



恶意软件分析

第4章: 静态分析高级技术



主要内容

- 4.1 x86反汇编速成班
- 4.2 IDA Pro
- 4.3 识别汇编中的C代码结构
- 4.4 分析恶意Windows程序



4.1 x86反汇编速成班



基础技术

- 静态分析基础技术
 - 从表面看恶意代码
- 动态分析基础技术
 - 仅仅能分析特定情况下的恶意代码的行为
- 反汇编
 - 查看恶意代码指令，弄明白它究竟是什么



抽象层次

Malware Analysis

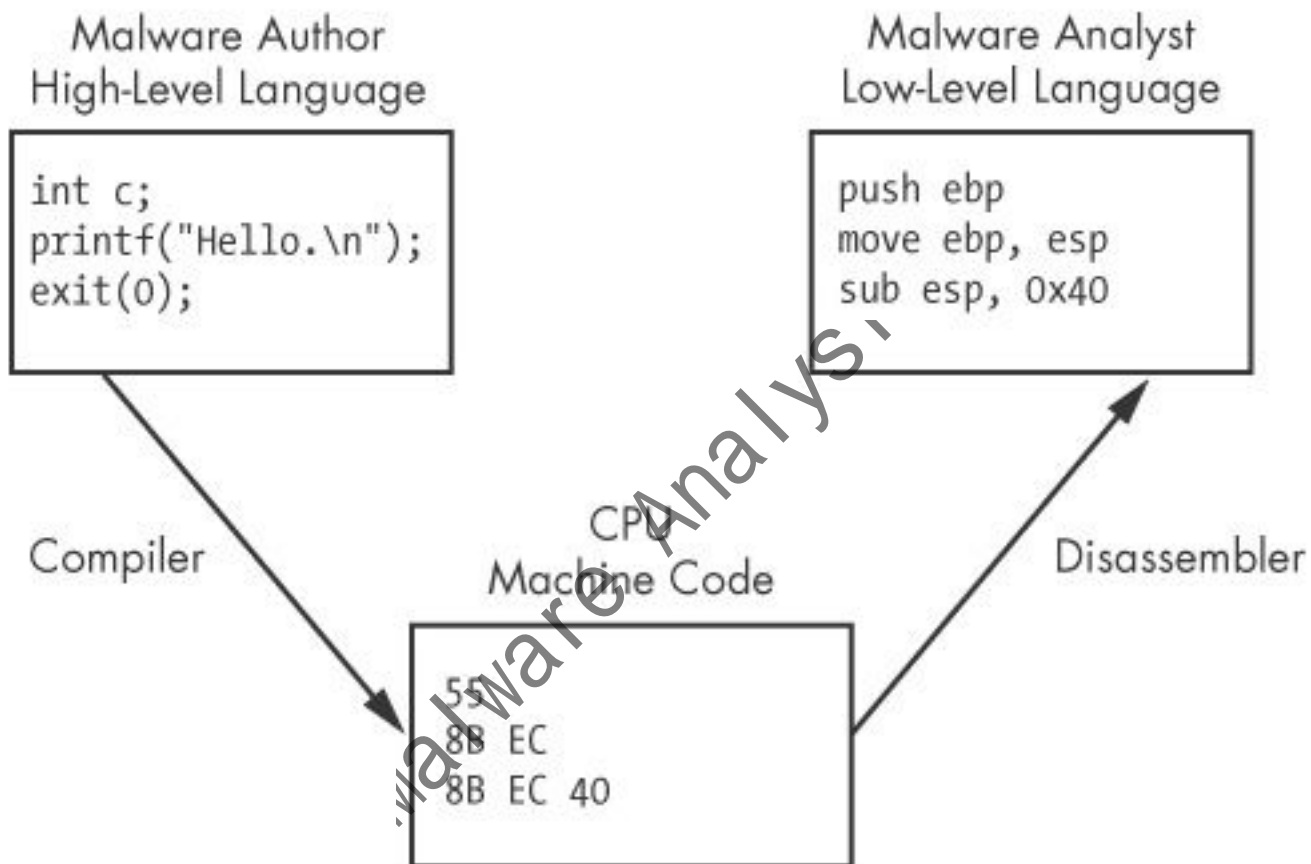


Figure 5-1. Code level examples



六个抽象层次

- 硬件
- 微指令
- 机器码
- 低级语言
- 高级语言
- 解释型语言

Malware Analysis



硬件

- 电子电路
- XOR、AND、OR、NOT 门
- 很难被软件操作

Malware Analysis



微指令

- 又称固件（**firmware**）
- 只能在为它设计的特定电路上执行
- 恶意代码分析时，通常不关心微指令



机器码

- 操作码 (Opcodes)
 - 告诉处理器想要做什么
 - 由高级语言编写的程序编译而来



低级语言

- 计算机体系结构指令集的人类易读版本
- 汇编语言
 - PUSH, POP, NOP, MOV, JMP ...
- 反汇编器生成汇编语言
- 在没有源代码的情况下，这是能从恶意代码中还原出来的最高层次语言



高级语言

- 大部分程序员使用它
- C、C++等
- 由编译器转换成机器码



解释型语言

- 位于最高层
- Java, C#, Perl, .NET, Python
- 代码不会被编译成机器码
- 会被翻译成字节码 (bytecode)
 - 一种中间表示
 - 不依赖硬件和操作系统
 - 字节码在解释器中执行。代码运行时解释器将字节码翻译成机器码
 - 例如: Java 虚拟机



逆向工程



反汇编

- 恶意代码存储在磁盘上，通常是机器码层的二进制形式
- 反汇编将二进制形式机器码转换成汇编语言
- **IDA Pro** 是最常用的反汇编器



汇编语言

- 每种处理器有不同的版本
- x86 – 32-bit Intel （最常见）
- x64 – 64-bit Intel
- SPARC, PowerPC, MIPS, ARM 等
- Windows 运行在x86 或 x64上
- x64 计算机能够运行 x86 程序
- 大部分恶意代码是为 x86设计的



x86 体系结构



- **CPU**（中央处理单元）执行代码
- **RAM** 存储数和代码
- **I/O** 输入输出系统，为硬盘、键盘、显示器等设备提供接口。

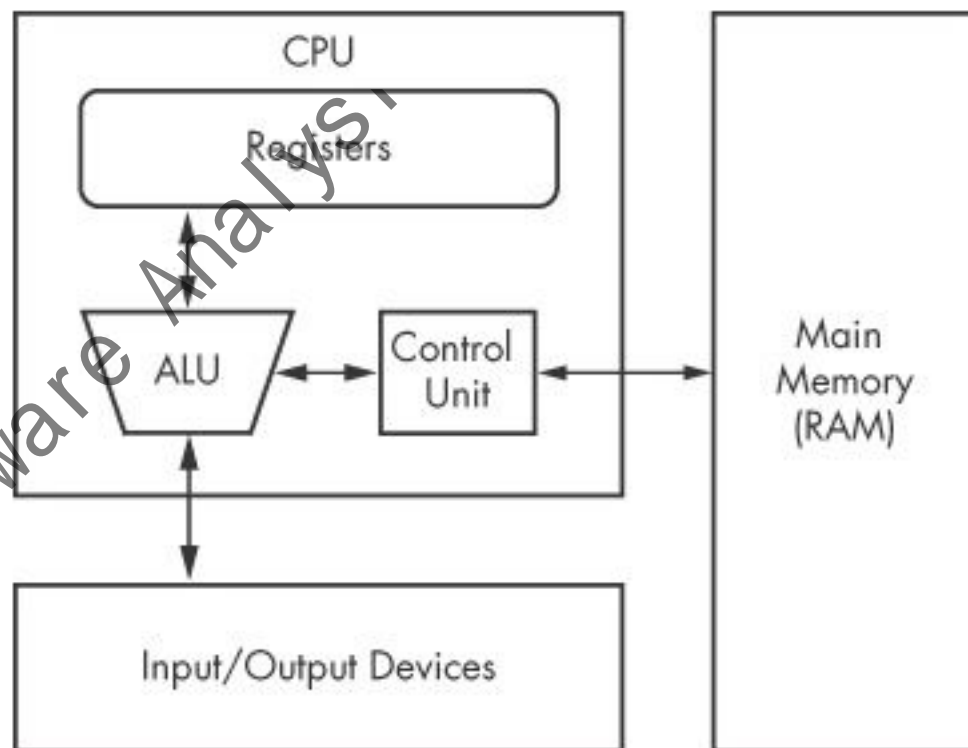


Figure 5-2. Von Neumann architecture



CPU组件

- 控制单元
 - 用指令指针寄存器从内存中取要执行的指令
- 寄存器
 - CPU中数据的基本存储单元
 - 存取速度比内存快
- 算术逻辑单元
 - 执行指令并将结果放在寄存器或内存中



内存(RAM)

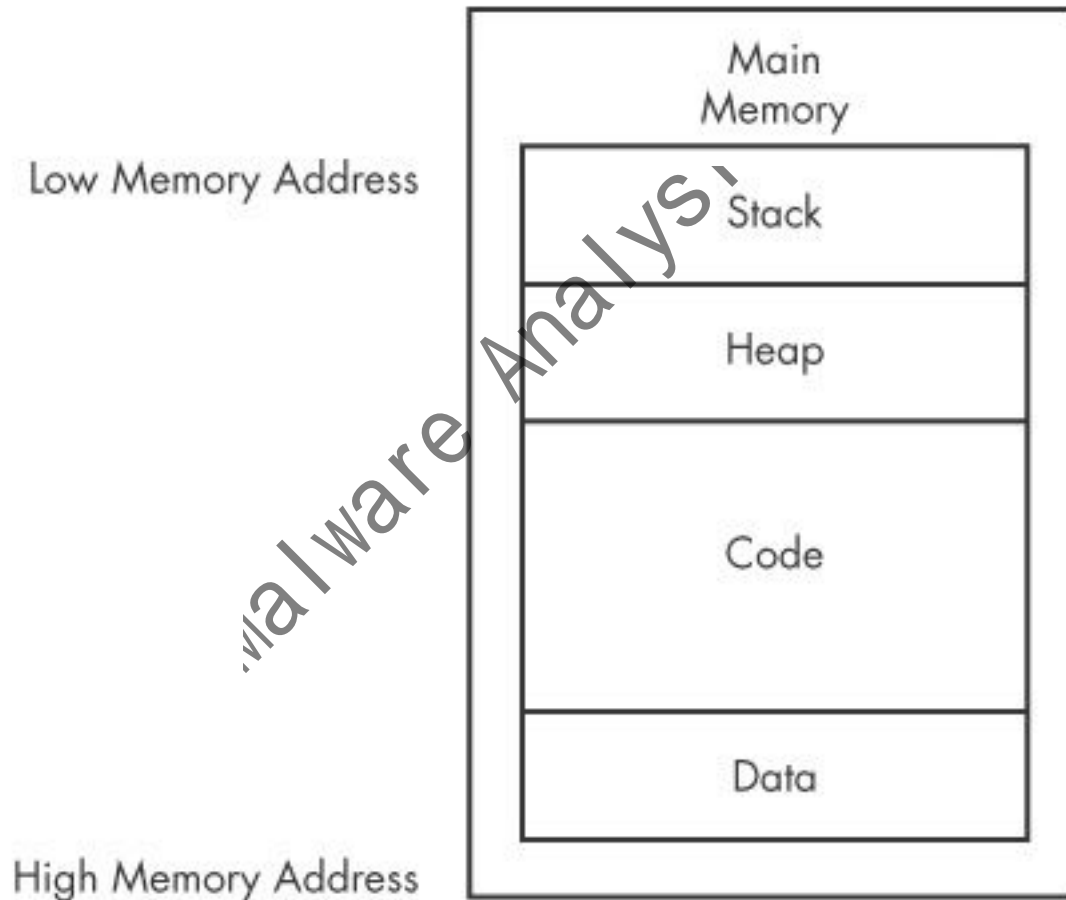


Figure 5-3. Basic memory layout for a program



数据

- 在程序加载时放入内存
- 静态值
 - 程序运行时不会被改变
- 也是全局值
 - 程序任何部分都可以使用它们

代码



- CPU指令
- 控制程序做什么

Malware Analysis

堆



- 动态内存
- 在程序运行过程中，其内容经常改变
- 程序用于分配新值和释放不需要的值

栈



- 函数的局部变量和参数
- 帮助控制程序执行流

Malware Analysis



指令

- 助记符后跟操作数
- `mov ecx 0x42`
 - 将十六进制数42存储到 `ecx` 寄存器
- “`mov ecx`” 为 十六进制 `0xB9`
- `0x42` 为 `0x4200000000`
- 这条指令的十六进制表示为
- `0xB942000000`



数据的字节序

- 大端
 - 最高字节放在最前边
 - 0x42 作为 64位数值表示为 0x00000042
- 小端
 - 最低字节放在最前边
 - 0x42作为 64位数值表示为 0x42000000
- 网络数据使用大端
- x86 程序使用小端



IP 地址

- 127.0.0.1十六进制为7F 00 00 01
- 在网络中传输时为0x7F000001
- 在内存中存储时为 0x0100007F



操作数

- 立即数
 - 固定值，如：x42
- 寄存器
 - eax, ebx, ecx等等
- 内存地址
 - Denoted with brackets, like [eax]

寄存器



Table 5-3. The x86 Registers

General registers	Segment registers	Status register	Instruction pointer
EAX (AX, AH, AL)	CS	EFLAGS	EIP
EBX (BX, BH, BL)	SS		
ECX (CX, CH, CL)	DS		
EDX (DX, DH, DL)	ES		
EBP (BP)	FS		
ESP (SP)	GS		
ESI (SI)			



寄存器

- 通用寄存器
 - CPU在执行期间使用
- 段寄存器
 - 用于定位内存节
- 状态标识
 - 用于做出决定
- 指令指针
 - 用于定位要执行的下一条指令



寄存器大小

- 通用寄存器大小均为 **32 位**
 - 可以在按32 (edx) 位或16 (dx) 位引用
- 四个寄存器 (**eax, ebx, ecx, edx**) 还可按8位引用
 - AL 为低8位
 - AH 为高8位

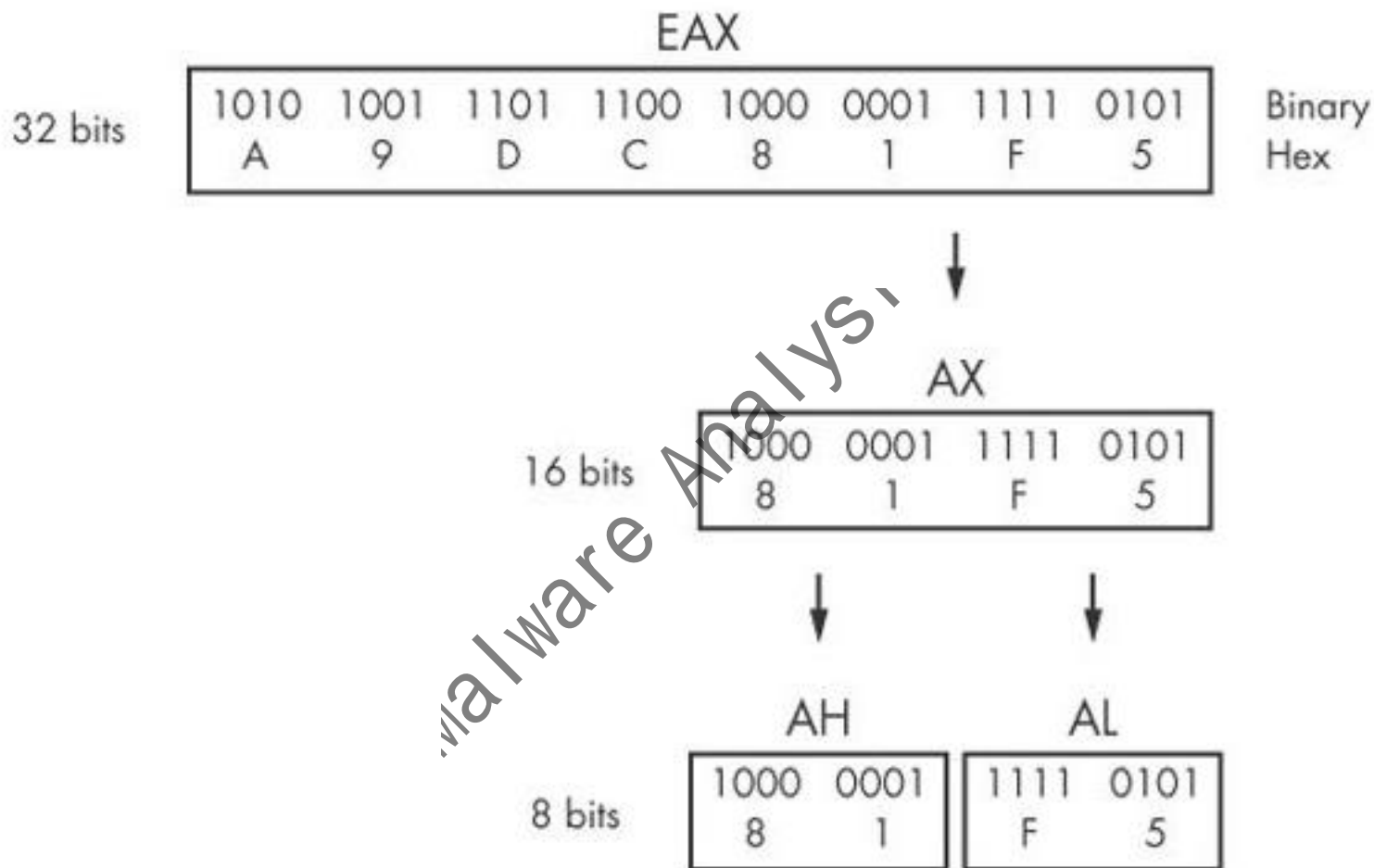


Figure 5-4. x86 EAX register breakdown



通用寄存器

- 一般用于存储数据或内存地址
- 经常交换着使用
- 一些指令只能使用特定寄存器
 - 乘法和除法指令只能使用**EAX**和**EDX**
- 约定
 - 编译器使用寄存器的一致方式
 - **EAX**存储函数返回值



标识寄存器

- EFLAGS是一个状态寄存器
- 32 位
- 每一位都是一个标识
- 置位为1或清除为0



重要标志位

- ZF
 - 运算结果为0，ZF被置位，否则被清除
- CF
 - 结果相对于目标操作数太大或者太小时CF被置位，否则被清除
- SF
 - 运算结果为负，或者运算结果最高位为1时，SF被置位
- TF
 - 用于调试，当它被置位时，x86处理器每次只执行一条指令



EIP 指令指针

- 保存程序要执行的下一条指令在内存中的地址
- 如果**EIP** 保存的是错误的数据CPU将获取到非法指令并崩溃
- 缓冲区溢出目标就是控制 **EIP**



简单指令

Malware Analysis



简单指令

- **mov** 目标操作数,源操作数
 - 将数据从一个位置移动到另一个位置
- 我们使用Intel汇编语法, 将目标操作数放在前面
- 间接寻址
 - **[ebx]** 为 **EBX**指向的内存位置



Table 5-4. mov Instruction Examples

Instruction	Description
<code>mov eax, ebx</code>	Copies the contents of EBX into the EAX register
<code>mov eax, 0x42</code>	Copies the value 0x42 into the EAX register
<code>mov eax, [0x4037C4]</code>	Copies the 4 bytes at the memory location 0x4037C4 into the EAX register
<code>mov eax, [ebx]</code>	Copies the 4 bytes at the memory location specified by the EBX register into the EAX register
<code>mov eax, [ebx+esi*4]</code>	Copies the 4 bytes at the memory location specified by the result of the equation $ebx+esi*4$ into the EAX register



lea （加载有效地址）

- lea 目标操作数, 源操作数
- lea eax, [ebx+8]
 - 将ebx存储的内存地址 + 8 付给eax
- 比较
 - mov eax, [ebx+8]
 - 将ebx+8内存地址指向的数据付给eax



