# Windbg常见问题-指令解法大全

Windbg common commands for common problems

By lichao89047

QQ:571652571

2015/11/8

目录

[Windbg常见问题-指令解法大全 1](#_Toc446684258)

[写在前面的话 5](#_Toc446684259)

[程序逻辑 7](#_Toc446684260)

[Windbg和C语法区别 7](#_Toc446684261)

[as宏定义 8](#_Toc446684262)

[变量和操作符 9](#_Toc446684263)

[正则表达式 9](#_Toc446684264)

[命令流程控制 10](#_Toc446684265)

[汇编&反汇编 11](#_Toc446684266)

[怎样打印某函数调用关系 11](#_Toc446684267)

[怎样显示函数指令数？ 12](#_Toc446684268)

[如何在X64系统中实现64位执行模式和虚拟86执行模式(wow)切换 12](#_Toc446684269)

[如何强制为16位反汇编？ 13](#_Toc446684270)

[如何爆搜某种模式的反汇编指令？ 14](#_Toc446684271)

[如何在由任意地址正确反汇编该地址附近的指令？ 14](#_Toc446684272)

[怎样查找某地址附近的符号 15](#_Toc446684273)

[指令执行&跟踪 16](#_Toc446684274)

[怎样执行/跟踪到本函数或上级函数返回？ 16](#_Toc446684275)

[怎样执行/跟踪到指定地址？ 16](#_Toc446684276)

[怎样执行/跟踪到下一个分支指令？ 16](#_Toc446684277)

[如何跟踪某函数执行过的所有子函数？ 16](#_Toc446684278)

[回溯栈 18](#_Toc446684279)

[如何查看wow64进程回溯栈？ 18](#_Toc446684280)

[断点设置 19](#_Toc446684281)

[如何在物理地址下断？ 19](#_Toc446684282)

[如何对照IDA地址下断？ 19](#_Toc446684283)

[如何在针对线程/进程下断？ 20](#_Toc446684284)

[Ntfs文件操作断点 （不通用形式） 20](#_Toc446684285)

[如何对形如Gen\*的函数下断？ 20](#_Toc446684286)

[如何对pe所有导出函数下断？ （不通用形式） 20](#_Toc446684287)

[如何在驱动入口下断？ （不通用形式） 21](#_Toc446684288)

[如何正确地下字符串断点？ 21](#_Toc446684289)

[异常&事件 22](#_Toc446684290)

[如何在加载模块后暂停在Windbg中？ 22](#_Toc446684291)

[线程进程 23](#_Toc446684292)

[如何设置内核态进程/线程上下文？ 24](#_Toc446684293)

[如何暂停/恢复线程执行？ 25](#_Toc446684294)

[如何切换到可执行进程/线程？ 25](#_Toc446684295)

[如何遍历模块？ 26](#_Toc446684296)

[如何遍历进程？ 26](#_Toc446684297)

[如何遍历线程？ 26](#_Toc446684298)

[如何遍历寄存器？ 27](#_Toc446684299)

[如何遍历系统句柄表？ 27](#_Toc446684300)

[如何列出所有进程EPROCESS地址？ 27](#_Toc446684301)

[如何对模块排序 27](#_Toc446684302)

[如何获取进程名、进程ID 对应的进程对象 28](#_Toc446684303)

[PE相关 30](#_Toc446684304)

[如何查看某映像(sys exe dll)的版本号、时间、公司等信息？ 30](#_Toc446684305)

[如何显示pe头信息？ 30](#_Toc446684306)

[如何查找内存中的PE头？ 31](#_Toc446684307)

[符号{结构体,函数,...}查看 32](#_Toc446684308)

[如何列出以T开头的模块=？ 32](#_Toc446684309)

[如何查看所有前缀为Rtl的符号？ 32](#_Toc446684310)

[如何查看SEH链 33](#_Toc446684311)

[指定基址如何查看结构体成员数值？ 33](#_Toc446684312)

[如何打印内核单向/双向链表？ 33](#_Toc446684313)

[如何获取某结构体大小？ 34](#_Toc446684314)

[如何打印STRING, ANSI\_STRING, UNICODE\_STRING结构？ 34](#_Toc446684315)

[如何查看进程环境块PEB结构？ 34](#_Toc446684316)

[如何查看线程环境块TEB结构？ 35](#_Toc446684317)

[如何查看内核进程控制块？ 36](#_Toc446684318)

[如何打印系统服务表SSDT, SSSDT? 37](#_Toc446684319)

[如何打印用户态回调表KernelCallbackTable? 38](#_Toc446684320)

[如何查看系统中断表？ 38](#_Toc446684321)

[如何查看指定地址所属模块？ 39](#_Toc446684322)

[如何快速加载/卸载指定符号 39](#_Toc446684323)

[句柄和对象 40](#_Toc446684324)

[如何根据 基址、名称获取对象(OBJECT)信息？ 40](#_Toc446684325)

[如何查看驱动对象、设备对象、文件对象信息？ 41](#_Toc446684326)

[如何根据句柄获取对象信息？ 41](#_Toc446684327)

[如何显示所有ObjectType类型名？ 42](#_Toc446684328)

[注册表信息 44](#_Toc446684329)

[如何查看注册表项键值？ 44](#_Toc446684330)

[内存操作 45](#_Toc446684331)

[查看物理内存使用 46](#_Toc446684332)

[查看虚拟内存使用 46](#_Toc446684333)

[如何获取Fs:[0]所在地址？ 47](#_Toc446684334)

[如何查看某虚拟内存地址对应的物理内存地址？ 47](#_Toc446684335)

[如何查看某物理内存地址对应的虚拟内存地址？ 48](#_Toc446684336)

[如何查看地址所在虚拟内存位于哪个模块？ 48](#_Toc446684337)

[如何以固定字节模式填充内存？ 49](#_Toc446684338)

[如何拷贝虚拟内存块？ 49](#_Toc446684339)

[如何比较虚拟内存块？ 49](#_Toc446684340)

[如何将文件内容读取到调试器内存/从调试器内存写入文件？ 49](#_Toc446684341)

[如何搜索内存？ 49](#_Toc446684342)

[如何查看内存池信息？ 50](#_Toc446684343)

[如何查找指定Tag的内存池？ 50](#_Toc446684344)

[如何查看内存池使用情况？ 51](#_Toc446684345)

[如何查看内存堆信息？ 51](#_Toc446684346)

[如何显示虚拟内存块及访问权限 52](#_Toc446684347)

[如何查找内存中某类的对象实例？ 52](#_Toc446684348)

[特殊调试法 53](#_Toc446684349)

[如何用内核态调试器控制用户态调试器进程联合调试？ 53](#_Toc446684350)

[如何控制目标系统？ 55](#_Toc446684351)

[如何在调试程序时无缝切换调试器以及实现多调试器？ 55](#_Toc446684352)

[如何调试当前调试器？ 56](#_Toc446684353)

[如何用应用程序控制调试器？ 56](#_Toc446684354)

[如何使用IDA调试Windows内核和驱动？ 56](#_Toc446684355)

[其他 58](#_Toc446684356)

[如何查看最耗费时间片的线程？ 58](#_Toc446684357)

[如何加载和卸载插件？ 58](#_Toc446684358)

[如何快速替换驱动文件? 58](#_Toc446684359)

[读写gflag 59](#_Toc446684360)

[分析蓝屏dump 59](#_Toc446684361)

[显示当前使用的系统定时器 59](#_Toc446684362)

[命令：!mapped\_file 60](#_Toc446684363)

[清屏 60](#_Toc446684364)

[如何输出pdb中所有结构体？ 61](#_Toc446684365)

[如何让Windbg识别已知然而不存在于当前调试环境的结构体？ 61](#_Toc446684366)

[如何查看错误代码含义？ 61](#_Toc446684367)

[如何扩展a指令为64位汇编？ 61](#_Toc446684368)

## 写在前面的话

Windbg符号设置：

设置系统变量\_NT\_SYMBOL\_PATH为SRV\*e:\symbol\*http://msdl.microsoft.com/download/symbols e:\symbol设置为你要存储pdb符号文件的目录

设置交互式插件扩展：

将winxp目录下的插件Kdexts.dll，拷贝到winext下，即可开启amli模式，可交互

## 程序逻辑

### Windbg和C语法区别

|  |  |  |
| --- | --- | --- |
|  | Windbg | C |
| 自由变量 | @$t1, @$t2, @$t3,, @$t19 | Int i,j,k,..... |
| 赋值 | r@$t1=0;r@$t2=@$t1 | i=0;j=i |
| 解引用 | Poi(@$t1) | \*(int\*)i |
| 宏定义 | as Name Val | #define Name Val |
| 打印字符串 | .echo str | puts(str) |
| 格式化输出 | .printf “%?%?%?”,arg1,args,... | printf(“%?%?%?”,arg1,arg2,...) |

% 指针

%ma ASCII字符串

%mu UNICODE字符串

%msa ANSI\_STRING字符串

%msu UNICODE\_STRING字符串

格式化输出：

命令：.formats 表达式

例：

0:000> .formats 1c407e62

Evaluate expression:

Hex: 1c407e62

Decimal: 473988706

Octal: 03420077142

Binary: 00011100 01000000 01111110 01100010

Chars: .@~b

Time: Mon Jan 07 15:31:46 1985

Float: low 6.36908e-022 high 0

Double: 2.34182e-315

预设宏：

$ntnsym ntoskrnl基址

$ntwsym ntdll基址

$ntsym 根据用户态/内核态自动选择$ntnsym/$ntwsym

? 计算普通masm表达式

?? 计算C++表达式

例：

0:000> ?? ((\_PEB\*)0x7f2cf000)->ImageBaseAddress

void \* 0x001f0000

显示所有寄存器 r

显示寄存器 r@寄存器名

修改寄存器 r@寄存器名=值

读写MSR寄存器 wrmsr rdmsr

### as宏定义

1. 如何使用as进行宏定义？

命令： as 宏名 字符串

as /ma 宏名 ASCII字符串地址

as /mu 宏名 UNICODE字符串地址

as /msa 宏名 ANSI\_STRING字符串地址

as /msu 宏名 UNICODE\_STRING字符串地址

as /x 宏名 表达式

as /f 宏名 文件 宏代文件内容

as /c 宏名 命令 宏代命令结果

1. 如何控制是否开启as宏定义展开？

命令：.block {命令}

as定义的宏，必须和展开所在表达式用block分开

1. 如何控制as宏定义展开结果，结果用result表示

命令： ${宏名} 等价于c语言：

#ifdef宏名

result=宏展开

#else

result=${/n:宏名}——字符串本身

#endif

${/d:宏名} 等价于c语言：

#ifdef宏名

result=1

#else

result=0

#endif

${/f:宏名} 等价于c语言：

#ifdef宏名

result=宏展开

#else

result=空字符串

#endif

${/n:宏名} 等价于c语言：

#ifdef宏名

result=宏名

#else

result=${/n:宏名}——字符串本身

#endif

${/n:宏名} 等价于c语言：

#ifdef宏名

result=宏名

#else

result=${/n:宏名}——字符串本身

#endif

${/v:宏名} 等价于：${/n:宏名}——字符串本身

提示：有了as 和 ${}的控制，就能控制多种字符串格式转换为ascii字符串，因此多数情况下命令只支持ascii字符串即可

### 变量和操作符

|  |  |
| --- | --- |
| $exentry | 进程入口点地址 |
| $proc | PEPROCESS地址 |
| $thread | PETHREAD地址 |
| $peb | PEB地址 |
| $teb | TEB地址 |
| $tpid | 当前线程所属进程Id |
| $tid | 当前线程Id |
| $bp断点号 | 该断点地址 |

数进制：

默认接受十六进制数，若输入十进制则需要在前面加0n

Masm和c++表达式对照表：

|  |  |  |  |
| --- | --- | --- | --- |
| Masm | C++ | Masm | C++ |
| not | ! | dwo | \*(DWORD\*) |
| hi | HIWORD() | qwo | \*(ULONGLONG\*) |
| low | LOWORD() | poi | \*(PVOID\*) |
| by | \*(BYTE\*) | wo | \*(WORD\*) |
| = | == | and | & |
| Xor | ^ | or | | |

Masm库函数：

|  |  |
| --- | --- |
|  |  |
| $iment(Address) | 由映像基址获取模块入口点地址 |
| $scmp(“str1”,”str2”) | strcmp |
| $sicmp(“str1,”str2”) | stricmp |
| $spat(“str1”,”pattern”) | 匹配正则表达式 |
| $vvalid(Address,Length) | 探测一块内存有效性 |

支持的c++宏：

|  |  |
| --- | --- |
| #CONTAINING\_RECORD(Address, Type, Field) | 内核LIST\_ENTRY结构常用宏 |
| #FIELD\_OFFSET(Type, Field) | &(((type\*)0)->member)取成员偏移 |
| #RTL\_CONTAINS\_FIELD (Struct, Size, Field) | 探测成员是否存在 |
| #RTL\_FIELD\_SIZE(Type, Field) | 由成员名返回成员大小 |

### 正则表达式

若命令可以用正则表达式，则下列规则成立：

\* 代0~∞个字符

? 代1个字符

[] 代1个字符，该字符可以是“[]”之间的任何一个，“-”符可以指定范围，例如“a-z”

