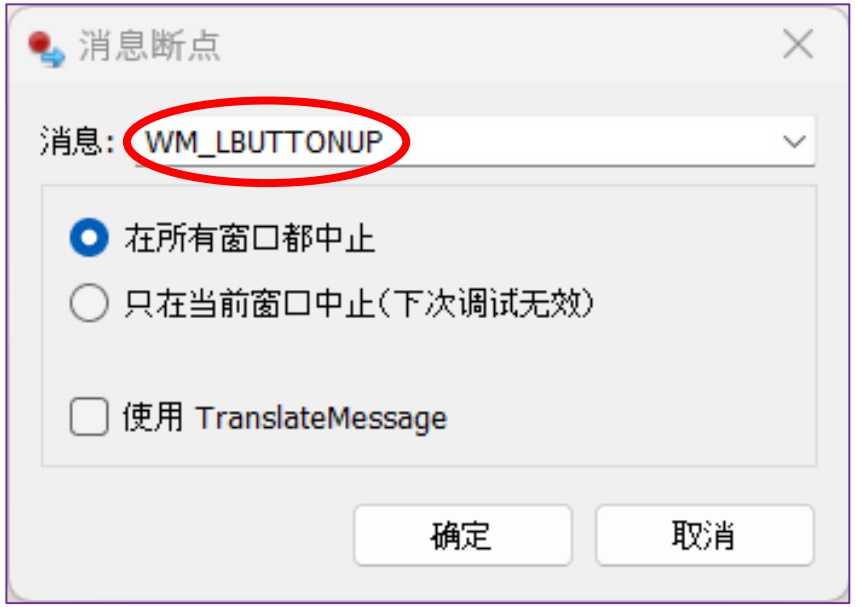
消息断点

右键单击Check按钮，在菜单中选择“消息断点"