- `mov eax, [ebx+8]` 执行后eax等于0x20
- `lea eax, [ebx+8]` 执行后eax等于0x00B30048

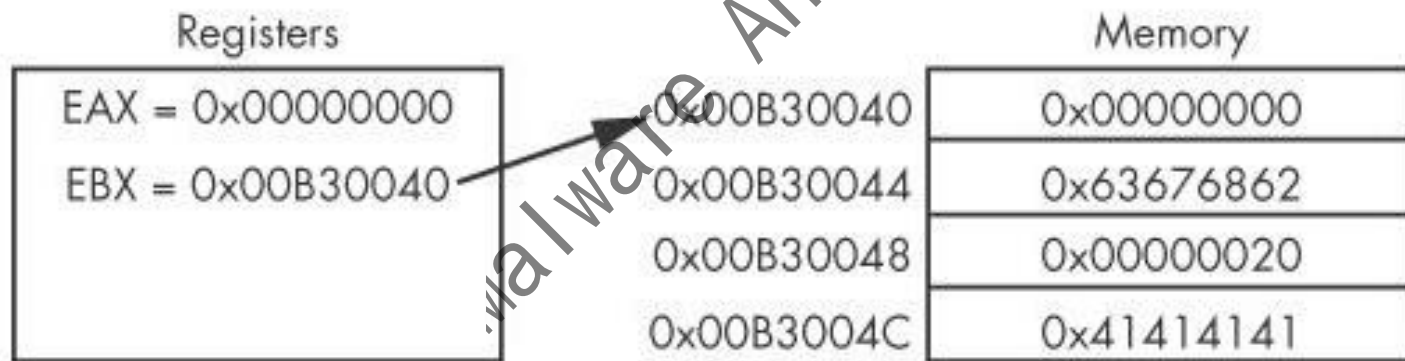


Figure 5-5. EBX register used to access memory



算术运算

- sub 减
- add 加
- inc 增加
- dec 减少
- mul 乘
- div 除

Malware Analysis



NOP指令

- 什么事情也不做
- 0x90
- 通常用做NOP滑板 (NOP Slid)
- 利用它, 攻击者即使他们不知道跳转的准确位置, 也可以运行代码



栈

- 函数的内存、局部变量、流控制结构
- 后进先出
- **ESP** 指向栈顶
- **EBP** 指向栈底
- **PUSH** 数据入栈
- **POP** 数据出栈



其他栈相关指令

- 函数都会使用
 - Call
 - Leave
 - Enter
 - Ret

Malware Analysis



函数调用

- 小的程序做一件事情并返回如: **printf()**
- “序言”
 - 函数开始处的指令用于为函数准备栈和寄存器
- “结语”
 - 函数结束前的指令用于恢复到函数调用前栈和寄存器的状态

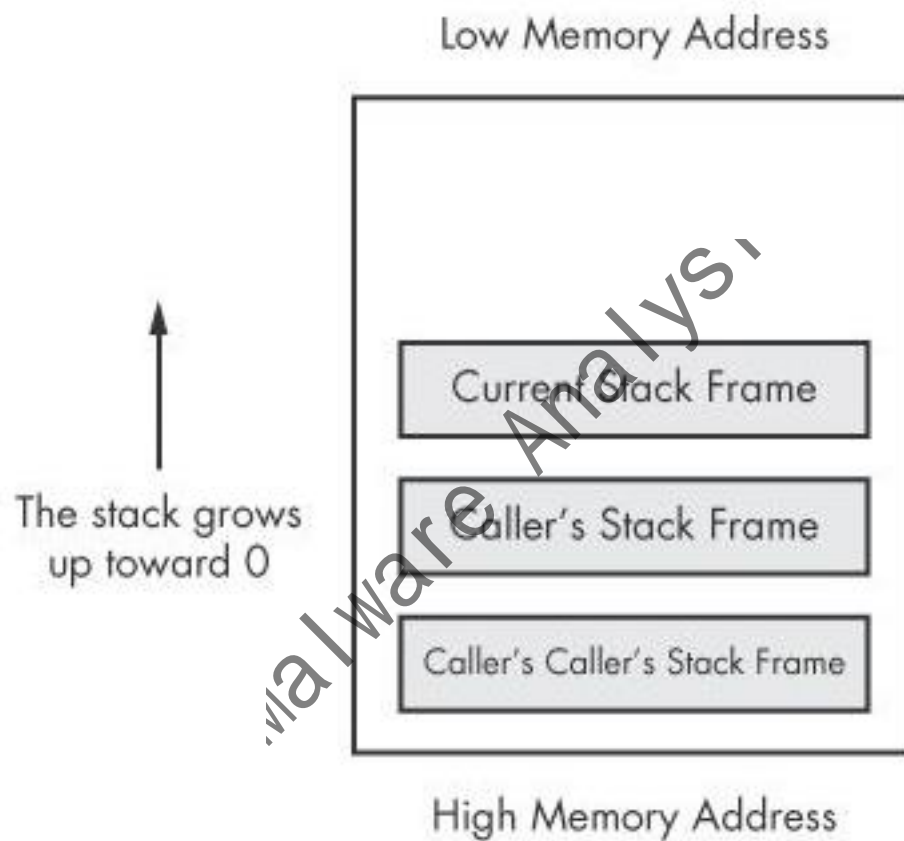


Figure 5-7. x86 stack layout

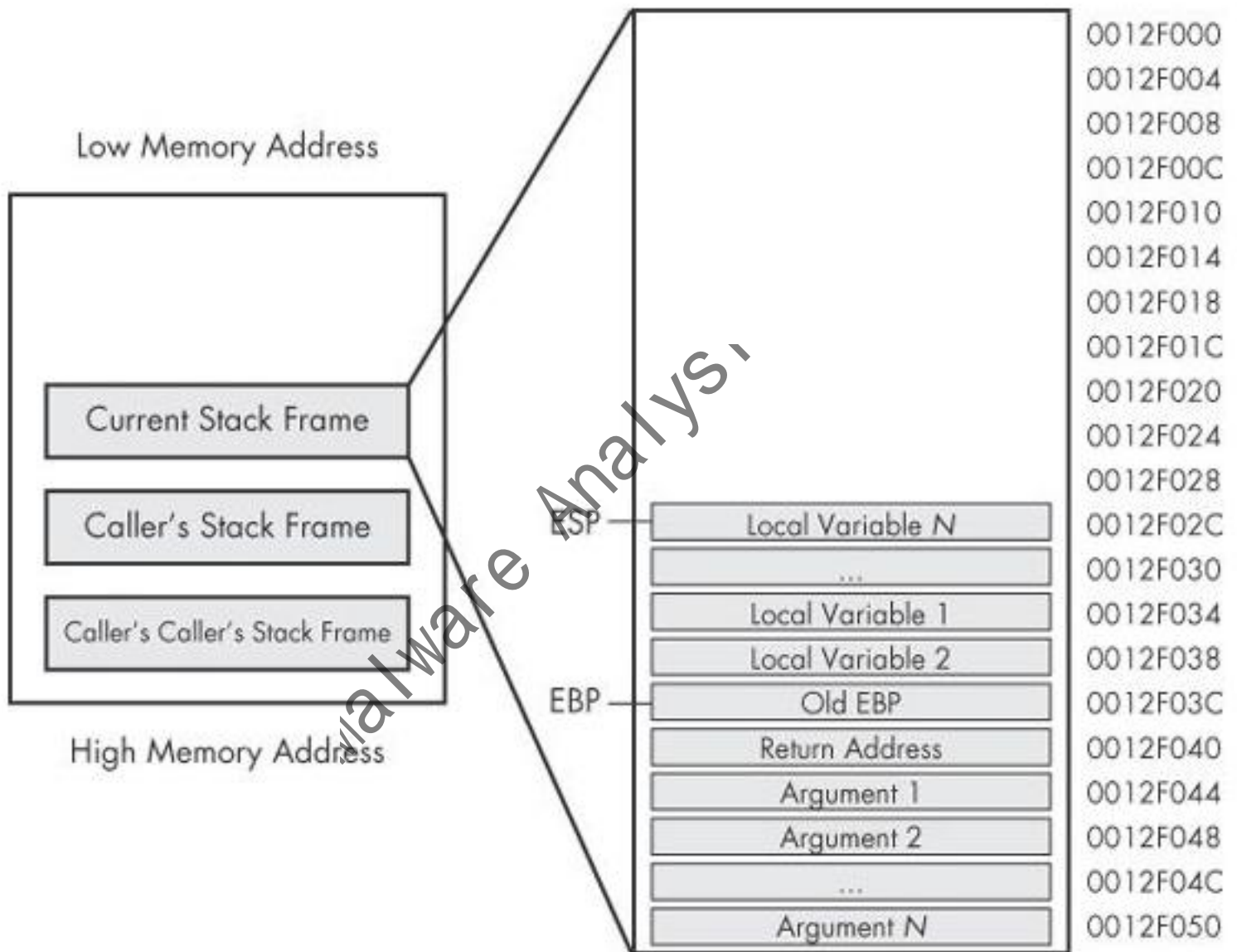


Figure 5-8. Individual stack frame



条件指令

- **test**
 - 和and指令功能一样比较两个数值，但它并不会修改其使用的操作数
 - **test eax, eax**
 - 如果eax为0 ZF标志位置位
- **cmp eax, ebx**
 - 如果两个参数相等，ZF标志位置位



分支指令

- jz
 - 如果ZF标志位置位则跳转
- jnz
 - 如果ZF标志位被清除则跳转



C 语言主函数

- 每一个C语言的程序都有一个main() 函数
- `int main(int argc, char** argv)`
 - `argc` 为命令行中参数的个数
 - `argv` 为字符串数组指针, 指向所有的命令行参数

例子

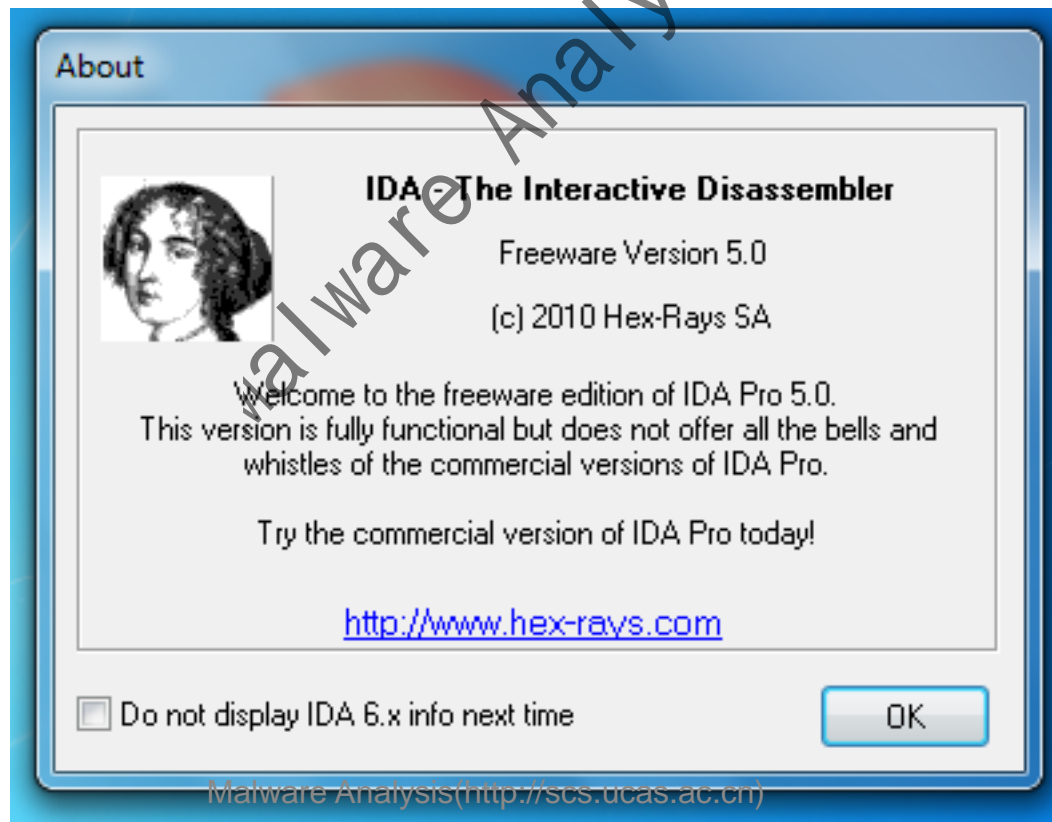


- `cp foo bar`
- `argc = 3`
- `argv[0] = cp`
- `argv[1] = foo`
- `argv[2] = bar`

Malware Analysis



4.2 IDA Pro



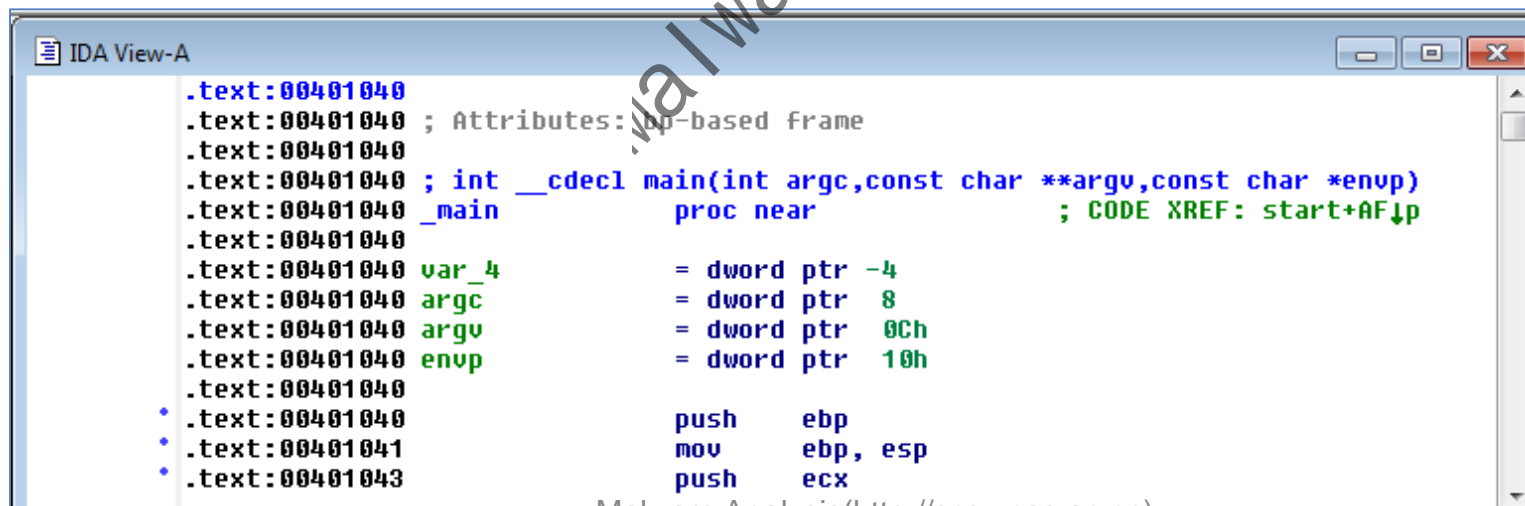
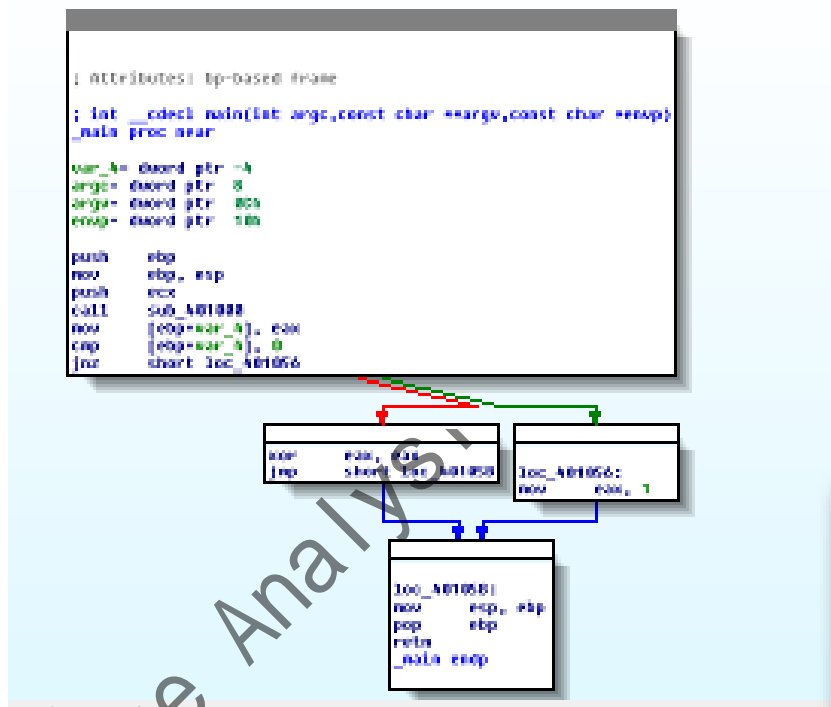


IDA Pro 版本

- 功能齐全的商业版本
- 旧的免费版本
 - 均支持x86
 - 商业版支持x64 和其他处理器如：手机处理器
- 快速库标记和识别技术（FLIRT）中包含了公用库的代码特征

图形和文本模式

- 空格键切换两种模式





默认的图形模式视图

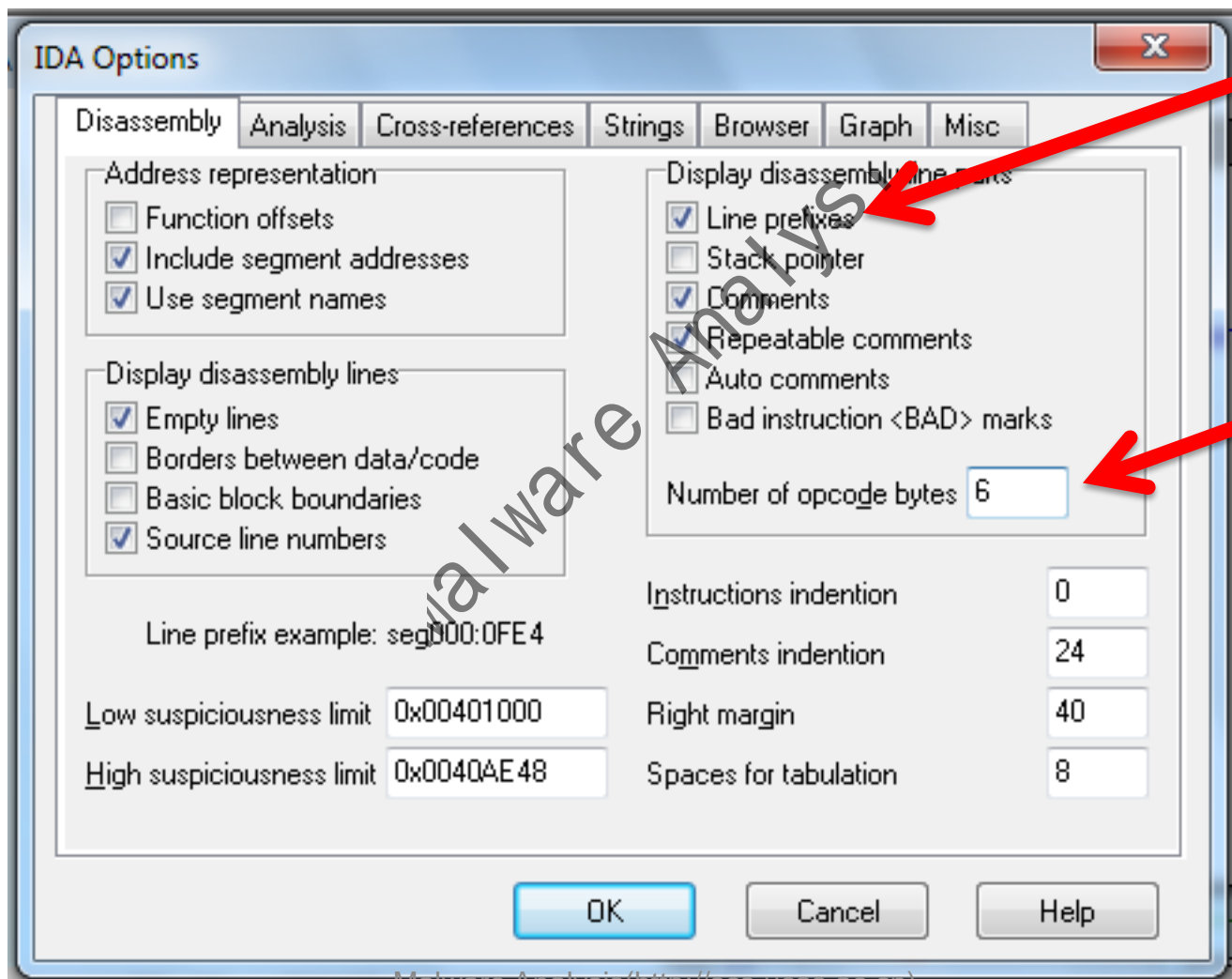
```
Windows [N]
; Attributes: bp-based frame

; int __cdecl main(int argc,const char **argv,const char *envp)
_main proc near

var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
push    ecx
call    sub_401000
mov     [ebp+var_4], eax
cmp     [ebp+var_4], 0
jnz     short loc_401056
```