# 代0~∞个字符的前缀

+ 代1~∞个字符

### 命令流程控制

判断逻辑：

.if (条件) {命令}

.if (条件) {命令} .else{命令}

.if (条件) {命令} .elsif(条件){命令}

.if (条件) {命令} .elsif(条件){命令} .else{命令}

循环逻辑：

.for(命令;条件;命令){命令}

.foreach (变量 {命令1}){命令2} 对命令1执行的每一条结果(空格或换行分开)，执行命令2

.foreach /s (变量 “字符串”){命令} 对字符串每条子串 (空格或换行分开)，执行命令2

.foreach /f (变量 “文件路径”){命令} 对文件中每条字符串 (空格或换行分开)，执行命令2

.while(条件) {命令}

.do{命令}(条件)

.break 用于.for .while .do中打破循环

.continue 用于.for .while .do中跳过本次循环

j 表达式 命令1; 命令2 等价于：.if (表达式!=0) {命令1} .else{命令2}

命令; z(表达式) 等价于：.do{命令}(表达式!=0)

异常处理：

.catch{命令} 相当于c语言的：

try{命令}

catch(...){}

{}

.leave 从.catch块中跳出

## 汇编&反汇编

命令： u 地址 [长度] 反汇编之后代码

Ub地址 [长度] 反汇编之前代码

Up地址 [长度] 从物理地址反汇编

Uf 地址 反汇编当前函数

命令： a 地址 在指定地址处写入汇编 16位

### 怎样打印某函数调用关系

命令：uf /c /D 地址 打印当前函数对其他函数的调用

适用范围：用户态/内核态

例：

kd> uf /c /D 0x804fa5e6

nt!KeDelayExecutionThread (804fa5e6)

nt!KeDelayExecutionThread+0x8f (804fa675):

call to nt!KiUnlockDispatcherDatabase (80542748)

nt!KeDelayExecutionThread+0xe9 (804fa6cf):

call to nt!KiInsertTreeTimer (80500f62)

nt!KeDelayExecutionThread+0x116 (804fa6fc):

call to nt!KiSetPriorityThread (80501bba)

nt!KeDelayExecutionThread+0x12f (804fa715):

call to nt!KiFindReadyThread (80501894)

nt!KeDelayExecutionThread+0x19f (804fa785):

call to nt!KiActivateWaiterQueue (804fc02a)

nt!KeDelayExecutionThread+0x1c4 (804fa7aa):

call to nt!KiSwapThread (80501ca0)

nt!KeDelayExecutionThread+0x1de (804fa7c4):

call to nt!KiComputeWaitInterval (804fa504)

nt!KeDelayExecutionThread+0x1e6 (804fa7cc):

call to hal!KeRaiseIrqlToDpcLevel (806d3298)

nt!KeDelayExecutionThread+0x26a (804fa850):

call to nt!KiUnlockDispatcherDatabase (80542748)

命令：# 函数名 起始地址 l长度 打印在某段地址范围内代码对该函数的引用

适用范围：内核态/用户态

例：

kd> # IopCreateFile 840554ae l10000

nt!NtCreateFile+0x2f:

840554dd e87340ffff call nt!IopCreateFile (84049555)

nt!IoCreateFileEx+0x99:

84081442 e80e81fcff call nt!IopCreateFile (84049555)

nt!NtOpenFile+0x25:

84084c97 e8b948fcff call nt!IopCreateFile (84049555)

### 怎样显示函数指令数？

命令：uf /i /m 地址

适用范围：用户态/内核态

例：

kd> uf /i ntcreatefile

21 instructions scanned

nt!NtCreateFile:

8056f2fc 8bff mov edi,edi

8056f2fe 55 push ebp

8056f2ff 8bec mov ebp,esp

8056f301 33c0 xor eax,eax

8056f303 50 push eax

8056f304 50 push eax

8056f305 50 push eax

8056f306 ff7530 push dword ptr [ebp+30h]

8056f309 ff752c push dword ptr [ebp+2Ch]

8056f30c ff7528 push dword ptr [ebp+28h]

8056f30f ff7524 push dword ptr [ebp+24h]

8056f312 ff7520 push dword ptr [ebp+20h]

8056f315 ff751c push dword ptr [ebp+1Ch]

8056f318 ff7518 push dword ptr [ebp+18h]

8056f31b ff7514 push dword ptr [ebp+14h]

8056f31e ff7510 push dword ptr [ebp+10h]

8056f321 ff750c push dword ptr [ebp+0Ch]

8056f324 ff7508 push dword ptr [ebp+8]

8056f327 e860d8ffff call nt!IoCreateFile (8056cb8c)

8056f32c 5d pop ebp

8056f32d c22c00 ret 2Ch

### 如何在X64系统中实现64位执行模式和虚拟86执行模式(wow)切换

命令：!sw

适用范围：内核态/用户态

例：

0:000> .load wow64exts

0:000> !sw

Switched to Guest (WoW) mode

0:000:x86> ? .

Evaluate expression: 1995360060 = 76eec73c

0:000:x86> !sw

Switched to Host mode

0:000> ? .

Evaluate expression: 1994597202 = 00000000`76e32352

0:000> .load wow64exts

0:000> u .

wow64cpu!CpupSyscallStub+0x2:

00000000`76e32352 c3 ret

00000000`76e32353 cc int 3

00000000`76e32354 b80d0000c0 mov eax,0C000000Dh

00000000`76e32359 e93ef0ffff jmp wow64cpu!CpuSetContext+0x15c (00000000`76e3139c)

00000000`76e3235e 488b876c010000 mov rax,qword ptr [rdi+16Ch]

00000000`76e32365 48898370010000 mov qword ptr [rbx+170h],rax

00000000`76e3236c 488b8774010000 mov rax,qword ptr [rdi+174h]

00000000`76e32373 48898378010000 mov qword ptr [rbx+178h],rax

0:000> !sw

Switched to Guest (WoW) mode

0:000:x86> u 00000000`76e32352

wow64cpu!CpupSyscallStub+0x2:

76e32352 c3 ret

76e32353 cc int 3

76e32354 b80d0000c0 mov eax,0C000000Dh

76e32359 e93ef0ffff jmp wow64cpu!CpuSetContext+0x15c (76e3139c)

76e3235e 48 dec eax

76e3235f 8b876c010000 mov eax,dword ptr [edi+16Ch]

76e32365 48 dec eax

76e32366 898370010000 mov dword ptr [ebx+170h],eax

提示：也可手动修改cs以达到相同效果

### 如何强制为16位反汇编？

命令：ur 地址

适用范围：用户态/内核态

例：

kd> u .

nt!ExpInterlockedPopEntrySListEnd+0x8:

80542e37 c3 ret

nt!ExInterlockedPushEntrySList:

80542e38 8f0424 pop dword ptr [esp]

80542e3b 90 nop

nt!InterlockedPushEntrySList:

80542e3c 53 push ebx

80542e3d 55 push ebp

80542e3e 8be9 mov ebp,ecx

80542e40 8bda mov ebx,edx

80542e42 8b5504 mov edx,dword ptr [ebp+4]

kd> ur .

nt!ExpInterlockedPopEntrySListEnd+0x8:

80542e37 c3 ret

nt!ExInterlockedPushEntrySList:

80542e38 8f04 pop word ptr [si]

80542e3a 2490 and al,90h

nt!InterlockedPushEntrySList:

80542e3c 53 push bx

80542e3d 55 push bp

80542e3e 8be9 mov bp,cx

80542e40 8bda mov bx,dx

80542e42 8b5504 mov dx,word ptr [di+4]

### 如何爆搜某种模式的反汇编指令？

命令：# 查找模式 起始地址 [l长度]

参数：查找模式为正则表达式，可以匹配该处反汇编代码，或其对应的16进制机器码

适用范围：内核态/用户态

例：

0:000> u .

ntdll!LdrpDoDebuggerBreak+0x2b:

76f63bad 6c ins byte ptr es:[edi],dx

76f63bae 006900 add byte ptr [ecx],ch

76f63bb1 6300 arpl word ptr [eax],ax

76f63bb3 68006b0069 push 69006B00h

76f63bb8 006e00 add byte ptr [esi],ch

76f63bbb 670000 add byte ptr [bx+si],al

76f63bbe 0000 add byte ptr [eax],al

76f63bc0 00f9 add cl,bh

匹配反汇编：push 69006B00h

0:000> # push\*69 .

ntdll!LdrpDoDebuggerBreak+0x31:

76f63bb3 68006b0069 push 69006B00h

匹配机器码：68006b0069

0:000> # 68\*6b .

ntdll!LdrpDoDebuggerBreak+0x31:

76f63bb3 68006b0069 push 69006B00h

### 如何在由任意地址正确反汇编该地址附近的指令？

问题描述：假设知道某地址840554b2，如下左边是该地址处反汇编，右边是正确的指令地址反汇编，显然该处不是一条指令的开始地址，此时如何仅由该地址得到正确的函数反汇编？传统的方式是前向反汇编，试探法，这里介绍另一种方法，在知道函数起始地址的前提下：

命令：.dml\_flow 函数起始地址 目标地址

适用范围：用户态/内核态

kd> u 840554b2

nt!NtCreateFile+0x4:

840554b2 ec in al,dx 840554ae 8bff mov edi,edi

840554b3 51 push ecx 840554b0 55 push ebp

840554b4 33c0 xor eax,eax 840554b1 8bec mov ebp,esp

840554b6 50 push eax 840554b3 51 push ecx

840554b7 6a20 push 20h 840554b4 33c0 xor eax,eax

840554b9 50 push eax 840554b6 50 push eax

840554ba 50 push eax 840554b7 6a20 push 20h

例：

kd> .dml\_flow nt!NtCreateFile 840554b2

<No previous node>

nt!NtCreateFile (840554ae):

840554ae mov edi,edi

840554b0 push ebp

840554b1 mov ebp,esp

840554b3 push ecx

840554b4 xor eax,eax

840554b6 push eax

840554b7 push 20h

840554b9 push eax

840554ba push eax

840554bb push eax

### 怎样查找某地址附近的符号

命令：ln 地址

适用范围：内核态/用户态

例：

kd> ln nt!ntcreatefile-1

Browse module

Set bu breakpoint

(84055482) nt!SeValidateSecurityQos+0x2b | (840554ae) nt!NtCreateFile

## 指令执行&跟踪

指令跟踪(trace)和指令执行(execute)的区别在于对待函数调用指令(call)，跟踪会导致步入，而执行会导致步过

命令：t [=开始地址] [跟踪指令数] 执行指令

命令：t [=开始地址] [跟踪指令数] 跟踪指令

命令：g [=开始地址] [目标地址] 执行到某地址

命令：gc 从条件断点处开始执行

命令：gu 执行到上一级函数

### 怎样执行/跟踪到本函数或上级函数返回？

命令：tt n 跟踪到返回 n级

命令：pt n 执行到返回n级

### 怎样执行/跟踪到指定地址？

命令：ta [=开始地址] 结束地址

命令：pa [=开始地址] 结束地址

例：

kd> ta =kifastcallentry kifastcallentry+60

nt!KiFastCallEntry+0x5:

83e95325 6a30 push 30h

nt!KiFastCallEntry+0x7:

83e95327 0fa1 pop fs

nt!KiFastCallEntry+0x9:

83e95329 8ed9 mov ds,cx

nt!KiFastCallEntry+0xb:

83e9532b 8ec1 mov es,cx

nt!KiFastCallEntry+0xd:

83e9532d 648b0d40000000 mov ecx,dword ptr fs:[40h]

nt!KiFastCallEntry+0x14:

83e95334 8b6104 mov esp,dword ptr [ecx+4]

nt!KiFastCallEntry+0x17:

83e95337 6a23 push 23h

nt!KiFastCallEntry+0x19:

### 怎样执行/跟踪到下一个分支指令？

分支指令：指令可根据环境不同执行到不同的eip，比如条件跳转指令

命令：th n 跟踪到第n分支指令

命令：ph n 执行到第n分支指令

### 如何跟踪某函数执行过的所有子函数？

命令：wt

例：

kd> wt

Tracing testdriver2!func to return address f89cb070

8 0 [ 0] testdriver2!func

7 0 [ 1] nt!ExAllocatePool

89 0 [ 2] nt!ExAllocatePoolWithTag

5 0 [ 3] hal!KeRaiseIrqlToDpcLevel

197 5 [ 2] nt!ExAllocatePoolWithTag

9 202 [ 1] nt!ExAllocatePool

13 211 [ 0] testdriver2!func

85 0 [ 1] nt!ExFreePoolWithTag

19 296 [ 0] testdriver2!func

315 instructions were executed in 7 events (0 from other threads)

Function Name Invocations MinInst MaxInst AvgInst

hal!KeRaiseIrqlToDpcLevel 1 5 5 5

nt!ExAllocatePool 1 9 9 9

nt!ExAllocatePoolWithTag 1 197 197 197

nt!ExFreePoolWithTag 1 85 85 85

testdriver2!func 1 19 19 19

## 回溯栈

回溯栈用来记录每一级函数返回地址

命令：k 显示当前线程回溯栈

命令：kb 显示当前线程回溯栈并打印前3个(可能的)参数

命令：!stacks 打印所有进程的所有线程回溯栈

命令：!uniqstack 打印当前进程所有线程回溯栈

### 如何查看wow64进程回溯栈？

命令：!k

例：

0:000> .load wow64exts

0:000> !k

Walking Native Stack...

# Child-SP RetAddr Call Site

00 00000000`00e7e928 00000000`76e32318 wow64cpu!CpupSyscallStub+0x2

01 00000000`00e7e930 00000000`76df219a wow64cpu!Thunk0Arg+0x5

02 00000000`00e7e9e0 00000000`76df20d2 wow64!RunCpuSimulation+0xa

03 00000000`00e7ea30 00007fff`10093a15 wow64!Wow64LdrpInitialize+0x172

04 00000000`00e7ef70 00007fff`10072f1e ntdll!LdrpInitializeProcess+0x1591

05 00000000`00e7f290 00007fff`0ffe8ece ntdll!\_LdrpInitialize+0x89ffe

06 00000000`00e7f300 00000000`00000000 ntdll!LdrInitializeThunk+0xe

Walking Guest (WoW) Stack...

