



南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异



# 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



允公允能 日新月异

## 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



南开大学  
Nankai University



南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异



# 动态逆向分析



允公允能 日新月异

# 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



南开大学  
Nankai University



允公允能 日新月异

# 用户模式调试器

- 调试用户模式的应用程序
- CPU **Ring3**级
- x64dbg



南开大学  
Nankai University



允公允能 日新月异

# 内核模式调试器

- 调试操作系统内核程序
- CPU **Ring0**级
- WinDbg



南开大学  
Nankai University



允公允能 日新月异

# 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试器之一
- 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



南开大学  
Nankai University



允公允能 日新月异

## 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



南开大学  
Nankai University



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, P.R. CHINA 1919

允公允能 日新月異



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





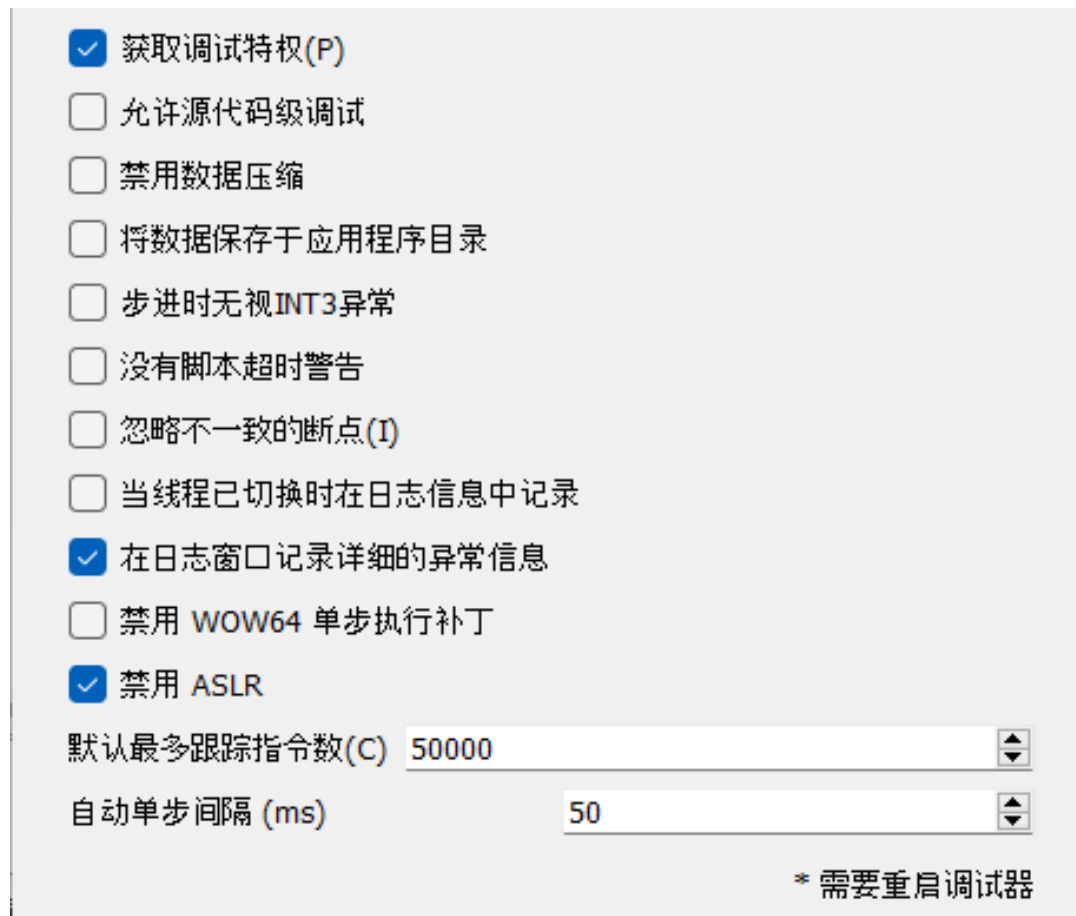
## x32dbg与x64dbg

- 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





南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异



x64dbg加载程序



允公允能 日新月异

# 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）



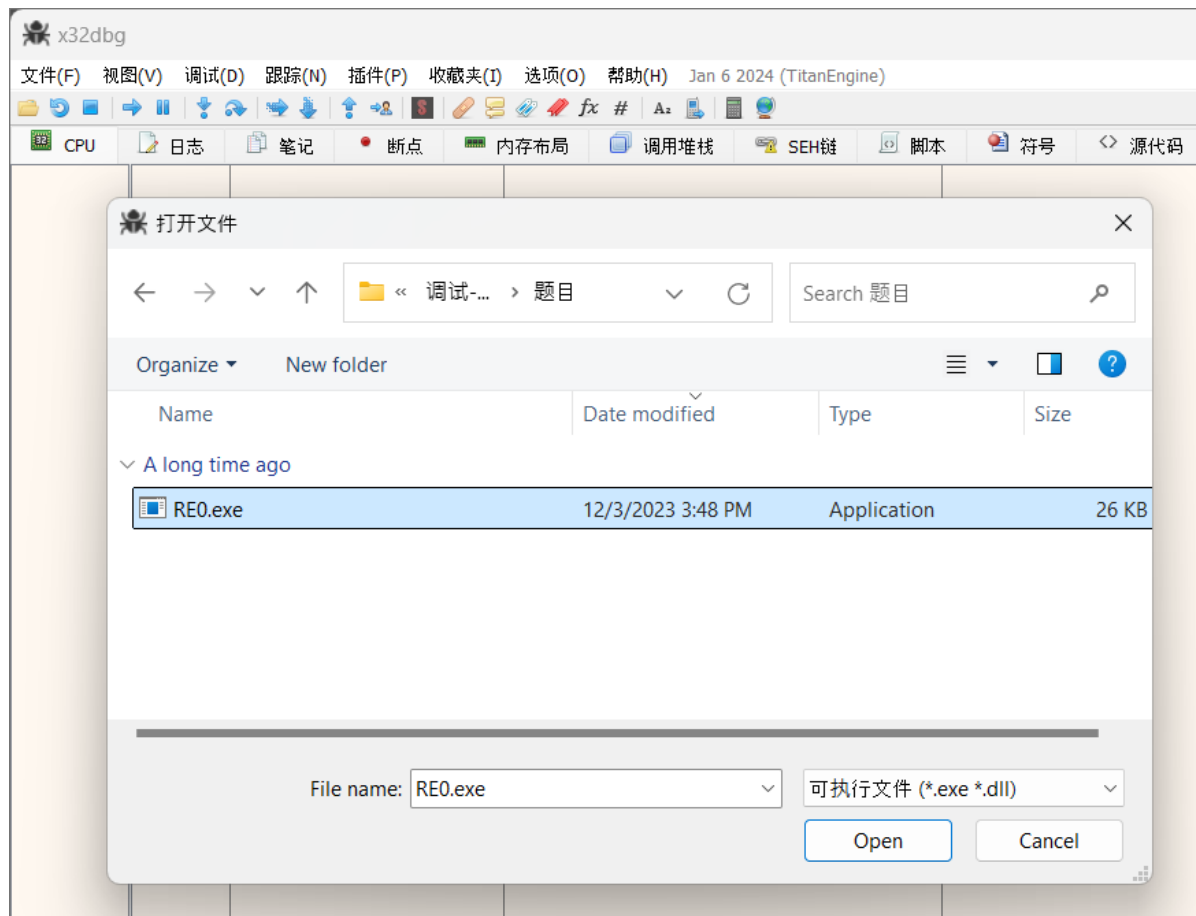
南开大学  
Nankai University





允公允能 日新月异

# 直接加载程序







允公允能 日新月异

## 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



南开大学  
Nankai University



允公允能 日新月异

# 附加进程

附加			
PID	名称	标题	路径
10392	smartscreen		C:\windows\system32\smartscreen
6744	RuntimeBroker		C:\windows\System32\RuntimeBroker
9276	msedge		C:\Program Files (x86)\Microsoft Edge
9448	msedge		C:\Program Files (x86)\Microsoft Edge
3104	backgroundTaskHost		C:\windows\system32\backgroundTaskHost
20516	msedge		C:\Program Files (x86)\Microsoft Edge
1176	msedge		C:\Program Files (x86)\Microsoft Edge
17224	msedge		C:\Program Files (x86)\Microsoft Edge
19168	msedge		C:\Program Files (x86)\Microsoft Edge
7876	msedge	Chrome_widgetwin_0	C:\Program Files (x86)\Microsoft Edge
18640	msedge	crashpad_SessionEndwatcher	C:\Program Files (x86)\Microsoft Edge
14412	msedge	Copilot (preview)	C:\Program Files (x86)\Microsoft Edge
4824	backgroundTaskHost		C:\windows\system32\backgroundTaskHost
13424	ai		C:\Program Files\Microsoft Office\Office16\ai
1560	UserOOBEBroker		C:\windows\System32\oobe\UserOOBEBroker
11304	svchost		C:\windows\System32\svchost
17852	SystemSettings	Settings	C:\windows\ImmersiveControlPanel\SystemSettings