设置显示行号和操作码选项





设置后的图形模式视图

```
00401040
00401040
00401040      ; Attributes: bp-based frame
00401040
00401040      ; int __cdecl main(int argc,const char **argv,const char *envp)
00401040      _main proc near
00401040
00401040      var_4= dword ptr -4
00401040      argc= dword ptr  8
00401040      argv= dword ptr  0Ch
00401040      envp= dword ptr  10h
00401040
00401040 55          push    ebp
00401041 8B EC      mov     ebp, esp
00401043 51          push    ecx
00401044 E8 B7 FF FF FF  call   sub_401000
00401049 89 45 FC      mov     [ebp+var_4], eax
0040104C 83 7D FC 00    cmp     [ebp+var_4], 0
00401050 75 04        jnz     short loc_401056
```

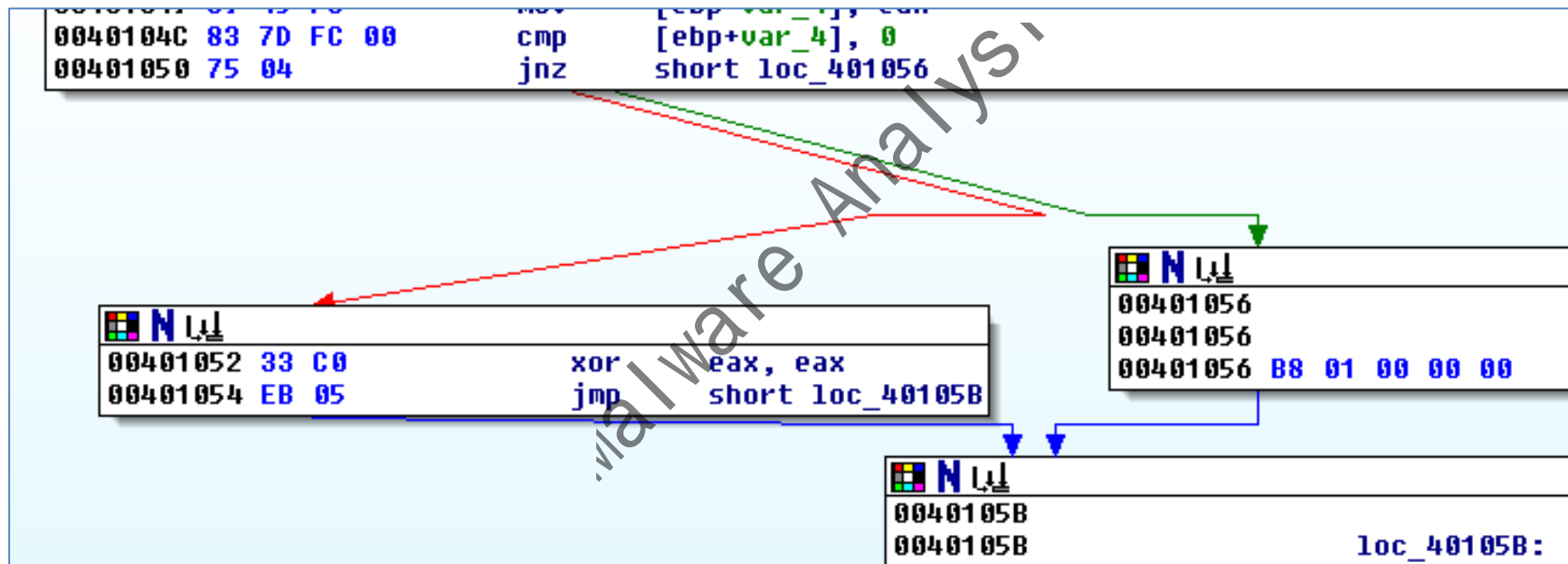


图形模式中的箭头

- 颜色
 - 红 表示条件跳转没有被采用
 - 绿 表示条件跳转被采用
 - 蓝 表示无条件跳转被采用
- 方向
 - 向上 表示循环



箭头颜色事例





高亮文本功能

- 图形模式中的高亮文本功能会高亮显示该文本的每个实例



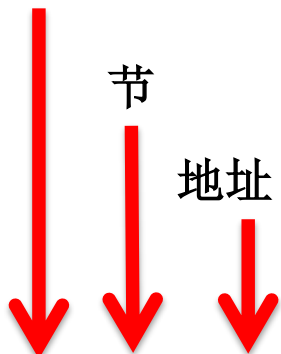
IDA Pro添加的
注释

文本模式

箭头
实线 = 无条件跳转
虚线 = 有条件跳转
向上 = 循环

节

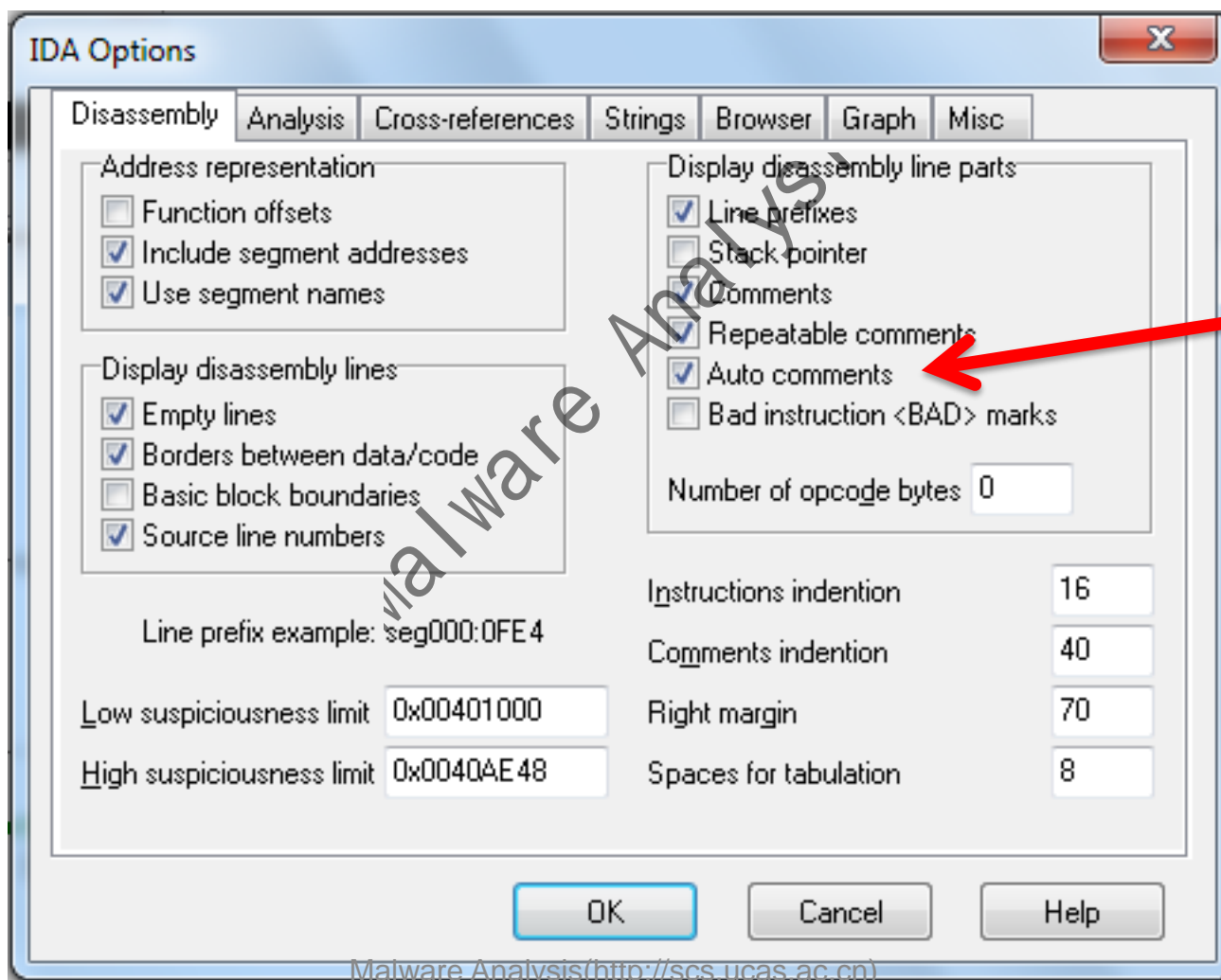
地址



```
.text:00401015      jz         short loc_40102B
.text:00401017      push        offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C      call       sub_40105F
.text:00401021      add         esp, 4
.text:00401024      mov         eax, 1
.text:00401029      jmp         short loc_40103A
.text:0040102B      ; -----
.text:0040102B      loc_40102B: ; CODE XREF: sub_401000+151j
.text:0040102B      push        offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:00401030      call       sub_40105F
.text:00401035      add         esp, 4
.text:00401038      xor         eax, eax
.text:0040103A      loc_40103A: ; CODE XREF: sub_401000+291j
.text:0040103A      mov         esp, ebp
.text:0040103C      pop         ebp
```



设置自动注释选项





为每条指令添加注释

```
.text:00401015      jz      short loc_40102B ; Jump if Zero (ZF=1)
.text:00401017      push     offset aSuccessInterne ; "Success: Internet Connection\n"
.text:0040101C      call    sub_40105F ; Call Procedure
.text:00401021      add     esp, 4 ; Add
.text:00401024      mov     eax, 1
.text:00401029      jmp     short loc_40103A ; Jump
.text:0040102B ; -----
.text:0040102B      loc_40102B: ; CODE XREF: sub_401000+15↑j
.text:0040102B      push     offset aError1_1NoInte ; "Error 1.1: No Internet\n"
.text:00401030      call    sub_40105F ; Call Procedure
.text:00401035      add     esp, 4 ; Add
.text:00401038      xor     eax, eax ; Logical Exclusive OR
.text:0040103A      loc_40103A: ; CODE XREF: sub_401000+29↑j
.text:0040103A      mov     esp, ebp
.text:0040103C      pop     ebp
```




对分析有用的窗口



函数窗口

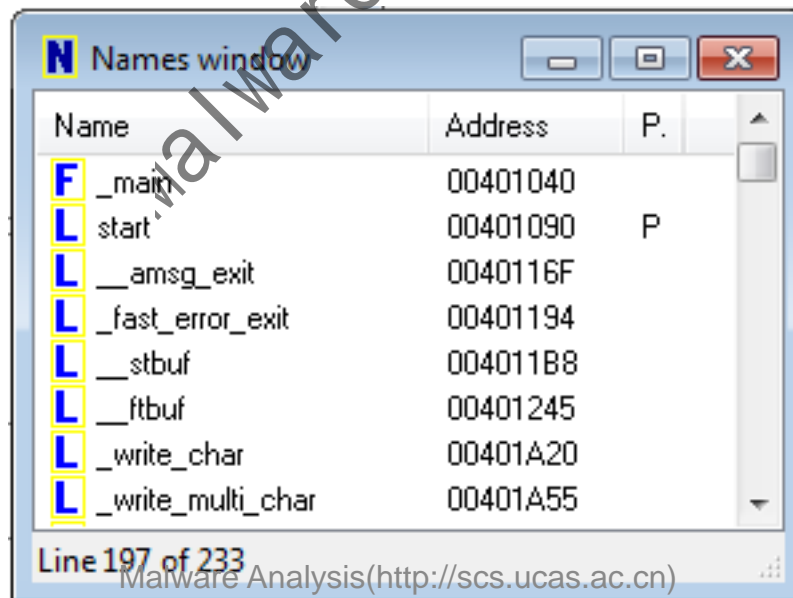
- 列举可执行文件中的所有函数，并显示每个函数的长度和标志
 - L = 库函数
- 排序
 - 规模庞大的函数通常更重要

Function name	Segment	Start	Length	R	F	L	S	B	T	=
CheckWindowsGenuineStatus()	.text	010091F9	0000007C	R	.	.	.	B	.	.
ControlBackgroundBrushInfo: 'scalar deletingtext	01032897	00000029	R	.	.	.	B	.	.
CreateDecoderFromResource(IWICImagingF...	.text	0101FB50	00000097	R	.	.	.	B	T	.



名字窗口

- 列举每个地址的名字
 - 函数、命名代码、命名数据和字符串





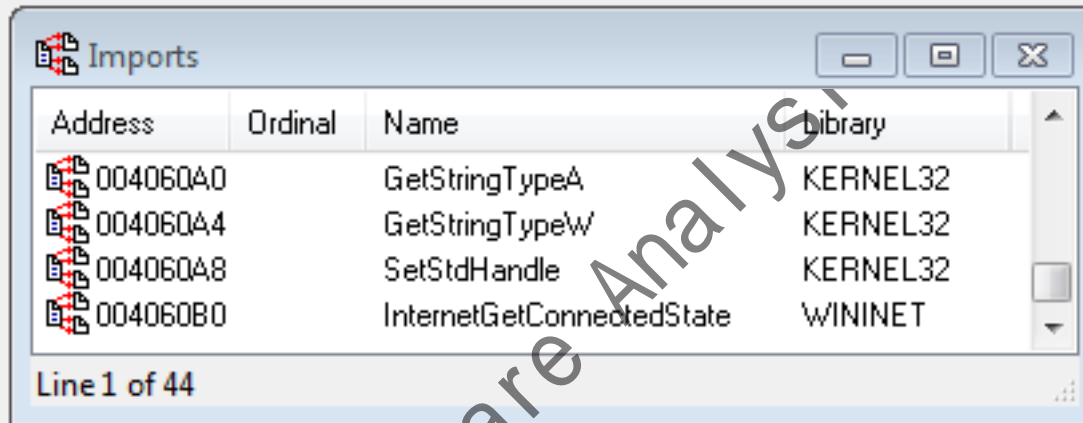
字符串窗口

Strings window

Address	Length	Type	String
00401000	0000000F	C	GetStringTypeW
0040100F	0000000D	C	SetStdHandle
0040101C	0000000C	C	CloseHandle
0040102D	0000000D	C	KERNEL32.dll
00401038	00000018	C	Error 1.1: No Internet\n
00401050	0000001E	C	Success: Internet Connection\n



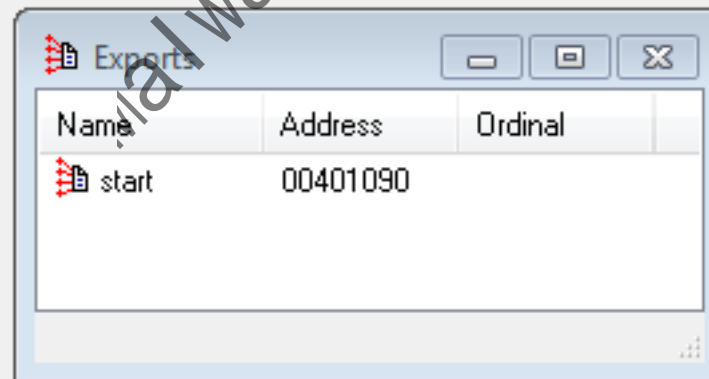
导入、导出表窗口



Imports

Address	Ordinal	Name	Library
004060A0		GetStringTypeA	KERNEL32
004060A4		GetStringTypeW	KERNEL32
004060A8		SetStdHandle	KERNEL32
004060B0		InternetGetConnectedState	WININET

Line 1 of 44



Exports

Name	Address	Ordinal
start	00401090	

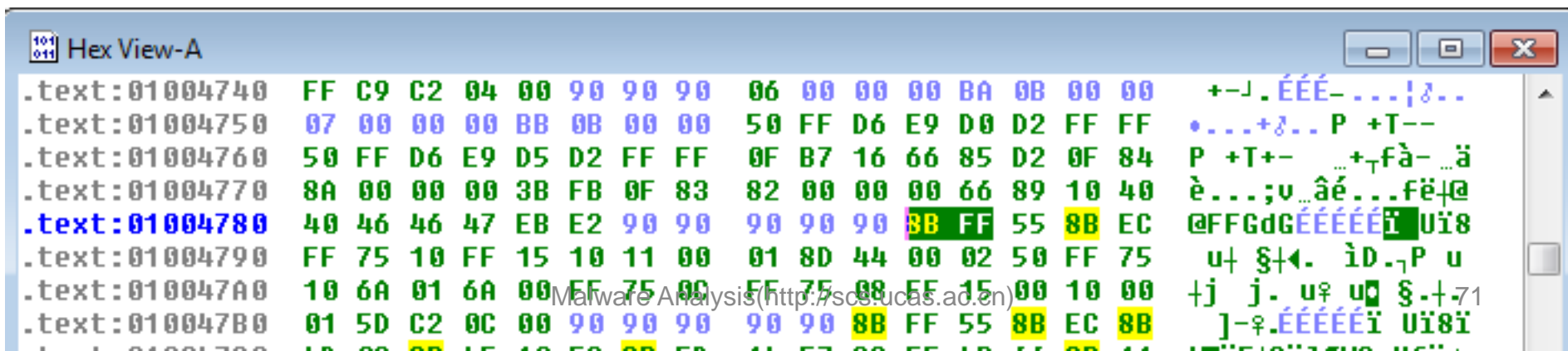
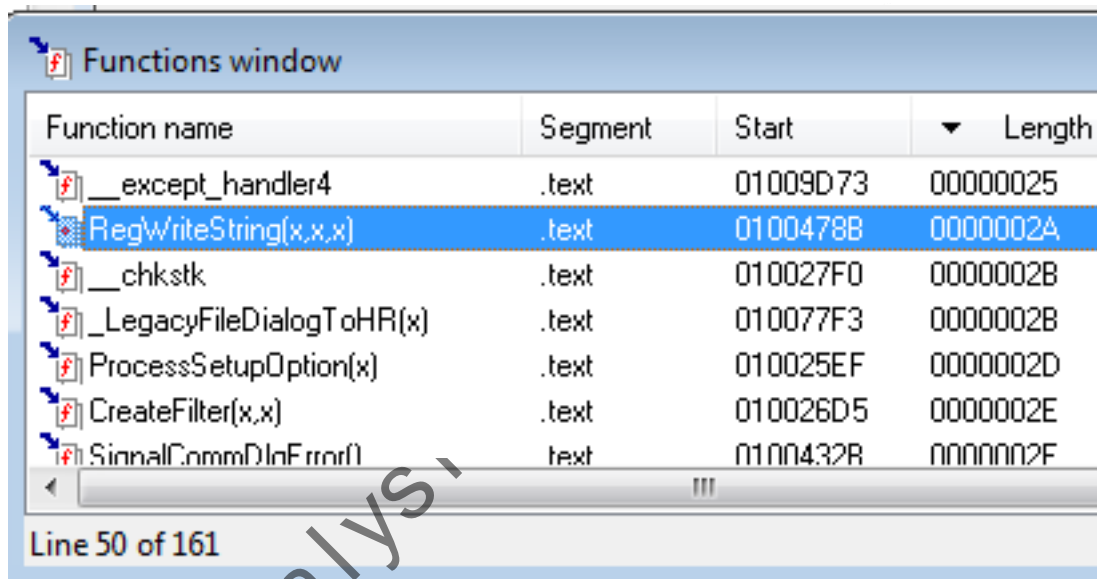


- 列举所有活跃数据结构的布局
 - 悬停看到黄色的弹出窗口



交叉引用

- 双击函数
- 跳转到其他视图中的代码





函数调用

- 参数压入栈
- 调用函数

```
0100478B
0100478B
0100478B      ; Attributes: 40- Based frame
0100478B
0100478B      ; int __stdcall RegWriteString(HKEY hKey,LPCWSTR lpValueName,BYTE *lpData)
0100478B      _RegWriteString@12 proc near
0100478B
0100478B      hKey= dword ptr  8
0100478B      lpValueName= dword ptr  0Ch
0100478B      lpData= dword ptr  10h
0100478B
0100478B 8B FF      mov     edi, edi
0100478D 55        push    ebp
0100478E 8B EC      mov     ebp, esp
01004790 FF 75 10    push    [ebp+lpData] ; lpString
01004793 FF 15 10 11 00 01 call    ds:__imp__lstrlenW@4 ; lstrlenW(x)
```

0.00% (-30,-41) (788,342) 00003B8B 0100478B: RegWriteString(x,x,x)