# ChildEBP RetAddr

00 00f7f868 76f1ce1b ntdll\_76eb0000!NtTerminateProcess+0xc

## 断点设置

bp 设置软件断点

bm 设置已加载符号断点 (/a 强制下断)

bu 设置未加载符号断点

ba 设置硬件断点

bl 列举断点

bd 禁用断点

be 启用断点

bc 清除断点

### 如何在物理地址下断？

如果在加载pe时采用了文件内存映射，那么一块物理内存会映射到不同虚拟内存，因此如果对方映射了多个相同的PE往往需要在不同虚拟地址下断，这里提出一种物理内存手动下断方式

适用范围：内核态

例：

kd> !pte 840554ae

VA 840554ae

PDE at C0602100 PTE at C04202A8

contains 00000000001DA063 contains 0000000004055121

pfn 1da ---DA--KWEV pfn 4055 -G--A--KREV

找到ntcreatefile的物理地址

kd> !db 40554ae

# 40554ae 8b ff 55 8b ec 51 33 c0-50 6a 20 50 50 50 ff 75 ..U..Q3.Pj PPP.u

# 40554be 30 ff 75 2c ff 75 28 ff-75 24 ff 75 20 ff 75 1c 0.u,.u(.u$.u .u.

# 40554ce ff 75 18 ff 75 14 ff 75-10 ff 75 0c ff 75 08 e8 .u..u..u..u..u..

# 40554de 73 40 ff ff 59 5d c2 2c-00 90 90 90 90 90 6a 40 s@..Y].,......j@

# 40554ee 68 28 42 e6 83 e8 70 51-e2 ff 8b 75 0c 8b 86 88 h(B...pQ...u....

# 40554fe 00 00 00 89 45 cc 8b 86-50 01 00 00 89 45 d0 8d ....E...P....E..

# 405550e 7d d8 89 7d d4 c6 45 e2-00 3b 75 08 74 33 8d 8e }..}..E..;u.t3..

# 405551e 70 02 00 00 8b 11 83 e2-fe 8d 42 02 8b f8 8b d9 p.........B.....

手动修改为软件断点

kd> !eb 40554ae cc

kd> g

Break instruction exception - code 80000003 (first chance)

nt!NtCreateFile:

840554ae cc int 3

中断后，需要手动改回物理内存

### 如何对照IDA地址下断？

若当前符号在IDA中地址为Va1，IDA View菜单 -> Open subviews -> Segments 中，查找到第一个节的虚拟地址Va1Begin，使用lm指令找到在当前内存中，该模块起始地址Va2Begin，则Va2=Va1 – Va1Begin + Va2Begin为所求

### 如何在针对线程/进程下断？

命令： bp /p EPROCESS地址 针对进程下断

bp /t ETHREAD地址 针对线程下断

适用范围：内核态

### Ntfs文件操作断点 （不通用形式）

拦截创建/打开文件

bp Ntfs!NtfsCommonCreate "du poi(poi(poi(poi(esp+8)+0x60)+0x18)+0x34);.echo \"FILE\_CREATE\_OR\_OPEN \n\";gc"

拦截普通删除

bp Ntfs!NtfsCommonSetInformation ".if poi(poi(poi(esp+8)+0x60)+0x8)==0xD {du poi(poi(poi(poi(esp+8)+0x60)+0x18)+0x34);.echo \"NORMAL\_DELETE \n\"} .else {gc}"

拦截NtDeleteFile

bp Ntfs!NtfsCommonCreate ".if (poi(poi(poi(esp+8)+0x60)+0x8)&0x1000)!=0 {du poi(poi(poi(poi(esp+8)+0x60)+0x18)+0x34);.echo \"FILE\_DELETE\_ON\_CLOSE \n\"};gc"

拦截设置文件

bp Ntfs!NtfsCommonSetInformation ".printf \"%d,%d\\n\",poi(poi(poi(esp+8)+0x60)),poi(poi(poi(esp+8)+0x60)+0x8);gc"

### 如何对形如Gen\*的函数下断？

命令：bm /a 符号

适用范围：内核态/用户态

0:000> bm /a ml64!Gen\*

1: 00000000`00c733c0 @!"ml64!genIntReloc"

2: 00000000`00c73694 @!"ml64!genDataDef"

3: 00000000`00c7160c @!"ml64!GenCodeJump"

4: 00000000`00c9a354 @!"ml64!genPrologue"

5: 00000000`00c73ef4 @!"ml64!GenCodeRet"

6: 00000000`00c9a620 @!"ml64!genEpilogue"

7: 00000000`00c73a60 @!"ml64!genNormReloc"

8: 00000000`00c71008 @!"ml64!GenCodeLoop"

9: 00000000`00c71710 @!"ml64!GenREXPrefix"

10: 00000000`00cda6d0 @!"ml64!genmcBuffT"

11: 00000000`00c71940 @!"ml64!GenCodeNormal"

12: 00000000`00c73434 @!"ml64!genReloc"

13: 00000000`00c98ffc @!"ml64!genProEpiMacroCall"

14: 00000000`00c73d00 @!"ml64!GenCodeString

### 如何对pe所有导出函数下断？ （不通用形式）

命令： 1. lm获取基址 base

2. 解析导出表 r@$t1=base+poi(base+poi(base+0x3c)+0x78)

3.遍历导出函数 .for(r@$t2=0;@$t2<poi(@$t1+0x18);r@$t2=@$t2+1) {bp base+poi(base+poi(@$t1+0x1c)+4\*@$t2)}

### 如何在驱动入口下断？ （不通用形式）

命令：在驱动加载之前，下断bp nt!MmLoadSystemImage "du poi(poi(esp+4)+4);r@$t1=poi(esp+0x18);gu;bp poi(@$t1)+poi(poi(@$t1)+poi(poi(@$t1)+0x3c)+0x28)"

### 如何正确地下字符串断点？

假设内存如下：

0:000> db .

76f63bad 6c 00 69 00 63 00 68 00-6b 00 69 00 6e 00 67 00 l.i.c.h.k.i.n.g.

76f63bbd 00 00 00 00 f9 ff c3 90-90 90 90 fe ff ff ff 00 ................

76f63bcd 24 00 7b 00 74 00 32 00-7d 00 00 00 ff ff ff b0 $.{.t.2.}.......

76f63bdd 3b f6 76 b4 3b f6 76 90-90 90 90 90 8b ff 55 8b ;.v.;.v.......U.

76f63bed ec 81 ec 3c 02 00 00 a1-50 32 fb 76 33 c5 89 45 ...<....P2.v3..E

76f63bfd fc 53 56 8b 35 a0 f0 fa-76 8b d9 57 6a 2a 58 66 .SV.5...v..Wj\*Xf

76f63c0d 89 85 dc fd ff ff 33 ff-89 bd ea fd ff ff 66 89 ......3.......f.

76f63c1d bd ee fd ff ff c7 85 e0-fd ff ff a8 b7 ef 76 c7 ..............v.

匹配写法：

0:000> .block{as /mu ${/v:tn2} 76f63bad};? $scmp("${tn2}","lichking")

Evaluate expression: 0 = 00000000

注意：一定要有.block，对于as语句必须用block隔开才能展开

## 异常&事件

sxe 事件异常名 开启事件异常捕获

sxd 事件异常名 关闭事件异常捕获

异常码：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| asrt | av | dz | c000008e | eh | gp | ii |
| 断言错误 | 访问异常 | 整数除0 | 浮点除0 | c++异常 | 页保护错误 | 指令错误 |
| iov | isc | sbo | sov | aph | 3c | chhc |
| 整数溢出 | 非法系统调用 | 栈缓冲区溢出 | 栈溢出 | 程序停止响应 | 子进程退出 | 非法句柄 |
| wos | wob | ssessec | bpebpec | ccecc |  |  |
| wow64单步异常 | wow64断点异常 | 单步异常 | 断点异常 | ctrl+c;ctrl+break |  |  |

事件码：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ser | cpr | epr | ct | et | ld | ud | out |
| 系统错误 | 进程创建 | 进程退出 | 线程创建 | 线程退出 | 加载模块 | 卸载模块 | 调试输出 |

命令：.eventlog 打印最近的异常和事件

适用范围：用户态/内核态

命令：.lastevent 打印上次异常和事件

适用范围：用户态/内核态

### 如何在加载模块后暂停在Windbg中？

命令： sxe ld [模块名]

适用范围：用户态/内核态

命令：菜单Debug->Event Filters，设置Load module Enabled, Handled

适用范围：用户态/内核态

## 线程进程

命令：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 命令 | | | |\* | |. | |# | |n |
| 显示进程 | 所有进程 | 所有进程 | 当前活动进程 | 触发异常进程 | n号进程 |
| 命令 | ~ | ~\* | ~. | ~# | ~n |
| 显示线程 | 所有线程 | 所有线程 | 当前活动线程 | 触发异常线程 | n号线程 |
| 命令 | |ns |  |  |  |  |
| 设置当前进程 | n号进程 |  |  |  |  |
| 命令 | ~ns |  |  |  |  |
| 设置当前线程 | n号线程 |  |  |  |  |

适用范围：用户态

命令：~ns 切换当前处理器为n号处理器

适用范围：内核态

命令：.process 查看当前进程PEPROCESS地址

适用范围：内核态

例：

0:000> .process

Implicit process is now 00000000`7ecf7000

命令：!process 查看指定进程信息

适用范围：内核态

例：

kd> !process 81e2dda0

Failed to get VAD root

PROCESS 81e2dda0 SessionId: 0 Cid: 0624 Peb: 7ffde000 ParentCid: 02a4

DirBase: 08a40220 ObjectTable: e24b1dc8 HandleCount: 269.

Image: vmtoolsd.exe

VadRoot 00000000 Vads 0 Clone 0 Private 1279. Modified 5. Locked 0.

DeviceMap e10086e8

Token e24b8570

ElapsedTime 00:19:03.573

UserTime 00:00:00.203

KernelTime 00:00:01.515

QuotaPoolUsage[PagedPool] 143628

QuotaPoolUsage[NonPagedPool] 9472

Working Set Sizes (now,min,max) (3054, 50, 345) (12216KB, 200KB, 1380KB)

PeakWorkingSetSize 3092

VirtualSize 87 Mb

PeakVirtualSize 88 Mb

PageFaultCount 4446

MemoryPriority BACKGROUND

BasePriority 13

CommitCharge 2366

THREAD 818aeda8 Cid 0624.0628 Teb: 7ffdd000 Win32Thread: e17ca2e0 WAIT: (Executive) UserMode Non-Alertable

82129c6c NotificationEvent

IRP List:

81d36b80: (0006,0094) Flags: 00000900 Mdl: 00000000

Not impersonating

DeviceMap e10086e8

Owning Process 0 Image: <Unknown>

Attached Process 81e2dda0 Image: vmtoolsd.exe

Wait Start TickCount 1367 Ticks: 15662 (0:00:04:04.718)

Context Switch Count 57 IdealProcessor: 0 LargeStack

UserTime 00:00:00.031

KernelTime 00:00:00.078

Win32 Start Address 0x004060d0

Start Address 0x7c810705

命令：.thread 查看当前线程PETHREAD地址

适用范围：内核态

例：

0:000> .thread

Implicit thread is now 00000000`7ecfb000

命令：!thread PETHREAD地址 !thread 线程Id 查看指定线程信息

适用范围：内核态

例：

kd> !thread 818c4020

THREAD 818c4020 Cid 0624.0648 Teb: 7ffdc000 Win32Thread: e17e2c90 RUNNING on processor 0

Not impersonating

DeviceMap e10086e8

Owning Process 0 Image: <Unknown>

Attached Process 81e2dda0 Image: vmtoolsd.exe

Wait Start TickCount 17004 Ticks: 25 (0:00:00:00.390)

Context Switch Count 2744 IdealProcessor: 0 LargeStack

UserTime 00:00:00.093

KernelTime 00:00:01.421

Win32 Start Address 0x77dc3539

Start Address 0x7c8106f9

Stack Init b2b48000 Current b2b47ba8 Base b2b48000 Limit b2b43000 Call 0

Priority 15 BasePriority 15 PriorityDecrement 0 DecrementCount 0

ChildEBP RetAddr Args to Child

b2b47be0 805462e1 00000000 b2b47d64 00000100 nt!ExpInterlockedPopEntrySListEnd+0x8 (FPO: [0,0,0])

b2b47c3c 8056bed3 00000000 ffdff120 704f6f49 nt!ExAllocatePoolWithTag+0x3e1 (FPO: [Non-Fpo])

### 如何设置内核态进程/线程上下文？

命令：.process PEPROCESS地址 设置当前进程上下文

适用范围：内核态

例：

kd> !process 0 0 smss.exe

Failed to get VAD root

PROCESS 81c38da0 SessionId: none Cid: 0220 Peb: 7ffd4000 ParentCid: 0004

DirBase: 08a40020 ObjectTable: e13bde58 HandleCount: 19.

