



汇编语言与逆向工程

付俊松、崔宝江

北京邮电大学

2018年10月



目录

@ 课程简介

@ 第一章 初识逆向工程



目录

@ 课程简介

□ 基础篇

- 初识逆向工程
- 基础知识
- 从C语言看汇编



目录

@ 课程简介

□ 进阶篇

- 逆向初体验
- 常见加密算法逆向分析
- 异常处理
- 代码混淆
- 脱壳



目录

@ 课程简介

□ 高级篇

- PE结构
- Hook
- IDA Python
- 反调试



目录

@ 课程简介

□ 参考书籍

- (1) 汇编语言（第3版），王爽 著 清华大学出版社,2013 年
- (2) 逆向工程权威指南，Dennis, Yurichev, 丹尼斯 著,Archer, 安天安全研究与应急处理中心 译,人民邮电出版社,2017
- (3) 逆向工程实战，[美] 邓（Bruce Dang）等著；单业 译,人民邮电出版社,2015



目录

@ 课程简介

- 考核方式
- 1-16周，周四，7-8节
1-16双周，周一，3-4节
- 3学分
- 讲授+动手实践
- 相关工具



常用工具下载

@ <https://down.52pojie.cn/Tools/>

爱盘 -- 在线破解工具包

<https://down.52pojie.cn/Tools/>

File Name ↓

Parent directory/

Android_Tools/

Anti_Rootkit/

Cryptography/

Debuggers/

OD、Windbg

Disassemblers/

IDA

Dongle/

Editors/

010 editor

NET/

Network_Analyzer/

OllyDbg_Plugin/

Other/

PEtools/

Packers/

Patchers/

Unpackers/



目录

@ 课程简介

@ 第一章 初识逆向工程



第一章 初识逆向工程

① 1.概述

② 2.常用工具简介



1.概述

@1.概述

- ❑ 什么是逆向工程
- ❑ 如何学习逆向工程
- ❑ 逆向分析的方法



1.概述

@ 逆向工程

□ Reverse Engineering

- 一种产品设计技术再现过程，即对一项目标产品进行逆向分析及研究，从而演绎并得出该产品的处理流程、组织结构、功能特性及技术规格等设计要素，以制作出功能相近，但又不完全一样的产品。



1.概述

@ 软件逆向工程

□ Software Reverse Engineering

- 是逆向工程在软件方面的应用，通过多种计算机技术的运用来实现对软件算法、流程、结构、代码的逆向拆解和分析，从而达到破解软件或扩展软件功能的目的。
- 逆向工程中设计到的知识主要包括加解密技术、代码混淆技术、脱壳技术、反调试技术等，后续将一一进行讲解。



1.概述

@ 1.概述

- 什么是逆向工程
- 如何学习逆向工程
- 逆向分析的方法



1.概述

@ 如何学习逆向工程

- ❑ 逆向工程涉及到很多知识，包括操作系统、汇编语言、加解密、编译原理等，同时要求逆向工程人员有丰富的编程经验和较强的程序理解、分析能力。
- ❑ 尽管逆向工程需要用到很多知识，但也不必担心自己没有基础，通常都是一边分析程序一边补充知识。



1.概述

@ 如何学习逆向工程

□ (1) 积累足量的汇编知识。

- 在分析软件时，主要的分析对象是汇编代码，逆向工程人员通过对汇编代码的分析得到软件的各种信息。因此，汇编语言的学习在逆向工程中是必不可少的。
- 一个优秀的逆向工程人员要能够掌握基础的汇编知识，熟悉常见的汇编指令，并且熟练的使用汇编指令。



1.概述

@ 如何学习逆向工程

□ (2) 多多练习编程能力

- 学习逆向需要有基本的编程能力，能够自己编写实用程序；
- 要对理解编程的理论知识，例如函数的工作原理、函数调用约定、参数传递方式、返回值、堆栈等。
- 只有懂得如何使用高级语言编写程序，才能更好的读懂汇编代码。



1.概述

@ 如何学习逆向工程

□ (3) 理解操作系统相关原理

- 要熟悉常见的**windows API**,一些敏感的系统调用往往是逆向过程中的关注点;
- 要理解**windows**底层的知识,例如进程管理、内存分配、堆栈等,理解了这些就能更好的掌握注入技术、钩子技术等逆向工程中的常见技术。



1.概述

@ 如何学习逆向工程

□ (4) 掌握分析工具的使用

- 在分析程序时，工具的使用是必不可少的。
- 常见的分析工具有**IDA**、**OlllyDbg**(俗称**OD**)、**WinDbg**等，这些都是分析程序的利器。
- **IDA**是静态分析工具，**OD**和**WinDbg**是动态分析工具。



1.概述

@ 如何学习逆向工程

□ (5) 熟悉程序保护技术

- 很多程序为了自己不被分析，往往会利用**PE**文件结构结合加密算法对自己进行保护，也就是加壳，同时可能配合反调试技术加强保护。
- 在分析这类程序时，就需要熟悉**PE**结构、常见的加解密算法、反调试技术。



1.概述

@ 如何学习逆向工程

□ (6) 不断的动手实践

- 逆向工程是一门技术，需要不断的实践才能逐渐掌握。
- 只有通过实践，才能真正理解并掌握各种各样的逆向技术，才能体会到逆向的乐趣。



1.概述

④ 如何学习逆向工程

- ❑ 除此之外，可以多去一些论坛上逛逛，学习一下别人的逆向思路和方法，也可以用论坛上的例子来练手，比如看雪论坛、吾爱破解等。



1.概述

@1.概述

- 什么是逆向工程
- 如何学习逆向工程
- 逆向分析的方法



1.概述

@ 逆向分析的方法

- ❑ 在对软件进行逆向分析时，需要综合使用各种分析工具从多个方面对软件进行分析。
- ❑ 软件逆向分析方法主要分为两类
 - 静态分析
 - 动态分析



1.概述

@ 静态分析

- ❑ 在不运行软件的情况下对软件进行分析，观察软件的外部特征
- ❑ 获取文件的类型（**EXE**、**DLI**、**DOC**等）、大小、**PE**头信息、导入导出函数、内部字符串、是否运行时解压缩、注册信息、调试信息、数字证书等多种信息



1.概述

@ 静态分析

- ❑ 静态分析所得的软件特征称为静态特征
- ❑ 通过静态特征能够推测软件的功能行为，以及软件运行在系统中留下的痕迹。
- ❑ **IDA Pro**是最常用的软件静态分析工具
- ❑ 除此之外，还应该学会使用**PEView**、**stud_PE**等专门分析**PE**头信息的工具。



静态分析



1.概述

@ 动态分析

- ❑ 在运行软件的情况下对软件的代码流、内存状态、程序内部结构、程序功能原理、系统行为等特征进行分析。
- ❑ 由于程序往往有多个分支，一次运行不能触发所有功能，因此动态分析中往往会多次运行程序。
- ❑ 与静态分析不同，动态分析能够观察到程序的真实功能。



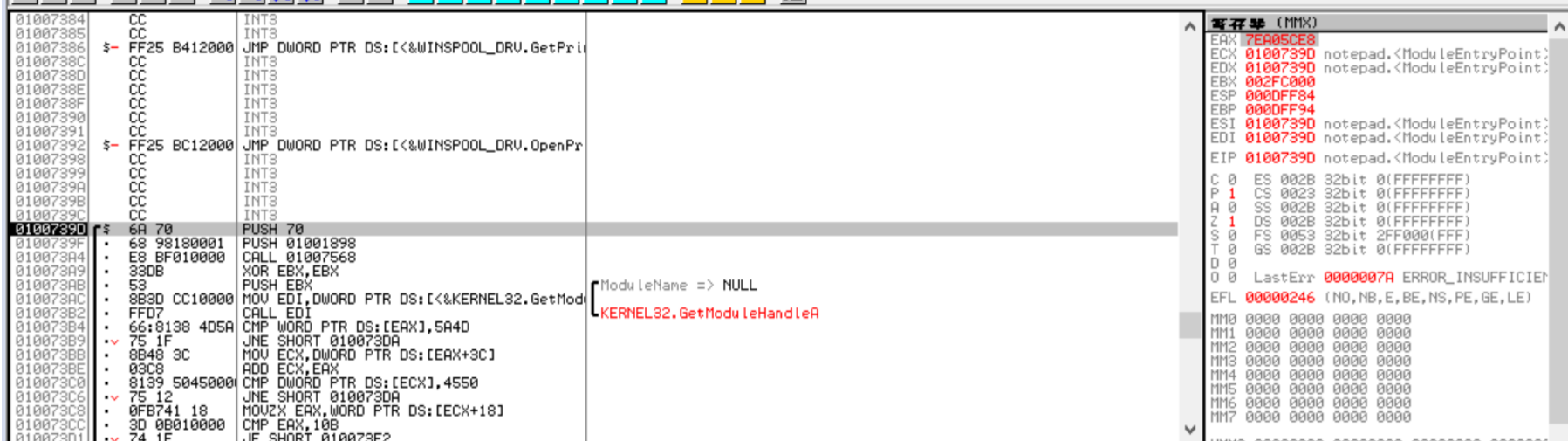
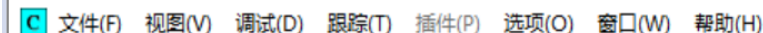
1.概述

@ 动态分析

- ❑ 恶意程序的运行有时会给分析机器和网络带来危险，因此往往搭建虚拟机或沙箱进行动态分析
- ❑ WinDbg、OllyDbg是常见的动态分析工具
- ❑ Norman沙箱、GFI沙箱、Comodo沙箱是常见的沙箱工具



④ 动态分析



```
notepad.<ModuleEntryPoint>
```

地址	十六进制数据	ASCII	注释
01000900	00 00 00 00 D4 70 00 01		
01000910	00 00 00 00 00 00 00 00		
01000920	4E 00 6F 00 74 00 65 00	N o t e	
01000930	FF FF FF FF 01 00 00 00		
01000940	04 00 00 00 05 00 00 00		
01000950	08 00 00 00 09 00 00 00		
01000960	0C 00 00 00 0D 00 00 00		
01000970	10 00 00 00 11 00 00 00		
01000980	2D 00 00 00 14 00 00 00		
01000990	17 00 00 00 18 00 00 00		
010009A0	1B 00 00 00 1C 00 00 00		
010009B0	1F 00 00 00 20 00 00 00		
010009C0	23 00 00 00 24 00 00 00		

1.概述

@ 动静结合分析

- ❑ 一般情况下，逆向分析要同时使用静态分析方法和动态分析方法，也就是动静结合。
- ❑ 首先使用静态分析收集软件的相关信息，然后通过这些信息推测软件的功能、结构和机制，之后再进行分析。
- ❑ 动静结合的分析方法能够有效的提高程序分析的效率，增加程序分析的效率。



第一章 初识逆向工程

① 1.概述

② 2.常用工具简介



2. 常用工具简介

④ 工欲善其事，必先利其器

- ❑ 在学习逆向分析的过程中，工具是必不可少的。
- ❑ 熟练掌握一些逆向分析工具，往往能在以后的实际运用中起到很好的效果。



2. 常用工具简介

- ☐ IDApro
- ☐ OllyDbg
- ☐ WinDbg
- ☐ GDB



常用工具下载

@ <https://down.52pojie.cn/Tools/>

爱盘 -- 在线破解工具包

<https://down.52pojie.cn/Tools/>

File Name ↓

Parent directory/

Android_Tools/

Anti_Rootkit/

Cryptography/

Debuggers/

OD、Windbg

Disassemblers/

IDA

Dongle/

Editors/

010 editor

NET/

Network_Analyzer/

OllyDbg_Plugin/

Other/

PEtools/

Packers/

Patchers/

Unpackers/



2. 常用工具简介

IDApro

- ❑ 逆向分析时，通常先用**IDApro**先对可执行文件进行静态分析。
- ❑ 这里所说的静态分析是在不执行程序的情况下借助反汇编工具来直接对可执行文件进行解析，得到汇编代码，进而从汇编层次上对程序进行分析。