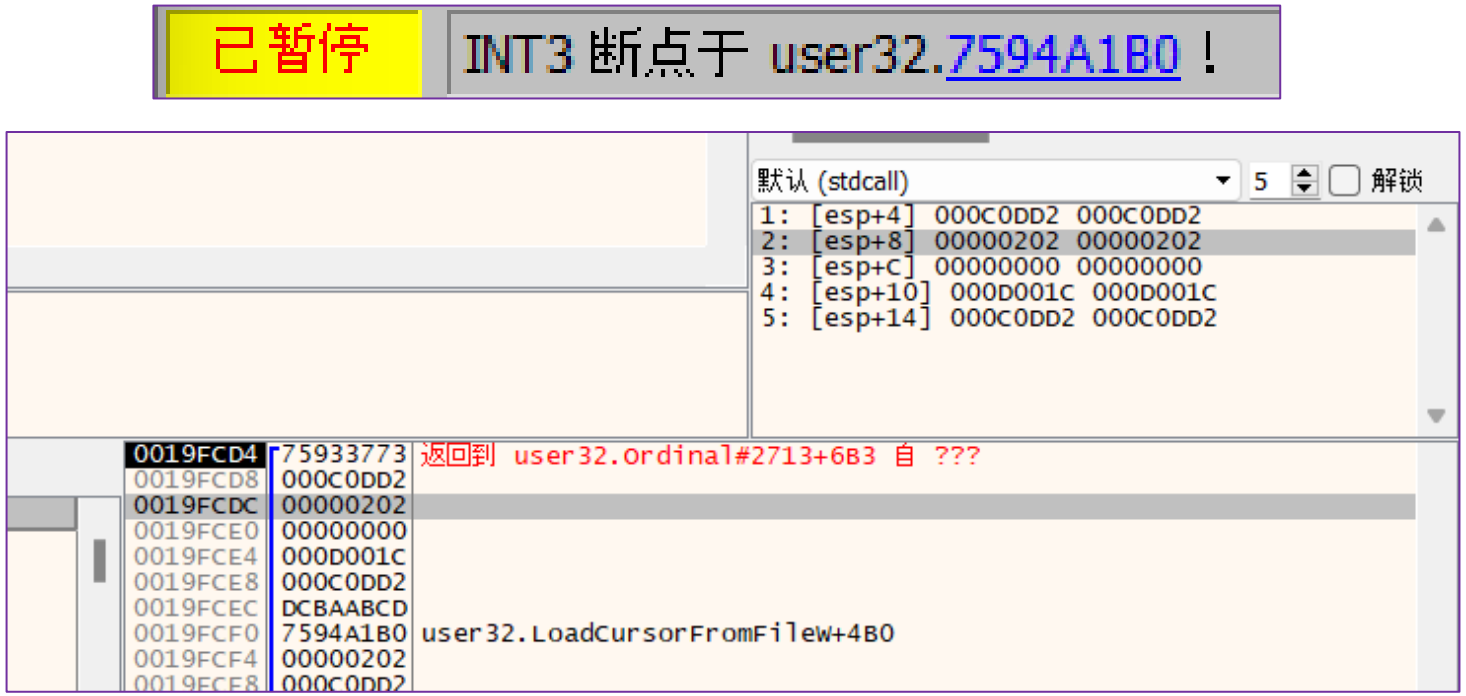
消息断点

- 在弹出的消息窗口中，输入“WM_COMMAND”消息。



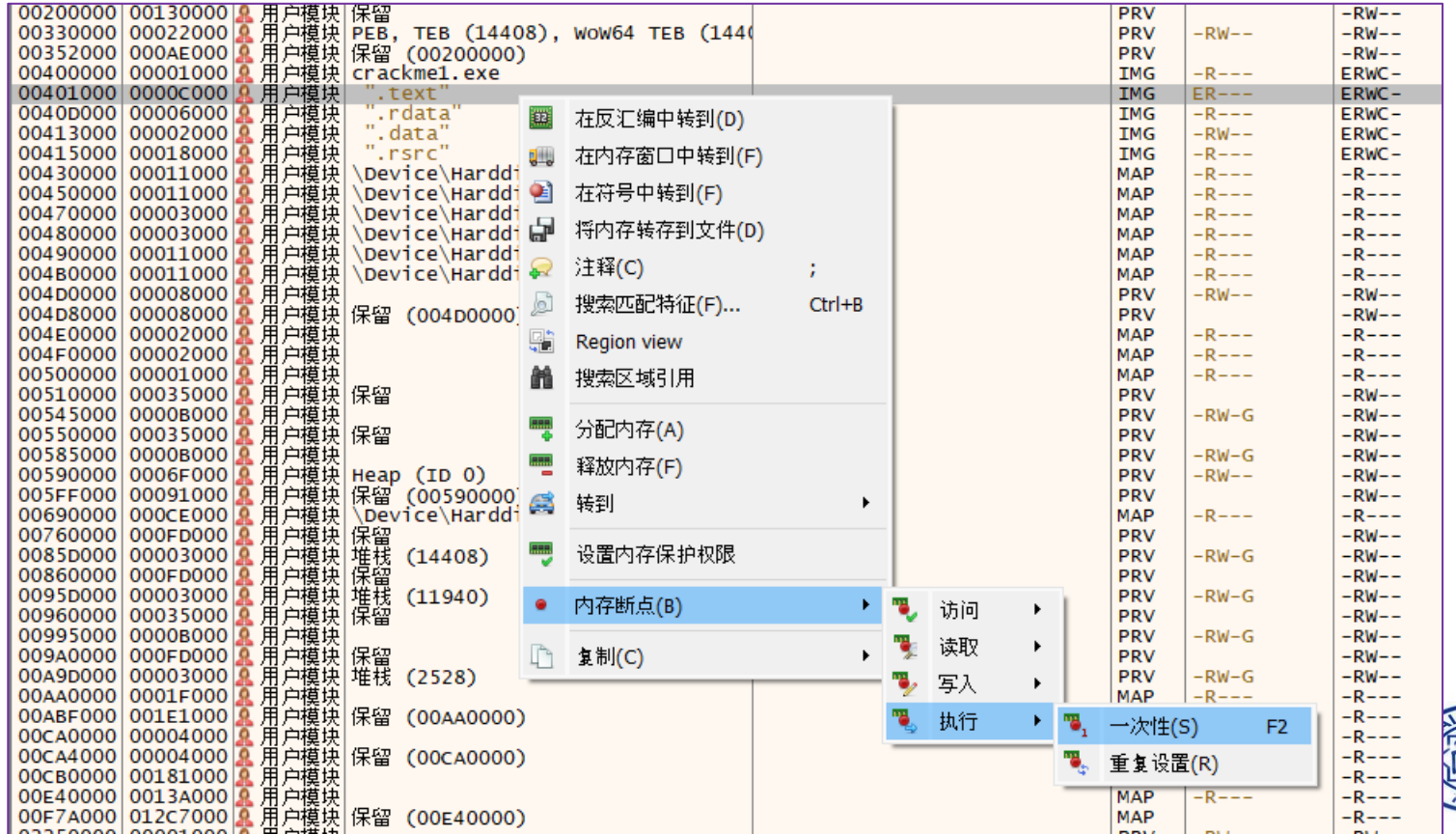
消息断点

- 点击Check按钮，程序中断在Windows系统代码



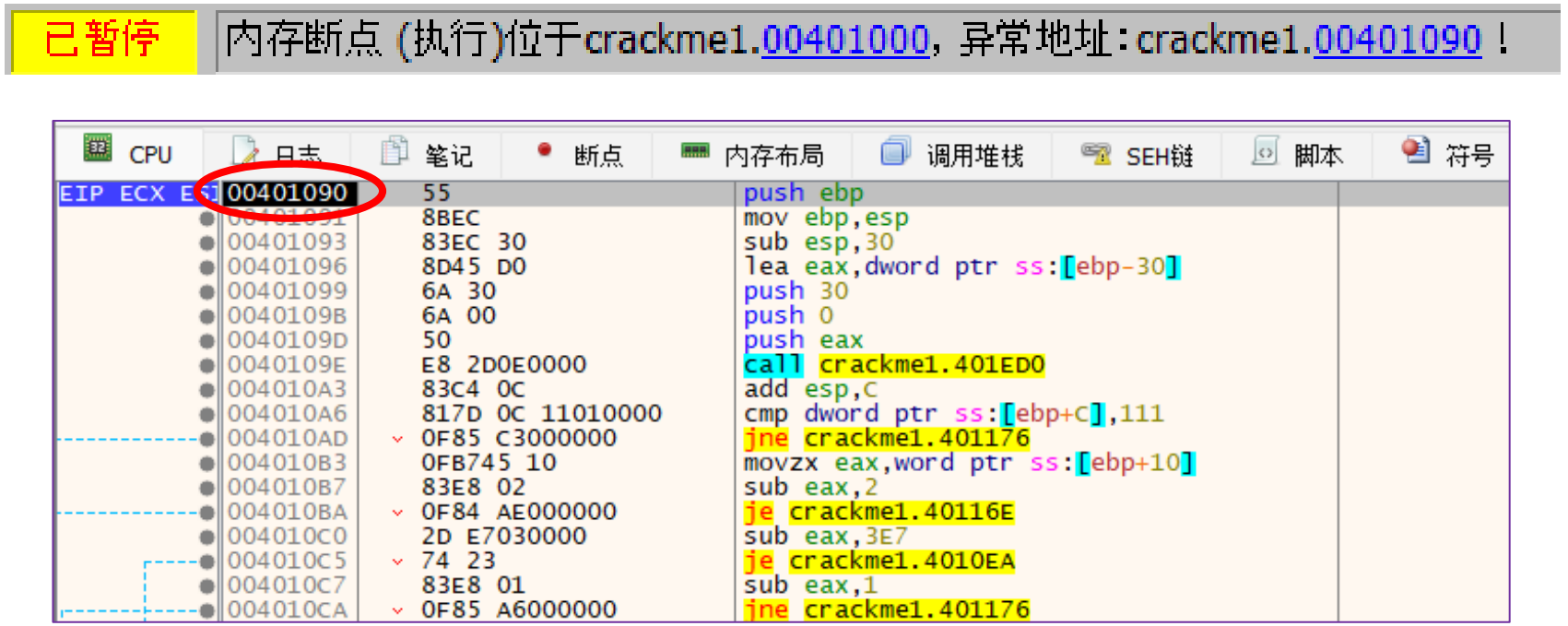
消息断点

- 打开内存窗口，对crackme1的.text段设置内存访问断点



消息断点

- F9运行程序，中断在0x00401090，crackme1的消息处理循环的位置



消息断点

- 浏览消息循环的代码，发现crackme1使用GetDlgItemTextA读取用户输入的字符串，并与一个常量字符串比较，然后根据结果弹出不同的MessageBoxA

```
004010EF    50              push eax
004010F0    68 E8030000     push 3E8
004010F5    FF35 A0424100   push dword ptr ds:[4142A0]
004010FB    FF15 10D14000   call dword ptr ds:[<GetDlgItemTextA>]
00401101    B9 D81A4100     mov ecx,crackme1.411AD8          411AD8:"cr4ckingL3ssons"
00401106    8D45 D0         lea eax,dword ptr ss:[ebp-30]
00401109    0F1F80 00000000 nop dword ptr ds:[eax],eax
00401110    8A10            mov dl,byte ptr ds:[eax]