Image: smss.exe

kd> .process 81c38da0

Implicit process is now 81c38da0

WARNING: .cache forcedecodeuser is not enabled

命令：.thread PETHREAD地址 设置当前线程上下文

适用范围：内核态

命令：.context 用户态上下文地址 设置当前进程用户态上下文(属于进程上下文)

例：

kd> !process 0 0

\*\*\*\* NT ACTIVE PROCESS DUMP \*\*\*\*

PROCESS fe5039e0 SessionId: 0 Cid: 0008 Peb: 00000000 ParentCid: 0000

DirBase: 00030000 ObjectTable: fe529b68 TableSize: 50.

Image: System

PROCESS fe3c0d60 SessionId: 0 Cid: 0208 Peb: 7ffdf000 ParentCid: 00d4

DirBase: 0011f000 ObjectTable: fe3d0f48 TableSize: 30.

Image: regsvc.exe

kd> .context 0011f000

### 如何暂停/恢复线程执行？

命令：~线程号n (通过将挂起计数减一达到在系统中暂停该线程执行的效果)

命令：~线程号m (通过将挂起计数加一达到在系统中恢复该线程执行的效果)

命令：~线程号f (通过将冻结计数减一达到在调试器中暂停该线程执行的效果)

命令：~线程号u (通过将冻结计数加一达到在调试器中恢复该线程执行的效果)

### 如何切换到可执行进程/线程？

命令：.process /p /r /i PEPROCESS地址

适用范围：内核调试

例：

kd> !process 0 0 smss.exe

Failed to get VAD root

PROCESS 81c38da0 SessionId: none Cid: 0220 Peb: 7ffd4000 ParentCid: 0004

DirBase: 08a40020 ObjectTable: e13bde58 HandleCount: 19.

Image: smss.exe

kd> .process /p /r /i 81c38da0

You need to continue execution (press 'g' <enter>) for the context

to be switched. When the debugger breaks in again, you will be in

the new process context.

kd> g

Break instruction exception - code 80000003 (first chance)

nt!RtlpBreakWithStatusInstruction:

80528bec cc int 3

命令：.thread /p /r PETHREAD地址

适用范围：内核调试

例：

kd> .thread /p /r 805537c0

Implicit thread is now 805537c0

Implicit process is now 80553a20

.cache forcedecodeuser done

Loading User Symbols

### 如何遍历模块？

命令：!for\_each\_module

选项：@#FileVersion @#ProductVersion @#ModuleIndex @#ModuleName @#ImageName @#Base @#Size @#End

例：

kd> !for\_each\_module .echo @#ModuleIndex : @#Base @#End @#ModuleName @#ImageName @#LoadedImageName

00 : 01000000 01060000 ntsd C:\Program Files\Debugging Tools for Windows (x86)\ntsd.exe ntsd.exe

01 : 01400000 016f9000 ext C:\Program Files\Debugging Tools for Windows (x86)\winext\ext.dll ext.dll

02 : 01800000 0181d000 uext C:\Program Files\Debugging Tools for Windows (x86)\winext\uext.dll uext.dll

03 : 01900000 01975000 exts C:\Program Files\Debugging Tools for Windows (x86)\WINXP\exts.dll exts.dll

04 : 02000000 0239b000 dbgeng C:\Program Files\Debugging Tools for Windows (x86)\dbgeng.dll dbgeng.dll

05 : 03000000 03141000 dbghelp C:\Program Files\Debugging Tools for Windows (x86)\dbghelp.dll dbghelp.dll

### 如何遍历进程？

命令：!for\_each\_process

选项：@#Process为EPROCESS结构

例：

kd> !for\_each\_process dt \_EPROCESS ImageFileName @#Process

nt!\_EPROCESS

+0x174 ImageFileName : [16] "System"

nt!\_EPROCESS

+0x174 ImageFileName : [16] "smss.exe"

nt!\_EPROCESS

+0x174 ImageFileName : [16] "autochk.exe"

nt!\_EPROCESS

+0x174 ImageFileName : [16] "csrss.exe"

nt!\_EPROCESS

+0x174 ImageFileName : [16] "winlogon.exe"

### 如何遍历线程？

命令：!for\_each\_thread “”

选项：@#Thread为ETHREAD结构

命令：!list -t nt!\_LIST\_ENTRY.Flink -x "dt nt!\_KTHREAD @@(#CONTAINING\_RECORD(@$extret,nt!\_KTHREAD,ThreadListEntry))" poi( EPROCESS地址 +@@(#FIELD\_OFFSET(nt!\_KPROCESS,ThreadListHead))) 手动遍历

### 如何遍历寄存器？

命令：!for\_each\_register “”

选项：@#RegisterName @#RegisterValue

### 如何遍历系统句柄表？

命令：!list -t nt!\_LIST\_ENTRY.Flink -x "dt nt!\_HANDLE\_TABLE @@(#CONTAINING\_RECORD(@$extret,nt!\_HANDLE\_TABLE,

HandleTableList))" nt!HandleTableListHead 手动遍历

### 如何列出所有进程EPROCESS地址？

命令：dml\_proc 或 !process

例：

kd> !dml\_proc

Address PID Image file name

821b9660 4 System

81c1cca8 2c0 smss.exe

81c3d660 2e0 autochk.exe

81cde760 304 csrss.exe

81f5c758 324 winlogon.exe

81f16628 350 services.exe

81dfdc08 360 lsass.exe

8200f020 444 vmacthlp.exe

81d7eda0 454 svchost.exe

81c46890 490 BaiduHips.exe

81f0eda0 4b0 BaiduSdSvc.exe

81e7e410 500 QQPXRTP.exe

81f5f638 510 logonui.exe

81f253c0 5f4 svchost.exe

81b73890 648 svchost.exe

81dff898 6dc svchost.exe

81e27020 780 userinit.exe

81bf7578 7f4 svchost.exe

81d2a020 f0 ZhuDongFangYu.e

81b78da0 148 explorer.exe

81394890 2e4 spoolsv.exe

### 如何对模块排序

命令：lmDksm 按模块名排序

例：

kd> lmDksm

start end module name

b2ae3000 b2b05000 360AntiHacker (deferred)

b2a21000 b2a38000 360boost (deferred)

b2f44000 b2f7a000 360Box (deferred)

b2e95000 b2ee6700 360netmon (deferred)

b2a38000 b2a6d000 360qpesv (deferred)

f8211000 f821d000 360reskit (deferred)

b2f1a000 b2f43900 360SelfProtection (deferred)

f84e7000 f8514500 ACPI (deferred)

b2dca000 b2debb80 afd (deferred)

f870a000 f8714580 agp440 (deferred)

f8466000 f847d900 atapi (deferred)

f8d21000 f8d21c00 audst进程 (pdb symbols) d:\symcachel\audstub.pdb\6B3BF8F0C8834E7E8EFE53B7A91E2A3F1\audstub.pdb

b2a6d000 b2a9a000 BAPIDRV (deferred)

f8ab2000 f8ab5f00 BATTC (deferred)

命令：!dml\_proc 按进程对象地址排序

例：

kd> !dml\_proc

Address PID Image file name

821b97c0 4 System

81dd1c80 264 smss.exe

81ce0950 284 autochk.exe

82015878 2a4 csrss.exe

81d5f7a0 2c4 winlogon.exe

81c225d0 2f0 services.exe

820be4b0 300 lsass.exe

81689020 3d4 vmacthlp.exe

81d5b2d8 3e4 svchost.exe

81f536f8 41c logonui.exe

816995f0 43c QQPCNTP.exe

81fbe500 484 svchost.exe

81c0ba60 538 svchost.exe

### 如何获取进程名、进程ID 对应的进程对象

命令：!process 0 Flags 进程名 根据进程名获取进程对象

适用范围：内核态

例：

kd> !process 0 0 explorer.exe

Failed to get VAD root

PROCESS 81ce8bd0 SessionId: 0 Cid: 0780 Peb: 7ffde000 ParentCid: 06a8

DirBase: 13e40220 ObjectTable: e2417298 HandleCount: 431.

Image: explorer.exe

命令：!process 进程Id 根据进程ID获取进程对象

适用范围：内核态

例：

kd> !process 4

Searching for Process with Cid == 4

PROCESS 865e6690 SessionId: none Cid: 0004 Peb: 00000000 ParentCid: 0000

DirBase: 00185000 ObjectTable: 8a001940 HandleCount: 1543.

Image: System

VadRoot 86c8a630 Vads 7 Clone 0 Private 3. Modified 6964. Locked 64.

DeviceMap 8a009fc8

Token 8a0010b0

ElapsedTime 00:00:46.509

UserTime 00:00:00.000

KernelTime 00:00:00.577

QuotaPoolUsage[PagedPool] 0

QuotaPoolUsage[NonPagedPool] 0

Working Set Sizes (now,min,max) (154, 0, 0) (616KB, 0KB, 0KB)

PeakWorkingSetSize 1562

VirtualSize 1 Mb

PeakVirtualSize 7 Mb

## PE相关

### 如何查看某映像(sys exe dll)的版本号、时间、公司等信息？

命令：Lm指令v选项可以显示版本信息，这样很方便的查找当前加载模块是不是预期模块

适用范围：内核态/用户态

例：

kd> lmvm nt\*

start end module name

804d8000 806d0480 nt (pdb symbols) d:\symcachel\ntkrnlpa.pdb\30B5FB31AE7E4ACAABA750AA241FF3311\ntkrnlpa.pdb

Loaded symbol image file: ntkrnlpa.exe

Image path: ntkrnlpa.exe

Image name: ntkrnlpa.exe

Timestamp: Mon Apr 14 02:31:06 2008 (4802516A)

CheckSum: 002050D3

ImageSize: 001F8480

File version: 5.1.2600.5512

Product version: 5.1.2600.5512

File flags: 0 (Mask 3F)

File OS: 40004 NT Win32

File type: 1.0 App

File date: 00000000.00000000

Translations: 0804.04b0

CompanyName: Microsoft Corporation

ProductName: Microsoft(R) Windows(R) Operating System

InternalName: ntkrnlpa.exe

OriginalFilename: ntkrnlpa.exe

ProductVersion: 5.1.2600.5512

FileVersion: 5.1.2600.5512 (xpsp.080413-2111)

FileDescription: NT Kernel & System

LegalCopyright: (C) Microsoft Corporation. All rights reserved.

### 如何显示pe头信息？

命令：!dh, !lmi

适用范围：内核态/用户态

里：

0:000> !dh 001f0000

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES

14C machine (i386)

7 number of sections

55C5B5A9 time date stamp Sat Aug 08 15:54:17 2015

0 file pointer to symbol table

0 number of symbols

E0 size of optional header

102 characteristics

Executable

32 bit word machine

OPTIONAL HEADER VALUES

10B magic #

10.00 linker version

3200 size of code

3A00 size of initialized data

0 size of uninitialized data

11069 address of entry point

1000 base of code

----- new -----

### 如何查找内存中的PE头？

检测PE可以用于查找内核重载，内存映射文件等

命令：.imgscan /l /v /r 起始地址 l搜索长度

例：

kd> .imgscan /l /v /r 80b9f000 88db6000

\*\*\* Checking 80b9f000 - 88db6000

MZ at 80b9f000 - size 2a000

Name: kdvm.dll

Loaded kdvm.dll module

MZ at 80bfe000

MZ at 83e0a000 - size 410000

Name: ntoskrnl.exe

Loaded ntoskrnl.exe module

MZ at 8421a000 - size 37000

Name: HAL.dll

Loaded HAL.dll module

MZ at 86b1d000 - size 26d00

MZ at 8708b000 - size 26d00

MZ at 87454000 - size 88000

Name: MZ?

Loaded MZ? module

MZ at 88c00000 - size 18000

Name: rasl2tp.exe

Loaded rasl2tp.exe module

## 符号{结构体,函数,...}查看

命令：.reload 重新加载符号信息

选项：/f 强制加载 /user 用户态模块

适用范围：用户态/内核态

### 如何列出以T开头的模块=？

命令：lm m 模块名

适用范围：用户态/内核态

例：

kd> lm m T\*

start end module name

b1d28000 b1d4d000 TAOKernelXP (deferred)

b1d75000 b1d8ec80 TAOAccelerator (deferred)

b2ce8000 b2d0a700 TFsFlt (deferred)

b2d0b000 b2d33580 TSDefenseBt (deferred)

b2d34000 b2d65160 TSKsp (deferred)

b2d8c000 b2da2980 TSSysKit (deferred)

b2e3c000 b2e94380 tcpip (deferred)

f8515000 f8531c00 TsFltMgr (deferred)

f889a000 f88a3f00 termdd (pdb symbols) d:\symcachel\termdd.pdb\C04E4855F20641ECB654BB1AD575B8611\termdd.pdb

f8992000 f8996a80 TDI (pdb symbols) d:\symcachel\tdi.pdb\545742C029D24374BD687966638629EB1\tdi.pdb

f8a6a000 f8a6f380 TS888 (deferred)

f8a8a000 f8a8f500 TDTCP (deferred)

### 如何查看所有前缀为Rtl的符号？

命令：x指令

选项：/1只显示符号名 /2只显示地址 (与.foreach搭配是极好的)