2. 常用工具简介

④ **IDA**打开待分析文件后，会进入到主窗口界面，其中最明显的就是最大的两个窗口：

❑ 函数窗口（**Function**）

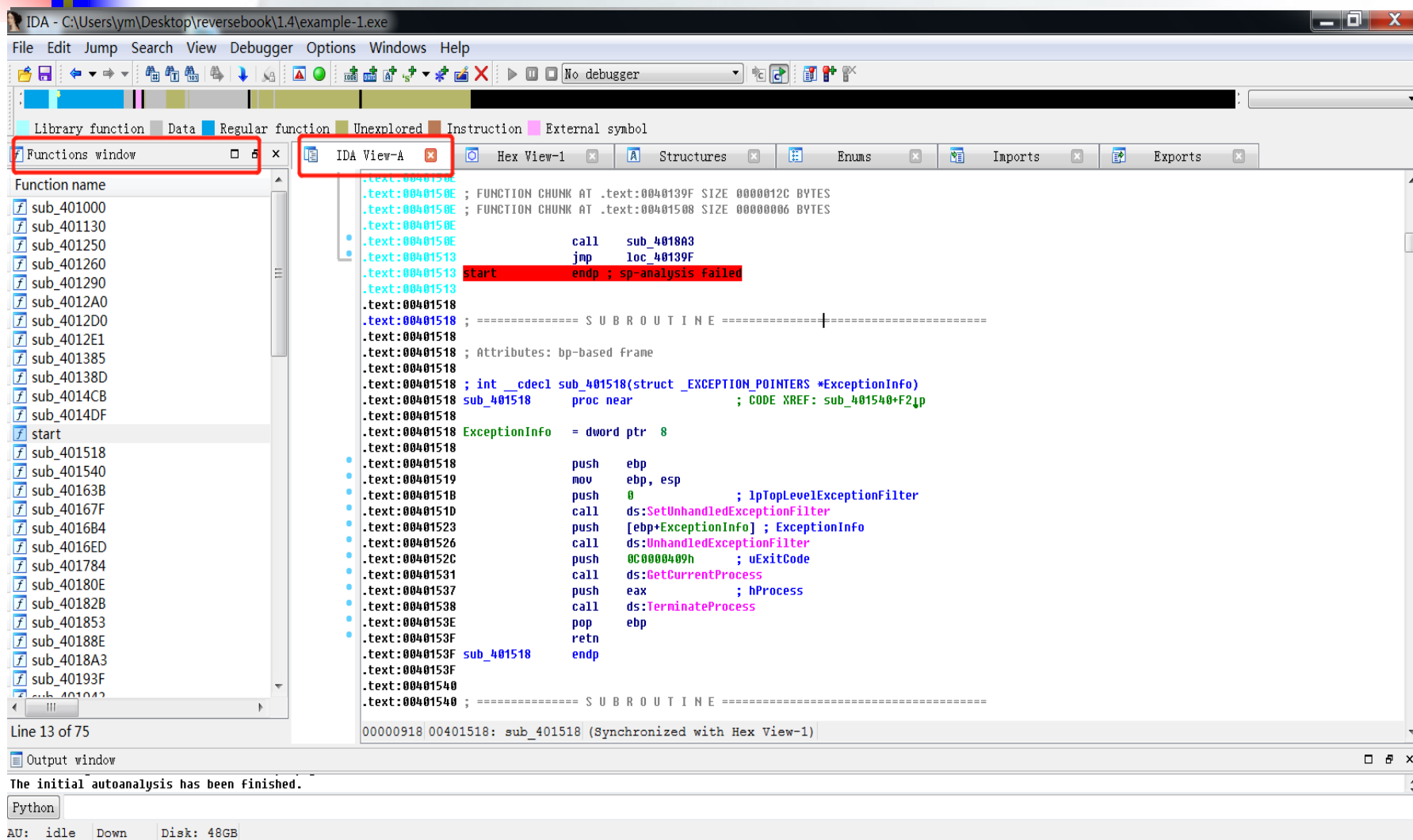
○ 函数窗口里显示的是**IDA**自动识别出来的一些函数

❑ 反汇编窗口

○ 反汇编窗口里显示的则是整个程序反汇编之后的结果



2. 常用工具简介



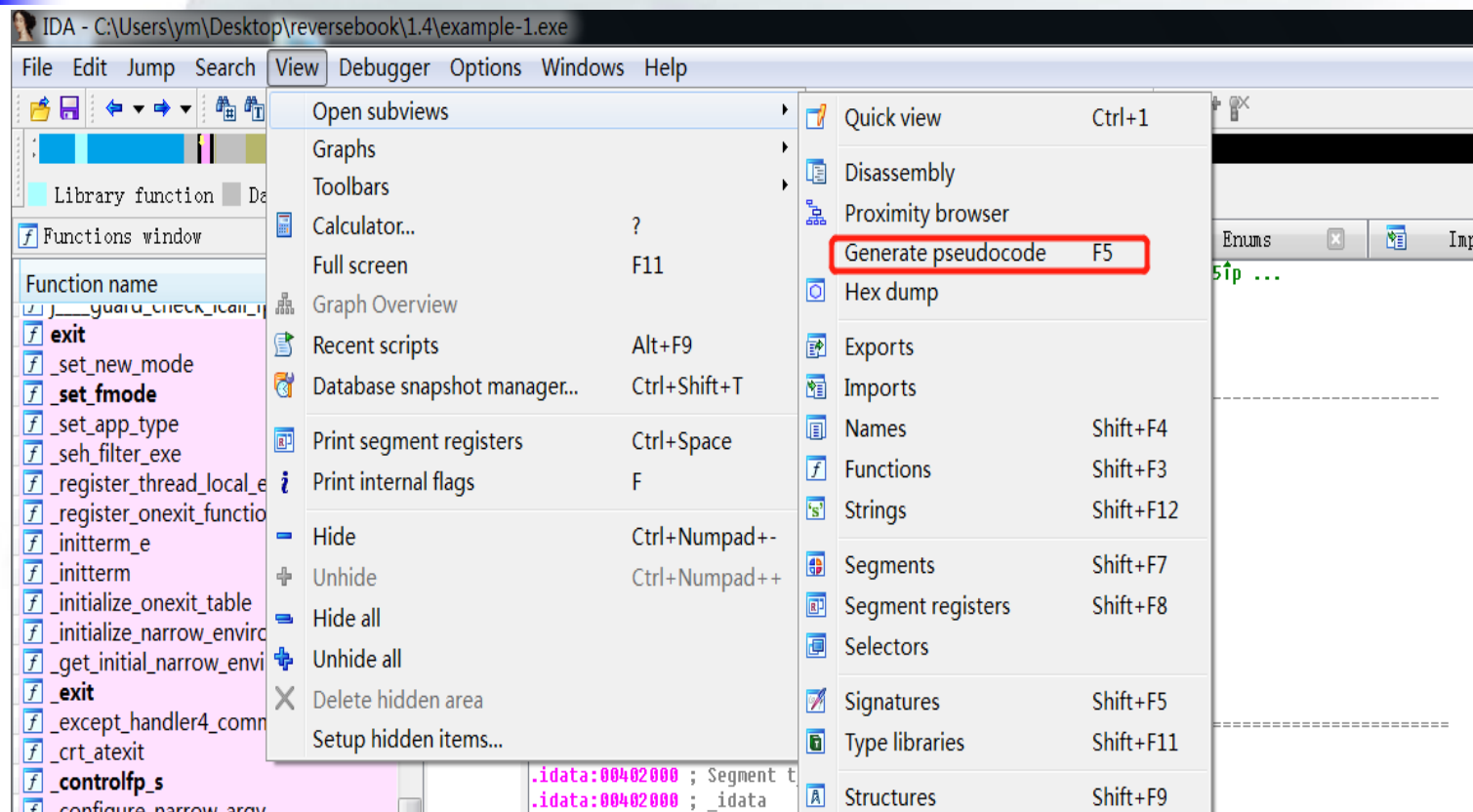
2. 常用工具简介

@F5插件介绍

- ❑ 直接对汇编语言进行分析可能难度较大，IDA里面的插件**Generate pseudocode**实现了将汇编代码转换为伪代码的功能
- ❑ 该插件的位置在**View→Open subview→Generate pseudocode**，对应的快捷键为**F5**。



2. 常用工具简介



2. 常用工具简介

④ 常用快捷键

□ Tab

- 有些时候**F5**插件得到的伪代码并不理想，这个时候需要自己去分析汇编代码。**Tab**键能够很方便的在伪代码和汇编代码之间进行切换。

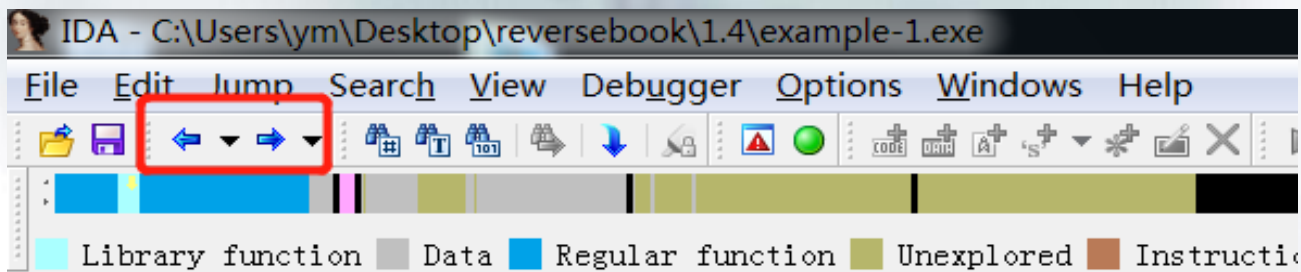


2. 常用工具简介

② 常用快捷键

□ 翻页

- 当在IDA里面进行了多个操作且想回到之前进行操作的地方时，点击工具栏里的翻页键可以达到想要的效果。



2. 常用工具简介

@ 常用快捷键

□ Esc键

- 如果仅进行了一个操作，如点击进入到了另一个函数，且这个时候想回到之前的状态，那么直接按也能达到相同的效果。



2. 常用工具简介

注释

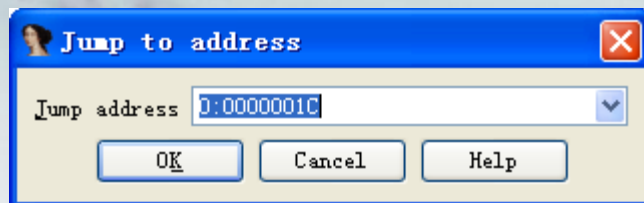
- ❑ 添加注释分为汇编代码的注释和伪代码的注释
- ❑ 如要添加汇编代码的注释，在需要添加的地方右键，可以看到**Entercomment**、**Enter repeatable comment**，对应的快捷键分别是冒号和分号。
- ❑ 按冒号添加的注释只在添加的地方显示，按分号添加的注释在所有交叉引用的地方都会出现
- ❑ 同样，如要添加伪代码的注释，只需按下对应的快捷键 **‘/’**，然后输入注释内容即可。



2. 常用工具简介

@ 跳转

- 要跳转到某个地址或某个函数时，按**G**并输入要跳转的地址或函数名称，可以直接实现跳转



视图切换

The image displays two side-by-side windows of the IDA Pro disassembler, showing the disassembly of a function named `1000B04: using guessed type int dword_1000BA4`.