14304	ShellExperienceHost		C:\windows\SystemApps\ShellExperienceHost
16656	WidgetService		C:\Program Files\WindowsApps\Microsoft.Windows.Common-Theme
9348	ApplicationFrameHost	Settings	C:\windows\System32\ApplicationFrameHost
5860	ai		C:\Program Files\Microsoft Office\Office16\ai
4636	WINWORD	x64dbg调试器入门.docx - word	C:\Program Files\Microsoft Office\Office16\WINWORD
10472	svchost		C:\windows\System32\svchost

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)





允公允能 日新月异

## 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；





允公允能 日新月异

## 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。



南开大学  
Nankai University

# 内存访问断点

The screenshot shows the Memory window in Visual Studio, displaying a memory dump for the process debug2.exe. The memory dump is organized into columns: Address, Size, Displacement, Page Information, Content, Type, Page Protection, and Initial Protection. A context menu is open over the memory dump, showing options for navigating and managing memory. The '内存断点(B)' (Memory Breakpoint) option is selected, and a sub-menu is open showing options for setting the breakpoint: '访问' (Access), '读取' (Read), '写入' (Write), and '执行' (Execute). The '执行' option is further expanded, showing '一次性(S)' (One-time) and '重复设置(R)' (Repeat setting).

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-G	-RW--
007F5000	00008000	用户模块	保留		PRV	-RW--	-RW--
00800000	00017000	用户模块	PEB, TEB (143		PRV	-RW--	-RW--
00817000	00012000	用户模块	保留 (0080000		PRV	-RW--	-RW--
00829000	001D7000	用户模块	保留		PRV	-RW--	-RW--
00A00000	000FB000	用户模块	保留		PRV	-RW--	-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			MAP	-R---	-R---
00C30000	00035000	用户模块	保留		MAP	-R---	-R---
00C65000	00008000	用户模块			MAP	-R---	-R---
00C70000	00008000	用户模块			MAP	-R---	-R---
00C78000	00008000	用户模块	保留 (00C70000)		MAP	-R---	-R---



允公允能 日新月异

# 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。



南开大学  
Nankai University



允公允能 日新月异

## 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



南开大学  
Nankai University



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







南開大學

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月異

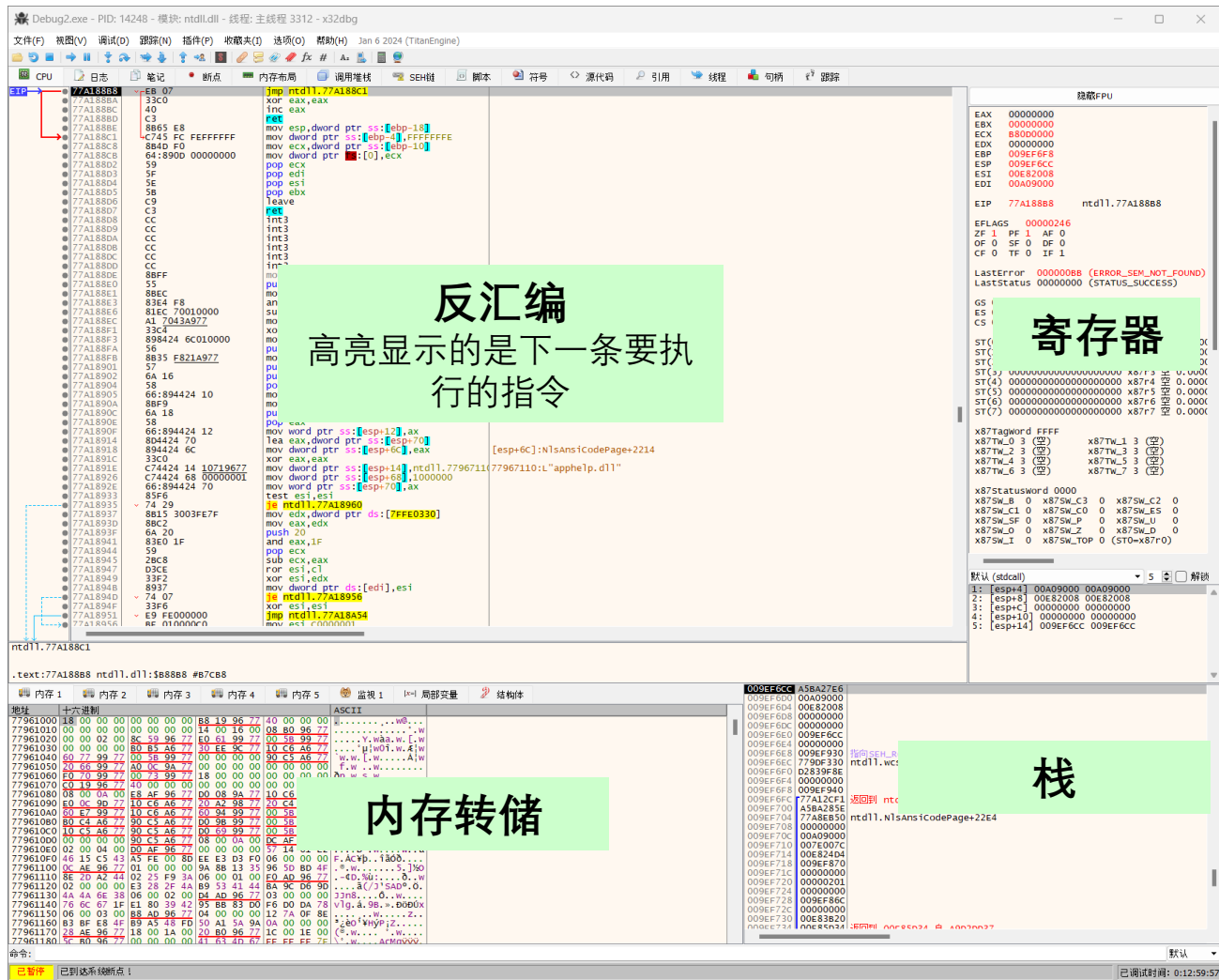


x64dbg的界面



允公允能 日新月异

# x64dbg界面



南开大学  
Nankai University



## 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

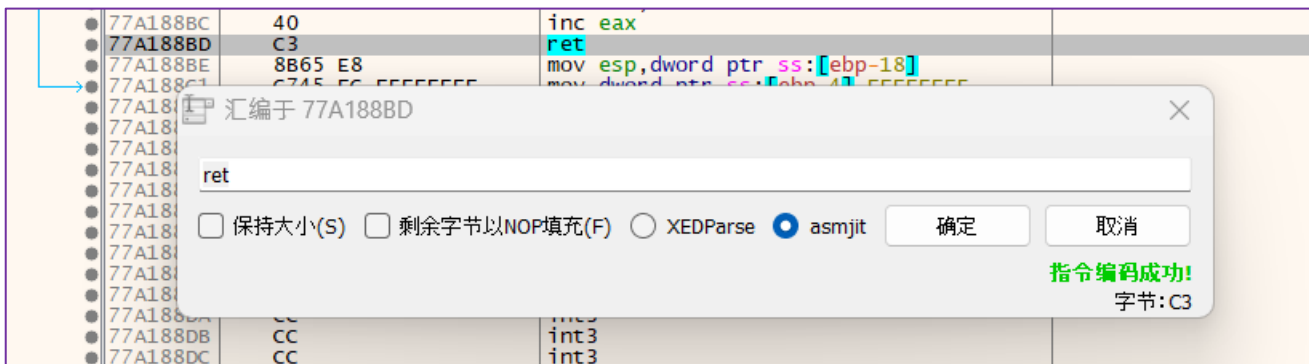
- Address列:显示相对地址，再次双击返回标准地址模式。
- Hex dump列:设置或取消无条件断点，对应的快捷键是“F2”键。