适用范围：用户态/内核态

例：

kd> x nt!rtl\*

805e1284 nt!RtlFreeHotPatchData = <no type information>

8052aa00 nt!RtlDelete = <no type information>

8052b612 nt!RtlpVerCompare = <no type information>

80529d14 nt!RtlNumberOfSetBits = <no type information>

805d3842 nt!RtlValidAcl = <no type information>

8069d942 nt!RtlInitializeRangeListPackage = <no type information>

805d2c72 nt!RtlInitializeUnicodePrefix = <no type information>

805d40c0 nt!RtlCreateAtomTable = <no type information>

8052dfbc nt!RtlpTraceDatabaseAllocate = <no type information>

8052b3ce nt!RtlDeleteElementGenericTableAvl = <no type information>

805d4e4a nt!RtlpCopyRangeListEntry = <no type information>

805e0532 nt!RtlGetSetBootStatusData = <no type information>

80543548 nt!RtlLargeIntegerShiftLeft = <no type information>

805dc642 nt!RtlpGenerateInheritAcl = <no type information>

8052d7ec nt!RtlLargeIntegerDivide = <no type information>

805da254 nt!RtlLengthSid = <no type information>

8052e702 nt!RtlUnwind = <no type information>

### 如何查看SEH链

命令：!exchain 打印异常链

适用范围：内核态/用户态

例：

0:000> !exchain

0012fea8: Prymes!\_except\_handler3+0 (00407604)

CRT scope 0, filter: Prymes!dzExcepError+e6 (00401576)

func: Prymes!dzExcepError+ec (0040157c)

0012ffb0: Prymes!\_except\_handler3+0 (00407604)

CRT scope 0, filter: Prymes!mainCRTStartup+f8 (004021b8)

func: Prymes!mainCRTStartup+113 (004021d3)

0012ffe0: KERNEL32!GetThreadContext+1c (77ea1856)

### 指定基址如何查看结构体成员数值？

命令：dt [-b] 模块名!结构名 子成员名 基址

选项：-b 打印子结构体 子成员名可以用通配符

适用范围：用户态/内核态

例：

kd> dt \_FILE\_OBJECT

nt!\_FILE\_OBJECT

+0x000 Type : Int2B

+0x002 Size : Int2B

+0x004 DeviceObject : Ptr32 \_DEVICE\_OBJECT

+0x008 Vpb : Ptr32 \_VPB

+0x00c FsContext : Ptr32 Void

+0x010 FsContext2 : Ptr32 Void

+0x014 SectionObjectPointer : Ptr32 \_SECTION\_OBJECT\_POINTERS

+0x018 PrivateCacheMap : Ptr32 Void

+0x01c FinalStatus : Int4B

+0x020 RelatedFileObject : Ptr32 \_FILE\_OBJECT

+0x024 LockOperation : UChar

+0x025 DeletePending : Uchar

kd> dt \_FILE\_OBJECT Size

nt!\_FILE\_OBJECT

+0x002 Size : Int2B

注意：常用该命令打印系统符号中的结构体，或者在有源码的情况下查看变量，直接dt 变量即可

### 如何打印内核单向/双向链表？

!list

!slist

!lookaside

!pplookaside

### 如何获取某结构体大小？

命令：1. dt –v 结构体 2. ?? sizeof(结构体)

适用范围：用户态/内核态

例：

0:000> dt -v \_PEB

teststack!\_PEB

struct \_PEB, 71 elements, 0x230 bytes

+0x000 InheritedAddressSpace : UChar

+0x001 ReadImageFileExecOptions : UChar

+0x002 BeingDebugged : UChar

+0x003 SpareBool : UChar

+0x004 Mutant : Ptr32 to Void

0:000> ?? sizeof(\_PEB)

unsigned int 0x230

### 如何打印STRING, ANSI\_STRING, UNICODE\_STRING结构？

命令： ds 地址 !str地址 打印ANSI\_STRING

dS 地址 !ustr 地址 打印UNICODE\_STRING

命令： .printf (见前述章节)

### 如何查看进程环境块PEB结构？

命令：.process 获取\_PEB基址

适用范围：用户态

命令：dt \_PEB @$peb 查看当前进程信息

例：

0:000> dt \_PEB

teststack!\_PEB

+0x000 InheritedAddressSpace : UChar

+0x001 ReadImageFileExecOptions : UChar

+0x002 BeingDebugged : UChar

+0x003 SpareBool : UChar

+0x004 Mutant : Ptr32 Void

+0x008 ImageBaseAddress : Ptr32 Void

命令：.process 获取\_EPROCESS基址

适用范围：内核态

命令：dt \_EPROCESS @$proc 或 !process @$proc 查看当前进程信息

例：

kd> dt \_EPROCESS @$proc

nt!\_EPROCESS

+0x000 Pcb : \_KPROCESS

+0x06c ProcessLock : \_EX\_PUSH\_LOCK

+0x070 CreateTime : \_LARGE\_INTEGER 0x0

+0x078 ExitTime : \_LARGE\_INTEGER 0x0

+0x080 RundownProtect : \_EX\_RUNDOWN\_REF

+0x084 UniqueProcessId : 0x00000004 Void

+0x088 ActiveProcessLinks : \_LIST\_ENTRY [ 0x81dd1d08 - 0x8055b1d8 ]

### 如何查看线程环境块TEB结构？

命令：.thread 获取\_TEB基址 x86下为FS:[0]

适用范围：用户态

命令：dt \_TEB @$teb 查看当前线程信息

例：

0:000> dt \_TEB @$teb

teststack!\_TEB

+0x000 NtTib : \_NT\_TIB

+0x01c EnvironmentPointer : (null)

+0x020 ClientId : \_CLIENT\_ID

+0x028 ActiveRpcHandle : (null)

+0x02c ThreadLocalStoragePointer : 0x7fe6f02c Void

+0x030 ProcessEnvironmentBlock : 0x7fe69000 \_PEB

+0x034 LastErrorValue : 0

+0x038 CountOfOwnedCriticalSections : 0

+0x03c CsrClientThread : (null)

+0x040 Win32ThreadInfo : (null)

注意：第一个元素为TIB结构

命令：.thread 获取\_ETHREAD基址

适用范围：内核态

命令： 1. dg fs获取\_TEB基址 (x86)

0:000> dg fs

P Si Gr Pr Lo

Sel Base Limit Type l ze an es ng Flags

---- -------- -------- ---------- - -- -- -- -- --------

0053 7fe6f000 00000fff Data RW Ac 3 Bg By P Nl 000004f3

2. dt \_PEB 7fe6f000

命令：dt \_ETHREAD @$thread 查看当前线程信息

例：

kd> dt \_ETHREAD @$thread

nt!\_ETHREAD

+0x000 Tcb : \_KTHREAD

+0x1c0 CreateTime : \_LARGE\_INTEGER 0x0e88cf0d`f3bc51d0

+0x1c0 NestedFaultCount : 0y00

+0x1c0 ApcNeeded : 0y0

+0x1c8 ExitTime : \_LARGE\_INTEGER 0x81be01e8`81be01e8

+0x1c8 LpcReplyChain : \_LIST\_ENTRY [ 0x81be01e8 - 0x81be01e8 ]

+0x1c8 KeyedWaitChain : \_LIST\_ENTRY [ 0x81be01e8 - 0x81be01e8 ]

+0x1d0 ExitStatus : 0n0

+0x1d0 OfsChain : (null)

+0x1d4 PostBlockList : \_LIST\_ENTRY [ 0x81be01f4 - 0x81be01f4 ]

### 如何查看内核进程控制块？

命令：!pcr 基址 x86下为FS:[0]

适用范围：内核态

kd> !pcr

KPCR for Processor 0 at ffdff000:

Major 1 Minor 1

NtTib.ExceptionList: b1b8c528

NtTib.StackBase: b1b8cdf0

NtTib.StackLimit: b1b8a000

NtTib.SubSystemTib: 00000000

NtTib.Version: 00000000

NtTib.UserPointer: 00000000

NtTib.SelfTib: 00000000

SelfPcr: ffdff000

Prcb: ffdff120

Irql: 00000000

IRR: 00000000

IDR: ffffffff

InterruptMode: 00000000

IDT: 8003f400

GDT: 8003f000

TSS: 80042000

CurrentThread: 81be0020

NextThread: 00000000

IdleThread: 805537c0

DpcQueue:

命令： 1. dg fs获取\_KPCR基址 (x86)

kd> dg fs

P Si Gr Pr Lo

Sel Base Limit Type l ze an es ng Flags

---- -------- -------- ---------- - -- -- -- -- --------

0030 ffdff000 00001fff Data RW Ac 0 Bg Pg P Nl 00000c93

2. dt \_KPCR ffdff000

适用范围：内核态

例：

kd> dt \_KPCR ffdff000

nt!\_KPCR

+0x000 NtTib : \_NT\_TIB

+0x01c SelfPcr : 0xffdff000 \_KPCR

+0x020 Prcb : 0xffdff120 \_KPRCB

+0x024 Irql : 0 ''

+0x028 IRR : 0

+0x02c IrrActive : 0

+0x030 IDR : 0xffffffff

+0x034 KdVersionBlock : 0x80546b38 Void

+0x038 IDT : 0x8003f400 \_KIDTENTRY

+0x03c GDT : 0x8003f000 \_KGDTENTRY

注意：第三个成员为\_KPRCB结构

### 如何打印系统服务表SSDT, SSSDT?

命令： dps nt!KiServiceTable l0x200

dps poi(KeServiceDescriptorTable) l0x200

适用范围：SSDT

例：

kd> dps poi(KeServiceDescriptorTable) l0x200

80502b9c 8059a9f4 nt!NtAcceptConnectPort

80502ba0 805e7e74 nt!NtAccessCheck

80502ba4 805eb6ba nt!NtAccessCheckAndAuditAlarm

80502ba8 805e7ea6 nt!NtAccessCheckByType

80502bac 805eb6f4 nt!NtAccessCheckByTypeAndAuditAlarm

80502bb0 805e7edc nt!NtAccessCheckByTypeResultList

80502bb4 805eb738 nt!NtAccessCheckByTypeResultListAndAuditAlarm

80502bb8 805eb77c nt!NtAccessCheckByTypeResultListAndAuditAlarmByHandle

1. 获取csrss进程对象

kd> !process 0 0 csrss.exe

Failed to get VAD root

PROCESS 82015878 SessionId: 0 Cid: 02a4 Peb: 7ffd8000 ParentCid: 0264

DirBase: 14700060 ObjectTable: e1672920 HandleCount: 482.

Image: csrss.exe

1. 将该进程设置为当前上下文

kd> .process 82015878

Implicit process is now 82015878

WARNING: .cache forcedecodeuser is not enabled

1. 读取sssdt

适用范围：SSSDT

例：

kd> dps poi(nt!KeServiceDescriptorTableShadow+0x10)

bf99ce80 bf937330 win32k!NtGdiAbortDoc

bf99ce84 bf9489d2 win32k!NtGdiAbortPath

bf99ce88 bf882d2f win32k!NtGdiAddFontResourceW

bf99ce8c bf94054d win32k!NtGdiAddRemoteFontToDC

bf99ce90 bf949fe9 win32k!NtGdiAddFontMemResourceEx

bf99ce94 bf9375c4 win32k!NtGdiRemoveMergeFont

bf99ce98 bf937669 win32k!NtGdiAddRemoteMMInstanceToDC

bf99ce9c bf83affa win32k!NtGdiAlphaBlend

bf99cea0 bf949910 win32k!NtGdiAngleArc

### 如何打印用户态回调表KernelCallbackTable?

1. 获取csrss进程对象

kd> !process 0 0 csrss.exe

Failed to get VAD root

PROCESS 82015878 SessionId: 0 Cid: 02a4 Peb: 7ffd8000 ParentCid: 0264

DirBase: 14700060 ObjectTable: e1672920 HandleCount: 482.

Image: csrss.exe

1. 将该进程设置为当前上下文

kd> .process 82015878

Implicit process is now 82015878

WARNING: .cache forcedecodeuser is not enabled

1. 加载用户态模块user32.dll

kd> .reload

Connected to Windows XP 2600 x86 compatible target at (Sun Nov 8 22:55:03.842 2015 (UTC + 8:00)), ptr64 FALSE

Loading Kernel Symbols

...............................................................

................................................................

...............

Loading User Symbols

..............

Loading unloaded module list

.............