Left Window (IDA - C:\WINDOWS\NOTEPAD.IDB (NOTEPAD.EXE))

The left window shows the disassembly of a function named `1000B04: using guessed type int dword_1000BA4`. The assembly code is as follows:

```

1000B04: using guessed type int dword_1000BA4;
Python
AU: idle Down Disk: 35GB

```

Right Window (IDA - C:\WINDOWS\NOTEPAD.IDB (NOTEPAD.EXE))

The right window shows the disassembly of a function named `1000B04: using guessed type int dword_1000BA4`. The assembly code is as follows:

```

1000B04: using guessed type int dword_1000BA4;
Python
AU: idle Down Disk: 35GB

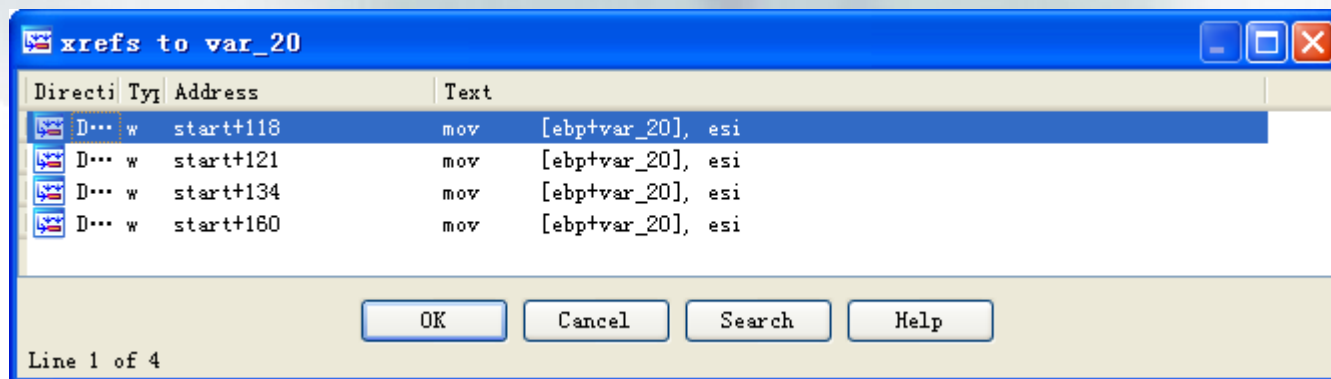
```

Both windows show the same assembly code, indicating that the disassembly is consistent across the two views.

2. 常用工具简介

@ 交叉引用

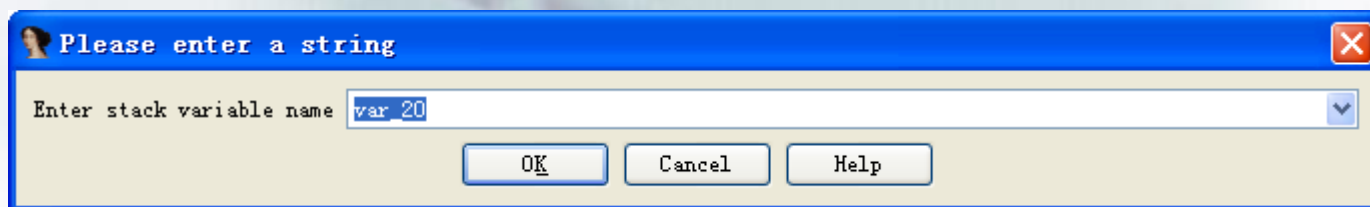
- ❑ 对于某个变量，有时需要查找它在哪里被引用了。
- ❑ 可以先将光标移至需要查看的变量处，按下X键，就会出现所有关于该变量交叉引用列表。查看函数的交叉引用时进行同样的操作即可。



2. 常用工具简介

@ 重命名

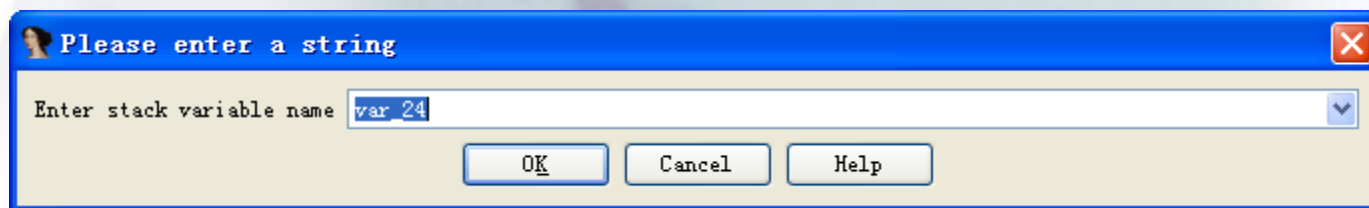
- ❑ 将光标移至要重命名的函数或变量处，按下N键，即可对函数和变量重命名。



2. 常用工具简介

@ 修改变量类型

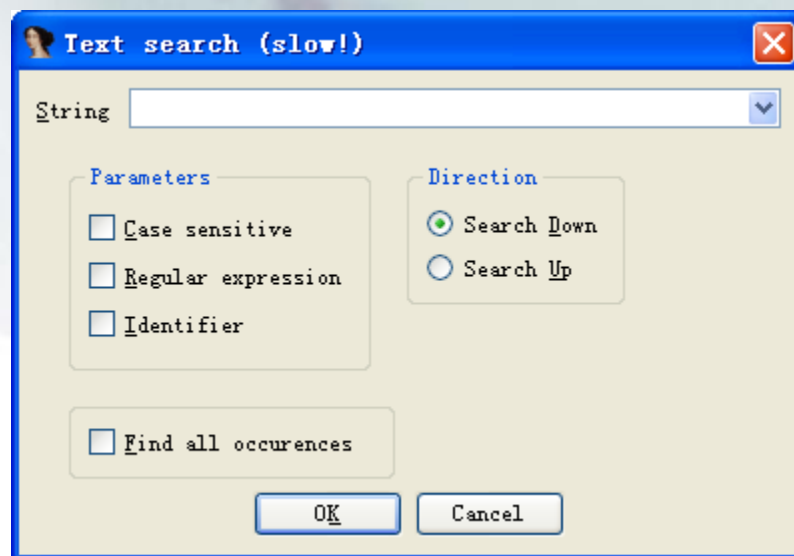
- 要修改变量类型时，将光标移至要修改的变量处，按下Y键，在弹出的窗口处输入新的变量类型。



2. 常用工具简介

@ 字符串相关

- ☐ 快捷键**Shift+F12**可以将程序中所有的字符串列出来。
- ☐ 快捷键**Alt+T**可以搜索某个特定的字符串。



2. 常用工具简介

@ 数据和指令之间的切换

- ❑ 如果确信某段十六进制数据是一段指令，那么可以按下**快捷键C**，它能将一段十六进制数据解释为汇编指令。
- ❑ **快捷键P**能够将一段代码定义为函数，并且定义的函数会在函数窗口中显示出来。
- ❑ 将数据定义为指令或函数后，如果想取消定义，按下**快捷键U**即可取消定义，并将指令或函数以十六进制的数据来显示。



2. 常用工具简介

@ 数据和字符串的切换

- 如果确信某段十六进制数据是字符串，**快捷键A**可以将该十六进制数据转换为字符串。如果要取消转换，按下**快捷键U**即可。



2. 常用工具简介

@ 进制转换

□ IDA里面的数据默认都是十六进制显示的，有时候需要将其转换为我们熟悉的十进制，只需将光标移至待转换的数据处并按下**快捷键H**即可实现进制转换。



2. 常用工具简介

@ 修改堆栈平衡

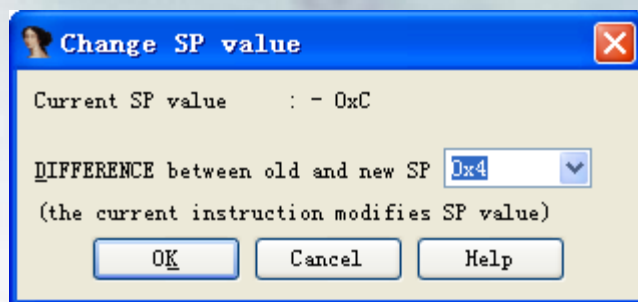
- ❑ **IDA**会跟踪函数内每一条指令上的栈指针的变化。如果**IDA**无法确定一条指令是否更改了栈指针，就需要手动调整栈指针了，否则**IDA**会为函数剩余部分提供一个错误的栈指针。
- ❑ 修改堆栈平衡的快捷键为**Alt+K**，打开窗口后会显示当前指令处栈顶指针**ESP**的值（**Current SP value**），并且可以对当前指令对栈的改变量进行修改。



2. 常用工具简介

@ 修改堆栈平衡

- ❑ 修改堆栈平衡的快捷键为**Alt+K**，打开窗口后会显示当前指令处栈顶指针**ESP**的值（**Current SP value**），并且可以对当前指令对栈的改变量进行修改。



2. 常用工具简介

- ❑ IDApro
- ❑ OllyDbg
- ❑ WinDbg
- ❑ GDB



2. 常用工具简介

@ OllyDbg调试器（简称OD）

- ❑ 是一款具有可视化界面的用户模式调试器，它结合了动态调试和静态分析，对异常的跟踪处理很灵活。
- ❑ 它的反汇编引擎能够识别出上千个经常使用的函数，加上爱好者不断的对其进行修改，OD也变得越来越大。