EIP	Address	Hex Dump	Disassembly
77A188B8	EB 07	33C0	jmp ntdll.77A188C1
77A188BA	40	8B65 E8	xor eax,eax
77A188BC	C3	C745 FC FFFFFFFF	inc eax
77A188BD		8B4D F0	ret
77A188BE		64:890D 00000000	mov esp,dword ptr ss:[ebp-18]
77A188C1			mov dword ptr ss:[ebp-4],FFFFFFFE
77A188C8			mov ecx,dword ptr ss:[ebp-10]
77A188CB			mov dword ptr [0],ecx
77A188D2			pop ecx
77A188D3			pop edi
77A188D4			pop esi
77A188D5			pop ebx
77A188D6			leave
77A188D7			ret

EIP	Address	Hex Dump	Disassembly
77A188B8	EB 07	33C0	jmp ntdll.77A188C1
77A188BA	40	8B65 E8	xor eax,eax
77A188BC	C3	C745 FC FFFFFFFF	inc eax
77A188BD		8B4D F0	ret
77A188BE		64:890D 00000000	mov esp,dword ptr ss:[ebp-18]
77A188C1			mov dword ptr ss:[ebp-4],FFFFFFFE
77A188C8			mov ecx,dword ptr ss:[ebp-10]
77A188CB			mov dword ptr [0],ecx
77A188D2			pop ecx
77A188D3			pop edi
77A188D4			pop esi
77A188D5			pop ebx
77A188D6			leave
77A188D7			ret

## 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

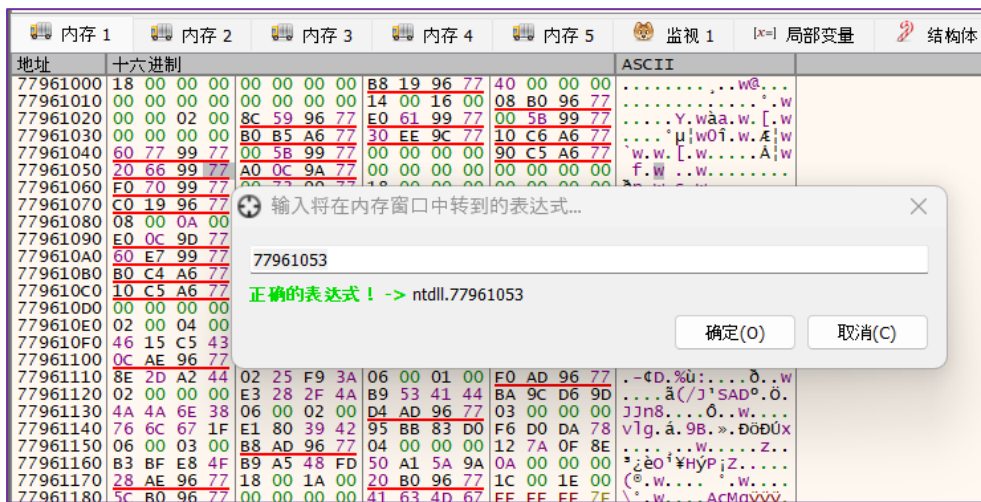
- 在进行动态跟踪时，信息窗口(Information window)显示与指令相关的各寄存器的值、API函数调用提示和跳转提示等信息。

	004014FE	8845 DC	mov byte ptr ss:[ebp-24],al	
EIP →	00401501	A1 30614000	mov eax,dword ptr ds:[<_s crt_current_	exe_common.inl:245
	00401506	33C9	xor ecx,ecx	ecx:"Actx "
	00401508	41	inc ecx	ecx:"Actx "
	00401509	3BC1	cmp eax,ecx	ecx:"Actx "
	0040150B	✓ 0F84 2F010000	je debug2.401640	
	00401511	85C0	test eax,eax	exe_common.inl:249
	00401513	✓ 75 49	jne debug2.40155E	
	00401515	890D 30614000	mov dword ptr ds:[<_s crt_current_native	exe_common.inl:251 ecx:"Actx "
eax=0				
dword ptr ds:[00406130 <debug2.__s crt_current_native_startup_state>]=0				
.text:00401501 debug2.exe:\$1501 #901				



# 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+G**”快捷键，打开地址窗口，输入地址





# 数据窗口

内存 1内存 2内存 3内存 4内存 5监视 1[x=] 局部变量结构体

地址	十六进制	ASCII
77961000	18 00 00 00	.....w@...
77961010	00 00 00 00	.....°.w
77961020	00 00 02 00	.....Y.wàa.w.[.w
77961030	00 00 00 00	.....µ w0î.w.Æ w
77961040	60 77 99 77	.....F.....
77961050	20 66 99 77	.....F.....
77961060	F0 70 99 77	.....F.....
77961070	C0 19 96 77	.....F.....
77961080	08 00 0A 00	.....F.....
77961090	E0 0C 9D 77	.....F.....
779610A0	60 E7 99 77	.....F.....
779610B0	B0 C4 A6 77	.....F.....
779610C0	10 C5 A6 77	.....F.....
779610D0	00 00 00 00	.....F.....
779610E0	02 00 04 00	.....F.....
779610F0	46 15 C5 43	F.À¢þ..îãóð....
77961100	0C AE 96 77	..°.w.....5.]]%0
77961110	8E 2D A2 44	.-¢D.%ù:....ð..w
77961120	02 00 00 00	....ã(/J¹SAD°.Ö.
77961130	4A 4A 6E 38	jJn8....ô..w....
77961140	76 6C 67 1F	vlg.á.9B.»..ðöðÚx
77961150	06 00 03 00	.....w.....Z..
77961160	B3 BF E8 4F	³zè0ï¥HýPjZ....
77961170	28 AE 96 77	(°.w.....°.w....
77961180	5C B0 96 77	\°.w.....AcMqVvVv

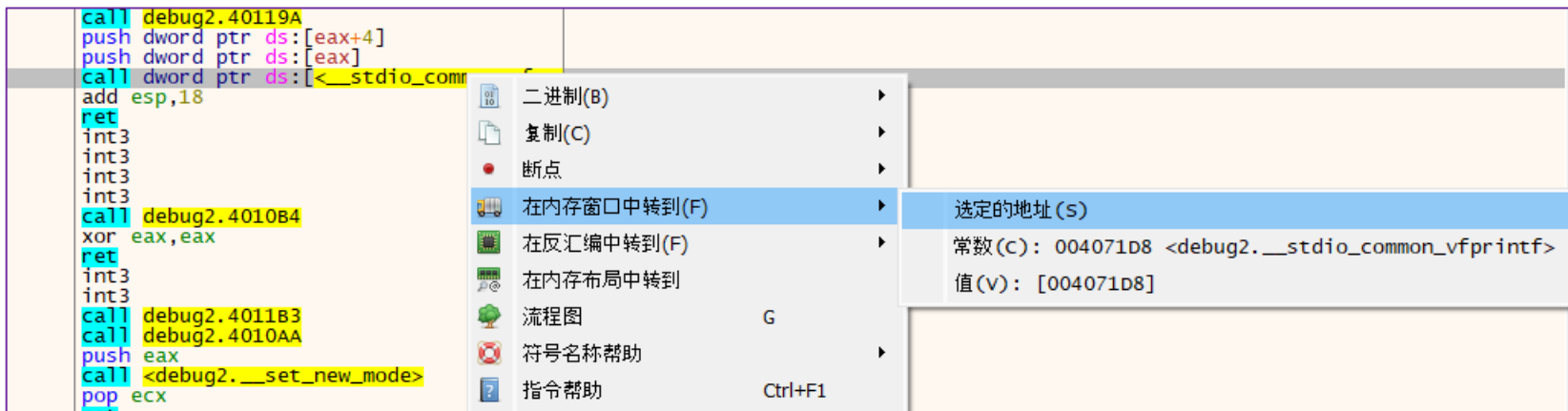
⊕ 输入要同步到的表达式...