返回到默认视图

- 选择Windows中的Reset Desktop返回默认视图
- 选择Windows中的Save Desktop保存这个新视图

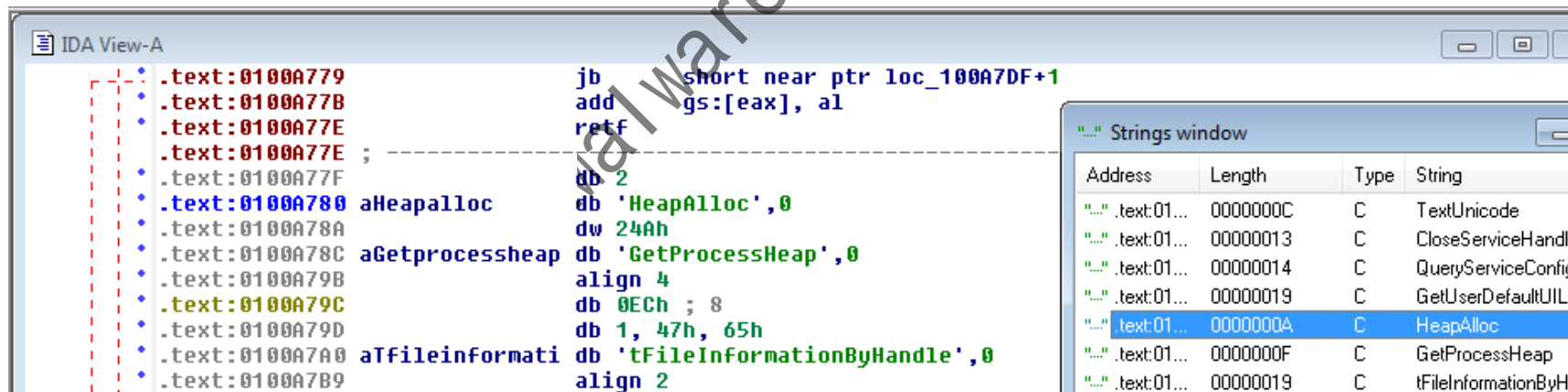


IDA Pro简介



导出表或者字符串

- 双击窗口中的项，会把你带到这个项的被使用的位置





使用链接

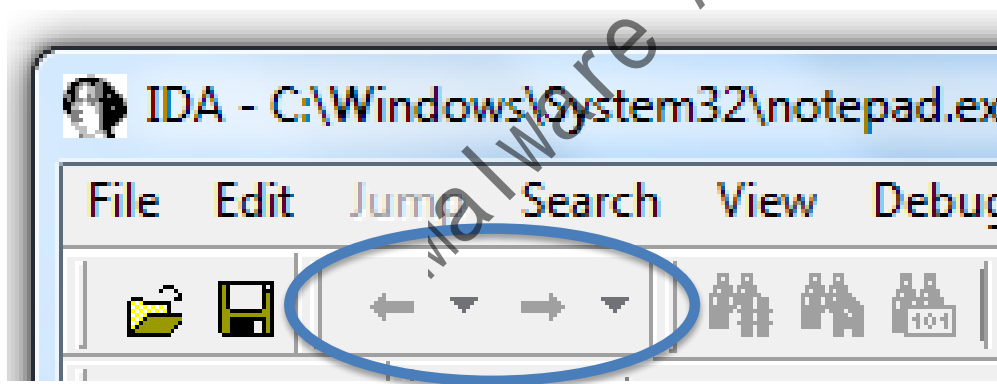
- 双击反汇编窗口中的任一地址，都会显示目标的位置

```
IDA View-A
• .text:010047A1      push     1           ; dwType
• .text:010047A3      push     0           ; Reserved
• .text:010047A5      push     [ebp+lpValueName] ; lpValueName
• .text:010047A8      push     [ebp+hKey]   ; hKey
• .text:010047AB      call     ds:imp_RegSetValueExW@324 ; RegSetValueExW(x,x,x,x,x,x)
• .text:010047B1      pop      ebp
• .text:010047B2      ret      0Ch
• .text:010047B2      _RegWriteString@12 endp
• .text:010047B2
```



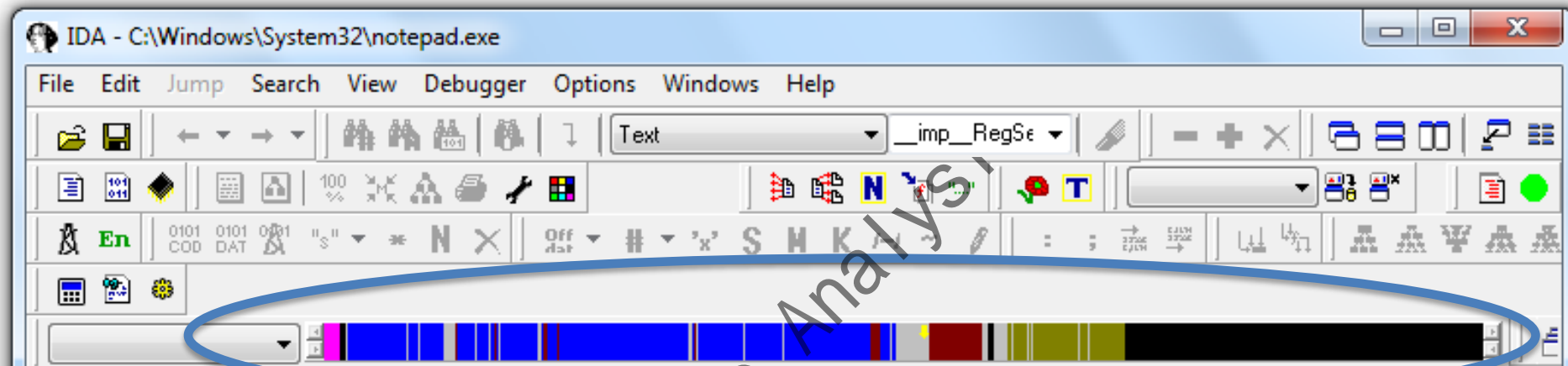
浏览历史

- 前进后退按钮，使得你在历史视图中来回移动，就像使用浏览器在网页访问历史中来回移动一样





导航栏

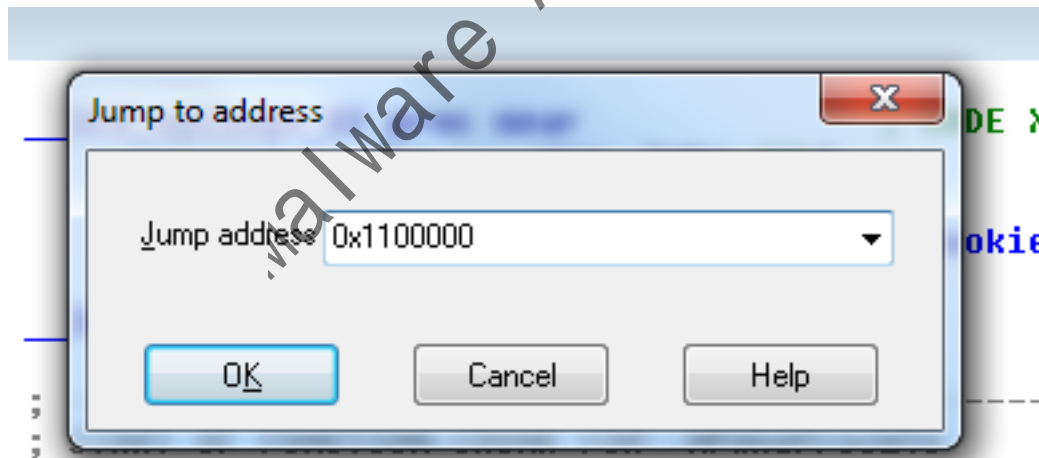


- 浅蓝色： 库代码
- 红色： 编译器生成代码
- 深蓝色： 用户编写的代码— 分析这里



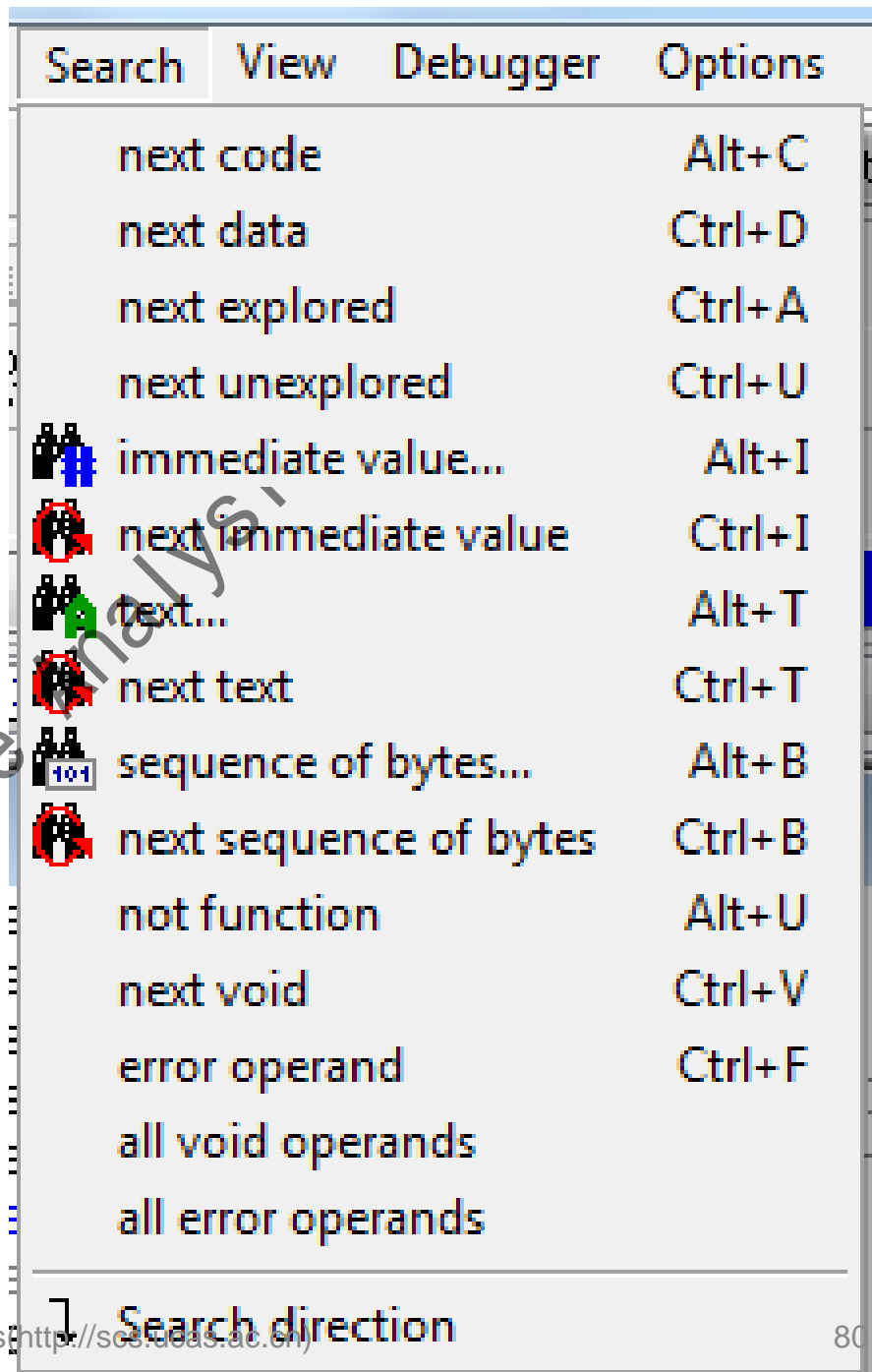
跳转到位置

- 按下G键
- 就能跳转到地址或命名的位置



搜索

- 很多选项
- 搜索文本很方便





使用交叉引用



代码交叉引用

```
.text:00401440
.text:00401440 ; !!!!!!!!!!!!!!! SUBROUTINE !!!!!!!!!!!!!!!
.text:00401440
.text:00401440
.text:00401440 ; int __cdecl main(int argc,const char **argv,const char *envp)
.text:00401440 _main          proc near          ; CODE XREF: start+DE↓
.text:00401440
.text:00401440 var_44          = dword ptr -44h
.text:00401440 var_40          = dword ptr -40h
.text:00401440 var_3C          = dword ptr -3Ch
.text:00401440 var_38          = dword ptr -38h
.text:00401440 var_34          = dword ptr -34h
.text:00401440 var_30          = dword ptr -30h
.text:00401440 var_2C          = dword ptr -2Ch
.text:00401440 var_28          = dword ptr -28h
.text:00401440 var_24          = dword ptr -24h
.text:00401440 var_20          = dword ptr -20h
.text:00401440 var_1C          = dword ptr -1Ch
.text:00401440 var_18          = dword ptr -18h

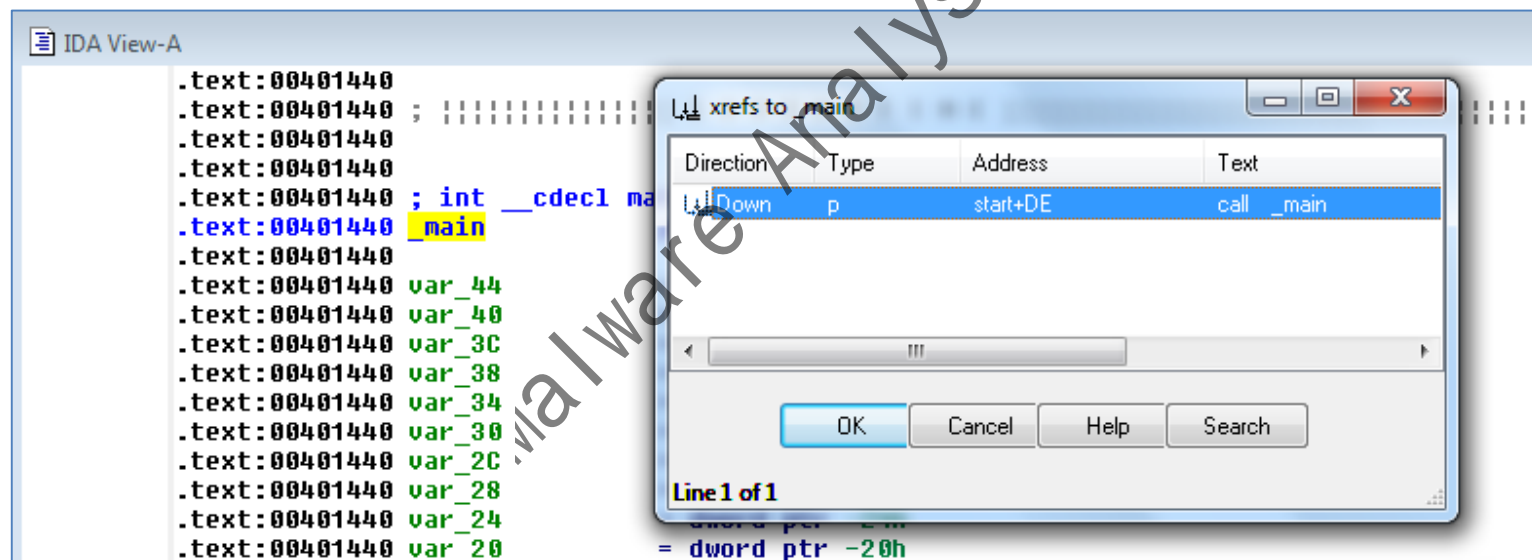
push    offset unk_403000
call     _initterm
call     ds:__p__initenv
mov     ecx,[ebp+envp]
mov     [eax],ecx
push    [ebp+envp]          ; envp
push    [ebp+argv]         ; argv
push    [ebp+argc]         ; argc
call     main
add     esp, 30h
```

- XREF 注释显示函数是被哪里调用的
- 但默认情况下只显示少数几个交叉引用



怎么查看所有的交叉引用

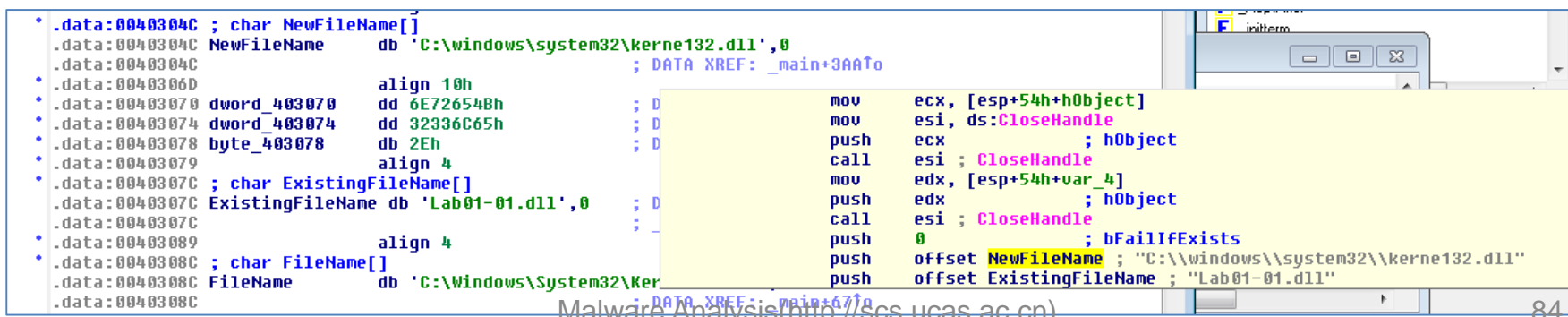
- 点击函数名并按 X 键





数据交叉引用

- 演示：
 - 从字符串开始
 - 双击一个感兴趣的字符串
 - 悬停在数据XREF，查看哪里使用了这个字符串
 - X 键显示所有引用



```
.data:0040304C ; char NewFileName[]
.data:0040304C NewFileName db 'C:\windows\system32\kerne132.dll',0
.data:0040304C ; DATA XREF: _main+3AAfo
.data:0040306D align 10h
.data:00403070 dword_403070 dd 6E726548h
.data:00403074 dword_403074 dd 32336C65h
.data:00403078 byte_403078 db 2Eh
.data:00403079 align 4
.data:0040307C ; char ExistingFileName[]
.data:0040307C ExistingFileName db 'Lab01-01.dll',0
.data:0040307C ;
.data:00403089 align 4
.data:0040308C ; char FileName[]
.data:0040308C FileName db 'C:\Windows\System32\Ker
.data:0040308C
```

```
mov ecx, [esp+54h+hObject]
mov esi, ds:CloseHandle
push ecx ; hObject
call esi ; CloseHandle
mov edx, [esp+54h+var_4]
push edx ; hObject
call esi ; CloseHandle
push 0 ; bFailIfExists
push offset NewFileName ; "C:\windows\system32\kerne132.dll"
push offset ExistingFileName ; "Lab01-01.dll"
```



分析函数



函数和参数识别

- IDA Pro 识别函数、标记函数和标记局部变量
- 并不总是正确的

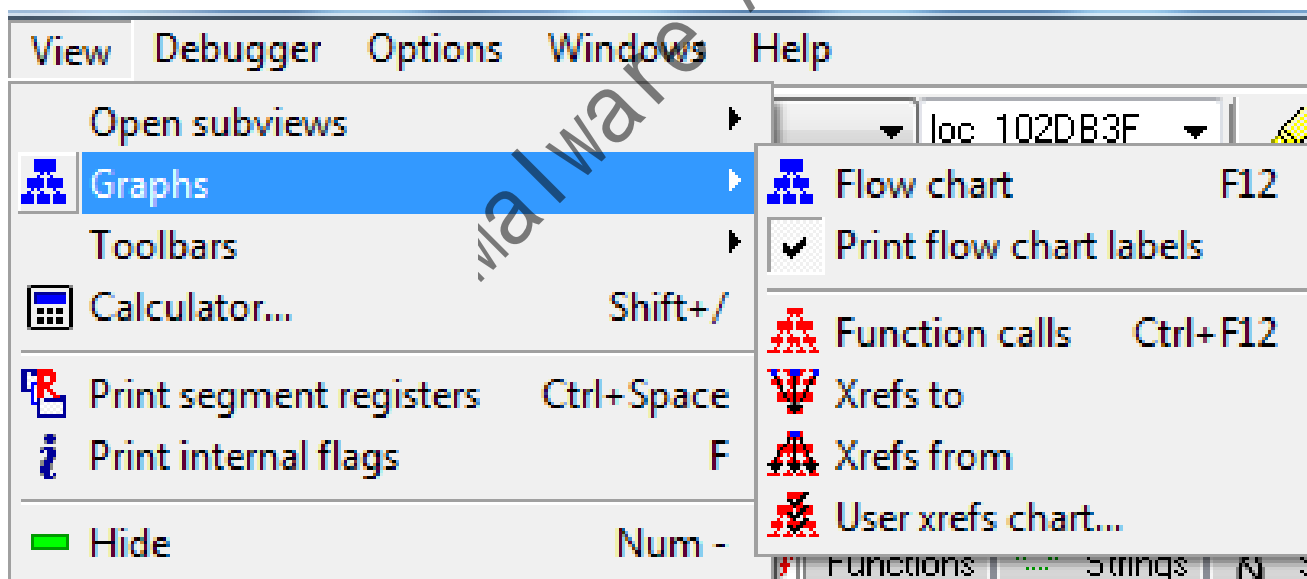
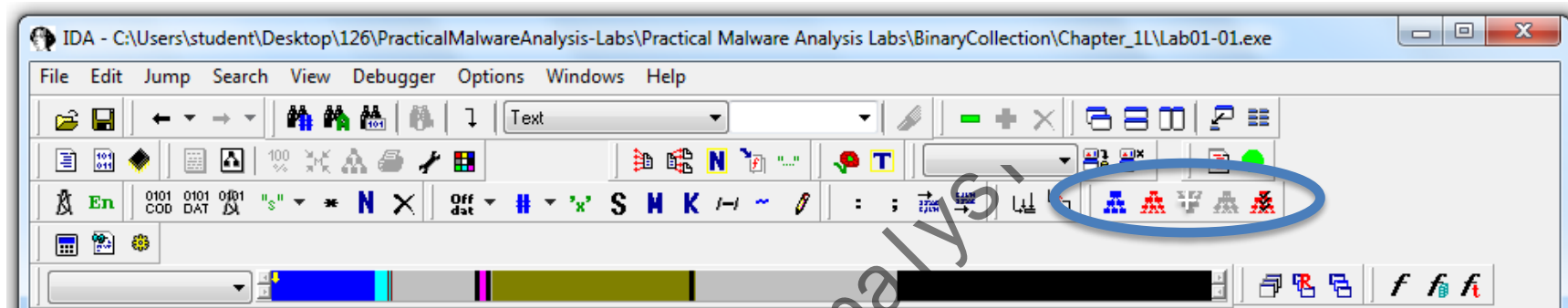
```
IDA View-A
.text:00401040
.text:00401040
.text:00401040 sub_401040 proc near ; CODE XREF: sub_4010A0+88↓p
.text:00401040 ; sub_4010A0+B7↓p ...
.text:00401040 arg_0 = dword ptr 4
.text:00401040 arg_4 = dword ptr 8
.text:00401040 arg_8 = dword ptr 0Ch
* .text:00401040 mov eax, [esp+arg_4]
* .text:00401044 push esi
* .text:00401045 mov esi, [esp+4+arg_0]
* .text:00401049 push eax
```