1. 从user32.dll获取符号

kd> x user32!\*apfnDispatch\*

77d12970 USER32!apfnDispatch = <no type information>

kd> dds apfnDispatch

77d12970 77d27f3c USER32!\_\_fnCOPYDATA

77d12974 77d587b3 USER32!\_\_fnCOPYGLOBALDATA

77d12978 77d28ec8 USER32!\_\_fnDWORD

77d1297c 77d2b149 USER32!\_\_fnNCDESTROY

77d12980 77d5876c USER32!\_\_fnDWORDOPTINLPMSG

77d12984 77d5896d USER32!\_\_fnINOUTDRAG

77d12988 77d3b84d USER32!\_\_fnGETTEXTLENGTHS

77d1298c 77d58c42 USER32!\_\_fnINCNTOUTSTRING

77d12990 77d285c1 USER32!\_\_fnINCNTOUTSTRINGNULL

77d12994 77d58b0f USER32!\_\_fnINLPCOMPAREITEMSTRUCT

77d12998 77d2ce26 USER32!\_\_fnINLPCREATESTRUCT

77d1299c 77d58b4d USER32!\_\_fnINLPDELETEITEMSTRUCT

77d129a0 77d4feec USER32!\_\_fnINLPDRAWITEMSTRUCT

77d129a4 77d58b8b USER32!\_\_fnINLPHELPINFOSTRUCT

77d129a8 77d58b8b USER32!\_\_fnINLPHELPINFOSTRUCT

### 如何查看系统中断表？

命令：!idt [中断号] 输出所有可用中断号

适用范围：内核态

例：

kd> !idt -a

Dumping IDT: 8003f400

287937b900000000: 8053f1ac nt!KiTrap00

287937b900000001: 8053f324 nt!KiTrap01

287937b900000002: Task Selector = 0x0000

287937b900000003: 8053f6f4 nt!KiTrap03

287937b900000004: 8053f874 nt!KiTrap04

287937b900000005: 8053f9d0 nt!KiTrap05

287937b900000006: 8053fb44 nt!KiTrap06

287937b900000007: 805401ac nt!KiTrap07

287937b900000029: 00000000

287937b90000002a: 8053e9ee nt!KiGetTickCount

287937b90000002b: 8053eaf0 nt!KiCallbackReturn

287937b90000002c: 8053ec90 nt!KiSetLowWaitHighThread

287937b90000002d: 8053f5d0 nt!KiDebugService

287937b90000002e: 8053e491 nt!KiSystemService

287937b90000002f: 80541790 nt!KiTrap0F

### 如何查看指定地址所属模块？

命令：lm a 地址

适用范围：用户态/内核态

例：

kd> lm m ntdll

Browse full module list

start end module name

7c920000 7c9b6000 ntdll (pdb symbols) e:\symbol\ntdll.pdb\99192024C5EB4830AC602195086637082\ntdll.pdb

kd> lm a 7c920010

Browse full module list

start end module name

7c920000 7c9b6000 ntdll (pdb symbols) e:\symbol\ntdll.pdb\99192024C5EB4830AC602195086637082\ntdll.pdb

### 如何快速加载/卸载指定符号

快速卸载符号的需求在于，正在调试的某个文件，因为某需求改动，重新编译时会发生pdb文件占用，导致无法编译成功，比较不好的做法是结束windbg

命令：.reload /u \*.dll 卸载某dll的符号

快速加载符号的需求在于，.reload指令有时会花费较长时间，而有时只需加载特定符号

命令：.reload \*.dll 加载某dll的符号

快速加载“本次调试未涉及的PE文件”的需求在于，可以查看&使用目前符号中不存在的结构

快速加载任意PE(dll/exe/sys)

命令：.reload /f 文件名.后缀=加载地址,长度

例：

0:000> .reload /f 2.exe=70000000,65536

\*\*\* WARNING: Unable to verify timestamp for 2.exe

## 句柄和对象

### 如何根据 基址、名称获取对象(OBJECT)信息？

命令： !object 对象地址

!object 对象路径 对象路径可以对照winobj对象目录结构

!object 对象类型名 Driver Device Directory Port Key SymbolicLink Event WaitablePort File.....需要设置gflag

适用范围：内核态

例：

kd> !object e100a478

Object: e100a478 Type: (821ed420) Directory

ObjectHeader: e100a460 (old version)

HandleCount: 0 PointerCount: 7

Directory Object: e10010e0 Name: ArcName

kd> !object \

Object: e10010e0 Type: (821ed420) Directory

ObjectHeader: e10010c8 (old version)

HandleCount: 0 PointerCount: 40

Directory Object: 00000000 Name: \

126 symbolic links snapped through this directory

Hash Address Type Name

---- ------- ---- ----

00 e100a478 Directory ArcName

8213b5a8 Device Ntfs

01 e13af030 Port SeLsaCommandPort

02 820b9738 Device FatCdrom

03 e1011490 Key \REGISTRY

05 e14ef870 Port ThemeApiPort

06 e2385460 Port XactSrvLpcPort

09 e152a490 Directory NLS

10 e1008660 SymbolicLink DosDevices

kd> !object \Driver

Object: e12bf480 Type: (821ed420) Directory

ObjectHeader: e12bf468 (old version)

HandleCount: 0 PointerCount: 83

Directory Object: e10010e0 Name: Driver

Hash Address Type Name

---- ------- ---- ----

00 81c051f8 Driver Beep

8213b2a8 Driver NDIS

81e45a08 Driver KSecDD

01 81d5ec40 Driver FsVga

81e73b10 Driver Raspti

81cb9610 Driver es1371

81cb9498 Driver Mouclass

02 81d5e898 Driver vmx\_svga

03 81ce5030 Driver Fips

81c35880 Driver Kbdclass

04 81ee86e8 Driver VgaSave

kd> !object \Device

Object: e100d748 Type: (821ed420) Directory

ObjectHeader: e100d730 (old version)

HandleCount: 0 PointerCount: 274

Directory Object: e10010e0 Name: Device

11 symbolic links snapped through this directory

Hash Address Type Name

---- ------- ---- ----

00 81fd59e8 Device KsecDD

8213a030 Device Ndis

81fbaa98 Device Beep

e13c3ac8 SymbolicLink ScsiPort2

821e7850 Device 00000032

821e8610 Device 00000025

821e92b0 Device 00000019

01 81e44060 Device Netbios

821e7610 Device 00000033

821e83d0 Device 00000026

02 81c2ff18 Device Ip

81c6e5d0 Device KSENUM#000

### 如何查看驱动对象、设备对象、文件对象信息？

命令：!drvobj [对象基址] !devobj [对象基址] !fileobj [对象基址]

适用范围：内核态

### 如何根据句柄获取对象信息？

命令：!handle [句柄 [标志位 [PEPROCESS [类型名]]]]

适用范围：用户态/内核态

例：

获取对象基址

kd> !handle 00cc

Failed to get VAD root

PROCESS 81bf9ba0 SessionId: 0 Cid: 0c44 Peb: 7ffdb000 ParentCid: 0884

DirBase: 14700820 ObjectTable: e17d6430 HandleCount: 169.

Image: 360Safe.exe

Handle table at e17d6430 with 169 entries in use

00cc: Object: e1604668 GrantedAccess: 00020019 Entry: e118e198

Object: e1604668 Type: (821b2708) Key

ObjectHeader: e1604650 (old version)

HandleCount: 1 PointerCount: 1

Directory Object: 00000000 Name: \REGISTRY\MACHINE\SYSTEM\CONTROLSET001\CONTROL\NLS\LOCALE\ALTERNATE SORTS

因为是Key类型，对应结构为\_CM\_KEY\_BODY

kd> dt \_CM\_KEY\_BODY e1604668

nt!\_CM\_KEY\_BODY

+0x000 Type : 0x6b793032

+0x004 KeyControlBlock : 0xe13f6698 \_CM\_KEY\_CONTROL\_BLOCK

+0x008 NotifyBlock : (null)

+0x00c ProcessID : 0x00000c44 Void

+0x010 Callers : 0

+0x014 CallerAddress : [10] 0x004f0053 Void

+0x03c KeyBodyList : \_LIST\_ENTRY [ 0xe13f66c8 - 0xe182860c ]

注意：!handle会显示所有进程所有句柄

### 如何显示所有ObjectType类型名？

.foreach (addr {x /q /0 nt!\*ObjectType}) {dt \_object\_type Name poi(${addr})}

nt!\_OBJECT\_TYPE

+0x040 Name : \_UNICODE\_STRING "SymbolicLink"

nt!\_OBJECT\_TYPE

+0x040 Name : \_UNICODE\_STRING "Semaphore"

nt!\_OBJECT\_TYPE

+0x040 Name : \_UNICODE\_STRING "Controller"

nt!\_OBJECT\_TYPE

+0x040 Name : \_UNICODE\_STRING "Key"

nt!\_OBJECT\_TYPE

+0x040 Name : \_UNICODE\_STRING "EventPair"

nt!\_OBJECT\_TYPE

+0x040 Name : \_UNICODE\_STRING "DebugObject"

nt!\_OBJECT\_TYPE

+0x040 Name : \_UNICODE\_STRING "Desktop"

nt!\_OBJECT\_TYPE

命令：ObpTypeObjectType链表

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。。

## 注册表信息

### 如何查看注册表项键值？

命令：!dreg

适用范围：用户态

例：

!dreg System\CurrentControlSet\Services\Tcpip!\*

## 内存操作

命令：!db, !dc, !dd, !dp, !dq, !du, !dw 分别按不同类型读取物理内存

适用范围：内核态

命令：db, dc, dd, dp, dq, du, dw 分别按不同类型读取虚拟内存

适用范围：内核态/用户态

命令： dds l[元素个数] 作为4字节地址数组打印

dqs l[元素个数] 作为8字节地址数组打印

dps l[元素个数] 根据指针大小自动判断

适用范围：内核态/用户态

例：

读取虚拟地址

kd> db f8da6000

f8da6000 4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00 MZ..............

f8da6010 b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......

f8da6020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

f8da6030 00 00 00 00 00 00 00 00-00 00 00 00 d0 00 00 00 ................

f8da6040 0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68 ........!..L.!Th

f8da6050 69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f is program canno

f8da6060 74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20 t be run in DOS

f8da6070 6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00 mode....$.......

由虚拟地址转换物理地址

kd> !pte f8da6000

VA f8da6000

PDE at C0603E30 PTE at C07C6D30

contains 0000000001034163 contains 0000000007FB9163

pfn 1034 -G-DA--KWEV pfn 7fb9 -G-DA--KWEV

读取物理地址

kd> !db 7FB9000

# 7fb9000 4d 5a 90 00 03 00 00 00-04 00 00 00 ff ff 00 00 MZ..............

# 7fb9010 b8 00 00 00 00 00 00 00-40 00 00 00 00 00 00 00 ........@.......

# 7fb9020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................

# 7fb9030 00 00 00 00 00 00 00 00-00 00 00 00 d0 00 00 00 ................

# 7fb9040 0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68 ........!..L.!Th

# 7fb9050 69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f is program canno

# 7fb9060 74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20 t be run in DOS

# 7fb9070 6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 00 mode....$.......

可以看出，内容是相同的

命令：!eb, !ed 分别按不同类型写入物理内存

适用范围：内核态调试

命令：e, ea, eb, ed, eD, ef, ep, eq, eu, ew, eza 分别按不同类型写入虚拟内存

适用范围：内核态/用户态调试

例：

写入字节：eb f8da6000 90 90 90 90 90

写入字符串：ea f8da6000 "my ass" eu f8da6000 "my ass"

### 查看物理内存使用

命令：!memusage

例：

kd> !memusage

loading PFN database

loading (100% complete)

Compiling memory usage data (99% Complete).

Zeroed: 40657 (162628 kb)

Free: 3646 ( 14584 kb)

Standby: 54142 (216568 kb)

Modified: 957 ( 3828 kb)

ModifiedNoWrite: 0 ( 0 kb)

Active/Valid: 31555 (126220 kb)

Transition: 0 ( 0 kb)

SLIST/Bad: 0 ( 0 kb)

Unknown: 0 ( 0 kb)

TOTAL: 130957 (523828 kb)

Building kernel map

Finished building kernel map

Scanning PFN database - (100% complete)

Usage Summary (in Kb):

Control Valid Standby Dirty Shared Locked PageTables name

8164b4a0 12 0 0 0 0 0 mapped\_file( qqpcrtp\_qmhipspolicyeng.log )

820b7d38 148 24 0 4 0 0 mapped\_file( SysEvent.Evt )

820f6728 332 0 0 0 0 0 mapped\_file( $LogFile )

81fe7d78 4 0 0 0 0 0 mapped\_file( $MftMirr )

81f98ae0 3956 1352 0 0 0 0 mapped\_file( $Mft )

8208f160 640 0 0 0 0 0 mapped\_file( $BitMap )

81e46098 4 0 0 0 0 0 mapped\_file( $Mft )

81e462a8 12 0 0 0 0 0 mapped\_file( $Directory )

81c63208 0 8 0 0 0 0 mapped\_file( No name for file )

81e46ae0 4 0 0 0 0 0 mapped\_file( $Directory )

821e3090 32 0 0 0 0 0 mapped\_file( No name for file )

81c63270 16 0 0 0 0 0 mapped\_file( $Directory )

81cf0230 328 0 0 0 0 0 mapped\_file( $Directory )

8219d4a8 304 72 0 276 0 0 mapped\_file( ntdll.dll )

### 查看虚拟内存使用

命令：!vm

例：

kd> !vm

\*\*\* Virtual Memory Usage \*\*\*

Physical Memory: 130940 ( 523760 Kb)

Page File: \??\C:\pagefile.sys

Current: 786432 Kb Free Space: 784332 Kb

Minimum: 786432 Kb Maximum: 1572864 Kb

Available Pages: 98445 ( 393780 Kb)

ResAvail Pages: 96643 ( 386572 Kb)

Locked IO Pages: 1105 ( 4420 Kb)

Free System PTEs: 226165 ( 904660 Kb)

Free NP PTEs: 28139 ( 112556 Kb)

Free Special NP: 0 ( 0 Kb)

Modified Pages: 957 ( 3828 Kb)

Modified PF Pages: 957 ( 3828 Kb)

NonPagedPool Usage: 3481 ( 13924 Kb)

NonPagedPool Max: 32768 ( 131072 Kb)

PagedPool 0 Usage: 4660 ( 18640 Kb)

PagedPool 1 Usage: 693 ( 2772 Kb)

PagedPool 2 Usage: 712 ( 2848 Kb)

PagedPool Usage: 6065 ( 24260 Kb)

PagedPool Maximum: 65536 ( 262144 Kb)

Session Commit: 526 ( 2104 Kb)

Shared Commit: 2984 ( 11936 Kb)

### 如何获取Fs:[0]所在地址？

命令：dg 段选择子

0:000> dg @fs

P Si Gr Pr Lo

Sel Base Limit Type l ze an es ng Flags

---- -------- -------- ---------- - -- -- -- -- --------

0053 7fe6f000 00000fff Data RW Ac 3 Bg By P Nl 000004f3

### 如何查看某虚拟内存地址对应的物理内存地址？

命令：!pte 虚拟地址 获取page table entry (PTE) 和page directory entry (PDE)信息

适用范围：内核态

例：

kd> !pte 804d8000

VA 804d8000

PDE at C0602010 PTE at C04026C0

contains 0000000000AEE023 contains 00000000004D8063

pfn aee ----A--KWEV pfn 4d8 ---DA—KWEV

命令： !vtop PFN 虚拟地址

!vtop 0 虚拟地址 使用当前进程PFN

适用范围：内核态

例：

kd> !process 0 0

\*\*\*\* NT ACTIVE PROCESS DUMP \*\*\*\*

....