无效的表达式.....

确定(O) 取消(C)



# 在数据窗口中跟踪



内存 1	内存 2	内存 3	内存 4	内存 5	监视 1	[x=] 局部变量
地址	十六进制	十六进制	十六进制	十六进制	ASCII	
0040149A	FF 15 D8 71	40 00 83 C4	18 C3 CC CC	CC CC E8 07	y.0q@..A.Aiiiiè.	
004014AA	FC FF FF 33	C0 C3 CC CC	E8 FC FC FF	FF E8 EE FB	üyy3AAiièüüyyèiü	
004014BA	FF FF 50 E8	68 14 00 00	59 C3 CC CC	CC CC E8 09	yyPèh...yAiiiiè.	
004014CA	FD FF FF E9	02 00 00 00	CC CC 6A 14	68 08 57 40	yyé....ii.j.h.w@	
004014DA	00 E8 8E FB	FF FF 6A 01	E8 81 FC FF	FF 59 84 C0	.è.üyyj.è.üyyY.A	
004014EA	0F 84 50 01	00 00 32 DB	88 5D E7 83	65 FC 00 E8	..P...20.]ç.eü.è	
004014FA	8D FC FF FF	88 45 DC A1	30 61 40 00	33 C9 41 3B	.üyy.EÜ;0a@.3ÉA;	
0040150A	C1 0F 84 2F	01 00 00 85	C0 75 49 89	0D 30 61 40	A../...AuI..0a@	
0040151A	00 68 18 46	40 00 68 0C	43 40 00 E8	C4 13 00 00	.h.F@.h.C@.èÄ...	
0040152A	59 59 85 C0	74 11 C7 45	FC FE FF FF	FF B8 FF 00	YY.At.çEüpyyy.y.	
0040153A	00 00 E9 EF	00 00 00 68	08 42 40 00	68 00 40 40	..éi...h.B@.h.@@	
0040154A	00 E8 98 13	00 00 59 59	C7 05 30 61	40 00 02 00	.è....YYÇ.0a@...	



# 数据编辑

The screenshot shows the Immunity Debugger interface. The assembly window displays the instruction at address 0040149A: `jmp <debug2.____scr..._main_seh>`. A context menu is open over this instruction, showing options like "二进制编辑(T)", "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)", "断点(B)", "搜索匹配特征(F)...", "搜索引用(R)", "与表达式同步(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)", and "反汇编(D)". The assembly code on the right includes instructions like `mov byte ptr ss:[ebp-19],b1` and `mov dword ptr ds:[<__scr..._current`.





允公允能 日新月异

## 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



南开大学  
Nankai University



## 寄存器窗口

- 通过双击一个寄存器值，修改该寄存器的数值

隐藏FPU		
<u>EAX</u>	00000000	
EBX	00726000	
<u>ECX</u>	00900000	"Actx "
<u>EDX</u>	00900000	"Actx "
EBP	008FFC7C	
<u>ESP</u>	008FFC48	
ESI	00401014	<debug2.optionalHeader
EDI	00401014	<debug2.optionalHeader
EIP	00401501	debug2.00401501



允公允能 日新月异

## 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



南开大学  
Nankai University

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



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



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







南開大學

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月異



# Memory Map

# Memory Map

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 (14660)		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\		MAP	-R---	-R---
00950000	00011000	用户模块	\Device\Harddiskvolume3\windows\		MAP	-R---	-R---
00970000	00003000	用户模块	\Device\Harddiskvolume3\windows\		MAP	-R---	-R---
00980000	00011000	用户模块	\Device\Harddiskvolume3\windows\		MAP	-R---	-R---
009A0000	00003000	用户模块	\Device\Harddiskvolume3\windows\		MAP	-R---	-R---
009B0000	00003000	用户模块	\Device\Harddiskvolume3\windows\		MAP	-R---	-R---
009C0000	000CE000	用户模块	\Device\Harddiskvolume3\windows\		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\		MAP	-R---	-R---
00AC0000	00011000	用户模块	\Device\Harddiskvolume3\windows\		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--