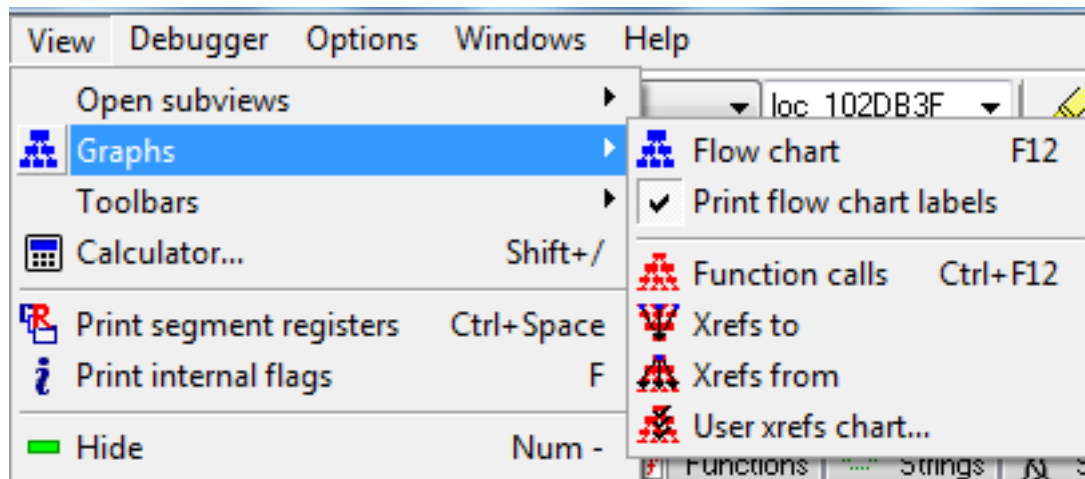
使用图形选项



图形化选项

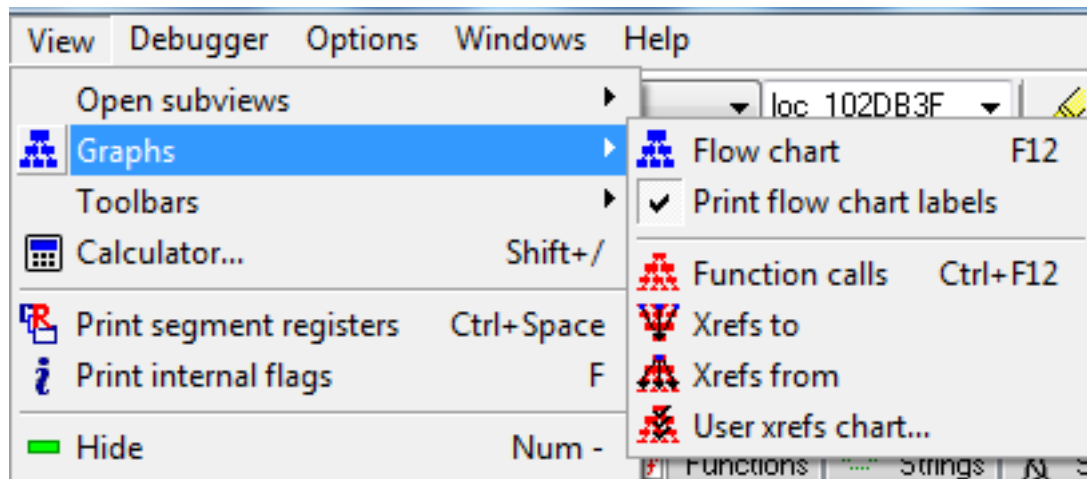


图形化选项



- 它们是遗留的图形，不能被IDA操作
- 前两个似乎过时了
 - Flow chart
 - 对当前函数创建一个流程图
 - Function calls
 - 对整个程序创建函数调用图

图形化选项

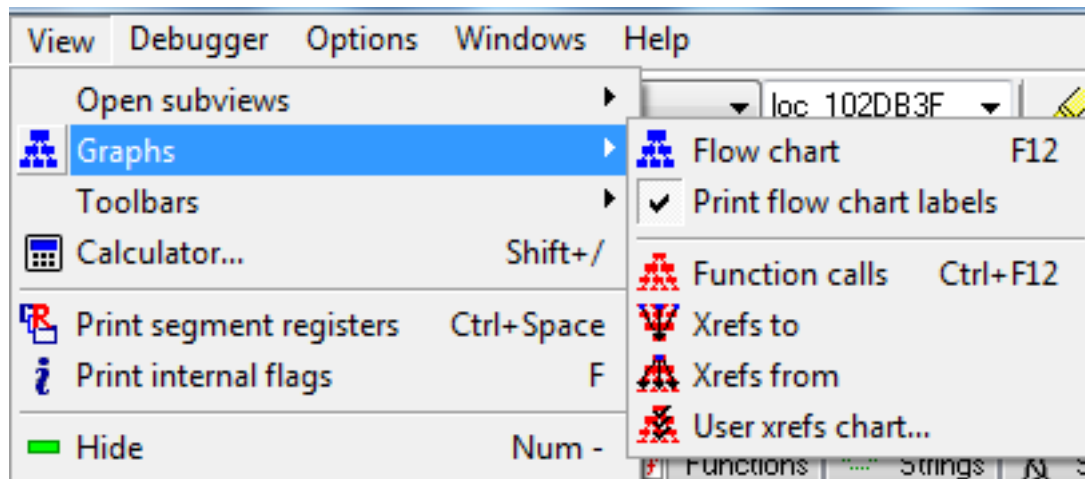


- Xrefs to
 - 对当前选择的交叉引用，生成所有指向这一引用的链接图
 - 能够显示到达某一函数的所有路径

数

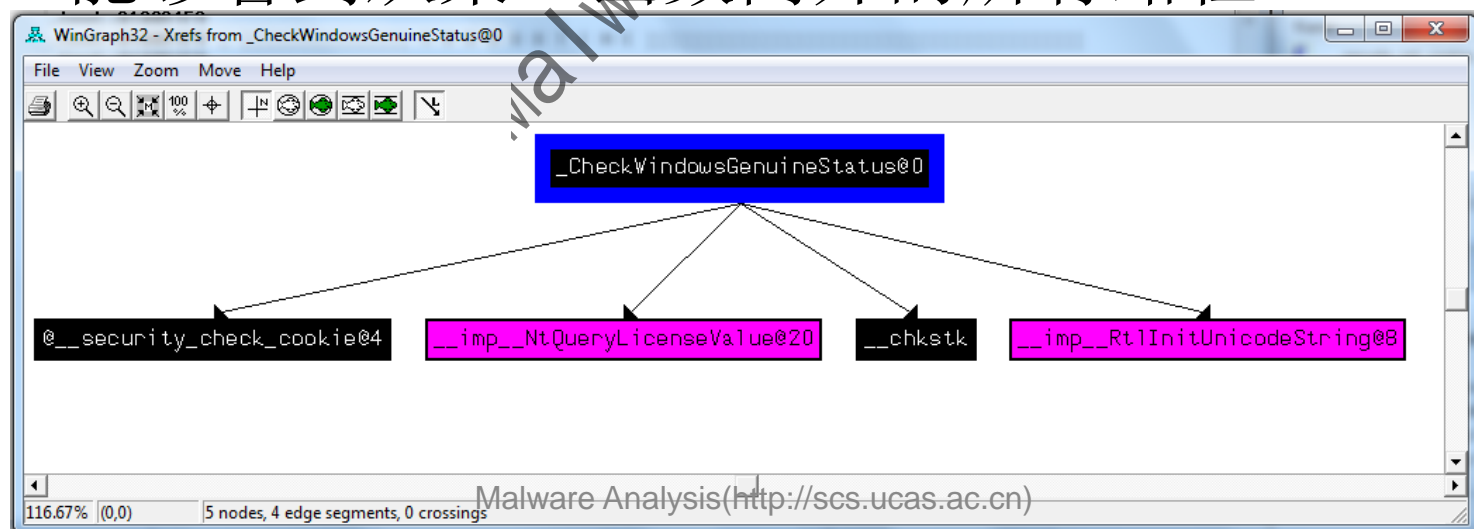


图形化选项

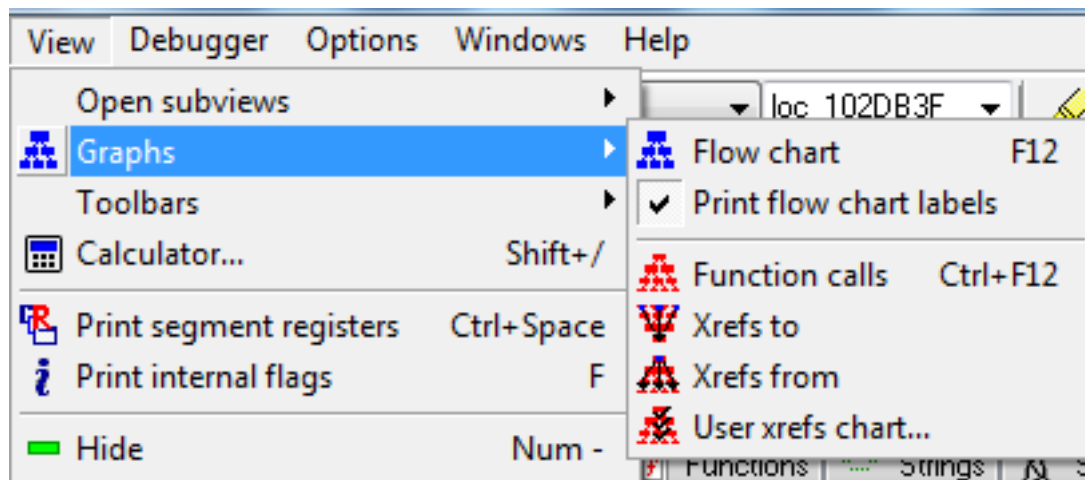


- Xrefs from

- 从当前选择的符号开始, 生成所有交叉引用的链接图
- 能够看到从某一函数离开的所有路径



图形化选项



- User xrefs chart...

- 创建一个用户指定的交叉引用图，可指定图形的递归深度、所使用的符号、去往或源自的符号等
- 修改遗留图形的唯一方法



增强反汇编

警告



- 没法撤销, 所以如果你做出修改并且搞得一团糟, 你可能会后悔

Malware Analysis



重命名位置

- 可以修改名字如：将 `sub_401000` 改为 `ReverseBackdoorThread`
- 在一个地方修改，IDA将会其他地方都修改

Table 6-2. Function Operand Manipulation

Without renamed arguments

```

004013C8 mov     eax, [ebp+arg_4]
004013CB push    eax
004013CC call   _atoi
004013D1 add     esp, 4
004013D4 mov     [ebp+var_598], ax
004013DB movzx  ecx, [ebp+var_598]
004013E2 test   ecx, ecx
004013E4 jnz     short loc_4013F8
004013E6 push    offset aError
004013EB call   printf
004013F0 add     esp, 4
004013F3 jmp     loc_4016FB
004013F8 ; -----
004013F8
004013F8 loc_4013F8:
004013F8 movzx  edx, [ebp+var_598]
004013FF push    edx
00401400 call   ds:htons

```

With renamed arguments

```

004013C8 mov     eax, [ebp+port_str]
004013CB push    eax
004013CC call   _atoi
004013D1 add     esp, 4
004013D4 mov     [ebp+port], ax
004013DB movzx  ecx, [ebp+port]
004013E2 test   ecx, ecx
004013E4 jnz     short loc_4013F8
004013E6 push    offset aError
004013EB call   printf
004013F0 add     esp, 4
004013F3 jmp     loc_4016FB
004013F8 ; -----
004013F8
004013F8 loc_4013F8:
004013F8 movzx  edx, [ebp+port]
004013FF push    edx
00401400 call   ds:htons

```



注释

- 按冒号(:) 加一条注释
- 按分号(;) 所有的交叉引用都会回显这个注释



格式化操作数

- 默认十六进制
- 右键点击使用其他格式

```
mov     edi, edi
push    ebp
mov     ebp, esp
mov     eax, 1320h
call    __chkstk
mov     eax, ___se
xor     eax, ebp
mov     [ebp+var_4], eax
push    offset aSe
```

Use standard symbolic constant

10	4896	H
8	11440o	
2	1001100100000b	B



使用命名的常量


- 使得 Windows API 参数更清晰

Before symbolic constants	After symbolic constants
<pre>mov esi, [esp+1Ch+argv] mov edx, [esi+4] mov edi, ds:CreateFileA push 0 ; hTemplateFile push 80h ; dwFlagsAndAttributes push 3 ; dwCreationDisposition push 0 ; lpSecurityAttributes push 1 ; dwShareMode</pre>	<pre>mov esi, [esp+1Ch+argv] mov edx, [esi+4] mov edi, ds:CreateFileA push NULL ; hTemplateFile push FILE_ATTRIBUTE_NORMAL ; dwFlagsAndAttributes push OPEN_EXISTING ; dwCreationDisposition push NULL ; lpSecurityAttributes push FILE_SHARE_READ ; dwShareMode</pre>



用插件扩展IDA

- 可以使用IDC (IDA的脚本语言) 和 Python 脚本

 www.openrce.org/downloads/browse/IDA_Scripts		
Decrypt Data	Unknown	IDA script to decipher data from HCU Millenium strainer stage 1 (AESFUL.EXE)
Delphi RTTI script	RedPlait	This script deals with Delphi RTTI structures
Export To Lib	Unknown	This script exports all functions to a lib file
Find Format String Vulnerabilities	Unknown	A small IDC script hacked from sprintf.idc to detect format bugs currently ...



4.3 识别汇编中的C代码结构



函数调用

```
#include "stdafx.h"
```

```
int _tmain(int argc, _TCHAR* argv[])  
{  
    printf("Hello! %d %d %d\n", 1, 2, 3);  
    return 0;  
}
```

```
C:\Windows\system32\cmd.exe  
Hello! 1 2 3  
Press any key to continue . . .
```

用IDA Pro找到代码

- IDA 只显示入口点

```
public start
start proc near
jmp     wmainCRTStartup
start endp
```

Application Type	Entry Point	Startup Function Embedded in Your Executable
GUI application that wants ANSI characters and strings	_tWinMain (WinMain)	WinMainCRTStartup
GUI application that wants Unicode characters and strings	_tWinMain (wWinMain)	wWinMainCRTStartup
CUI application that wants ANSI characters and strings	_tmain (Main)	mainCRTStartup
CUI application that wants Unicode characters and strings	_tmain (Wmain)	wmainCRTStartup



技巧：使用字符串窗口和XREF

Address	Length	Type	String
"..." .text:00...	00000F76	C
"..." .rdata:0...	00000014	C	YOURNAME-8a: %d %d\n
"..." .rdata:0...	0000001B	C	tack around the variable '
"..." .rdata:0...	00000011	C	' was corrupted.
"..." .rdata:0...	0000000E	C	he variable '

```
.rdata:00415858 ; char aYourname8aDD[]
.rdata:00415858 aYourname8aDD db 'YOURNAME-8a: %d %d',0Ah,0 ; DATA XREF: wmain+32fo
.rdata:0041586C align 10h
.rdata:00415870 a_native_start:
.rdata:00415870 unicode 0,
.rdata:004158C0 db 0
.rdata:004158C1 db 0
.rdata:004158C2 db 0
.rdata:004158C3 db 0
.rdata:004158C4 db 0
.rdata:004158C5 db 0
.rdata:004158C6 db 0
.rdata:004158C7 db 0
.rdata:004158C8 db 0
```

```
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_8], 2
mov     esi, esp
mov     eax, [ebp+var_8]
push    eax
mov     ecx, i
push    ecx
push    offset aYourname8aDD ; "YOURNAME-8a: %d %d\n"
call    ds:imp_printf
```

用IDA Pro 反汇编

- printf() 函数的4个参数
- 压入栈
- 倒序
- call 调用函数

```
wmain proc near
var_C0= dword ptr -0C0h

push    ebp
mov     ebp, esp
sub     esp, 0C0h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_C0]
mov     ecx, 30h
mov     eax, 0CCCCCCCch
rep stosd
mov     esi, esp
push    3
push    2
push    1
push    offset aHelloDDD ; "Hello! %d %d %d\n"
call    ds:__imp_printf
add     esp, 10h
cmp     esi, esp
call    j__RTC_CheckEsp
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 0C0h
cmp     ebp, esp
call    j__RTC_CheckEsp
mov     esp, ebp
pop     ebp
retn
wmain endp
```

printf("Hello! %d %d %d\n", 1, 2, 3);



全局与局部变量

- 全局变量
 - 可被程序中的任意函数访问和使用
- 局部变量
 - 在函数中定义并仅供该函数使用



全局与局部变量

```
#include "stdafx.h"

int i=1; // GLOBAL VARIABLE

int _tmain(int argc, _TCHAR* argv[])
{
    int j=2; // LOCAL VARIABLE
    printf("YOURNAME-8a: %d %d\n", i, j);
    return 0;
}
```

C:\Windows\system32\cmd.exe

YOURNAME-8a: 1 2
Press any key to continue . . .