2. 常用工具简介

@ 主窗口界面

- 用**OD**打开程序后默认打开的是**CPU**窗口。可以看到对应的有几个子面板窗口，分别是反汇编面板、寄存器面板、堆栈面板、数据面板、信息面板。



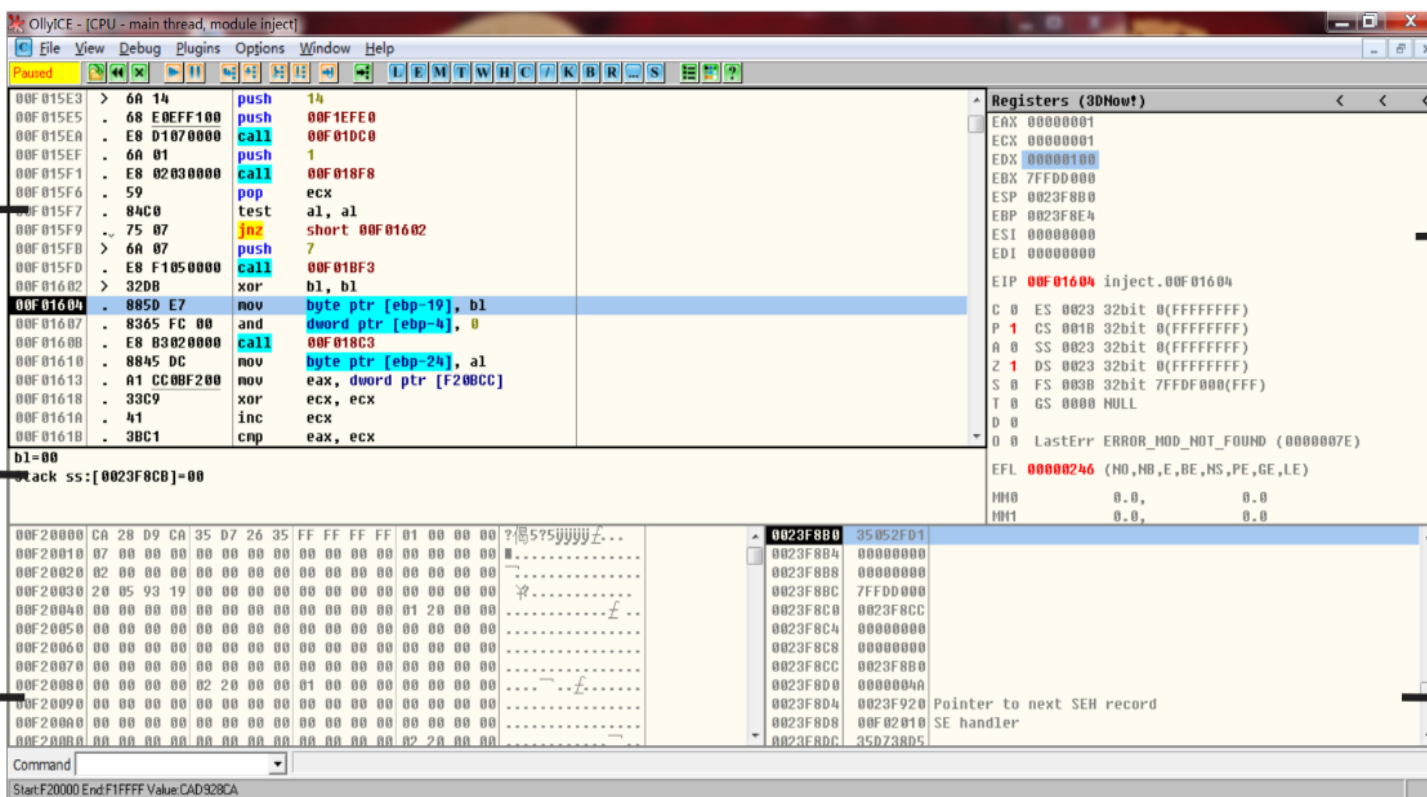
2. 常用工具简介

主窗口界面

反汇编面板

信息面板

数据面板



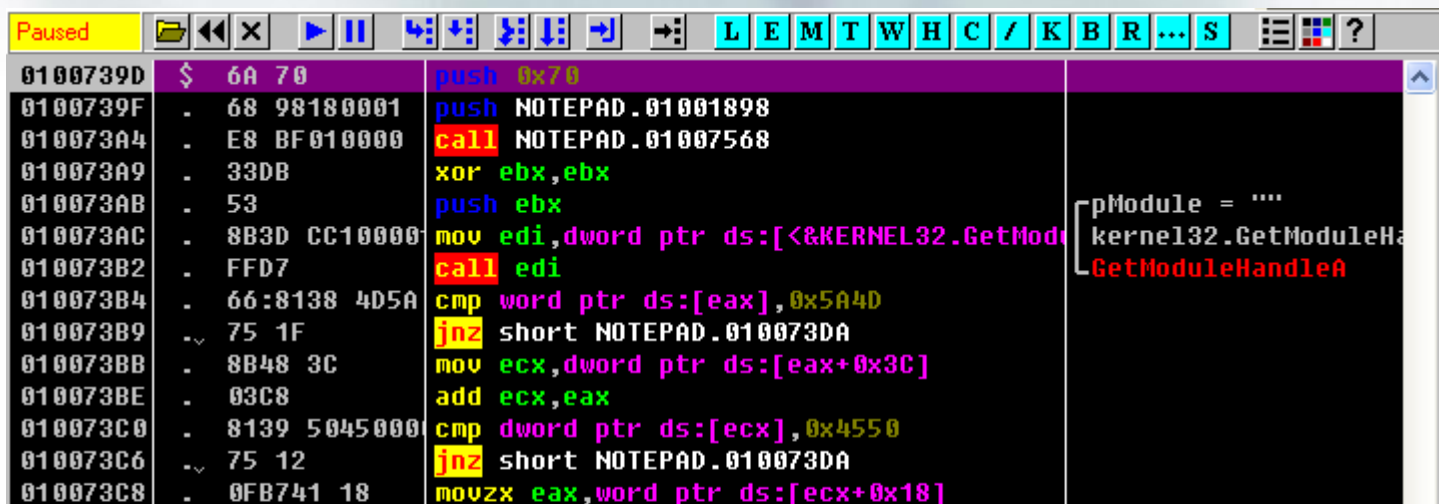
寄存器面板

堆栈面板

2. 常用工具简介

@ 反汇编面板窗口

- ❑ 被调试程序在反汇编后显示显示在该面板窗口中，显示的信息分为四列，从左至右依次为：地址、数据、反汇编代码和注释。
- ❑ 主要关注的是反汇编代码列，该列将指令以汇编代码的形式显示，从而让用户在汇编层次上对程序进行调试。



```
Paused
[Icons] L E M T W H C / K B R ... S [Icons] ?

0100739D  $ 6A 70      push 0x70
0100739F  . 68 98180001  push NOTEPAD.01001898
010073A4  . E8 BF010000  call NOTEPAD.01007568
010073A9  . 33DB        xor ebx,ebx
010073AB  . 53          push ebx
010073AC  . 8B3D CC1000  mov edi,dword ptr ds:[&KERNEL32.GetMod
010073B2  . FFD7        call edi
010073B4  . 66:8138 4D5A cmp word ptr ds:[eax],0x5A4D
010073B9  ~ 75 1F        jnz short NOTEPAD.010073DA
010073BB  . 8B48 3C      mov ecx,dword ptr ds:[eax+0x3C]
010073BE  . 03C8        add ecx,eax
010073C0  . 8139 504500  cmp dword ptr ds:[ecx],0x4550
010073C6  ~ 75 12        jnz short NOTEPAD.010073DA
010073C8  . 0FB741 18    movzx eax,word ptr ds:[ecx+0x18]
```

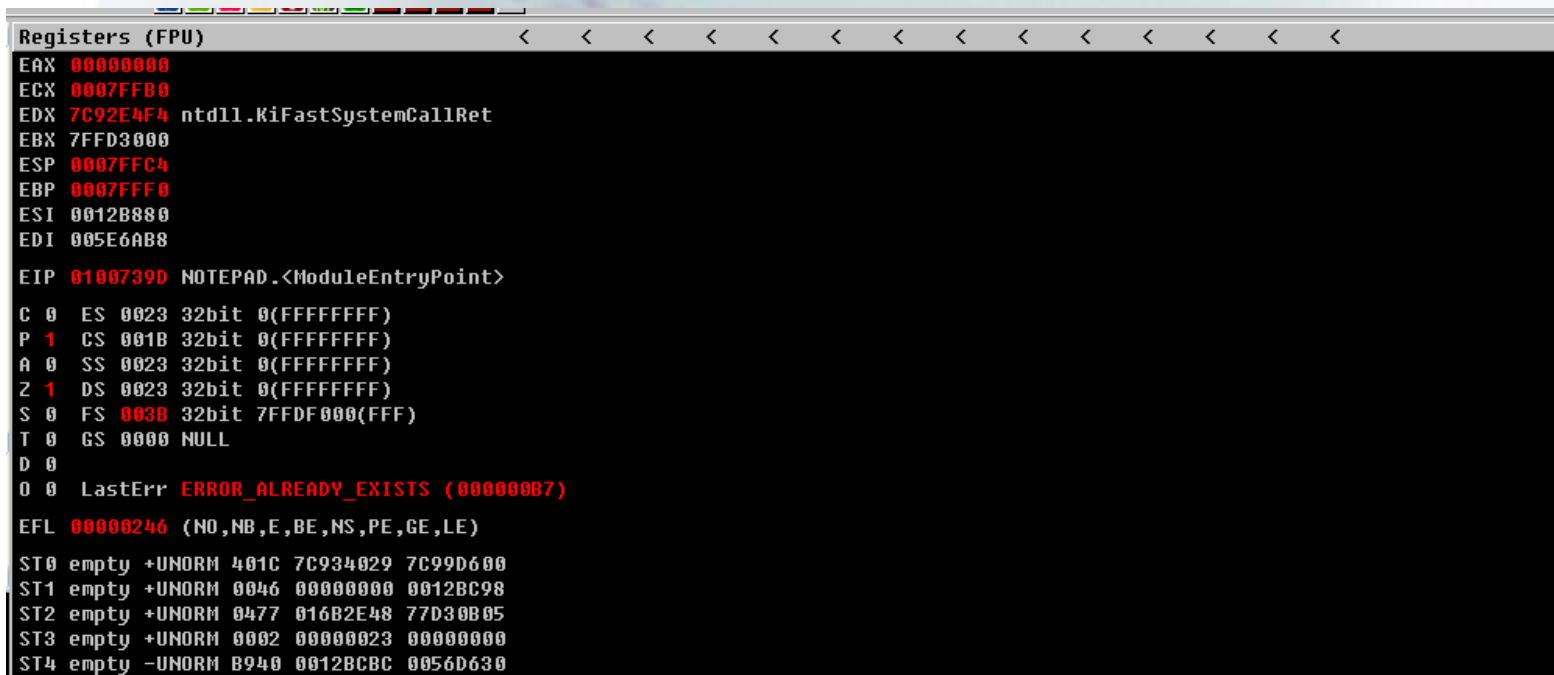
pModule = ''
kernel32.GetModuleHa
GetModuleHandleA



2. 常用工具简介

@ 寄存器面板窗口

- 该窗口将程序运行时用到的寄存器的值都列了出来，其中寄存器的值发生变化时颜色会变为红色。

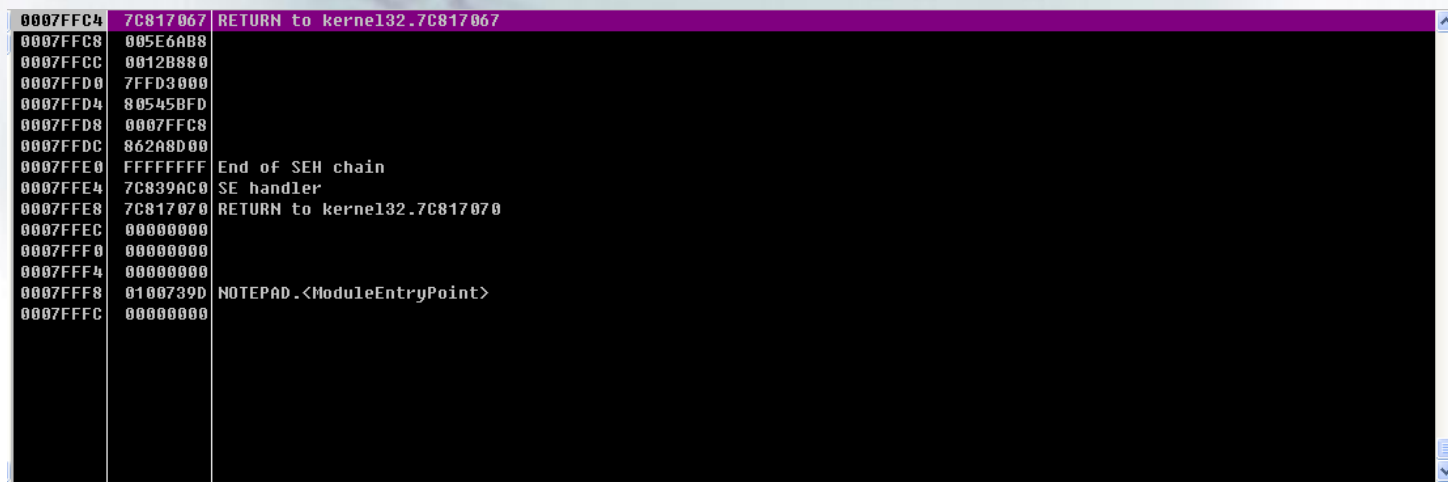


```
Registers (FPU)
EAX 00000000
ECX 0007FFB0
EDX 7C92E4F4 ntdll.KiFastSystemCallRet
EBX 7FFD3000
ESP 0007FFC4
EBP 0007FFF0
ESI 0012B880
EDI 005E6AB8
EIP 0100739D NOTEPAD.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_ALREADY_EXISTS (00000007)
EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty +UNORM 401C 7C934029 7C99D600
ST1 empty +UNORM 0046 00000000 0012BC98
ST2 empty +UNORM 0477 016B2E48 77D30B05
ST3 empty +UNORM 0002 00000023 00000000
ST4 empty -UNORM B940 0012BCBC 0056D630
```