允公允能 日新月异

# 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



南开大学  
Nankai University



允公允能 日新月异

## 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）



南开大学  
Nankai University



允公允能 日新月异

# 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



南开大学  
Nankai University



# 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



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      1mov eax, dword_40CF60
```

- ☒ A [ebp + var\_8]
- ☒ B [ebp + var\_4]
- ☒ C jnz loc\_0040120
- ☐ D dword\_40CF60

提交







允公允能 日新月异

## 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



南开大学  
Nankai University



允公允能 日新月异

# .reloc节

2020/2/26 11:34 应用程序扩展 110 KB

PEiD v0.95

File: C:\Windows\System32\user32.dll ... KB

Entrypoint: 000414D0 EP Section: .text > KB

File Offset: 000408D0 First Bytes: 8B,FF,55,8B > KB

Linker Info: 14.13 Subsystem: Win32 GUI > KB

Nothing found [Overlay] \* KB

Multi Scan Task Viewer Options About Exit KB

☒ Stay on top ?? -> KB

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





允公允能 日新月异

## 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。



南开大学  
Nankai University



# 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 (14660)		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--

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

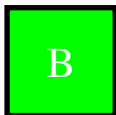


以下哪些是线程私有的？



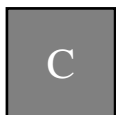
A

栈



B

寄存器



C

内存空间



D

代码

提交







南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异

执行指令



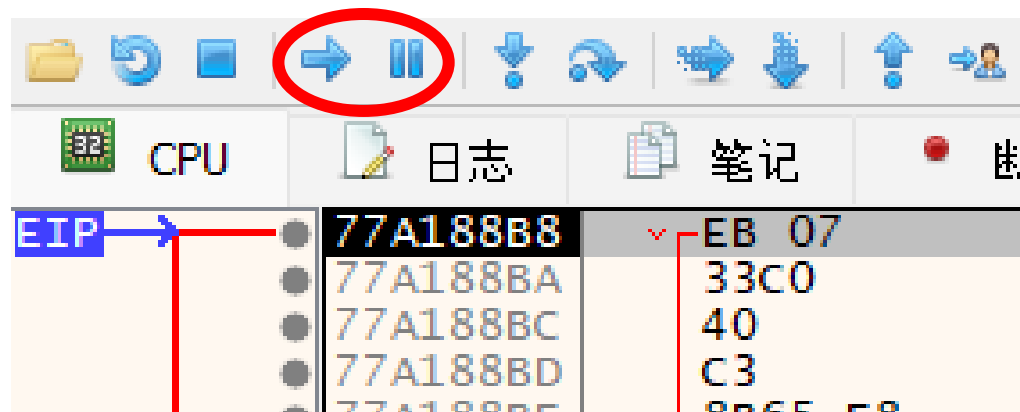
Function	Menu	Hotkey	Button
Run/Play	Debug ▶ Run	F9	
Pause	Debug ▶ Pause	F12	
Run to selection	Breakpoint ▶ Run to Selection	F4	
Run until return	Debug ▶ Execute till Return	CTRL-F9	
Run until user code	Debug ▶ Execute till User Code	ALT-F9	
Single-step/step-into	Debug ▶ Step Into	F7	
Step-over	Debug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





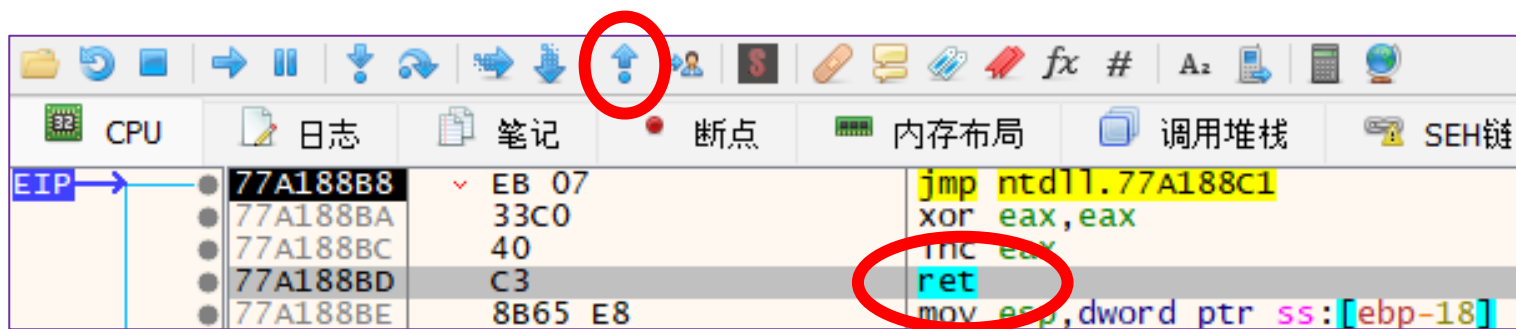
## 运行和暂停

- 运行调试应用程序（F9）
- 暂停调试程序（F12）



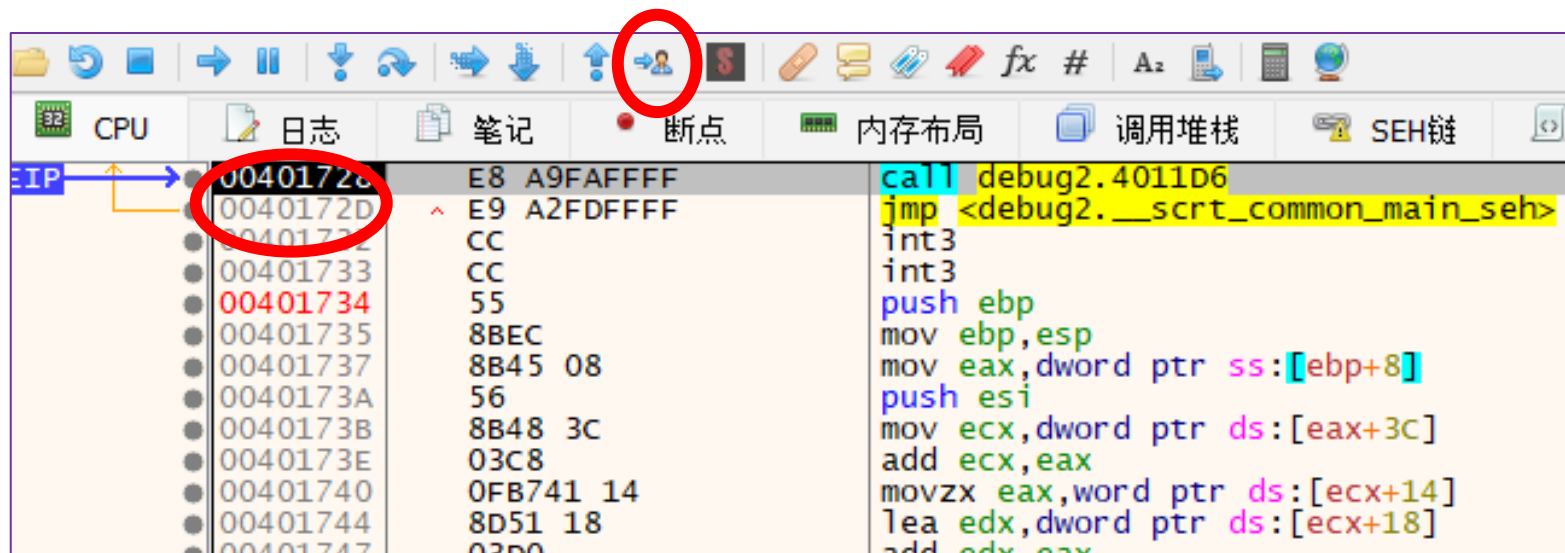
## 执行直到返回

- “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



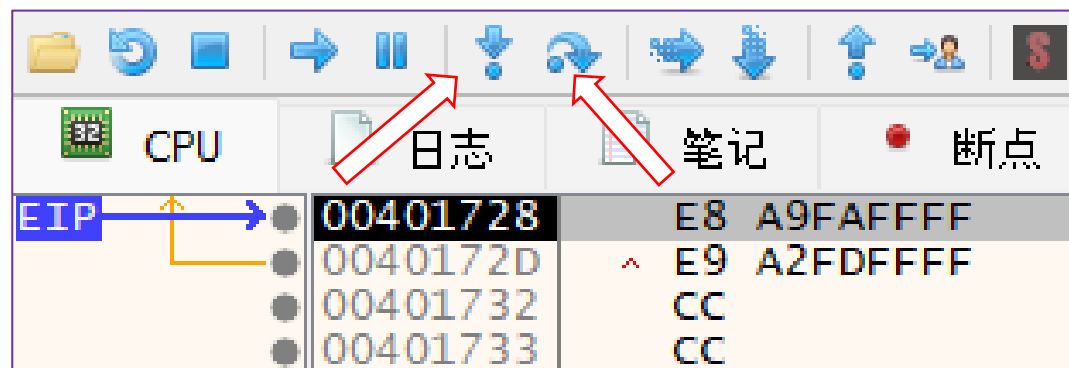
## 执行到用户代码

- “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。



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





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



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



南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异

断点



允公允能 日新月异

## 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



南开大学  
Nankai University



允公允能 日新月异














# 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		



允公允能 日新月异

## 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。



南开大学  
Nankai University





# 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





允公允能 日新月异

# 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



南开大学  
Nankai University



## 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
```