PROCESS ff779190 SessionId: 0 Cid: 04fc Peb: 7ffdf000 ParentCid: 0394

DirBase: 098fd000 ObjectTable: e1646b30 TableSize: 8.

Image: MyApp.exe

kd> !vtop 98fd 12f980

Pdi 0 Pti 12f

0012f980 09de9000 pfn(09de9)

### 如何查看某物理内存地址对应的虚拟内存地址？

命令：!pte2va PTE地址 查看PTE对应虚拟内存基址

适用范围：内核态

例：

kd> !pte2va C04026C0

804d8000

命令：!ptov DirBase 查看某进程物理内存到虚拟内存映射表

适用范围：内核态

例：

1: kd> .process

Implicit process is now 852b4040

1: kd> !process 852b4040 1

PROCESS 852b4040 SessionId: none Cid: 0004 Peb: 00000000 ParentCid: 0000

DirBase: 00185000 ObjectTable: 83203000 HandleCount: 663.

Image: System

...

1: kd> !ptov 185000

X86PtoV: pagedir 185000, PAE enabled.

15e11000 10000

549e6000 20000

...

60a000 210000

40b000 211000

...

54ad3000 25f000

548d3000 260000

...

d71000 77510000

...

### 如何查看地址所在虚拟内存位于哪个模块？

命令：!address 地址

0:000> !address 77c00000

例：

Usage: Image

Base Address: 77c00000

End Address: 77c01000

Region Size: 00001000

State: 00001000 MEM\_COMMIT

Protect: 00000002 PAGE\_READONLY

Type: 01000000 MEM\_IMAGE

Allocation Base: 77c00000

Allocation Protect: 00000080 PAGE\_EXECUTE\_WRITECOPY

Image Path: ntdll.dll

Module Name: ntdll

Loaded Image Name: C:\WINDOWS\SYSTEM32\ntdll.dll

Mapped Image Name:

More info: lmv m ntdll

More info: !lmi ntdll

More info: ln 0x77c00000

More info: !dh 0x77c00000

### 如何以固定字节模式填充内存？

命令：

填充虚拟地址 f 地址 l长度 字节

填充物理地址 fp 地址 l长度 字节

适用范围：f 内核态/用户态 fp 内核态

例：

kd> f f8a9b05b l0x100 0x12

Filled 0x100 bytes

### 如何拷贝虚拟内存块？

命令：m 源地址 l长度 目的地址

### 如何比较虚拟内存块？

命令：c源地址 l长度 目的地址

### 如何将文件内容读取到调试器内存/从调试器内存写入文件？

注意这里的读写没有pe映射之类的操作，而是二进制读写

命令：.readmem 文件路径 加载基址 l长度 将文件内容拷贝到被调试目标内存

命令：.writemem 文件路径 加载基址 l长度 从调试目标内存拷贝到文件

例：

0:000> .writemem 1234.bin 00000000`76eb0000 l0x20000

Writing 20000 bytes................................................................

### 如何搜索内存？

虚拟内存

命令：s [-[[Flags]Type]] 搜索基址 长度 搜索模式 按给定模式搜索内存

命令：s -[[Flags]]v 搜索基址 长度 对象实例 搜索内存块与给定对象的类虚表相同的对象实例

命令：s -[[Flags]]sa搜索基址 长度 搜索ASCII字符串

命令：s -[[Flags]]su搜索基址 长度 搜索UNICODE字符串

物理内存

命令：!search 目标值 [波动偏差 [起始PFN [结束PFN]]]

参数：目标值范围0~0xFFFFFFFFFFFFFFFF

例：

0:000> db 76f63bad

76f63bad 6c 00 69 00 63 00 68 00-6b 00 69 00 6e 00 67 00 l.i.c.h.k.i.n.g.

76f63bbd 00 00 00 00 f9 ff c3 90-90 90 90 fe ff ff ff 00 ................

76f63bcd 24 00 7b 00 74 00 32 00-7d 00 00 00 ff ff ff b0 $.{.t.2.}.......

76f63bdd 3b f6 76 b4 3b f6 76 90-90 90 90 90 8b ff 55 8b ;.v.;.v.......U.

76f63bed ec 81 ec 3c 02 00 00 a1-50 32 fb 76 33 c5 89 45 ...<....P2.v3..E

76f63bfd fc 53 56 8b 35 a0 f0 fa-76 8b d9 57 6a 2a 58 66 .SV.5...v..Wj\*Xf

76f63c0d 89 85 dc fd ff ff 33 ff-89 bd ea fd ff ff 66 89 ......3.......f.

76f63c1d bd ee fd ff ff c7 85 e0-fd ff ff a8 b7 ef 76 c7 ..............v.

0:000> s -u 76f63bad l10000 "lichking"

76f63bad 006c 0069 0063 0068 006b 0069 006e 0067 l.i.c.h.k.i.n.g.

### 如何查看内存池信息？

命令：!pool 地址

适用范围：内核态

例：

kd> !pool e1001050

e1001000 size: 40 previous size: 0 (Allocated) MmDT

e1001040 size: 10 previous size: 40 (Free) Mm

\*e1001050 size: 10 previous size: 10 (Allocated) \*ObDi

e1001060 size: 10 previous size: 10 (Allocated) ObDi

e1001070 size: 10 previous size: 10 (Allocated) Symt

e1001080 size: 40 previous size: 10 (Allocated) ObDm

e10010c0 size: 10 previous size: 40 (Allocated) ObDi

### 如何查找指定Tag的内存池？

命令：!poolfind Tag字符串/Tag值 [选项] [-x “命令”]

参数：选项

-nonpaged 非分页内存 -paged 分页内存

-global 全局池 -session 会话池

-small -large

-process tag值作为EPROCESS指针

适用范围：内核态

例：

!poolfind Mm\* - Find all Mm allocations in nonpaged pool.

!poolfind MmSt -paged - Find MmSt allocations in paged pool.

!poolfind Gla1 -session - Find Gla1 allocations in session pool.

!poolfind -tag "AB C" - Find pool tag which contains a space.

!poolfind -x "dt nt!\_MDL @$extret" Mdl - Find and print MDL allocations.

kd> !poolfind \* -nonpaged

\*\*\* CacheSize too low - increasing to 51 MB

Max cache size is : 53657600 bytes (0xccb0 KB)

Total memory in cache : 8917 bytes (0x9 KB)

Number of regions cached: 32

81 full reads broken into 93 partial reads

counts: 56 cached/37 uncached, 60.22% cached

bytes : 4456 cached/7109 uncached, 38.53% cached

\*\* Transition PTEs are implicitly decoded

\*\* Prototype PTEs are implicitly decoded

Scanning large pool allocation table for tag 0x2020202a (\* ) (afc00000 : b0000000)

86619000 : tag XPPH, size 0x79e8, Nonpaged pool

866209f0 : tag Frag, size 0, Nonpaged pool

86620a00 : tag IdeP, size 0x600, Nonpaged pool

87a1e000 : tag Cont, size 0xa000, Nonpaged pool

### 如何查看内存池使用情况？

命令：!poolused

适用范围：内核态

例：

0: kd> !poolused

Sorting by Tag

Pool Used:

NonPaged Paged

Tag Allocs Used Allocs Used

1394 1 520 0 0UNKNOWN pooltag '1394', please update pooltag.txt

1MEM 1 3368 0 0UNKNOWN pooltag '1MEM', please update pooltag.txt

2MEM 1 3944 0 0UNKNOWN pooltag '2MEM', please update pooltag.txt

3MEM 3 248 0 0UNKNOWN pooltag '3MEM', please update pooltag.txt

8042 4 3944 0 0PS/2 kb and mouse , Binary: i8042prt.sys

AGP 1 344 2 384UNKNOWN pooltag 'AGP ', please update pooltag.txt

AcdN 2 1072 0 0TDI AcdObjectInfoG

AcpA 3 192 1 504ACPI Pooltags , Binary: acpi.sys

AcpB 0 0 4 576ACPI Pooltags , Binary: acpi.sys

AcpD 40 13280 0 0ACPI Pooltags , Binary: acpi.sys

AcpF 6 240 0 0ACPI Pooltags , Binary: acpi.sys

AcpM 0 0 1 128ACPI Pooltags , Binary: acpi.sys

AcpO 4 208 0 0ACPI Pooltags , Binary: acpi.sys

### 如何查看内存堆信息？

命令：!heap

参数

适用范围：用户态/内核态

### 如何显示虚拟内存块及访问权限

命令：!vadump –v 显示所有虚拟内存块及访问权限

适用范围：用户态

例：

0:000> !vadump -v

BaseAddress: 00000000

AllocationBase: 00000000

RegionSize: 00010000

State: 00010000 MEM\_FREE

Protect: 00000001 PAGE\_NOACCESS

BaseAddress: 00010000

AllocationBase: 00010000

AllocationProtect: 00000004 PAGE\_READWRITE

RegionSize: 00001000

State: 00001000 MEM\_COMMIT

Protect: 00000004 PAGE\_READWRITE

Type: 00020000 MEM\_PRIVATE

命令：!vprot [虚拟地址] 显示某地址所属虚拟内存块及访问权限

适用范围：用户态

例：

0:000> !vprot 7ffe1000

BaseAddress: 7ffe1000

AllocationBase: 7ffe0000

AllocationProtect: 00000002 PAGE\_READONLY

RegionSize: 0000f000

State: 00002000 MEM\_RESERVE

Type: 00020000 MEM\_PRIVATE

### 如何查找内存中某类的对象实例？

## 特殊调试法

### 如何用内核态调试器控制用户态调试器进程联合调试？

用内核态调试器控制远程用户态调试器，此外还可以在远程机器执行shell命令、

准备工作：在远程机器(或vmware虚拟机)上安装windbg，并把环境变量path设置为该目录(必须能找到ntsd.exe)，之后重启机器即可

操作步骤：

1. 在本地主机建立远程内核态调试
2. !bpid [进程Id] 命令用户态调试器附加调试进程 例子如下：

kd> !bpid 0794

Finding winlogon.exe (0)...

Waiting for winlogon.exe to break. This can take a couple of minutes...

Break instruction exception - code 80000003 (first chance)

Stepping to g\_AttachProcessId check...

Break into process 794 set. The next break should be in the desired process.

Microsoft (R) Windows Debugger Version 6.12.0002.633 X86

Copyright (c) Microsoft Corporation. All rights reserved.

\*\*\* wait with pending attach

Symbol search path is: \*\*\* Invalid \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Symbol loading may be unreliable without a symbol search path. \*

\* Use .symfix to have the debugger choose a symbol path. \*

\* After setting your symbol path, use .reload to refresh symbol locations. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Executable search path is:

ModLoad: 01000000 010f1000 C:\WINDOWS\Explorer.EXE

ModLoad: 7c920000 7c9b6000 C:\WINDOWS\system32\ntdll.dll

ModLoad: 7c800000 7c91e000 C:\WINDOWS\system32\kernel32.dll

(794.f04): Break instruction exception - code 80000003 (first chance)

eax=7ffde000 ebx=00000001 ecx=00000002 edx=00000003 esi=00000004 edi=00000005

eip=7c92120e esp=0327ffcc ebp=0327fff4 iopl=0 nv up ei pl zr na pe nc

cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00000246

\*\*\* ERROR: Symbol file could not be found. Defaulted to export symbols for C:\WINDOWS\system32\ntdll.dll -

ntdll!DbgBreakPoint:

7c92120e cc int 3

0:025>

可见，本地内核态调试器已经勾住了远程用户态调试器的输入输出，此时进入用户态调试模式，在这种模式下，可以通过.shell命令对远程机器资源进行访问，如下所示：

0:025> .shell

.shell

Microsoft Windows XP [°?±? 5.1.2600]

(C) °?è¨?ùóD 1985-2001 Microsoft Corp.

C:\WINDOWS\system32><.shell waiting 1 second(s) for process>

<.shell process may need input>

ir

dir

<.shell waiting 1 second(s) for process>

?y?ˉ?÷ C ?Dμ??í??óD±ê???￡

?íμ?DòáDo?ê? BCE9-44CC

C:\WINDOWS\system32 μ?????

2015-11-08 12:50 <DIR> .

2015-11-08 12:50 <DIR> ..