00401112    3A11            cmp dl,byte ptr ds:[ecx]
00401114    75 1A           jne crackme1.401130
00401116    84D2            test dl,dl
00401118    74 12           je crackme1.40112C
0040111A    8A50 01         mov dl,byte ptr ds:[eax+1]
0040111D    3A51 01         cmp dl,byte ptr ds:[ecx+1]
00401120    75 0E           jne crackme1.401130
00401122    83C0 02         add eax,2
00401125    83C1 02         add ecx,2
00401128    84D2            test dl,dl
0040112A    75 E4           jne crackme1.401110
0040112C    33C0            xor eax,eax
0040112E    EB 05           jmp crackme1.401135
00401130    1BC0            sbb eax,eax
00401132    83C8 01         or eax,1
00401135    6A 00           push 0
00401137    85C0            test eax,eax
00401139    75 19           jne crackme1.401154
0040113B    68 E81A4100     push crackme1.411AE8             411AE8:"Congrats!"
00401140    68 F41A4100     push crackme1.411AF4             411AF4:"Well done!"
00401145    50              push eax
00401146    FF15 18D14000   call dword ptr ds:[<MessageBoxA>]
0040114C    33C0            xor eax,eax
0040114E    8BE5            mov esp,ebp
00401150    5D              pop ebp
00401151    C2 1000         ret 10
00401154    68 001B4100     push crackme1.411B00             411B00:"Sorry"
00401159    68 081B4100     push crackme1.411B08             411B08:"Wrong serial key. Try again."
0040115E    6A 00           push 0
00401160    FF15 18D14000   call dword ptr ds:[<MessageBoxA>]
00401166    33C0            xor eax,eax
00401168    8BE5            mov esp,ebp
0040116A    5D              pop ebp
```