允公允能 日新月异

# 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



南开大学  
Nankai University

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





允公允能 日新月异

# 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



南开大学  
Nankai University



允公允能 日新月异

# 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

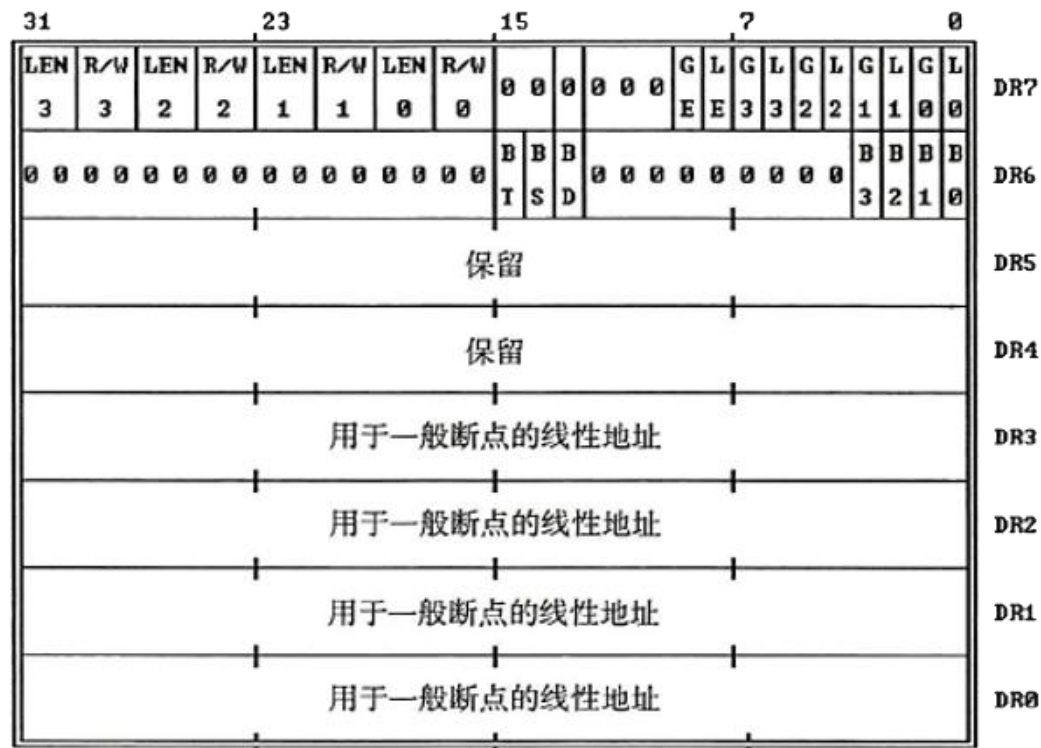


南开大学  
Nankai University





# 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DR<sub>x</sub>寄存器，包括后面介绍的硬件读取/写入断点





允公允能 日新月异

## 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”



南开大学  
Nankai University



# 硬件执行断点

●	004014E0	6A 01	push 1	exe_common.inl:237
●	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	mon.inl:240
●	004014F2	885D E7	mov byte p	
●	004014F5	8365 FC 00	and dword	
●	004014F9	E8 8DFCFFFF	call debug	eadInit
●	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	mon.inl:249
●	00401513	✓ 75 49	jne debug2	

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)





允公允能 日新月异

## 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



南开大学  
Nankai University



允公允能 日新月异

## 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

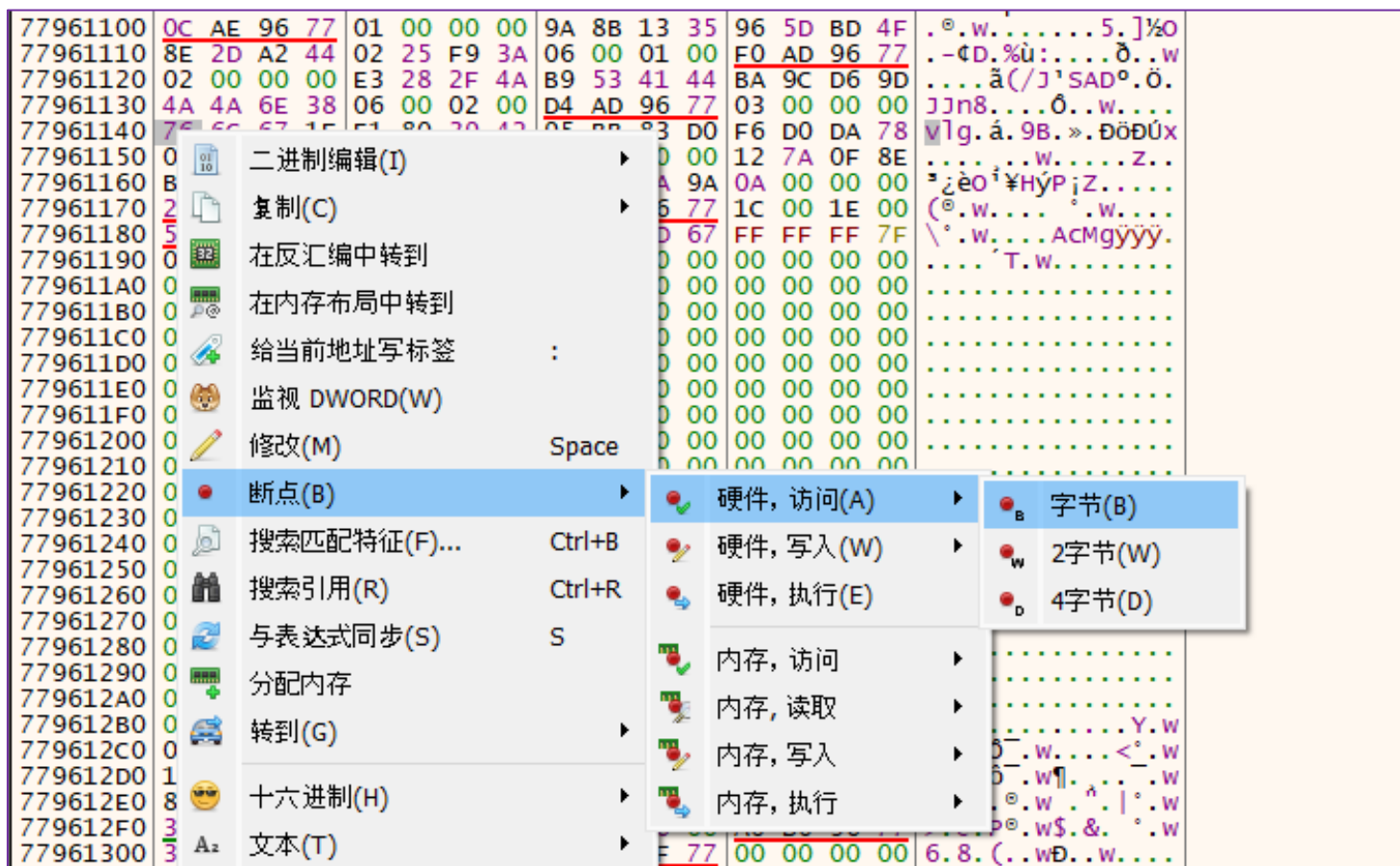


南开大学  
Nankai University



## 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





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





允公允能 日新月异

## 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



南开大学  
Nankai University



允公允能 日新月异

## 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



南开大学  
Nankai University



# 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	.....w@..
77961010	00 00 00 00 00 00 00 00 14 00 16 00 08 B0 96 77	.....w
77961020	00 00 02 00 8C 59 96 77 E0 61 99 77 00 5B 99 77	...Y.waa.w.[.w
77961030	00 00 00 00 B0 B5 A6 77 30 EE 9C 77 10 C6 A6 77	...µ w0i.w.Æ w
77961040	00 00 00 00 00 00 00 00 90 C5 A6 77	w.w.[.w.....A w
77961050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	f.w..w.....
77961060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	ðp.w.s.w.....
77961070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	A..w@..
77961080	A 77 10 C6 A6 77	...è..wÐ..w.Æ w
77961090	8 77 20 C4 A6 77	à..w.Æ w ¢.w A w
779610A0	9 77 00 5B 99 77	..c.w.Æ w'.w.[.w
779610B0	9 77 00 5B 99 77	°A w.A wÐ..w.[.w
779610C0	9 77 00 5B 99 77	.A w.A wÐi.w.[.w
779610D0	A 00 DC AF 96 77	...A w...Û..w
779610E0	0 00 57 14 01 E2	...Ð..w...W..â
779610F0	3 F0 06 00 00 00	F.ÄC¥p...iäóð..
77961100	3 35 96 5D BD 4F	.@.w.....5.]%O
77961110		..%û:....ð..w
77961120		ã(/J'SAD°.ð.
77961130		...ð..w...
77961140		ã.9B.»..ÐðÐÚx
77961150		..w.....Z..
77961160		i¥HyPjZ.....
77961170		
77961180		
77961190		
779611A0		
779611B0		
779611C0		
779611D0		
779611E0		
779611F0	Az 文本(T)	
77961200		
77961210	42 整数(I)	
77961220		