2015-05-17 18:33 1,570 $winnt$.inf

2015-05-17 22:58 <DIR> 1025

2015-05-17 22:58 <DIR> 1028

2015-05-17 22:58 <DIR> 1031

此时已经进入了shell控制模式，要退出该模式用exit命令即可(+Enter)，如下所示：

C:\WINDOWS\system32><.shell waiting 1 second(s) for process>

<.shell process may need input>exit

exit

exit

<.shell waiting 1 second(s) for process>

.shell: Process exited

Press ENTER to continue

<.shell process may need input>

0:025>

现在回到了用户态调试模式，如果要返回内核态调试模式，可以用.sleep 1000，并迅速手动暂停内核调试器，这样就回到了内核调试器模式，如下所示：

0:025> .sleep 10000

.sleep 10000

Break instruction exception - code 80000003 (first chance)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*

\* You are seeing this message because you pressed either \*

\* CTRL+C (if you run console kernel debugger) or, \*

\* CTRL+BREAK (if you run GUI kernel debugger), \*

\* on your debugger machine's keyboard. \*

\* \*

\* THIS IS NOT A BUG OR A SYSTEM CRASH \*

\* \*

\* If you did not intend to break into the debugger, press the "g" key, then \*

\* press the "Enter" key now. This message might immediately reappear. If it \*

\* does, press "g" and "Enter" again. \*

\* \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

nt!RtlpBreakWithStatusInstruction:

80528bec cc int 3

### 如何控制目标系统？

命令：

.shell 在目标系统执行命令行

.breakin 从用户态中断到内核态调试器

适用范围：内核态/用户态

命令：

.crash 在目标系统崩溃

.reboot 重启目标系统

适用范围：内核态

### 如何在调试程序时无缝切换调试器以及实现多调试器？

1. 从windbg无缝切换到windbg

适用于用户态调试。以InstDrv.exe为例，现有一个Windbg.exe，命名为A，之后的Windbg命名为B

A附加调试InstDrv.exe，假设断在NtCreateFile，

0:004> g

Breakpoint 0 hit

ntdll!NtCreateFile:

00007fff`10061720 4c8bd1 mov r10,rcx

现在想将这个暂停状态接管给B，则以windbg –pe –p pid为参数启动B：

.....

Loading Wow64 Symbols

.........................................

(5cbc.468c): Wake debugger - code 80000007 (first chance)

No .natvis files found at C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\Visualizers.

ntdll!NtCreateFile+0x1:

00007fff`10061721 8bd1 mov edx,ecx

之后再使用windbg –pe –p 进程Id附加，之后对A执行g后关闭，此时控制权交给B，完成了无缝替换Windbg调试

1. 从ollydbg无缝切换到windbg

先使用ollydbg附加InstDrv.exeF9运行，之后使用windbg –pe –p 进程Id附加，停在初始断点后执行g：

.....

Loading Wow64 Symbols

....................................................

(e84.422c): Wake debugger - code 80000007 (first chance)

No .natvis files found at C:\Program Files (x86)\Windows Kits\10\Debuggers\x64\Visualizers.

wow64win!NtUserGetMessage+0xa:

00000000`76e65a2a c3 ret

0:000> g

(e84.227c): WOW64 breakpoint - code 4000001f (first chance)

First chance exceptions are reported before any exception handling.

This exception may be expected and handled.

ntdll\_76eb0000!NtQueryInformationProcess:

76eec600 cc int 3

此时将Ollydbg关闭即可，此时关闭并不会导致进程退出，之后便可以只用Windbg进行调试。

1. 多个windbg调试同一个进程

使用对于多调试器原理相同，均使用-pe进行附加即可，停在初始断点wow64win!NtUserGetMessage+0xa，便执行g即可成功接管进程。多个调试器使用的时候一定要小心，很容易导致内存损坏的问题。

1. 一个ollydbg多个windbg调试同一个进程

与③类似，只不过Ollydbg必须第一个附加该进程

### 如何调试当前调试器？

命令：.dbgdbg

适用范围：用户态/内核态

### 如何用应用程序控制调试器？

命令：.ocommand 命令标志前缀

适用范围：用户态

例：

用户程序代码为：OutputDebugStringA("test .echo 应用程序控制调试器;lm");

Windbg先执行命令：.ocommand test

在执行用户代码时，会输出以下信息并暂停：

应用程序控制调试器

start end module name

00000000`009c0000 00000000`009e3000 ConsoleApplication2 C (private pdb symbols) C:\Users\Administrator\Documents\Visual Studio 2015\Projects\ConsoleApplication2\Debug\ConsoleApplication2.pdb

00000000`0f100000 00000000`0f273000 ucrtbased (deferred)

00000000`57e40000 00000000`57ef9000 MSVCP140D (deferred)

00000000`58030000 00000000`5804c000 VCRUNTIME140D (deferred)

00000000`74630000 00000000`74684000 bcryptPrimitives (deferred)

00000000`74690000 00000000`7469a000 CRYPTBASE (deferred)

00000000`746a0000 00000000`746be000 SspiCli (deferred)

00000000`74760000 00000000`747dc000 ADVAPI32 (deferred)

00000000`76170000 00000000`7622a000 RPCRT4 (deferred)

00000000`76480000 00000000`764c1000 sechost (deferred)

00000000`76860000 00000000`76937000 KERNELBASE (deferred)

00000000`76b80000 00000000`76c43000 msvcrt (deferred)

00000000`76ca0000 00000000`76de0000 KERNEL32 (deferred)

00000000`76de0000 00000000`76e2b000 wow64 (deferred)

00000000`76e30000 00000000`76e39000 wow64cpu (deferred)

00000000`76e40000 00000000`76ea8000 wow64win (deferred)

00000000`76eb0000 00000000`7701e000 ntdll\_76eb0000 (pdb symbols) e:\symbol\wntdll.pdb\8C67971C1474490580FC7B7918183B462\wntdll.pdb

00007fff`0ffd0000 00007fff`1017c000 ntdll (pdb symbols) e:\symbol\ntdll.pdb\FA53ECC41AEA4238870E88A34FDA3C6C1\ntdll.pdb

wow64!Wow64NotifyDebugger+0x1d:

00000000`76df0309 65488b042530000000 mov rax,qword ptr gs:[30h] gs:00000000`00000030=????????????????

### 如何使用IDA调试Windows内核和驱动？

由于调试功能只作为IDA的插件，因此IDA的能力完全取决于目标调试软件的调试能力，这里以Windbg为例。

1. 先使用Windbg连接内核(确保Windbg可以连接和调试)，记录下命令行参数，例如com:pipe,resets=0,reconnect,port=\\.\pipe\kd\_xp。之后关闭Windbg，这一步只为获取参数用于后面IDA给Windbg传参
2. 设置IDA。——若IDA处于无文件反汇编的情况，此时Debug菜单只有Run和Attach两项，选择菜单Debugger->Attach->Windbg debugger，在Connection string中填入①得到的连接命令，接着点开Debug options按钮，这个界面里的选项根据情况选择。再点开Set specific options按钮后选择Kernel mode debugging，下面的Output flags根据情况选择即可。 此时点击确定后会出现对话框，选择唯一的一项Kernel即可进入！——若IDA处于反汇编状态，依然在Debugger->Debugger Options...-> Set specific options中做同样处理(设置Kernel mode debugging)，之后Debugger->Process options...的Connection string中填入前面得到的命令行参数，最后Debugger->Attach to process...中，选择唯一的一项Kernel即可进入！

## 其他

### 如何查看最耗费时间片的线程？

命令：!runaway 7

适用范围：用户态

例：

0:001> !runaway 7

User Mode Time

Thread Time

0:55c 0:00:00.0093

1:1a4 0:00:00.0000

Kernel Mode Time

Thread Time

0:55c 0:00:00.0140

1:1a4 0:00:00.0000

Elapsed Time

Thread Time

0:55c 0:00:43.0533

1:1a4 0:00:25.0876

### 如何加载和卸载插件？

命令： .load 插件名 加载插件

.unload 插件名 卸载插件

### 如何快速替换驱动文件?

是否存在这种情况困扰你：调试一个驱动，而发现某处需要改动，于是需要重新编译，拖到虚拟机里替换文件，如果该驱动是系统启动型的，就更麻烦一些，先关机然后映射成本地盘替换。Windbg提供了一种方式替换要加载的驱动，这样就免去了为了测试驱动而手动替换虚拟机文件的麻烦。

命令： .kdfiles –m 旧文件路径 新文件路径 指定文件路径替换

.kdfiles Map文件 指定映射文件替换

选项： 旧文件：为符号路径，必须和该驱动注册表服务项的ImagePath一致！，路径根据驱动启动类型不同可以是\Systemroot\....或\??\c:\....等格式

新文件：可以是本机文件或网络文件

Map文件：格式如下(d:\Map\_Files\mymap.ini)

map

\Systemroot\system32\drivers\videoprt.sys

e:\MyNewDriver\binaries\videoprt.sys

map

\Systemroot\system32\mydriver.sys

\\myserver\myshare\new\_drivers\mydriver0031.sys

# Here is a comment

map

\??\c:\windows\system32\beep.sys

[\\myserver\myshare\new\_drivers\new\_beep.sys](file:///\\myserver\myshare\new_drivers\new_beep.sys)

之后通过设置环境变量\_NT\_KD\_FILES，或.kdfiles命令设置map文件

适用范围：远程调试

触发时机：系统尝试加载被替换模块时

例：

kd> .kdfiles d:\Map\_Files\mymap.ini

KD file associations loaded from 'd:\Map\_Files\mymap.ini'

### 读写gflag

!gflag

### 分析蓝屏dump

命令：.dump 选项 dmp文件名 创建内存转储文件

选项：/m 创建minidump /f 创建full dump

!analyze –v

从内存文件映射地址获取文件名

### 显示当前使用的系统定时器

命令：!timer

适用范围：内核态

例：

kd> !timer

Dump system timers

Interrupt time: b77af511 00000020 [11/14/2015 00:50:19.756]

List Timer Interrupt Low/High Fire Time DPC/thread

PROCESSOR 0 (nt!\_KTIMER\_TABLE 83f35680)

0 870e1870 ce024890 00000020 [11/14/2015 00:50:57.553] thread 870e17e0

1 869ffb00 c6e108a8 00000020 [11/14/2015 00:50:45.591] thread 869ffa70

2 8858d590 3b094108 00008f0d [ 5/13/2016 22:01:06.813] thread 8858d500

8 86ab1610 d9fc34f1 00000020 [11/14/2015 00:51:17.646] thread 86ab1580

10 88a91608 0f3b27d5 0000002f [11/14/2015 02:32:59.932] thread 88a89a18

12 88988310 bd748dd0 00000020 [11/14/2015 00:50:29.781] thread 88987780

16 885ba518 7aa15e20 00000022 [11/14/2015 01:02:56.660] thread 885ba488

20 884316f8 aae6c787 0000005e [11/14/2015 08:13:47.450] thread 88434030

22 8863c188 adf6f3bb 00000021 [11/14/2015 00:57:13.288] thread 885fad48

23 83f44860 9169c708 00000021 [11/14/2015 00:56:25.387] nt!ExpTimeRefreshDpcRoutine (DPC @ 83f448a0)

25 8660f890 2d74bb94 0000002c [11/14/2015 02:12:22.151] thread 8660f800

29 86f401d8 P c25f9f00 00000020 [11/14/2015 00:50:38.032] afd!AfdCheckLookasideLists (DPC @ 86f40200)

888220c0 c723dc01 00000020 [11/14/2015 00:50:46.029] thread 88822030

### 命令：!mapped\_file

0:000> !mapped\_file 4121ec

Mapped file name for 004121ec: '\Device\HarddiskVolume2\CODE\TimeTest\Debug\TimeTest.exe'

开启调试子进程

命令：.childdbg 1/0 1开启 2关闭

Windbg插件相关：

插件要放在windbg根目录或插件文件夹中，加载后可以用命令“!插件名.help”来查看帮助，“!导出函数”来使用功能。

命令：.load 插件dll名 加载插件

命令：.unload 插件dll名 卸载插件

### **清屏**

命令：.cls

!htrace

!obtrace

!memlist

!irp !irpfind

!processirps

### 如何输出pdb中所有结构体？

### 如何让Windbg识别已知然而不存在于当前调试环境的结构体？

适用条件：假设正在调试a.exe，其中某地址是MY\_DATA结构体的一个实例，而a.exe对应的a.pdb中未存储MY\_DATA结构体，而结构体是已知的

若一个结构能在.pdb中存储，则需要是全符号的，且代码中存在该类型的变量。

那么强制Windbg加载某结构体符号的过程就可以描述为：

1. 在代码中使用结构体定义变量并以Debug编译成pe文件(dll/sys)
2. 选定空隙内存，使用.reload命令强制加载pe和符号

实例：

2.cpp内容：

typedef struct \_MY\_DATA

{

int a;

int b;

} MY\_DATA;

typedef MY\_DATA \*PMY\_DATA;

void main()

{

MY\_DATA data;

}

下面假设已经用windbg调试a.exe，则做如下操作：

0:000> .reload /f 2.exe=70000000,65536

\*\*\* WARNING: Unable to verify timestamp for 2.exe

0:000> dt \_MY\_DATA

2!\_MY\_DATA

+0x000 a : Int4B

+0x004 b : Int4B

### 如何查看错误代码含义？

在使用!error命令时我发现该指令并不能正常解析应用层错误，会返回“Error code: (Win32) 0x5 (5) - <Unable to get error code text>”类似的错误，因此自己实现了一个识别插件，实现起来并不难，先把ntstatus和winerror头文件中的错误号，用正则表达式处理成结构体即可。顺便设置了自动查找并解析应用层lasterror。详情见我编写的!WDbgLiExts.err

usage: !err [-c code][-l]

Default code is $retreg; l stands for Api LastError

example:!err -c C0000001 !err !err -l

### 如何扩展a指令为64位汇编？

在实践过程中发现windbg的a指令，只能实现32位x86指令汇编功能，对于其他平台只提供了接口却并没有实现，而x64作为Windows常用的平台却不能进行汇编不得不让人恼火，因此我对照a指令做了一个可以汇编x64指令的工具，提供符号解析(例如nt!NtCreateFile解析成地址)，原理是利用ml64编译得到机器码，后期打算扩展为更多平台。详情见我编写的!WDbgLiExts.a

usage: !a [-s ProcessorType] [-a Address]

Optional ProcessorType:I386|ARM|IA64|AMD64|EBC

Default ProcessorType is I386;Default Address is current $ip

example:!a -s AMD64 -a .