2. 常用工具简介

@ 堆栈面板窗口

- 该窗口显示的是程序执行时的堆栈信息，左边一列是堆栈地址，右边一列是该地址存放的数据。



0007FFC4	7C817067	RETURN to kernel32.7C817067
0007FFC8	005E6A88	
0007FFCC	0012B880	
0007FFD0	7FFD3000	
0007FFD4	80545BFD	
0007FFD8	0007FFC8	
0007FFDC	862A8D00	
0007FFE0	FFFFFFFF	End of SEH chain
0007FFE4	7C839AC0	SE handler
0007FFE8	7C817070	RETURN to kernel32.7C817070
0007FFEC	00000000	
0007FFF0	00000000	
0007FFF4	00000000	
0007FFF8	0100739D	NOTEPAD.<ModuleEntryPoint>
0007FFFC	00000000	

2. 常用工具简介

④ 数据面板窗口

- 该窗口以十六进制和字符串的形式显示程序在内存中的数据。

Address	Hex dump				ASCII
01009000	00 00 00 00	D4 70 00 01	00 00 00 00	00 00 00 00詐.ㄟ.....
01009010	00 00 00 00	00 00 00 00	78 00 00 00	01 00 00 00x.ㄟ...
01009020	4E 00 6F 00	74 00 65 00	70 00 61 00	64 00 00 00	N.o.t.e.p.a.d...
01009030	FF FF FF FF	01 00 00 00	02 00 00 00	03 00 00 00	yyyyㄟ...ㄟ...
01009040	04 00 00 00	05 00 00 00	06 00 00 00	07 00 00 00	!...ㄟ.■.....
01009050	08 00 00 00	09 00 00 00	0A 00 00 00	0B 00 00 00	■.....■.....
01009060	0C 00 00 00	0D 00 00 00	0E 00 00 00	0F 00 00 00■.....
01009070	10 00 00 00	11 00 00 00	12 00 00 00	13 00 00 00	■.....■.....
01009080	2D 00 00 00	14 00 00 00	15 00 00 00	16 00 00 00	-...■.....■...
01009090	17 00 00 00	18 00 00 00	19 00 00 00	1A 00 00 00	■.....■.....
010090A0	1B 00 00 00	1C 00 00 00	1D 00 00 00	1E 00 00 00	■.....■.....
010090B0	1F 00 00 00	20 00 00 00	21 00 00 00	22 00 00 00	■... ..?...'...
010090C0	23 00 00 00	24 00 00 00	25 00 00 00	26 00 00 00	#...\$.%...&...
010090D0	27 00 00 00	28 00 00 00	29 00 00 00	2A 00 00 00	'...(...)*...
010090E0	2B 00 00 00	2C 00 00 00	34 90 00 01	38 90 00 01	+...,.4?ㄟ?
010090F0	3C 90 00 01	40 90 00 01	4C 90 00 01	48 90 00 01	<?ㄟ?ㄟ?ㄟ?
01009100	44 90 00 01	50 90 00 01	54 90 00 01	58 90 00 01	D?ㄟ?ㄟ?ㄟ?
01009110	5C 90 00 01	60 90 00 01	64 90 00 01	68 90 00 01	\?ㄟ?ㄟ?ㄟ?
01009120	6C 90 00 01	70 90 00 01	74 90 00 01	84 90 00 01	1?ㄟ?ㄟ?ㄟ?
01009130	88 90 00 01	8C 90 00 01	90 90 00 01	94 90 00 01	垚.ㄟ.ㄟ.ㄟ.ㄟ.
01009140	98 90 00 01	9C 90 00 01	A0 90 00 01	A8 90 00 01	槓.ㄟ.ㄟ.ㄟ.ㄟ.



2. 常用工具简介

④ 信息面板窗口

- 在调试跟踪时，该窗口显示与指令相关的寄存器值、API函数调用提示和跳转提示等信息

The screenshot shows the 'Information Panel' window in a debugger, titled 'KernelMode - NOTEPAD.EXE - [*G.P.U* - main thread, module NOTEPAD]'. The window displays assembly instructions and their corresponding register values. The instructions are listed in a table with columns for address, disassembly, and comment. The current instruction being executed is at address 010073BE, which is highlighted in purple. The register values are shown at the bottom of the window: eax=00000000 and ecx=0007FFB0.

Address	Disassembly	Comment
0100739D	push 0x70	
0100739F	push NOTEPAD.01001898	
010073A4	call NOTEPAD.01007568	
010073A9	xor ebx,ebx	
010073AB	push ebx	
010073AC	mov edi,dword ptr ds:[<&KERNEL32.GetMod	pModule = ""
010073B2	call edi	kernel32.GetModuleHa
010073B4	cmp word ptr ds:[eax],0x5A4D	GetModuleHandleA
010073B9	jnz short NOTEPAD.010073DA	
010073BB	mov ecx,dword ptr ds:[eax+0x3C]	
010073BE	add ecx,ecx	
010073C0	cmp dword ptr ds:[ecx],0x4550	
010073C6	jnz short NOTEPAD.010073DA	
010073C8	movzx eax,word ptr ds:[ecx+0x18]	
010073CC	cmp eax,0x10B	
010073D1	je short NOTEPAD.010073F2	
010073D3	cmp eax,0x20B	
010073D8	je short NOTEPAD.010073DF	
010073DA	mov dword ptr ss:[ebp-0x1C],ebx	
010073DD	jmp short NOTEPAD.01007406	
010073DF	cmp dword ptr ds:[ecx+0x84],0xE	
010073E6	jbe short NOTEPAD.010073DA	
010073E8	xor eax,ecx	

eax=00000000
ecx=0007FFB0



2. 常用工具简介

@ 开始调试

- ❑ 使用OD调试程序一般有两种打开方式。
- ❑ (1) 打开方式是：依次点击**File→Open**，选择待调试的程序，若程序执行时需要输入参数，则可以在**Arguments**栏中填入。
- ❑ (2) 打开方式是附加到正在执行中的进程，依次点击**File→Attach**，选择需要调试的进行即可。



2. 常用工具简介

@INT3断点

- ❑ 断点的一种，在诸如Olllydbg中的快捷键是F2，是一种很常用的断点类型。
- ❑ INT3指令的机器码为CC，所以通常也称之为CC指令。当被调试进程执行INT3指令导致一个异常时，调试器就会捕捉这个异常从而停在断点处，然后将断点处的指令恢复成原来的指令。



2. 常用工具简介

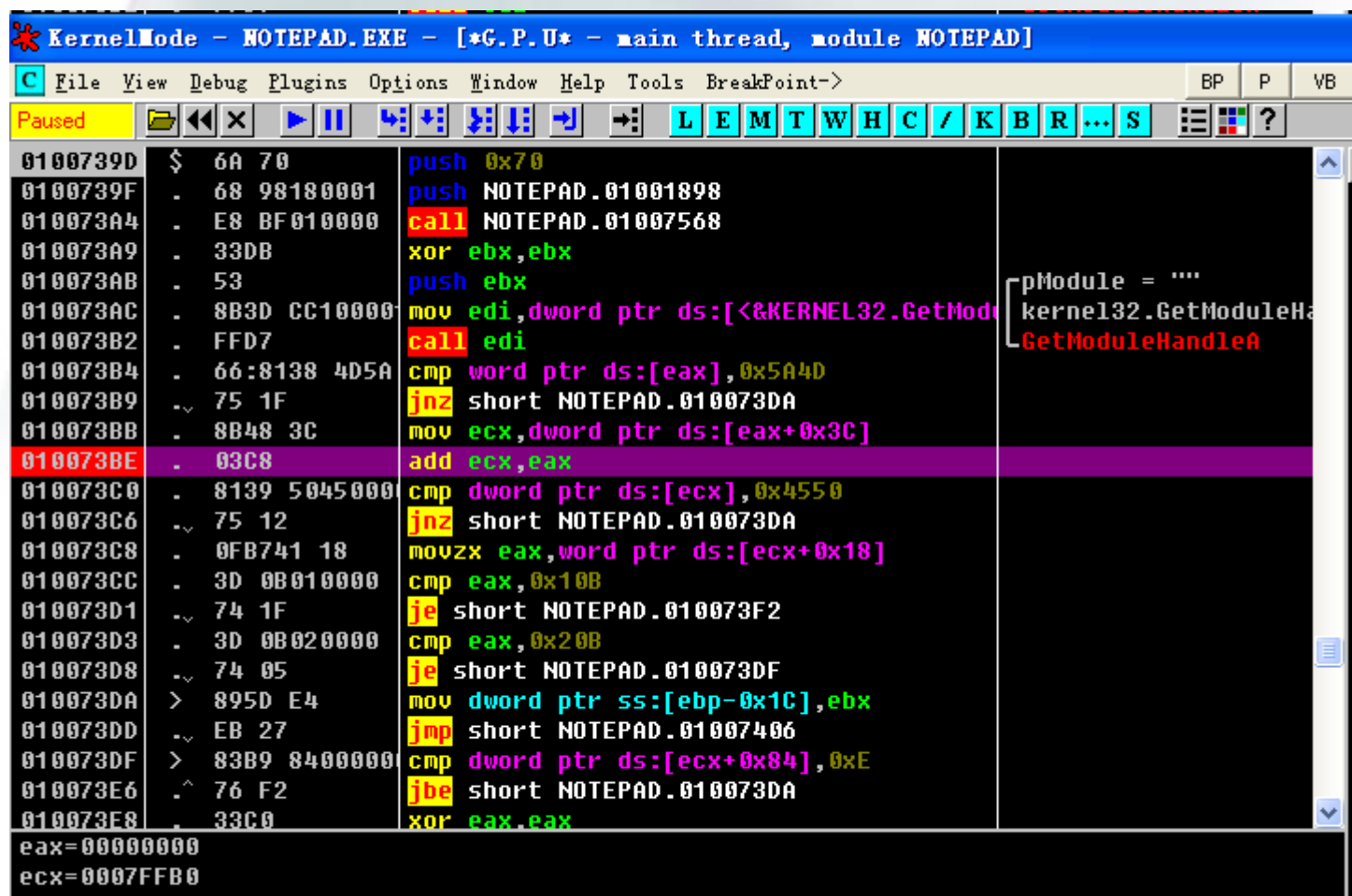
@INT3断点

- ❑ 用**INT3**断点的好处是可以设置无数个断点，缺点是改变了原程序指令，容易被软件检测到。
- ❑ 例如，为了防范**API**被下断，一些软件会检测**API**的首地址是否为**CCh**，以此来判断是否被下了断点。