全局与局部变量

```
mov     [ebp+var_8], 2           Local – on stack
mov     esi, esp
mov     eax, [ebp+var_8]        Local – on stack
push    eax
mov     ecx, i                  Global – in memory
push    ecx
push    offset aYourname8aDD ; "YOURNAME-8a: %d %d\n"
call    ds:imp__printf
```



算术运算

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int i=10;
    int j=2;
    int k;
    i = i + 2;
    k = i / j;
    printf("YOURNAME-9a: %d %d %d\n", i, j, k);
    return 0;
}
```

C:\Windows\system32\cmd.exe

```
YOURNAME-9a: 12 2 6
Press any key to continue . . .
```

算术运算



```
mov     [ebp+var_8], 0Ah
mov     [ebp+var_14], 2
mov     eax, [ebp+var_8]
add     eax, 2
mov     [ebp+var_8], eax
mov     eax, [ebp+var_8]
cdq
idiv    [ebp+var_14]
mov     [ebp+var_20], eax
```

```
int i=10;
int j=2;

i = i + 2;

k = i / j;
```

算术运算



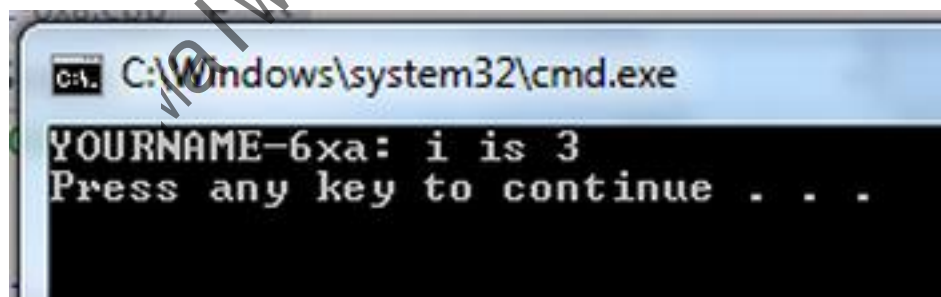
ASM Code	Explanation	C Code
mov [ebp+var_8], 0Ah	Put the number 10 into a local variable (i)	int i=10;
mov [ebp+var_14], 2	Put the number 2 into a local variable (j)	int j=2;
mov eax, [ebp+var_8]	Put i into eax	i = i + 2;
add eax, 2	Add 2 to eax	
mov [ebp+var_8], eax	Put eax (the result) into a local variable (i)	
mov eax, [ebp+var_8]	Put i into eax	k = i / j;
cdq	Convert double to quad (required for division)	
idiv [ebp+var_14]	Divide the value in eax by a local variable (j)	
mov [ebp+var_20], eax	Put eax (the result) into a local variable (k)	



识别if语句

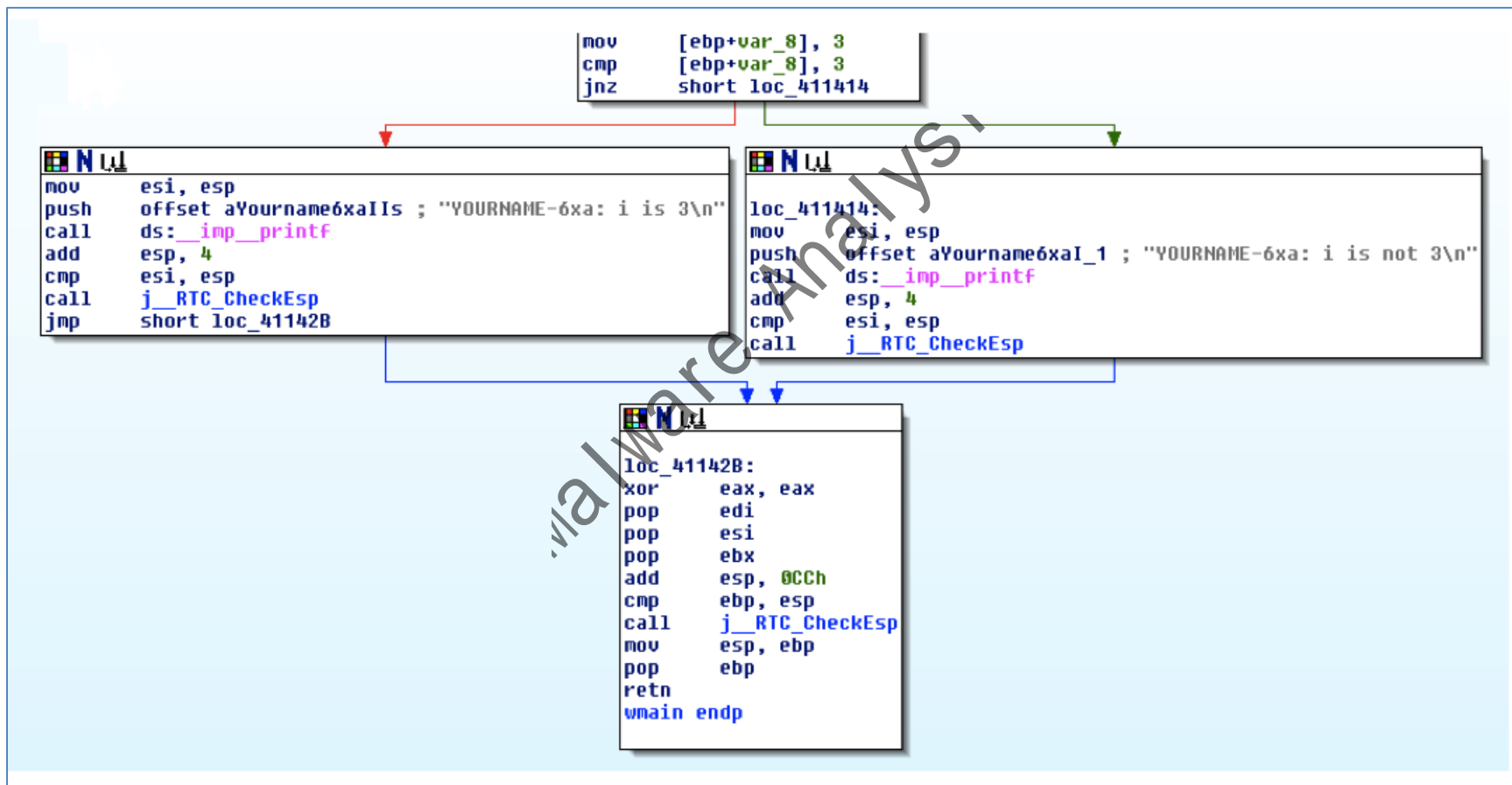
```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    int i=3;
    if (i == 3){ printf("YOURNAME-6xa: i is 3\n"); }
    else { printf("YOURNAME-6xa: i is not 3\n"); }
    return 0;
}
```





识别if语句





总结

- 查找代码
 - 先用字符串窗口然后使用交叉引用功能（XREF）
- 函数调用
 - 参数压入栈
 - 倒序
 - 调用函数
- 变量
 - 全局: 在内存中, 可被所有函数访问和使用
 - 局部: 在栈上, 仅能被该函数访问和使用

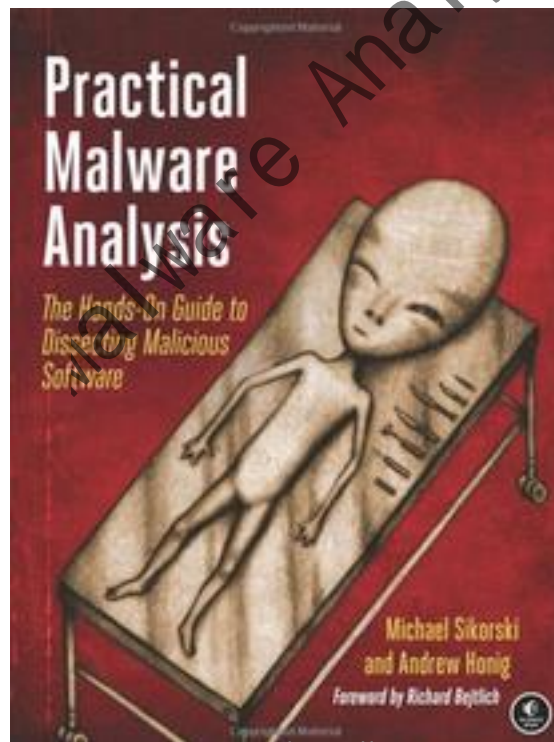


总结

- 算术操作
 - 将变量移动到寄存器
 - 执行算术运算 (**add**、**sub**、**idiv**等)
 - 将结果移动到变量
- If语句
 - 比较 (**cmp**、**test**等)
 - 条件跳转 (**jz**、**jnz**等)
 - 红箭头代表if语句的**false**，绿色箭头代表if语句的 **true**



4.4 分析恶意Windows程序





Windows API

Malware Analysis



什么是API?

- 处理程序与微软库之间的交互
- 概念
 - 类型和匈牙利表达法
 - 句柄
 - 文件系统函数
 - 特殊文件



类型和匈牙利表达法

- Windows API 使用它自己的名字，来表示C语言类型
 - 如：DWORD 和 WORD类型分别标识32位与16位无符号整数
- 匈牙利表达式
 - 包含一个32位无符号整数的变量会以dw开头



API 常见类型

- 类型和前缀
- WORD (w) 16位 无符号数值
- DWORD (dw) 32位 无符号数值
- Handle (H) 一个对象索引
- Long Pointer (LP) 指向另一类型的指针



句柄

- 操作系统中被打开或创建的项，如：
 - 窗口、进程、模块、菜单、文件等
- 句柄像指向这些对象的指针
 - 然而它们和指针不同
- 你能够对句柄做的唯一的事情，就是保存它并在后续函数调用中使用它来引用同一个对象



句柄事例

- CreateWindowEx 函数返回一个HWND，这是一个窗口的句柄
- 想对那个窗口做点什么，如调用 DestroyWindow函数时，需要使用这个句柄



文件系统函数

- CreateFile、 ReadFile、 WriteFile
 - 标准的文件输入/输出
- CreateFileMapping、 MapViewOfFile
 - 被恶意代码用来将文件加载到内存
 - 能够在不使用Windows加载器的情况下，用来执行文件



特殊文件

- 共享文件如 `\\server\share`
 - 或者 `\\?\server\share`
 - 禁用字符串解析，允许更长的文件名
- 名字空间
 - Windows 文件系统中的特殊文件夹
 - `\` 最低的名字空间，包含一切
 - `\\.\` 设备名字空间用于对磁盘直接的输入输出
 - Witty蠕虫向 `\\.\PhysicalDisk1` 写入数据，来破坏磁盘文件



特殊文件

- 备用数据流
 - 流数据被附加在一个文件中
 - File.txt:otherfile.txt

The screenshot shows a Windows Administrator Command Prompt window and a Notepad window. The Command Prompt shows the following commands and output:

```
C:\Users\sam\ads>echo 1 > foo
C:\Users\sam\ads>dir foo
Volume in drive C is Win7
Volume Serial Number is 80F8-F717

Directory of C:\Users\sam\ads
09/23/2013  05:31 PM                4 foo
               1 File(s)                4 bytes
               0 Dir(s)  78,679,588,864 bytes free

C:\Users\sam\ads>echo 22222222222222222222222222222222 > foo:bar.txt
C:\Users\sam\ads>dir foo
Volume in drive C is Win7
Volume Serial Number is 80F8-F717

Directory of C:\Users\sam\ads
09/23/2013  05:31 PM                4 foo
               1 File(s)                4 bytes
               0 Dir(s)  78,679,588,864 bytes free

C:\Users\sam\ads>notepad foo:bar.txt
C:\Users\sam\ads>
```

The Notepad window, titled "foo:bar.txt - Notepad", shows the text "22222222222222222222222222222222" entered into the text area.



Windows 注册表

Malware Analysis

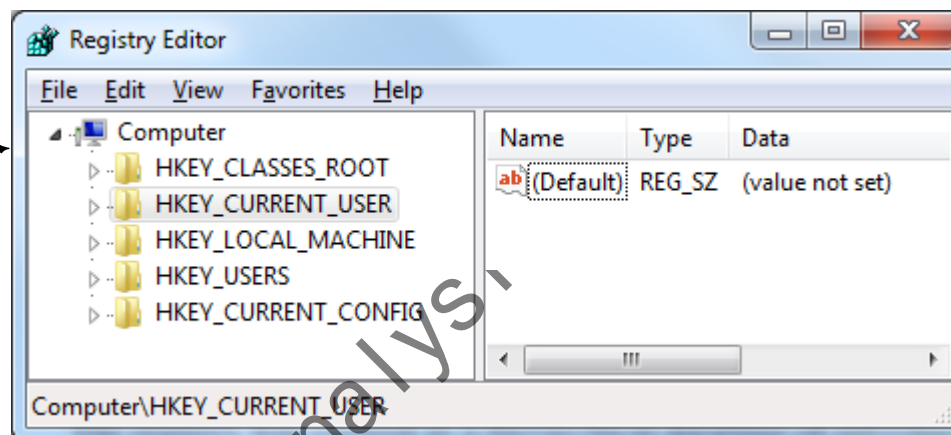


注册表用途

- 存储操作系统和程序配置信息
 - 桌面背景、鼠标参数等
- 恶意代码使用注册表来完成持久驻留
 - 当系统重启时重启恶意代码

注册表项

- 根键 这5个



- 子键 像文件夹中的子文件夹
- 键 包含文件夹或键值
- 值项 包含两部分：名字和值
- 值或数据 存储在注册表项中的数据
- REGEDIT 查看或编辑注册表的工具



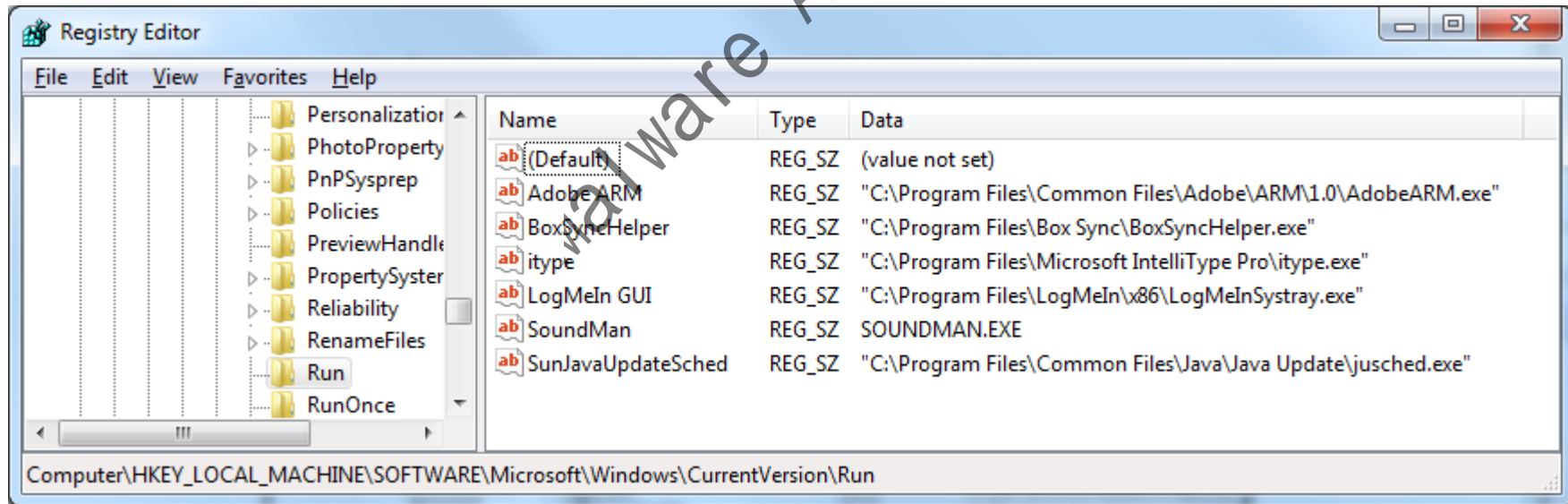
根键

- 注册表根键，注册表被划分为下面5个根键
 - HKEY_LOCAL_MACHINE (HKLM)
 - 保存对本地机器全局设置
 - HKEY_CURRENT_USER (HKCU)
 - 保存当前用户特定设置
 - HKEY_CLASSES_ROOT
 - 保存定义的类型信息
 - HKEY_CURRENT_CONFIG
 - 保存当前硬件配置的设置，特别是前面和标准配置之间不同的部分
 - HKEY_USERS
 - 定义默认用户、新用户和当前用户的配置



Run 子键

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
 - 当用户登录时自动启动的可执行程序





Autoruns

- Sysinternals系列工具之一
- 列举在操作系统启动时会自动运行的代码
 - 可执行文件
 - 加载到IE或其他程序中的DLL
 - 加载到内核中的驱动
 - 它会检查25-30个注册表中的位置
 - 不一定会找到所有自动运行代码



Autoruns

Autoruns [sam-c216\sam] - Sysinternals: www.sysinternals.com

File Entry Options User Help

Codecs Boot Execute Image Hijacks AppInit KnownDLLs Winlogon Winsock Providers
Print Monitors LSA Providers Network Providers Sidebar Gadgets
Everything Logon Explorer Internet Explorer Scheduled Tasks Services Drivers

Autorun Entry	Description	Publisher	Image Path	Timestamp
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run				6/10/2013 10:28 AM
<input checked="" type="checkbox"/> Adobe ARM	Adobe Reader and Acrobat...	Adobe Systems Incorporated	c:\program files\common fil...	4/4/2013 2:05 PM
<input checked="" type="checkbox"/> BoxSyncHelper	Box Sync Helper Process	Box, Inc.	c:\program files\box sync\b...	6/7/2013 9:19 PM
<input checked="" type="checkbox"/> itype	IType.exe	Microsoft Corporation	c:\program files\microsoft in...	5/20/2009 7:36 PM
<input checked="" type="checkbox"/> LogMeIn GUI	LogMeIn Desktop Application	LogMeIn, Inc.	c:\program files\logmein\x8...	4/12/2007 10:44 AM
<input checked="" type="checkbox"/> SoundMan	Realtek Sound Manager	Realtek Semiconductor Corp.	c:\windows\soundman.exe	3/8/2009 9:29 PM
<input checked="" type="checkbox"/> SunJavaUpdat...	Java(TM) Update Scheduler	Oracle Corporation	c:\program files\common fil...	3/12/2013 8:32 AM
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup				8/12/2013 4:03 PM
<input checked="" type="checkbox"/> Box Sync.Ink	Box Sync	Box, Inc.	c:\program files\box sync\b...	6/7/2013 9:19 PM
C:\Users\sam\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup				9/12/2013 8:07 AM
<input checked="" type="checkbox"/> Dropbox.Ink	Dropbox	Dropbox, Inc.	c:\users\sam\appdata\roa...	4/5/2013 1:44 PM
HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components				9/14/2009 6:01 PM
<input checked="" type="checkbox"/> Microsoft Wind...	Windows Mail	Microsoft Corporation	c:\program files\windows m...	7/13/2009 4:42 PM
HKCU\Software\Microsoft\Windows\CurrentVersion\Run				1/13/2012 11:02 AM
<input checked="" type="checkbox"/> Google Update	Google Installer	Google Inc.	c:\users\sam\appdata\loca...	8/22/2008 12:35 PM
<input checked="" type="checkbox"/> SkyDrive	Microsoft SkyDrive	Microsoft Corporation	c:\users\sam\appdata\loca...	8/11/2013 5:55 PM
HKLM\SOFTWARE\Classes\Protocols\Filter				7/13/2009 9:41 PM
<input checked="" type="checkbox"/> text/xml	Microsoft Office XML MIME...	Microsoft Corporation	c:\program files\common fil...	7/12/2003 3:19 AM
HKLM\SOFTWARE\Classes\Protocols\Handler				7/13/2009 9:41 PM
<input checked="" type="checkbox"/> mso-offdap	Microsoft Office XP Web C...	Microsoft Corporation	c:\program files\common fil...	8/4/2003 12:27 PM
<input checked="" type="checkbox"/> mso-offdap11	Microsoft Office Web Comp...	Microsoft Corporation	c:\program files\common fil...	8/1/2003 3:01 PM
HKCU\Software\Classes*\ShellEx\ContextMenuHandlers				9/14/2009 10:21 PM
<input checked="" type="checkbox"/> SkyDriveEx	Microsoft SkyDrive Shell Ex...	Microsoft Corporation	c:\users\sam\appdata\loca...	8/11/2013 5:55 PM

(Escape to cancel) Scanning... Malware Analysis(<http://scs.ucas.ac.cn/>) Windows Entries Hidden.



常用注册表函数

- RegOpenKeyEx
 - 打开一个注册表键, 用于编辑或查询
- RegSetValueEx
 - 添加一个新值到注册表, 并设置它的数值
- RegGetValue
 - 返回注册表中的一个值项的数值
- 注意: 文档在函数调用中将省略后面的 W (宽字符) 或者 A (ASCII) 字符, 如:
RegOpenKeyExW



Ex、A和W 后缀

- 函数命名规则

- 当遇到不熟悉的Windows函数时，一些函数命名规则值得注意；因为它们经常出现，在没有认出它们时还可能把你弄糊涂。举个例子，你经常会碰到函数名字包含后缀Ex，如CreateWindowsEx。当微软更新函数时，新函数与旧函数不兼容，微软继续支持旧函数。新函数名字为旧函数名字加上后缀Ex。名字以两个Ex结尾函数明显是更新了两次。
- 许多名字以A或者W结尾的函数把字符串作为参数，如CreateDirectoryW。最后这个字符在函数的文档中并不出现，它表示这个函数接受字符串参数并且有两个不同的版本：ASCII字符版或宽字符版。记住在微软文档中搜索函数时丢弃末尾的A或者W字符。



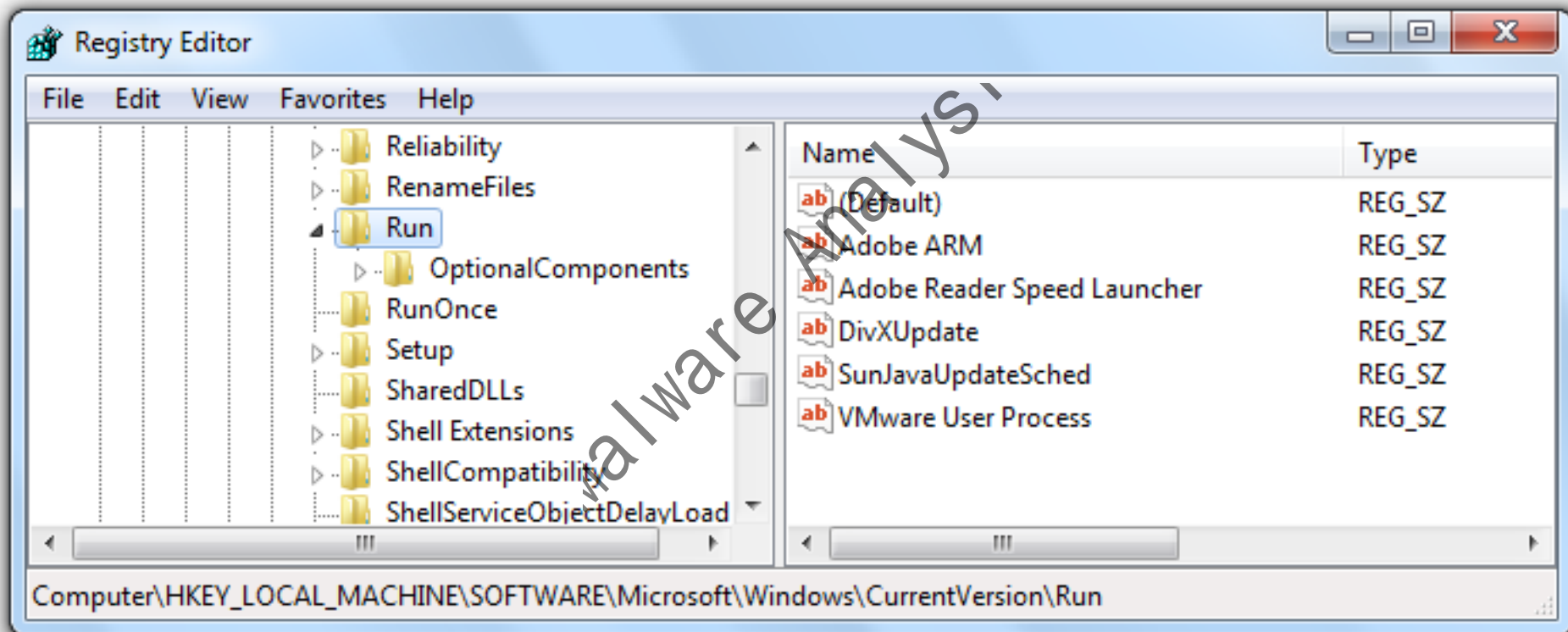
注册表操作代码

Example 8-1. Code that modifies registry settings

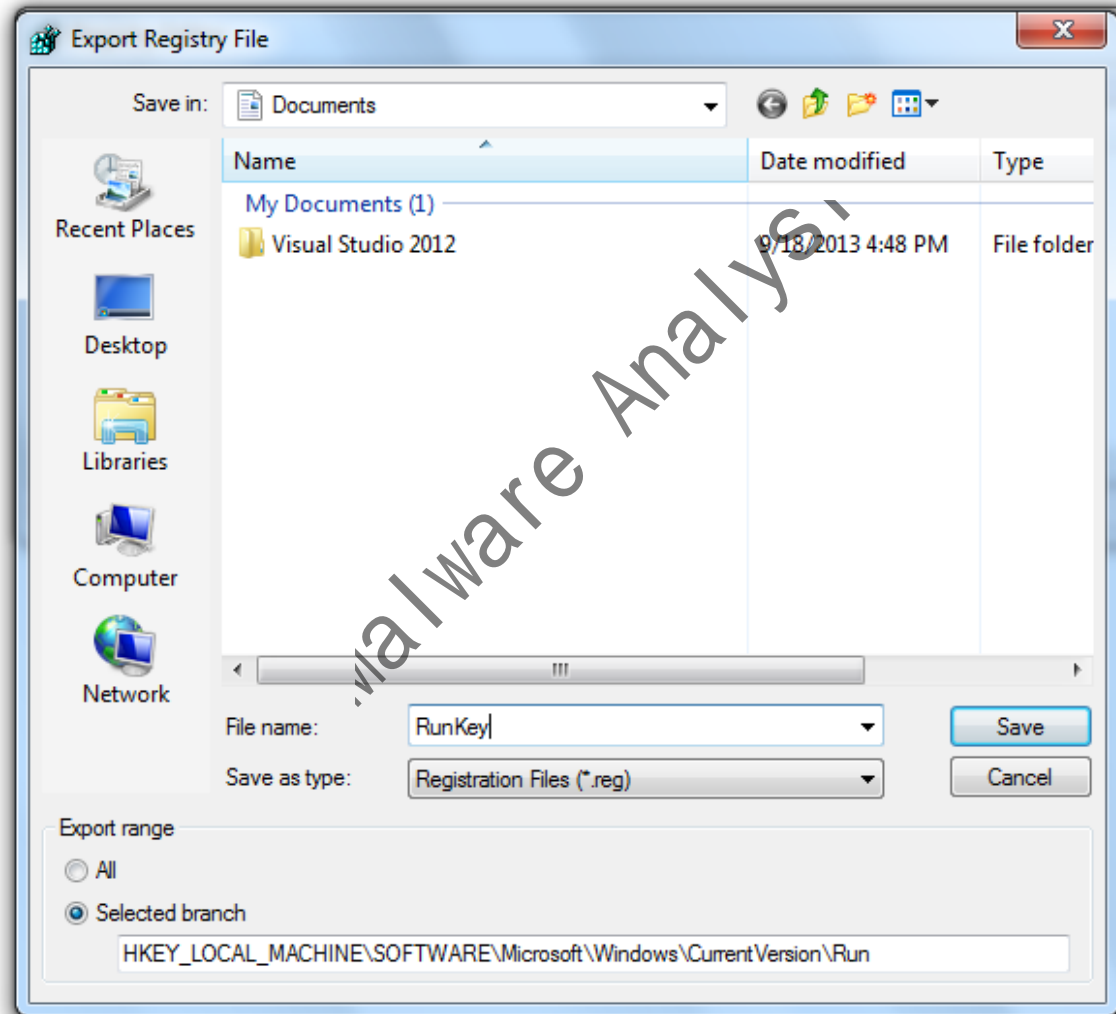
```
0040286F    push     2                      ; samDesired
00402871    push     eax                    ; ulOptions
00402872    push     offset SubKey         ;
"Software\\Microsoft\\Windows\\CurrentVersion\\Run"
00402877    push     HKEY_LOCAL_MACHINE    ; hKey
```




. REG 文件



. REG 文件



. REG 文件