允公允能 日新月异

## 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



南开大学  
Nankai University



允公允能 日新月异

## 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



南开大学  
Nankai University





允公允能 日新月异

# 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



南开大学  
Nankai University
















允公允能 日新月异

# 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\s																				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
<div><div> 转到断点</div><div> 删除(R) Del</div><div> 禁用 Space</div><div> 编辑(E) Shift+F2</div></div>						



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

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





允公允能 日新月异

## 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



南开大学  
Nankai University





允公允能 日新月异

# 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。



南开大学  
Nankai University



# 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


<div> <div>CPU</div> <div>日志</div> <div>笔记</div> <div>断点</div> <div>内存布局</div> <div>调用堆栈</div> <div>SEH链</div> <div>脚本</div> <div>符号</div> <div>源代码</div> <div>引用</div> <div>线程</div> <div>句柄</div> <div>跟踪</div> </div>										
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	禁用
75980980	70D42	Please enter the serial key:	Static	主线程	(19 50020000	4	1050A44	(81,88);173x16	已禁用	禁用
759252F0	270BB6	Default IME	IME	主线程	(19 8C000000	0	1050A44	(0,0);0x0	已禁用	禁用
7594A1B0	C0DD2	check	Button	主线程	(19 50010001	4	1050A44	(81,171);94x29	已禁用	禁用
7594A1B0	50D0E	About	Button	主线程	(19 50010000	4	1050A44	(193,171);94x29	已禁用	禁用
75961120	50CE8		Edit	主线程	(19 50010080	204	1050A44	(80,124);199x29	已禁用	禁用
769F72E0	50BE8	MSCTFIME UI	MSCTFIME UI	主线程	(19 8C000000	0	270BB6	(0,0);0x0	已禁用	禁用



# 消息断点

右键单击Check按钮，在菜单中选择“消息断点”

窗口过程	句柄	标题	窗口类名	线程
7595B460	1050A44	CrackMe1 by crackinglessons.com	#32770	主线程
75980980	70D42	Please enter the serial key:	Static	主线程
759252F0	270BB6	Default IME	IME	主线程
7594A1B0	C0DD2	check	Button	主线程
7594A1B0	50D0E	About		主线程
75961120	50CE8			主线程
769F72E0	50BE8	MSCTFIME UI		主线程

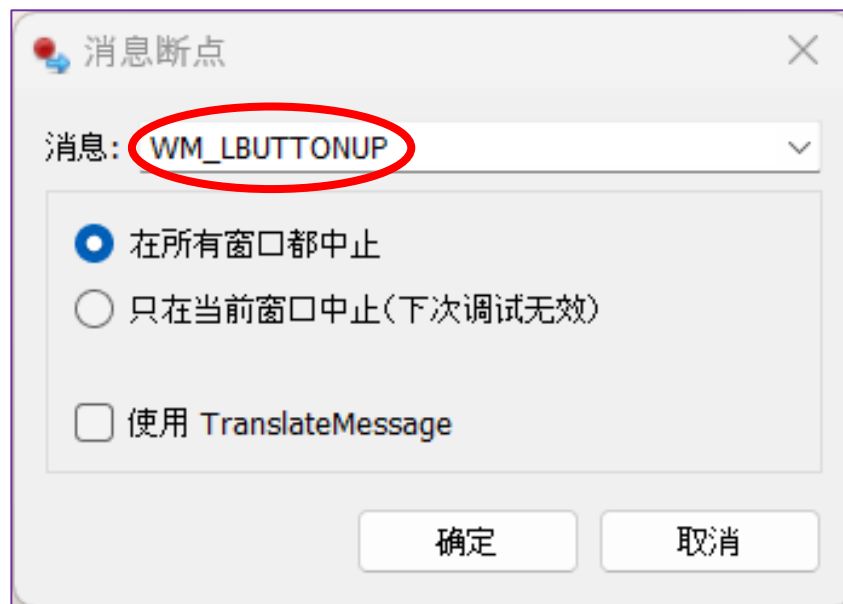


- 刷新(R) F5
- 禁用窗口
- 在反汇编中转到窗口过程 Return
- Follow in Threads
- 在窗口过程切换断点 F2
- 消息断点**
- 搜索...
- 复制(C)



## 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

已暂停 INT3 断点于 user32.7594A1B0 !

默认 (stdcall) 5 ☐ 解锁

1:	[esp+4]	000C0DD2	000C0DD2
2:	[esp+8]	00000202	00000202
3:	[esp+C]	00000000	00000000
4:	[esp+10]	000D001C	000D001C
5:	[esp+14]	000C0DD2	000C0DD2

0019FCD4	75933773	返回到 user32.Ordinal#2713+6B3 自 ???
0019FCD8	000C0DD2	
0019FCD8	00000202	
0019FCE0	00000000	
0019FCE4	000D001C	
0019FCE8	000C0DD2	
0019FCEC	DCBAABCD	
0019FCF0	7594A1B0	user32.LoadCursorFromFile+4B0
0019FCF4	00000202	
0019FCE8	000C0DD2	



# 消息断点

- 打开内存窗口，对crackme1的.text段设置内存访问断点

Address	Offset	Module	Section	Permissions	Access	Read	Write	Execute
00200000	00130000	用户模块	保留	PRV	-RW--	-RW--	-RW--	-RW--
00330000	00022000	用户模块	PEB, TEB (14408), Wow64 TEB (14408)	PRV	-RW--	-RW--	-RW--	-RW--
00352000	000AE000	用户模块	保留 (00200000)	PRV	-RW--	-RW--	-RW--	-RW--
00400000	00001000	用户模块	crackme1.exe	IMG	-R---	ERWC-	ERWC-	ERWC-
00401000	0000C000	用户模块	".text"	IMG	ER---	ERWC-	ERWC-	ERWC-
0040D000	00006000	用户模块	".rdata"	IMG	-R---	ERWC-	ERWC-	ERWC-
00413000	00002000	用户模块	".data"	IMG	-RW--	ERWC-	ERWC-	ERWC-
00415000	00018000	用户模块	".rsrc"	IMG	-R---	ERWC-	ERWC-	ERWC-
00430000	00011000	用户模块	\Device\Harddisk...	MAP	-R---	-R---	-R---	-R---
00450000	00011000	用户模块	\Device\Harddisk...	MAP	-R---	-R---	-R---	-R---
00470000	00003000	用户模块	\Device\Harddisk...	MAP	-R---	-R---	-R---	-R---
00480000	00003000	用户模块	\Device\Harddisk...	MAP	-R---	-R---	-R---	-R---
00490000	00011000	用户模块	\Device\Harddisk...	MAP	-R---	-R---	-R---	-R---
004B0000	00011000	用户模块	\Device\Harddisk...	MAP	-R---	-R---	-R---	-R---
004D0000	00008000	用户模块	保留 (004D0000)	PRV	-RW--	-RW--	-RW--	-RW--
004D8000	00008000	用户模块	保留 (004D0000)	PRV	-RW--	-RW--	-RW--	-RW--
004E0000	00002000	用户模块	保留 (004D0000)	MAP	-R---	-R---	-R---	-R---
004F0000	00002000	用户模块	保留 (004D0000)	MAP	-R---	-R---	-R---	-R---
00500000	00001000	用户模块	保留 (004D0000)	MAP	-R---	-R---	-R---	-R---
00510000	00035000	用户模块	保留 (004D0000)	PRV	-RW--	-RW--	-RW--	-RW--
00545000	0000B000	用户模块	保留 (004D0000)	PRV	-RW-G	-RW--	-RW--	-RW--
00550000	00035000	用户模块	保留 (004D0000)	PRV	-RW-G	-RW--	-RW--	-RW--
00585000	0000B000	用户模块	保留 (004D0000)	PRV	-RW-G	-RW--	-RW--	-RW--
00590000	0006F000	用户模块	Heap (ID 0)	PRV	-RW--	-RW--	-RW--	-RW--
005FF000	00091000	用户模块	保留 (00590000)	PRV	-RW--	-RW--	-RW--	-RW--
00690000	000CE000	用户模块	\Device\Harddisk...	MAP	-R---	-R---	-R---	-R---
00760000	000FD000	用户模块	保留 (004D0000)	PRV	-RW--	-RW--	-RW--	-RW--
0085D000	00003000	用户模块	堆栈 (14408)	PRV	-RW-G	-RW--	-RW--	-RW--
00860000	000FD000	用户模块	保留 (004D0000)	PRV	-RW--	-RW--	-RW--	-RW--
0095D000	00003000	用户模块	堆栈 (11940)	PRV	-RW-G	-RW--	-RW--	-RW--
00960000	00035000	用户模块	保留 (004D0000)	PRV	-RW-G	-RW--	-RW--	-RW--
00995000	0000B000	用户模块	保留 (004D0000)	PRV	-RW-G	-RW--	-RW--	-RW--
009A0000	000FD000	用户模块	保留 (004D0000)	PRV	-RW-G	-RW--	-RW--	-RW--
00A9D000	00003000	用户模块	堆栈 (2528)	PRV	-RW-G	-RW--	-RW--	-RW--
00AA0000	0001F000	用户模块	保留 (00AA0000)	MAP	-R---	-R---	-R---	-R---
00ABF000	001E1000	用户模块	保留 (00AA0000)	MAP	-R---	-R---	-R---	-R---
00CA0000	00004000	用户模块	保留 (00CA0000)	MAP	-R---	-R---	-R---	-R---
00CA4000	00004000	用户模块	保留 (00CA0000)	MAP	-R---	-R---	-R---	-R---
00CB0000	00181000	用户模块	保留 (00E40000)	MAP	-R---	-R---	-R---	-R---
00E40000	0013A000	用户模块	保留 (00E40000)	MAP	-R---	-R---	-R---	-R---
00F7A000	012C7000	用户模块	保留 (00E40000)	MAP	-R---	-R---	-R---	-R---
022F0000	022F0000	用户模块	保留 (00E40000)	MAP	-R---	-R---	-R---	-R---