2. 常用工具简介

@INT3断点



```
KernelMode - NOTEPAD.EXE - [*G.P.U* - main thread, module NOTEPAD]
File View Debug Plugins Options Window Help Tools BreakPoint-> BP P VB
Paused
0100739D $ 6A 70 push 0x70
0100739F . 68 98180001 push NOTEPAD.01001898
010073A4 . E8 BF010000 call NOTEPAD.01007568
010073A9 . 33DB xor ebx,ebx
010073AB . 53 push ebx
010073AC . 8B3D CC10000 mov edi,dword ptr ds:[<&KERNEL32.GetMod
010073B2 . FFD7 call edi
010073B4 . 66:8138 4D5A cmp word ptr ds:[eax],0x5A4D
010073B9 ~ 75 1F jnz short NOTEPAD.010073DA
010073BB . 8B48 3C mov ecx,dword ptr ds:[eax+0x3C]
010073BE . 03C8 add ecx,eax
010073C0 . 8139 5045000 cmp dword ptr ds:[ecx],0x4550
010073C6 ~ 75 12 jnz short NOTEPAD.010073DA
010073C8 . 0FB741 18 movzx eax,word ptr ds:[ecx+0x18]
010073CC . 3D 0B010000 cmp eax,0x10B
010073D1 ~ 74 1F je short NOTEPAD.010073F2
010073D3 . 3D 0B020000 cmp eax,0x20B
010073D8 ~ 74 05 je short NOTEPAD.010073DF
010073DA > 895D E4 mov dword ptr ss:[ebp-0x1C],ebx
010073DD ~ EB 27 jmp short NOTEPAD.01007406
010073DF > 83B9 8400000 cmp dword ptr ds:[ecx+0x84],0xE
010073E6 . 76 F2 jbe short NOTEPAD.010073DA
010073F8 . 33C0 xor eax,eax
eax=00000000
ecx=0007FFB0
```



2. 常用工具简介

@ 硬件断点

- ❑ 由硬件提供给我们的调试寄存器组，可以对这些硬件寄存器设置相应的值，然后让硬件帮我们断在需要下断点的地址。
- ❑ 硬件断点是**CPU**提供的功能，**Intel 80386**以上的**CPU** 提供了调试寄存器以用于软件调试。
- ❑ 从**Intel CPU**中，**DRx**调试寄存器总共有**8**个，从**DRx0**到**DRx7**。其中，**DR0~DR3**是调试地址寄存器，保存设置的硬件断点。



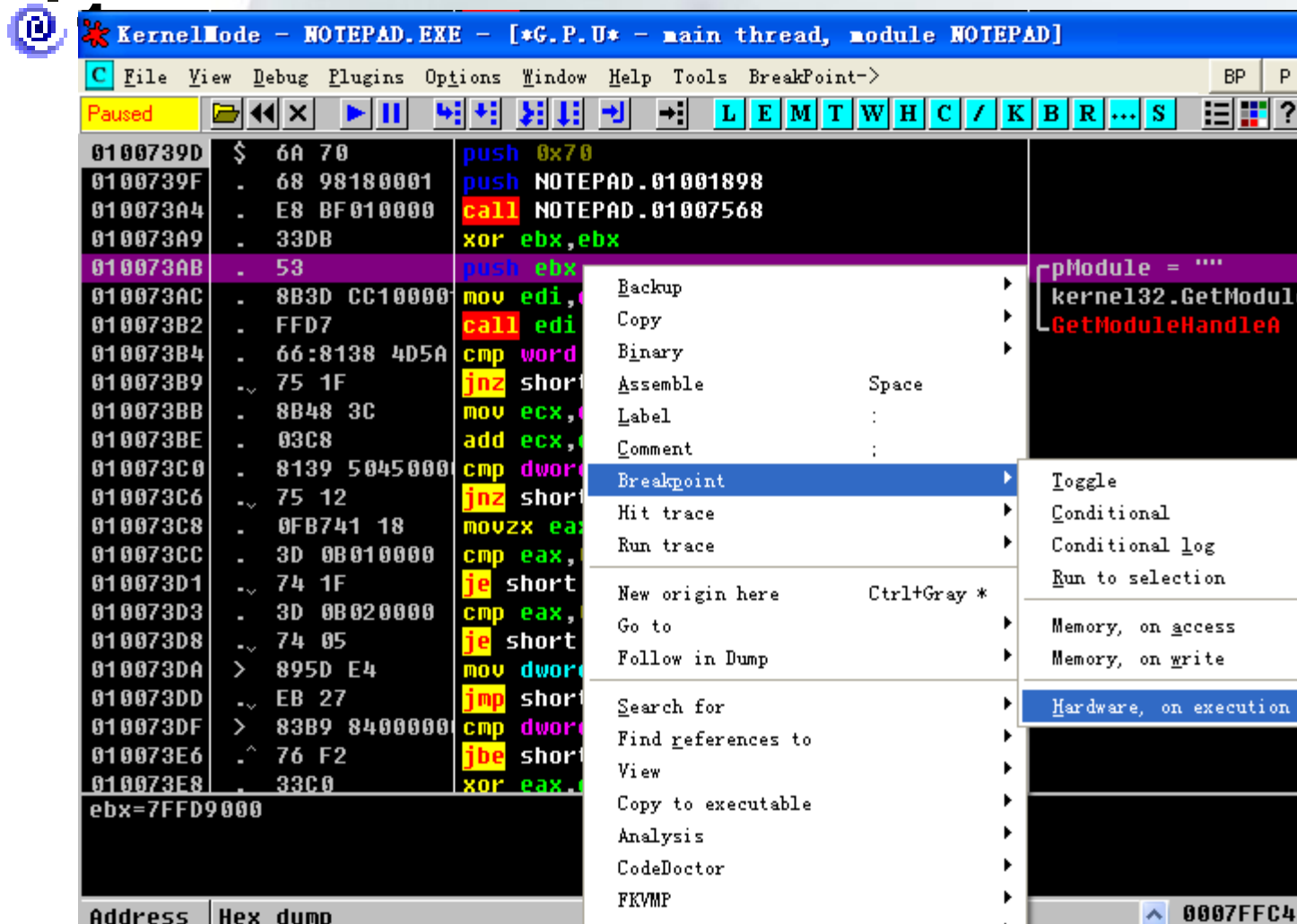
2. 常用工具简介

@ 硬件断点

- ❑ 硬件断点的设置只需鼠标右键并依次点击：**Breakpoint→Hardware on execution**即可。取消断点的操作则需依次点击：**Debug→Hardware breakpoints**，然后在弹出的窗口中删除硬件断点（或者**Breakpoint→Remove Hardware breakpoints**）
- ❑ 硬件断点优点是速度快，在INT3断点容易被发现的地方，使用硬件断点来代替会有很好的效果，缺点就是最多只能设置4个断点。



2. 常用工具简介



2. 常用工具简介

KernelMode - NOTEPAD.EXE - [*G.P.U* - main thread, module NOTEPAD]

File View Debug Plugins Options Window Help Tools BreakPoint-> BP P VB Notepad Calc Folder CMD

Paused

0100739B CC int3
0100739C CC int3
0100739D \$ 6A 70 push 0x70
0100739F . 68 98180001 push NOTEPAD.01001898
010073A4 . E8 BF010000 call NOTEPAD.01007568
010073A9 . 33DB xor ebx,ebx
010073AB . 53 push ebx
010073AC . 8B3D CC10000 mov edi,dword ptr ds:[<&
010073B2 . FFD7 call edi
010073B4 . 66:8138 4D5A cmp word ptr ds:[eax],0x
010073B9 . 75 1F jnz short NOTEPAD.010073
010073BB . 8B48 3C mov ecx,dword ptr ds:[ea
010073BE . 03C8 add ecx,eax
010073C0 . 8139 50450001 cmp dword ptr ds:[ecx],0
010073C6 . 75 12 jnz short NOTEPAD.010073
010073C8 . 0FB741 18 movzx eax,word ptr ds:[e
010073CC . 3D 0B010000 cmp eax,0x10B
010073D1 . 74 1F je short NOTEPAD.010073F
010073D3 . 3D 0B020000 cmp eax,0x20B
010073D8 . 74 05 je short NOTEPAD.010073D
010073DA > 895D E4 mov dword ptr ss:[ebp-0x
010073DD . EB 27 jmp short NOTEPAD.010074
010073DF > 83B9 84000000 cmp dword ptr ds:[ecx+0x
ebx=7FFD9000

Registers (FPU)
EAX 00000000
ECX 0007FFB0
EDX 7C92E4F4 ntdll.KiFa
EBX 7FFD9000
ESP 0007FFC4
EBP 0007FFF0
ESI 0012B880
EDI 005E63D0
EIP 0100739D NOTEPAD.<M
C 0 ES 0023 32bit 0(FF
P 1 CS 001B 32bit 0(FF

Backup
Copy
Binary
Assemble Space
Label
Comment
Breakpoint
Hit trace
Run trace
New origin here Ctrl+Gray *
Go to
Follow in Dump
Search for
Find references to

Toggle F2
Conditional Shift+F2
Conditional log Shift+F4
Run to selection F4
Memory, on access
Memory, on write
Hardware, on execution
Remove hardware breakpoint



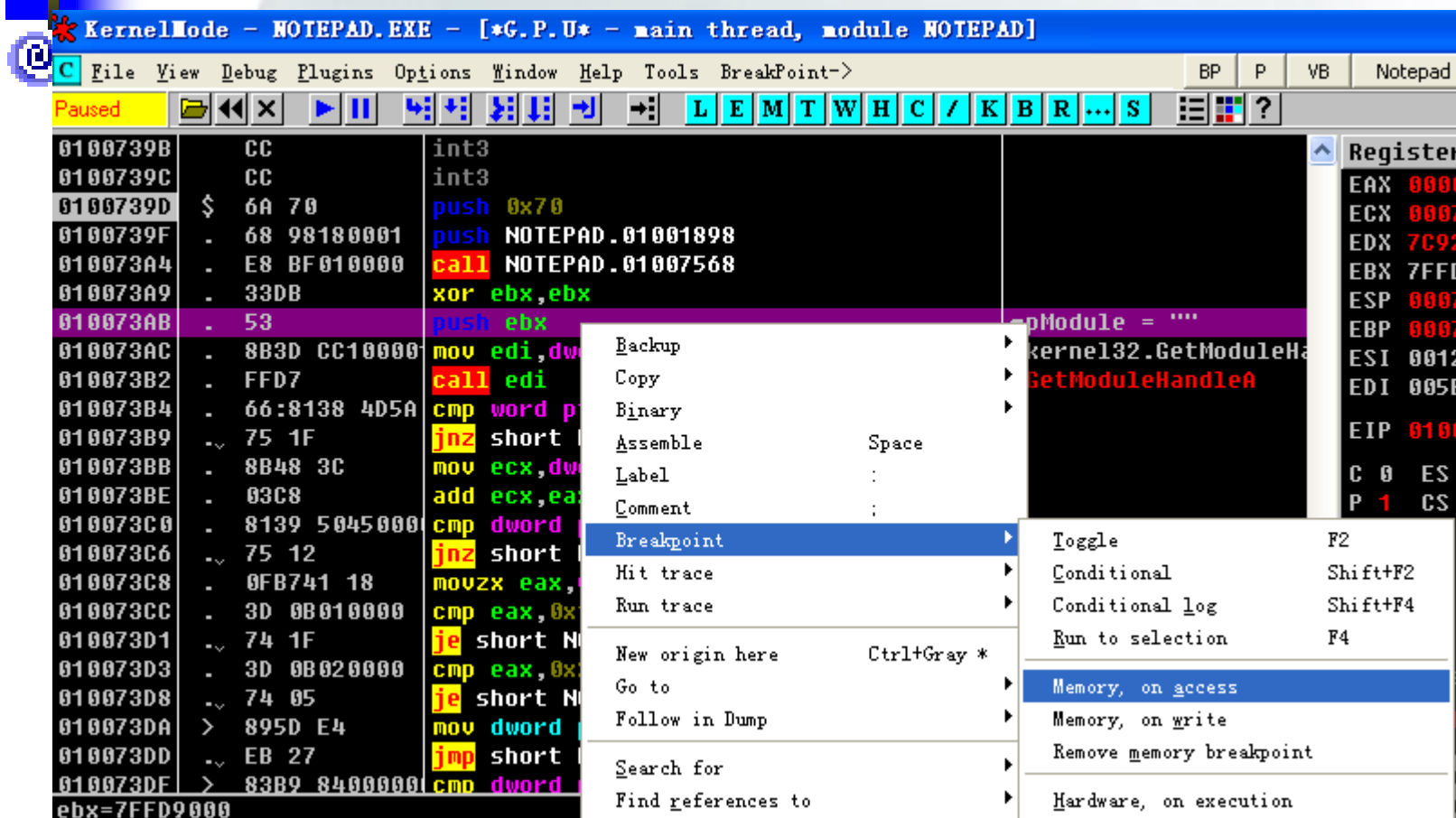
2. 常用工具简介

@ 内存断点

- ❑ 修改内存访问属性来触发内存访问错误而设置的断点称为内存断点。
- ❑ 设置内存断点只需鼠标右键并依次点击：**Breakpoint→Memory, on access/Memory, on write**。