```
RunKey.reg - Notepad
File Edit Format View Help
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"VMware User Process"="\"C:\\Program Files\\VMware\\VMware Tools\\vmtoolsd.exe\" -n
vmusr"
"SunJavaUpdateSched"="\"C:\\Program Files\\Common Files\\Java\\Java Update\\
\\jusched.exe\"""
"DivXUpdate"="\"C:\\Program Files\\DivX\\DivX Update\\DivXUpdate.exe\" /CHECKNOW"
"Adobe Reader Speed Launcher"="\"C:\\Program Files\\Adobe\\Reader 9.0\\Reader\\
\\Reader_sl.exe\"""
"Adobe ARM"="\"C:\\Program Files\\Common Files\\Adobe\\ARM\\1.0\\AdobeARM.exe\"""

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents]
@=""

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents
\IMAIL]
@=""
"Installed"="1"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents
\MAPI]
@=""
"Installed"="1"
"NoChange"="1"

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\OptionalComponents
\MSFS]
@=""
"Installed"="1"
```



网络API



伯克利兼容套接字

- Winsock 库， 主要在ws2_32.dll中
 - 它们的功能在Windows和Unix中几乎完全相同

Malware Analysis



Function Description

socket	Creates a socket
bind	Attaches a socket to a particular port, prior to the accept call
listen	Indicates that a socket will be listening for incoming connections
accept	Opens a connection to a remote socket and accepts the connection
connect	Opens a connection to a remote socket; the remote socket must be waiting for the connection
recv	Receives data from the remote socket
send	Sends data to the remote socket

注意

WSAStartup函数必须要在其他网络函数之前被调用，以便为这些网络库分配资源。当在调试代码查找网络连接入口时，在WSAStartup函数上设置一个断点是非常有用的，因为网络入口应该在后面不远的地方。



网络的服务器和客户端

- 服务器端
 - 维护一个开放的套接字，等待连接
 - 调用顺序为： `socket`、`bind`、`listen`、`accept`
 - 如果有需要的话后面跟着 `send` 和 `recv`
- 客户端
 - 连接到正在等待的套接字
 - 调用顺序为 `socket`、`connect`
 - 如果有需要的话后面跟着 `send` 和 `recv`

简化的服务器端程序样例



- 实际代码会多次调用
WSAGetLastError
函数

```
00401041  push    ecx                ; lpWSAData
00401042  push    202h              ; wVersionRequested
00401047  mov     word ptr [esp+250h+name.sa_data], ax
0040104C  call    ds:WSAStartup
00401052  push    0                 ; protocol
00401054  push    1                 ; type
00401056  push    2                 ; af
00401058  call    ds:socket
0040105E  push    10h              ; namelen
00401060  lea     edx, [esp+24Ch+name]
00401064  mov     ebx, eax
00401066  push    edx               ; name
00401067  push    ebx               ; s
00401068  call    ds:bind
0040106E  mov     esi, ds:listen
00401074  push    5                 ; backlog
00401076  push    ebx               ; s
00401077  call    esi ; listen
00401079  lea     eax, [esp+248h+addrlen]
0040107D  push    eax               ; addrlen
0040107E  lea     ecx, [esp+24Ch+hostshort]
00401082  push    ecx               ; addr
00401083  push    ebx               ; s
00401084  call    ds:accept
```




WinINet API

- 比Winsock更高级的API
- 函数在Wininet.dll中
- 实现了应用层协议HTTP和FTP
- InternetOpen - 连接到互联网
- InternetOpenURL - 连接到URL
- InternetReadFile - 从下载的文件中读取数据



跟踪恶意代码的运行



转移执行

- 使用跳转（`jmp`）和指令调用（`call`）转移到代码的其他部分执行，但还有其他方式：
 - 动态链接库DLL
 - 进程
 - 线程
 - 互斥量
 - 服务
 - 组件对象模型（COM）
 - 异常



DLL

- 多个应用程序之间共享代码
- 动态链接库导出代码能够被其他应用使用
- 静态库在动态链接库出现前就被使用
 - 还在使用，但不太常见
 - 它们不能在正运行的进程之间共享内存
 - 静态链接库比动态链接库更占内存



DLL 优点

- 使用Windows系统已有的DLL使代码更小
- 软件公司也可以自定义DLL
 - DLL文件同EXE文件一起发布

恶意代码作者如何使用DLL



- 将恶意代码存在DLL中
 - 有时将恶意的DLL加载到另外的进程
- 使用Windows DLL
 - 几乎所有的恶意代码使用Windows 基础的DLL
- 使用第三方DLL
 - 使用Firefox DLL 代替Windows API连接服务器



基本DLL架构

- DLL与EXE非常类似
- PE 文件格式
- 一个标志表明这是一个DLL，而不是一个EXE
- DLL有较多的导出函数和较少的导入函数
- DllMain是主函数不是导出函数，但在PE头中被指定为入口点
 - 当函数加载或卸载库时被调用

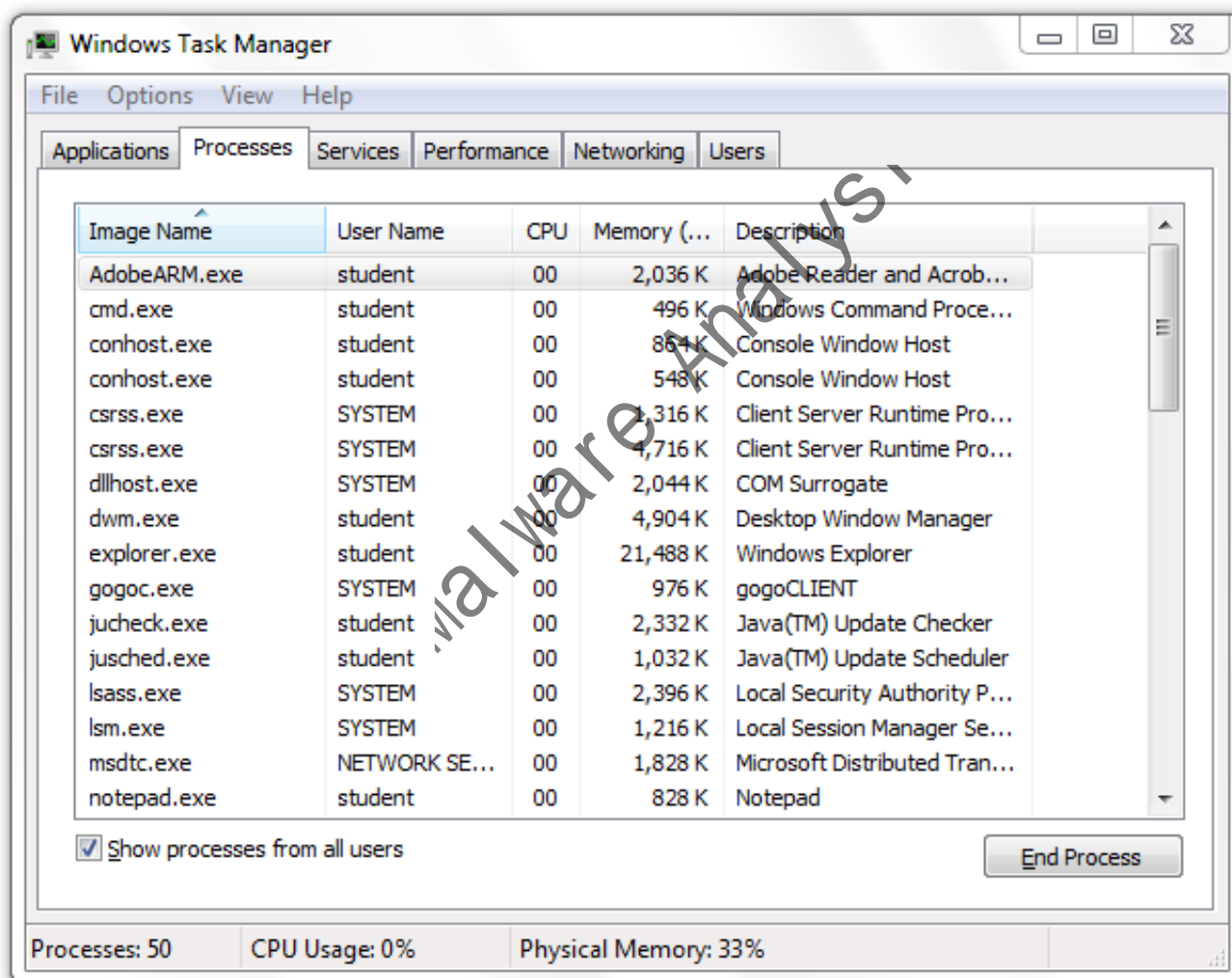
进程



- 每一个被Windows系统执行的程序就是一个进程
- 每个进程都有自己的资源
 - 句柄、内存等
- 每个进程有一个或者多个线程
- 老的恶意代码作为一个独立的进程运行
- 新的恶意代码将其代码作为其他进程的一部分执行



许多进程同时运行





内存管理

- 每个进程使用资源如：CPU、文件系统和内存等
- 操作系统为每个进程分配内存
- 两个进程访问同一个内存地址, 而实际上访问的是内存中的不同位置



创建一个新进程

- CreateProcess
 - 用一个函数调用可以创建一个简单的远程shell
 - STARTUPINFO 参数包含标准输入、标准输出以及标准错误流的句柄
 - 可以设置为一个套接字，创建一个远程shell



创建Shell的代码

- 将sockethandle、StdError、StdOutput和StdInput 存入lpProcessInformation中

Example 8-4. Sample code using the CreateProcess call

```
004010DA  mov     eax, dword ptr [esp+58h+SocketHandle]
004010DE  lea     edx, [esp+58h+StartupInfo]
004010E2  push    ecx                ; lpProcessInformation
004010E3  push    edx                ; lpStartupInfo
004010E4  1mov    [esp+60h+StartupInfo.hStdError], eax
004010E8  2mov    [esp+60h+StartupInfo.hStdOutput], eax
004010EC  3mov    [esp+60h+StartupInfo.hStdInput], eax
004010F0  4mov    eax, dword_403098
004010F5  push    0                  ; lpCurrentDirectory
004010F7  push    0                  ; lpEnvironment
004010F9  push    0                  ; dwCreationFlags
004010FB  mov     dword ptr [esp+6Ch+CommandLine], eax
```



- CommandLine 包含了命令行
- 在CreateProcess被调用时执行

```
004010FF  push    1                ; bInheritHandles
00401101  push    0                ; lpThreadAttributes
00401103  lea     eax, [esp+74h+CommandLine]
00401107  push    0                ; lpProcessAttributes
00401109  5push   eax              ; lpCommandLine
0040110A  push    0                ; lpApplicationName
0040110C  mov     [esp+80h+StartupInfo.dwFlags], 101h
00401114  6call   ds:CreateProcessA
```



线程

- 进程是容器
 - 每一个进程包含一个或多个线程
- 线程是Windows实际执行的内容
- 线程
 - 独立的指令序列
 - 被CPU执行，不用等待其他线程
 - 一个进程中的多个线程共享相同的内存空间
 - 每个线程都有自己的寄存器和堆栈



线程上下文

- 当一个线程运行时，它对CPU有完全的控制权
- 其他线程不能影响CPU的状态
- 当一个线程改变寄存器的值时，它不会影响任何其他线程
- 当操作系统切换到另一个线程时，所有CPU值被保存到一个结构体中，该结构体称为线程上下文



创建一个线程

- CreateThread
 - 调用者指定起始地址，也称为start函数



恶意代码如何使用线程

- 使用CreateThread加载恶意DLL到进程
- 创建两个线程，用于输入和输出
 - 用于与正在运行的应用程序通信

使用互斥量的进程间协作



- 互斥量 (Mutex) 是全局对象, 用于协调多个进程和线程
- 内核中称为互斥门 (mutant)
- 互斥量经常使用硬编码的名字, 可以用来识别恶意代码



互斥量函数

- WaitForSingleObject
 - 获取对互斥量的访问
 - 任何后续线程试图获取对它的访问时都必须等待
- ReleaseMutex
 - 当一个线程完成对互斥量的使用后，调用它
- CreateMutex
- OpenMutex
 - 获取另一个进程中互斥量的句柄



确保只有一份恶意代码实例在运行

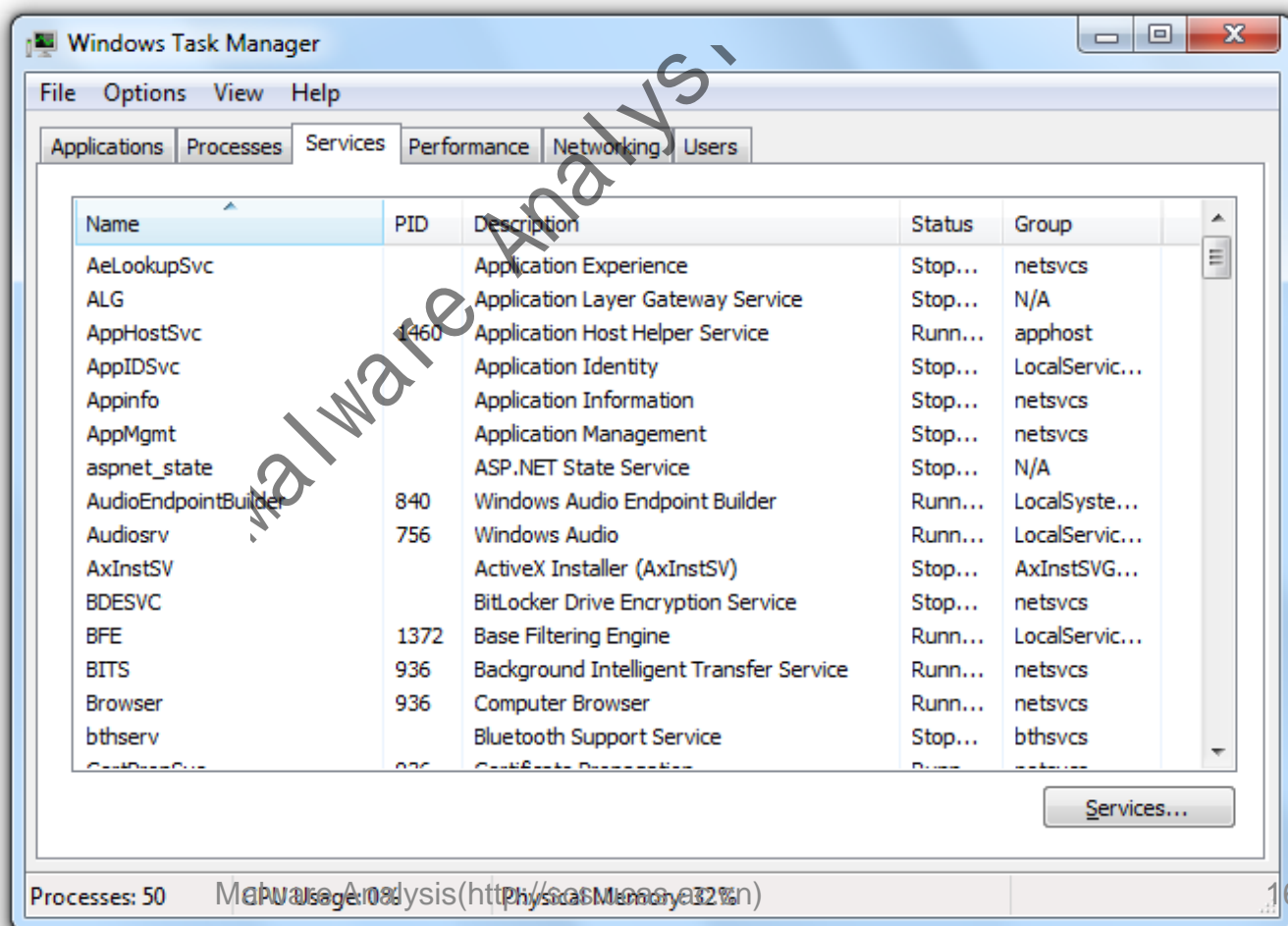
- OpenMutex 检测互斥量HGL345是否存在
- 如果不存在使用CreateMutex创建该互斥量
- test eax, eax
若eax为0则 Z 标志置位