条件断点

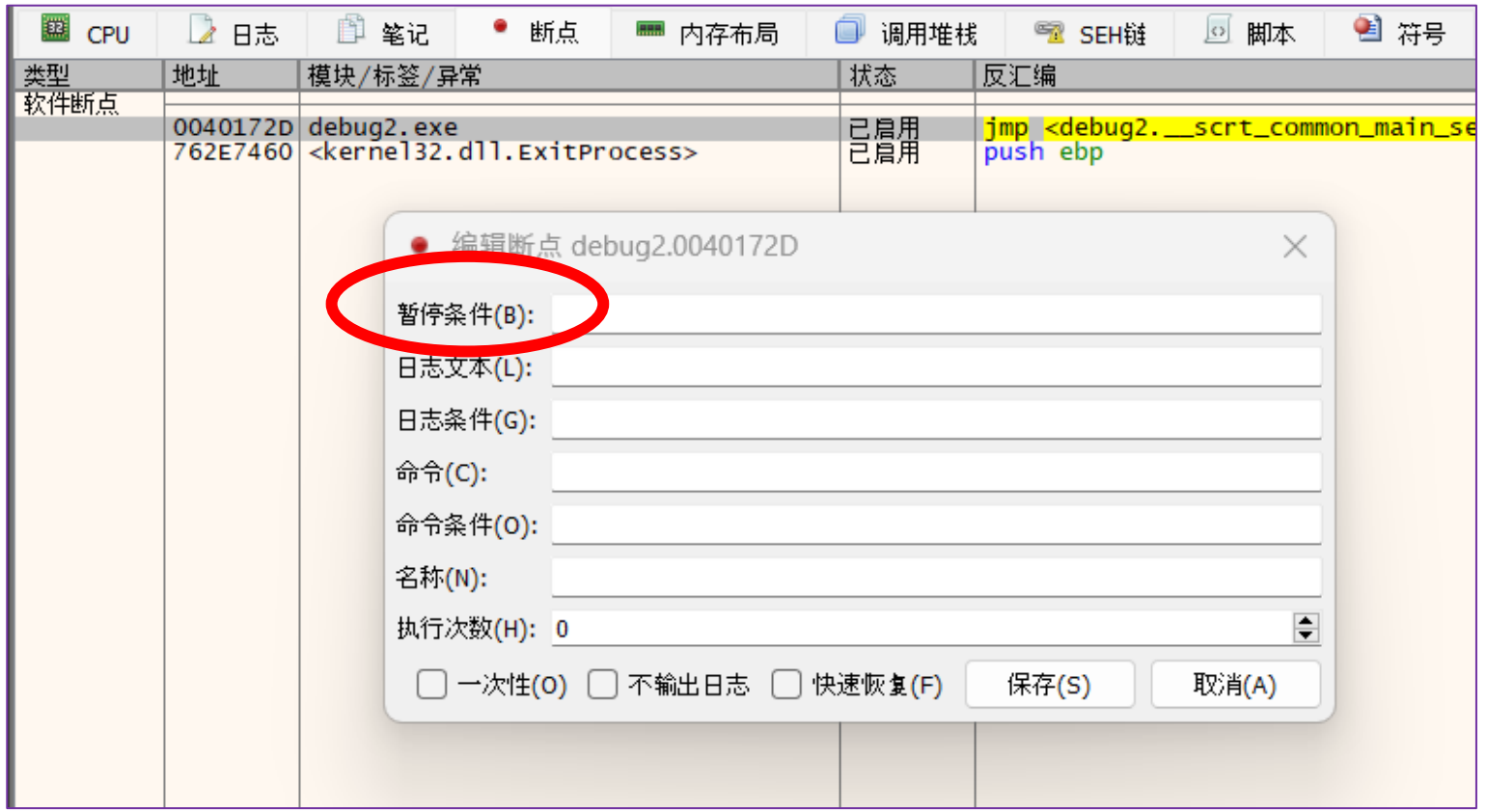
- 在调试过程中，在满足一定条件时断点才会触发，这类断点称为条件断点

  x64dbg的条件断点可以按寄存器、内存、消息等设断点

条件断点

- 条件断点是带有条件表达式的INT 3断点
  - 在INT3断点的基础上添加一个条件表达式
  - INT3断点中断程序执行
  - x64dbg判断表达式的值，如果是非零的值，则继续中断程序
  - 如果是0，恢复程序继续执行

条件断点


CPU
日志
笔记
断点
内存布局
调用堆栈
SEH链
脚本
符号
类型
地址
模块/标签/异常
状态
反汇编
软件断点
0040172D
debug2.exe
已启用
jmp <debug2.__scrt_common_main_se
762E7460
<kernel32.dll.ExitProcess>
已启用
push ebp
编辑断点 debug2.0040172D
暂停条件(B):
日志文本(L):
日志条件(G):
命令(C):
命令条件(O):
名称(N):
执行次数(H):
0
一次性(O)
不输出日志
快速恢复(F)
保存(S)
取消(A)

条件断点

- 命令行设置条件断点

如果安装了命令行插件，可以直接输入如下代码。

```
bp CreateFileA,[STRING [esp+4]]=="c:\\1212.txt"
```

如果是 Unicode 字符串，可以输入如下命令。

```
bp CreateFileW,[UNICODE [esp+4]]=="C:\\1212.txt"
```

条件断点

- 除了具有条件断点的功能
- 还能记录断点处函数表达式或参数的值
- 可以设置断点的次数，每次符合条件的中断，记数器的值就减一

多选题 10分

设置

x64dbg支持的断点类型？

A 软件断点

B 硬件断点

C 条件断点

D 内存访问断点

提交

南開大學 Nankai University

多选题 10分

设置

下面哪些断点会使程序执行速度变慢？

A 软件断点

B 硬件断点

C 条件断点

D 内存访问断点

提交

汇编语言与逆向技术

第10章 动态逆向分析技术

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2024-2025学年

本章知识点

- 动态逆向分析
- x64dbg加载程序
- x64dbg的界面
- Memory Map
  - 难点：重定位
- 执行指令
  - 难点：执行到返回、执行到用户代码
- 断点 Breakpoint
  - 难点：软件断点、硬件断点、内存断点、条件断点、消息断点