2. 常用工具简介



2. 常用工具简介

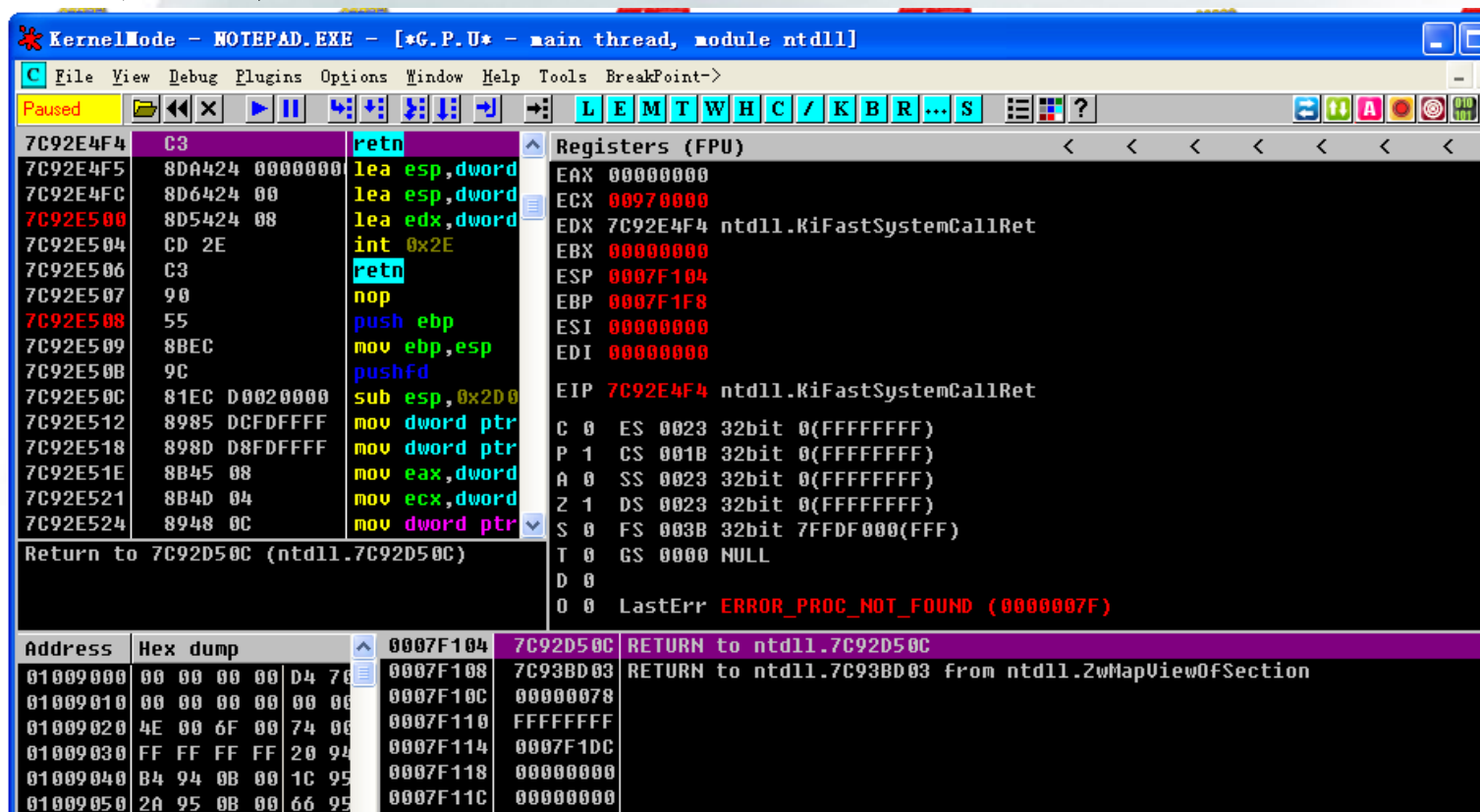
@跟踪运行

OllyDbg快捷键	功能
F4	运行到指定地址
F7	单步步进，遇到CALL跟进
F8	单步步过，遇到CALL跳过
F9	运行程序
Ctrl+F9	遇到第一个RET指令时中断



2. 常用工具简介

@跟踪运行



The screenshot displays a Windows debugger window titled "KernelMode - NOTEPAD.EXE - [*G.P.U* - main thread, module ntdll]". The interface includes a menu bar (File, View, Debug, Plugins, Options, Window, Help, Tools, BreakPoint->), a toolbar, and a status bar. The main window is divided into several panes:

- Assembly View:** Shows a list of instructions with their addresses and hex values. The current instruction is `retn` at address `7C92E4F4`.
- Registers (FPU):** Displays the state of various registers. For example, `EAX` is `00000000`, `ECX` is `00970000`, and `EIP` is `7C92E4F4`.
- Memory Dump:** Shows a hex dump of memory starting at address `01009000`. The dump includes hex values and their corresponding ASCII representations.

The assembly view shows the following instructions:

Address	Hex	Instruction
7C92E4F4	C3	<code>retn</code>
7C92E4F5	8D A4 24 00	<code>lea esp, dword [00000000]</code>
7C92E4FC	8D 64 24 00	<code>lea esp, dword [00000000]</code>
7C92E500	8D 54 24 00	<code>lea edx, dword [00000000]</code>
7C92E504	CD 2E	<code>int 0x2E</code>
7C92E506	C3	<code>retn</code>
7C92E507	90	<code>nop</code>
7C92E508	55	<code>push ebp</code>
7C92E509	8B EC	<code>mov ebp, esp</code>
7C92E50B	9C	<code>pushfd</code>
7C92E50C	81 EC D0 02 00 00	<code>sub esp, 0x200</code>
7C92E512	89 85 DC FD FF FF	<code>mov dword ptr [00000000], eax</code>
7C92E518	89 8D D8 FD FF FF	<code>mov dword ptr [00000000], ecx</code>
7C92E51E	8B 45 08	<code>mov eax, dword [00000000]</code>
7C92E521	8B 4D 04	<code>mov ecx, dword [00000000]</code>
7C92E524	89 48 0C	<code>mov dword ptr [00000000], eax</code>

The registers pane shows the following values:

Register	Value
EAX	00000000
ECX	00970000
EDX	7C92E4F4
EBX	00000000
ESP	0007F104
EBP	0007F1F8
ESI	00000000
EDI	00000000
EIP	7C92E4F4

The memory dump shows the following data:

Address	Hex	ASCII
01009000	00 00 00 00 D4 76	
01009010	00 00 00 00 00 00	
01009020	4E 00 6F 00 74 00	
01009030	FF FF FF FF 20 94	
01009040	B4 94 0B 00 1C 95	
01009050	2A 95 0B 00 66 95	



2. 常用工具简介

@地址跳转

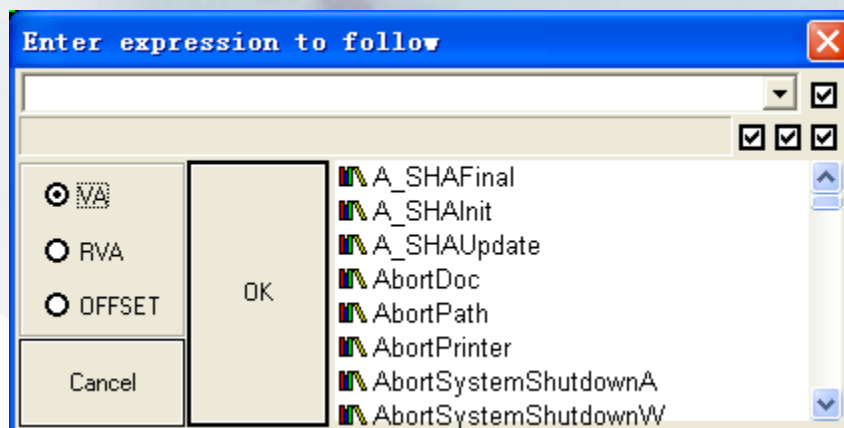
- ❑ 在OD中按下快捷键**Ctrl+G**可以直接跳转到对应的地址处。
- ❑ 如果要查看某个地址的数据，那么点击一下数据窗口，然后按下快捷键便可跳转到对应地址。
- ❑ 同理，如果要查看汇编代码或是堆栈数据，采取同样的操作即可。



2. 常用工具简介

@地址跳转

- ❑ 执行 **Go to(Ctrl+G)**命令,打开一个**Enter expression to follow**(对话框),输入地址, 单击**OK**按钮。



2. 常用工具简介

@堆栈相关操作

- ❑ 堆栈区显示的数据总是从栈顶指针**ESP**处显示
 - 选择**Address -> Relative to ESP**（子菜单一般只显示上面4个中的3个，根据需要，动态的变化），这时堆栈窗口就以**ESP**为指引，显示你想要的数。
 - 随着程序单步执行时（**F7**或**F8**），窗口中**ESP**总是滚动的，总是将栈顶置于顶部
 - 在观察某个地址处数据的变动时会不方便，如果你不想滚动，而是想观察某个位置上的堆栈值，这时可以使用**Lock Stack**来锁定滚动。



2. 常用工具简介

@ 堆栈相关操作

The screenshot displays the Immunity Debugger interface with the following components:

- Assembly View:** Shows assembly instructions for the `7C92E8C6` to `7C92E8F1` range. Instructions include `push esi`, `push edi`, `mov eax, dword ptr ss:[ebp-0x8]`, `mov dword ptr ss:[ebp-0x18], esp`, `push eax`, `mov eax, dword ptr ss:[ebp-0x4]`, `mov dword ptr ss:[ebp-0x4], -0x1`, `mov dword ptr ss:[ebp-0x8], eax`, `lea eax, dword ptr ss:[ebp-0x10]`, `mov dword ptr fs:[0], eax`, `ret`, `mov ecx, dword ptr ss:[ebp-0x10]`, `mov dword ptr fs:[0], ecx`, `pop ecx`, and `pop edi`.
- Registers View:** Shows the state of registers. `EIP` is `7C92E8C3`. `EDX` is `00000000`. `ESI` is `00000000`. `EIP` is `7C92E8C3`. `ntdll.7C9302C5` and `ntdll.7C930308` are listed. `eax=00000000` and `esp=0007F118` are shown.
- Hex Dump View:** Shows a hex dump of memory starting at address `01009000`. The dump contains data in hexadecimal and ASCII format.
- Registers View (Right):** Shows the state of registers. `EIP` is `7C92E8C3`. `ntdll.7C9302C5` and `ntdll.7C930308` are listed. `eax=00000000` and `esp=0007F118` are shown.
- Registers View (Bottom):** Shows the state of registers. `EIP` is `7C92E8C3`. `ntdll.7C9302C5` and `ntdll.7C930308` are listed. `eax=00000000` and `esp=0007F118` are shown.