```
00401007  push  1F0001h          ; dwDesiredAccess
0040100C  1call  ds:__imp__OpenMutexW@12 ;
OpenMutexW(x,x,x)
00401012  2test  eax, eax
00401014  3jz    short loc_40101E
00401016  push  0                ; int
00401018  4call  ds:__imp__exit
0040101E  push  offset Name      ; "HGL345"
00401023  push  0                ; bInitialOwner
00401025  push  0                ; lpMutexAttributes
00401027  5call  ds:__imp__CreateMutexW@12 ;
CreateMutexW(x,x,x)
```



服务

- 服务运行在后台，不需要用户输入





SYSTEM 账户

- 服务经常以比Administrator更强大的SYSTEM权限运行
- 服务能够在Windows系统启动时自动运行
 - 恶意代码维护持久化驻留的一种简单方法
 - 持久化驻留恶意代码，系统重启后还能存活



服务相关的 API 函数

- OpenSCManager
 - 返回一个服务控制管理器的句柄
- CreateService
 - 添加一个新的服务到服务控制管理器
 - 可以指定服务是否会在引导时自动启动
- StartService
 - 仅在服务设置为手动启动时使用



Svchost.exe

- WIN32_SHARE_PROCESS
 - 恶意代码使用的最常见的服务类型
 - 服务代码保存在DLL中
 - 组合多个服务到一个共享的名为svchost.exe的进程

在Process Explorer中查看 Svchost.exe



Process Explorer - Sysinternals: www.sysinternals.com [W7\student]

File Options View Process Find DLL Users Help

Process	PID	CPU	Private Bytes	Working Set	Description
System Idle Process	0	97.61	0 K	24 K	
System	4	0.15	44 K	672 K	
Interrupts	n/a	0.42	0 K	0 K	Hardware Interrupts
smss.exe	260		224 K	792 K	Windows Session Manager
csrss.exe	352		2,472 K	4,160 K	Client Server Runtime
wininit.exe	404		892 K	3,360 K	Windows Start
services.exe	508		4,312 K	6,512 K	Services and Control
svchost.exe	640		2,904 K	7,208 K	Host Process for
WmiPrvSE.exe	3736		1,768 K	4,752 K	WMI Provider
svchost.exe	708		3,196 K	6,716 K	Host Process for
svchost.exe	756		14,268 K	14,420 K	Host Process for
audiodg.exe	1680		15,016 K	14,024 K	Windows Audio
svchost.exe	840	< 0.01	44,436 K	50,672 K	Host Process for
dwm.exe	2848	0.20	88,212 K	34,328 K	Desktop Window
svchost.exe					
svchost.exe					
svchost.exe					
spoolsv.exe					
svchost.exe					
svchost.exe					
gogoc.exe					
sqlwriter.exe					

Command Line:
C:\Windows\System32\svchost.exe -k LocalSystemNetworkRestricted

Path:
C:\Windows\System32\svchost.exe (LocalSystemNetworkRestricted)

Services:
Desktop Window Manager Session Manager [UxSms]
Distributed Link Tracking Client [TrkWks]
Network Connections [Netman]
Offline Files [CscService]
Program Compatibility Assistant Service [PcaSvc]
Remote Desktop Services UserMode Port Redirector [UmRdpService]
Superfetch [SysMain]
Windows Audio Endpoint Builder [AudioEndpointBuilder]
Windows Driver Foundation - User-mode Driver Framework [wudfsvc]



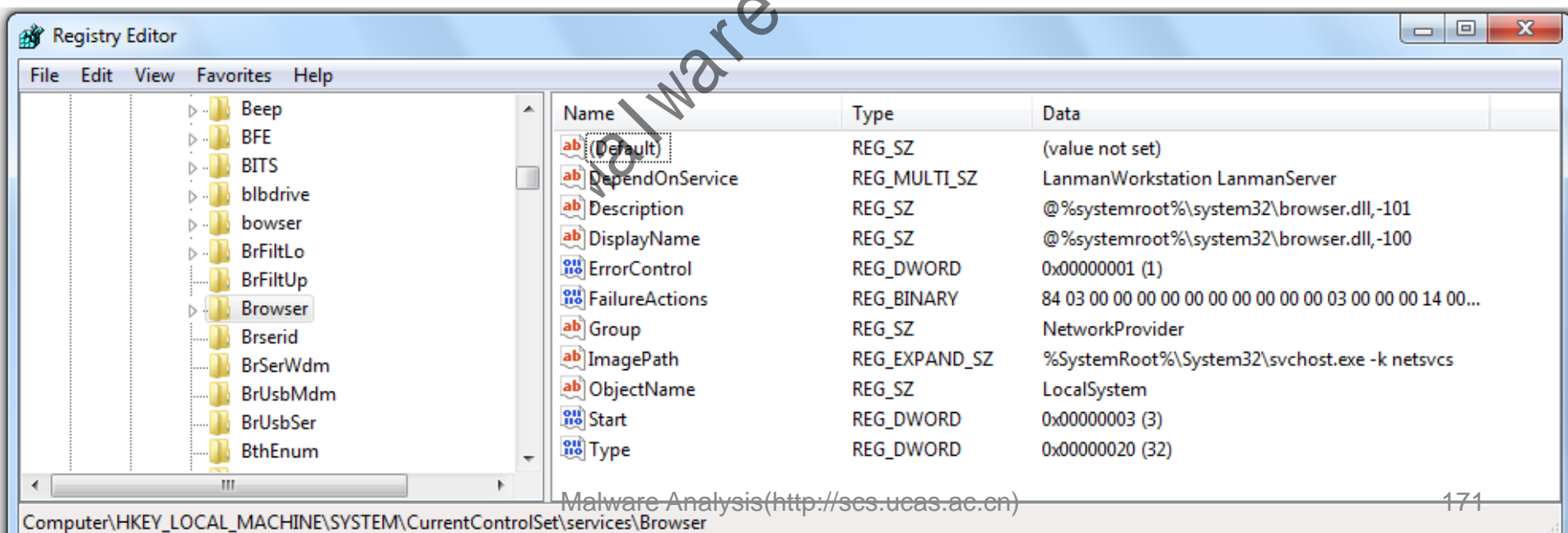
其他常见的服务类型

- WIN32_OWN_PROCESS
 - EXE文件，作为一个独立的进程运行
- KERNEL_DRIVER
 - 用于加载代码到内核



注册表中的服务信息

- HKLM\System\CurrentControlSet\Services
 - Start 值= 0x03
 - Type = 0x20 for WIN32_SHARE_PROCESS





SC 命令

- Windows包含的命令行工具
- 显示服务的信息

```
C:\Windows\System32>sc qc Browser
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: Browser
        TYPE               : 20    WIN32_SHARE_PROCESS
        START_TYPE           : 3     DEMAND_START
        ERROR_CONTROL        : 1     NORMAL
        BINARY_PATH_NAME     : C:\Windows\System32\svchost.exe -k netsvcs
        LOAD_ORDER_GROUP     : NetworkProvider
        TAG                  : 0
        DISPLAY_NAME         : Computer Browser
        DEPENDENCIES         : LanmanWorkstation
                           : LanmanServer
        SERVICE_START_NAME   : LocalSystem

C:\Windows\System32>
```



组件对象模型 (COM)

- 允许不同的软件组件之间共享代码
- 每个使用COM的线程在调用其他COM库前必须调用OleInitialize 或者 CoInitializeEx

GUIDs、 CLSIDs、 IIDs



- COM 对象通过全局唯一标识符 (GUIDs) 进行访问
- 有多种类型的GUID, 包括:
 - 类型标识符 (CLSIDs)
 - 在注册表的HKEY_CLASSES_ROOT\CLSID中
 - 接口标识符 (IIDs)
 - 在注册表的HKEY_CLASSES_ROOT\Interface中

异常



- 异常是由错误引起的，如被0除或无效的内存访问
- 当异常发生时，执行转移到结构化异常处理程序



保存异常处理信息到fs:0

- FS 是六个段寄存器之一

Example 8-13. Storing exception-handling information in fs:0

```
01006170  push  1offset loc 10061C0
01006175  mov    eax, large fs:0
0100617B  push  2eax
0100617C  mov    large fs:0, esp
```

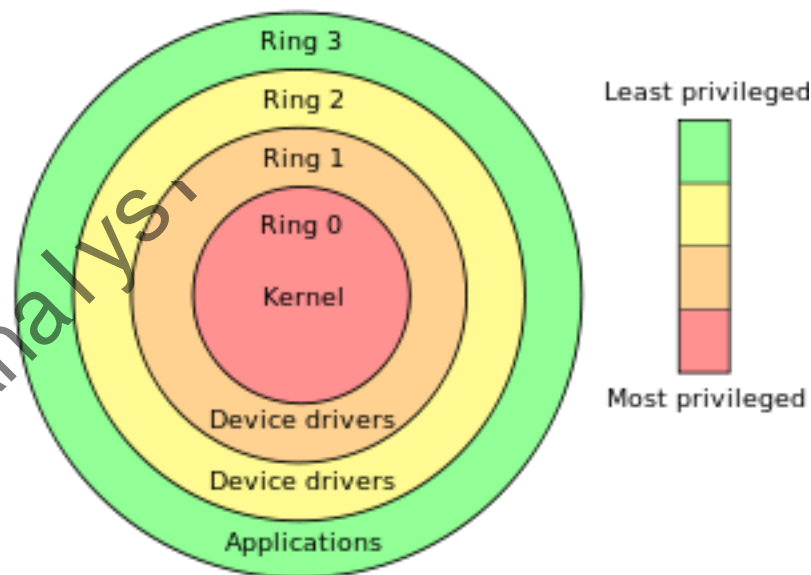



内核与用户模式



两个特权级别

- Ring 0: 内核模式
- Ring 3: 用户模式
- Rings 1和2
 - Windows中没有使用





用户模式

- 几乎所有的代码在用户模式下运行
 - 除了操作系统和硬件驱动运行在内核模式
- 用户模式不能直接访问硬件
- 限于CPU指令的一个子集
- 只能通过Windows API操纵硬件吗？



用户模式进程

- 每个进程都有其自己的内存、安全权限和资源
- 如果一个用户模式程序执行一个无效的指令并崩溃，Windows可以回收其资源并终止该程序



调用内核

- 不可能直接从用户模式跳到内核模式
- SYSENTER、SYSCALL或者INT 0x2E指令使用查找表找到预定义的函数



内核进程

- 所有内核进程共享资源和内存地址
- 更少的安全检查
- 如果内核代码执行无效指令，操作系统崩溃蓝屏死机
- 反病毒软件和防火墙运行在内核模式



内核模式的恶意代码

- 比用户模式恶意代码更强大
- 审计并不适用于内核
- 几乎所有的rootkits使用内核模式
- 大多数恶意代码不使用内核模式



原生API

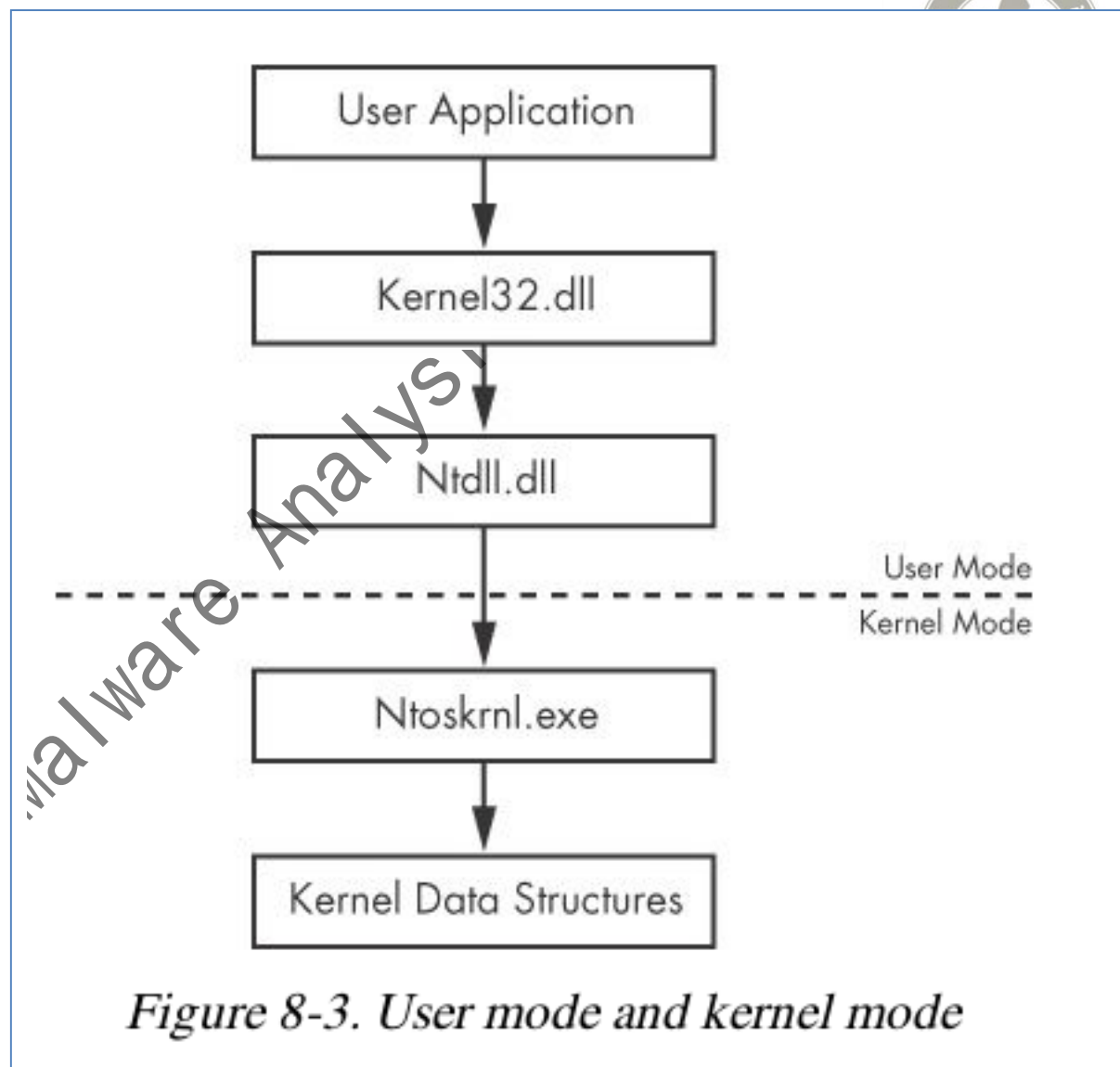


原生API

- 与Windows系统交互的低层接口
- 很少被非恶意程序使用
- 受恶意代码作者欢迎

Malware Analysis

- Ntdll.dll 管理用户空间和内核之间的交互
- Ntdll 函数组成原生 API





原生API

- 没有官方文档
- 用于Windows内部使用
- 能够被程序使用
- 原生API调用比Windows API调用更强大、更隐秘

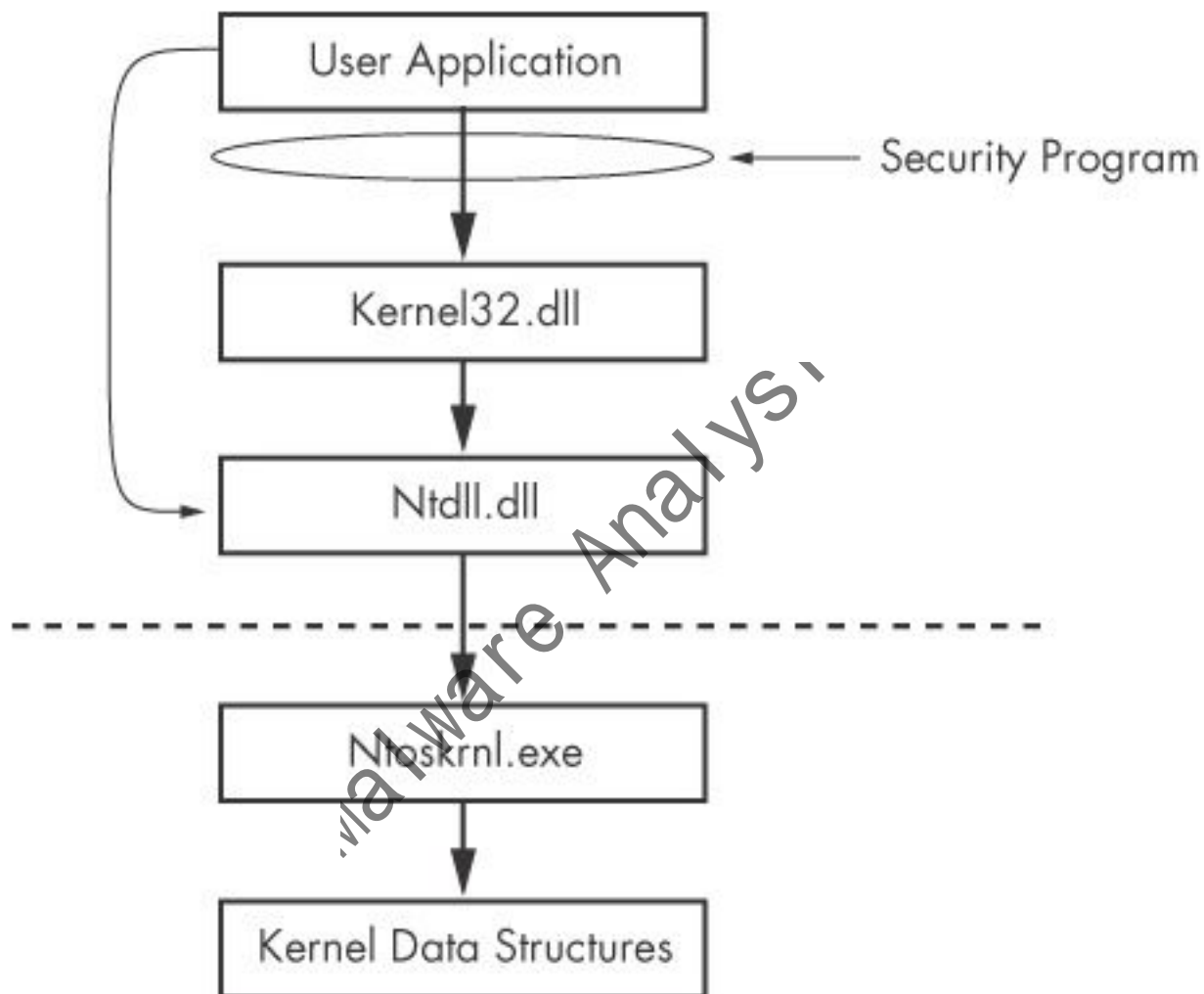
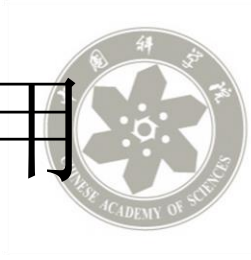
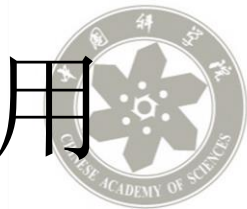


Figure 8-4. Using the Native API to avoid detection



恶意代码中流行的原生API调用

- NtQuerySystemInformation
- NtQueryInformationProcess
- NtQueryInformationThread
- NtQueryInformationFile
- NtQueryInformationKey
 - 比任何可用的Win32调用提供更多的信息



恶意代码中流行的原生API调用

- NtContinue
 - 用来从一个异常处理返回
 - 可用于复杂的方式转移执行
 - 用来迷惑分析师并使程序更加难调试