# 消息断点

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

已暂停

内存断点 (执行)位于crackme1.00401000, 异常地址:crackme1.00401090 !

CPU		日志	笔记	断点	内存布局	调用堆栈	SEH链	脚本	符号
EIP	ECX	ESI	00401090	55		push ebp			
			00401091	8BEC		mov ebp,esp			
			00401093	83EC 30		sub esp,30			
			00401096	8D45 D0		lea eax,dword ptr ss:[ebp-30]			
			00401099	6A 30		push 30			
			0040109B	6A 00		push 0			
			0040109D	50		push eax			
			0040109E	E8 2D0E0000		call crackme1.401ED0			
			004010A3	83C4 0C		add esp,C			
			004010A6	817D 0C 11010000		cmp dword ptr ss:[ebp+C],111			
			004010AD	0F85 C3000000		jne crackme1.401176			
			004010B3	0FB745 10		movzx eax,word ptr ss:[ebp+10]			
			004010B7	83E8 02		sub eax,2			
			004010BA	0F84 AE000000		je crackme1.40116E			
			004010C0	2D E7030000		sub eax,3E7			
			004010C5	74 23		je crackme1.4010EA			
			004010C7	83E8 01		sub eax,1			
			004010CA	0F85 A6000000		jne crackme1.401176			

## 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 A024100	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:"we'll 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	



允公允能 日新月异

## 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



南开大学  
Nankai University



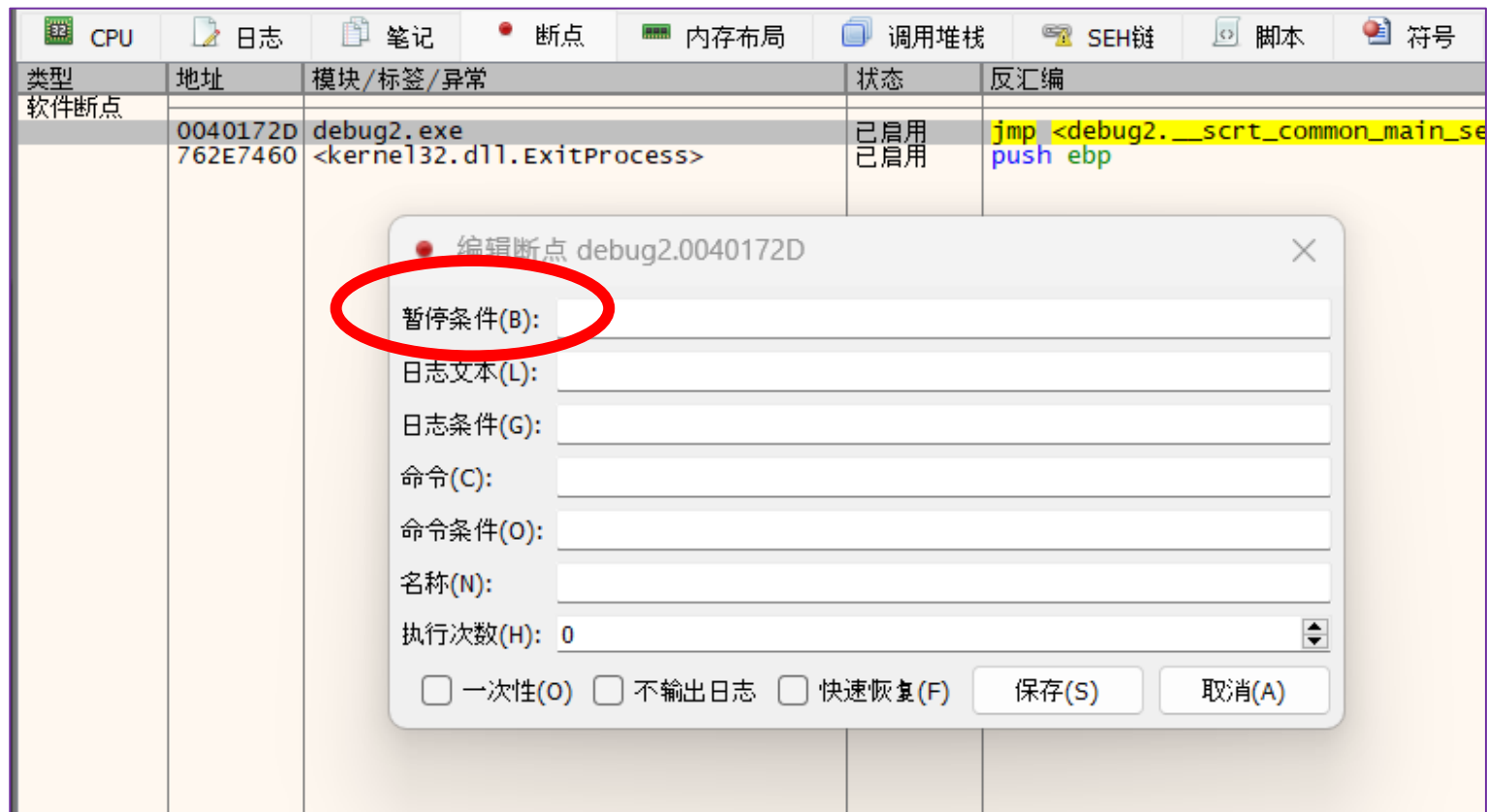


## 条件断点

- 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







# 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"
```

---





允公允能 日新月异

## 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，计数器的值就减一



南开大学  
Nankai University

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

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





南开大学

NANKAI UNIVERSITY, P.R. CHINA 1919

允公允能 日新月异



# 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



允公允能 日新月异

## 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



南开大学  
Nankai University