2. 常用工具简介

- ❑ IDApro
- ❑ OllyDbg
- ❑ WinDbg
- ❑ GDB



2. 常用工具简介

WinDbg

- ❑ 微软发布的一款免费调试工具，支持用户模式调试和内核模式调试，主要用于内核模式调试



2. 常用工具简介

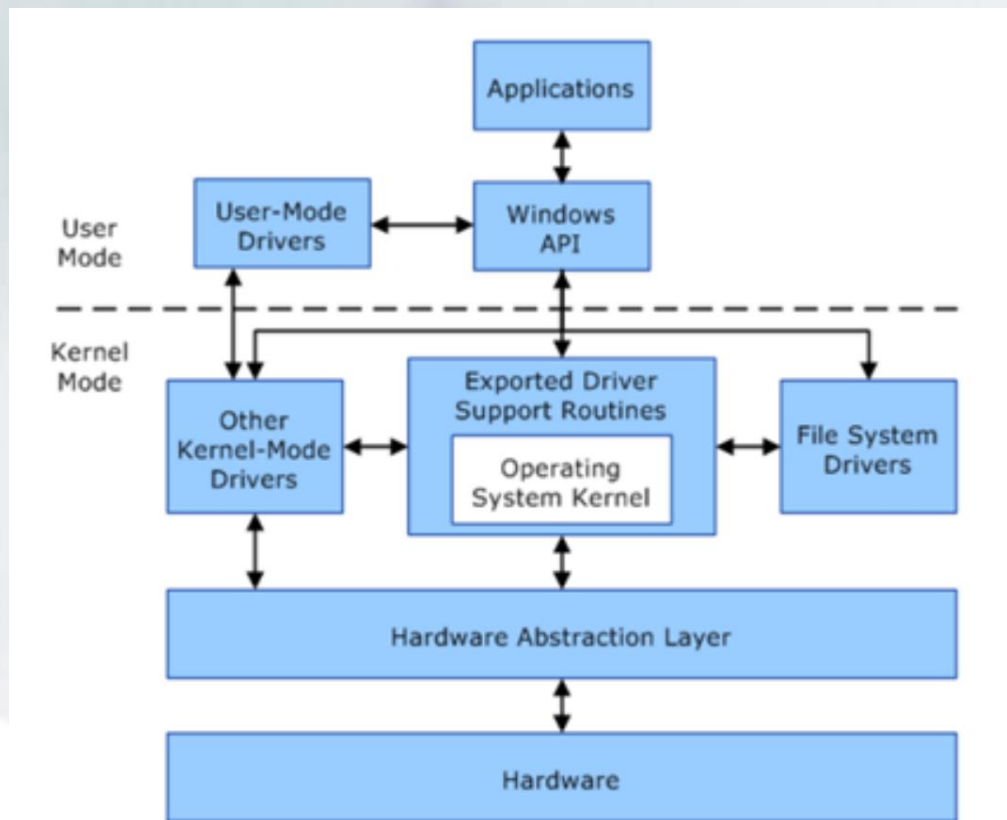
@ “用户模式” 和 “内核模式”

❑ 运行 **Windows** 计算机中的处理器有两个不同模式：“用户模式” 和 “内核模式”。

- 根据处理器上运行的代码的类型，处理器在两个模式之间切换。应用程序在用户模式下运行，核心操作系统组件在内核模式下运行。多个驱动程序在内核模式下运行，但某些驱动程序在用户模式下运行。
- 在内核模式下运行的所有代码都共享单个虚拟地址空间。如果内核模式驱动程序意外写入错误的虚拟地址，则属于操作系统或其他驱动程序的数据可能会受到损坏。



2. 常用工具简介



2. 常用工具简介

② “用户模式”和“内核模式”

- 当启动用户模式的应用程序时，**Windows** 会为该应用程序创建“进程”。进程为应用程序提供专用的“虚拟地址空间”和专用的“句柄表格”。
- 应用程序的虚拟地址空间为专用空间，一个应用程序无法更改属于其他应用程序的数据。每个应用程序都孤立运行。
- 在用户模式下运行的处理器无法访问为操作系统保留的虚拟地址，防止应用程序更改并且可能损坏关键的操作系统数据。



2. 常用工具简介

@ 开始调试

- ❑ 用WinDbg调试程序和OD一样，可以打开一个可执行文件来调试，或是附加到正在运行的进程上来调试。
- ❑ 具体操作为：**File→Open Executable/Attach to a Process。**
- ❑ 终止调试的操作为：**Debug→Stop Debugging/Detach Debuggee。**



2. 常用工具简介

@ WinDbg窗口的名称与用途如下

名称	热键	用途
Command	Atl+1	输入命令、显示命令结果和调试信息输出
Watch	Alt+2	观察指令全局变量、局部变量和寄存器的信息
Locals	Alt+3	自动显示当前函数的所有局部变量
Registers	Atl+4	观察和修改寄存器的值
Memory	Alt+5	观察和修改内存数据
Call Stack	Alt+6	栈中记录的函数调用序列
Disassembly	Alt+7	反汇编
Scratch Pad	Alt+8	白板，可以用来做调试笔记等
Processes and Threads	Alt+9	显示所有调试目标的列表，包括进程和线程等
Command Browser	Alt+N	执行和浏览命令



2. 常用工具简介

@断点相关

- ❑ 在WinDbg中下断点的命令为: **<bp address 或ModuleName!APIName>**
- ❑ 例如**bp 401000**,
- ❑ 例如**bp kernel32!GetCommandLineW**

断点相关命令	命令功能
bl	列出所有断点
bd/be	禁止/启用断点（Disable），bd 1表示禁止1号断点；be 1表示启用1号断点
bc	清除断点，bc 1表示清除1号断点，清除多个可用‘，’将断点编号隔开；用*匹配待清除断点，用-表示范围；
bp\$exentry	WinDbg中的伪寄存器记录了程序入口点，该命令表示在程序入口处下断点。



2. 常用工具简介

@ 内存查看

□ WinDbg提供了非常丰富的内存查看命令

○ d[类型] [地址范围]格式

命令	功能
db	以字节和ASCII的格式列出内存信息
dw	以双字节WORD格式列出内存信息
dd	以四字节DWORD格式列出内存信息
da	以ASCII字符串格式列出内存信息
du	以Unicode字符串格式列出内存信息



2. 常用工具简介

@ 指令反汇编

❑ 命令 **u (Unassemble)** 用来对指令进行反汇编

命令	功能
u	显示当前指令之后的若干指令
ub	显示当前指令之前的若干指令
uaddress	显示地址之后的指令
uL20	显示20行指令，该用法可以和上面两条指令结合起来使用，如u address L20将显示指定地址后的20行指令



2. 常用工具简介

@跟踪运行

□ WinDbg中使用命令g可以使程序开始运行，跟踪运行的命令主要为p、t，

p相关命令	功能
p	单步执行，遇到call步过
pa	执行到指定地址
pc	执行到下一个call的入口
ph	执行到下一个跳转处
pt	执行到返回

t相关命令	功能
t	单步执行，遇到call跟进
ta	单步执行到指定地址
tb	执行到分支处
tc	执行到下一个call的开始处



2. 常用工具简介

@堆栈信息

- ❑ 查看堆栈信息的命令为**ddsesp**，该命令将打印从从栈顶开始打印栈内的数据。
- ❑ 查看函数调用栈信息的命令为**kb**，默认会显示若干条函数调用信息，也可以指定显示的数目



2. 常用工具简介

@ 模块信息

❑ 命令 **lm[选项][a Address][m Pattern | M pattern]** 会列出当前已加载的所有模块。

- 使用参数 **a**，会列出指定地址所在的模块；
- 使用参数 **m**，后面跟一个表示模块名的通配符，如 **lm m *Name*** 将列出所有包含 **Name** 字符串的模块。



2. 常用工具简介

@ 字符串搜索

- 1. 寻找内存泄露的线索。比如知道当前内存泄露的内容是一些固定的字符串，就可以在**DLL** 区域搜索这些字符串出现的地址，然后再搜索这些地址用到什么代码中，找出这些内存是在什么地方开始分配的。
- 2. 寻找错误代码的根源。比如知道当前程序返回了 **0x80074015** 这样的一个代码，但是不知道这个代码是由哪一个内层函数返回的。就可以在代码区搜索 **0x80074015**，找到可能返回这个代码的函数。



2. 常用工具简介

@ 字符串搜索

- ❑ **s (Search)** 命令用于搜索内存，查找指定模板。
- ❑ **s -sa**与**s -su**命令用于搜索未指定的**ASCII**和**Unicode**字符串，可以检查一段内存中是否包含可打印字符。
- ❑ **s -a**与**s -u**命令用于搜索指定的**ASCII**和**Unicode**字符串。



2. 常用工具简介

GDB

□ **UNIX**下的调试工具，和**WinDbg**一样，也是通过命令行的形式来对程序进行调试。



2. 常用工具简介

@ 启动调试

□ **gdb fileName**

- 该命令将启动并调试名为 '**fileName**' 的程序，也可以在输入命令 **gdb** 后输入 **file** 命令来启动程序开始调试。

□ **gdb processName PID**

- 该命令会附加到 **PID** 所表示的进程上并调试该进程。相同功能的命令还有 **gdb -p <PID>**。



2. 常用工具简介

@断点相关

命令	功能
b *address	在地址address处设置断点
b functionName	在函数functionName入口处设置断点。
info b	列出所有断点
d Num (delete breakpoint)	删除编号为Num的断点



2. 常用工具简介

@ 查看内存

□ 查看内存的命令为：**x /nfu <address>**

○ 其中n表示要显示的内存单元的个数

○ f指示以什么样的格式显示

○ u则表示一个地址单元的长度

参数	取值	功能
f	x	以十六进制格式显示
	d	以十进制格式显示
	o	以八进制格式显示
	c	以字符格式显示
	i	以地址指令的格式显示
u	b	单字节
	h	双字节
	w	四字节（默认值）
	q	八字节



2. 常用工具简介

@ 调试跟踪

命令	功能
r	启动并运行程序
c	运行程序到下个断点
ni<N>	单步执行，遇到call指令步过，加上参数N可往后执行N行指令
si<N>	单步执行，遇到call指令跟进，加上参数N看往后执行N行指令
finish	执行完当前函数后停止
until<address>	执行完当前循环后停止，若加上地址参数后调试器将执行到指定地址停止



2. 常用工具简介

@ 其他常用命令

命令	功能
detach	断开与当前正在调试进程之间的链接
i r (info register)	列出所有寄存器以及寄存器的值
p /f \$eax	按照格式f显示指定寄存器的值
bt	显示栈帧信息，可指定显示的数量
disass \$pc (disasmble)	反汇编当前指令所在的函数



2. 常用工具简介

@ 插件介绍

□ **peda**是**GDB**的一个插件，使用该插件可以很大的提高**GDB**的使用体验，能够实时的列出寄存器、堆栈以反汇编代码。该插件的常用命令：

命令	功能
start	停在程序入口处
goto	执行到指定地址
session save/restore fileName	保存/恢复断点信息
finish	执行到函数返回地址处
checksec	查看程序开启的保护措施
vmmap	查看区段地址信息
find	查找字符串
pattern_create	构造非重复字符串
pattern_offset	计算偏移



2. 常用工具简介

@ 小结

- 本节对几种逆向分析中常用的工具的基本用法进行了介绍
- 工具的使用不在多而在精，相信熟练掌握这些工具后逆向分析能力会有很大的提升。
- 这些工具的具体使用及一些技巧将在后面章节的具体实例中有所体现。



2. 常用工具简介

@ 参考资料

- ❑ [1] <https://baike.baidu.com/item/%E9%80%86%E5%90%91%E5%B7%A5%E7%A8%8B/5097433?fr=aladdin>
- ❑ [2] <http://blog.csdn.net/huidurelease/article/details/65440501>
- ❑ [3] 逆向工程核心原理
- ❑ [4] 恶意代码分析实战





Q & A

谢谢!