【看雪】第一课 逆向分析基础知识

原创

yeomanry 于 2010-02-02 11:47:00 发布 1073 收藏 1
文章标签： windows 编译器 汇编 数据结构 任务调度 delphi

本文链接：https://blog.csdn.net/yeomanry/article/details/6486199

第一课 逆向分析基础知识

1.1 调用约定

在分析汇编代码时总是要遇到无数的Call，对于这些Call，尽量要根据Call之前传递的参数和Call的返回值来判断Call的功能。传递参数的工 作必须由函数调用者和函数本身来协调，计算机提供了一种被称为栈的数据结构来支持参数传递。

当参数个数多于一个时，按照什么顺序把参数压入堆栈。函数调用后，由谁来把堆栈恢复。在高级语言中，通过函数调用约定来说明这两个问题。常见的调用约定 有：

__cdecl 调用约定	PASCAL 调用约定	__stdcall 调用约定
push par3 ; 参数按右到左传递	push par1 ; 参数按左到右传递	push par3 ; 参数按右到左传递
push par2	push par2	push par2
push par1	push par3	push par1
call test1	call test1 ; 函数内平衡堆栈	call test1 ; 函数内平衡堆栈
add esp,0C ; 平衡堆栈		

【例】按__stdcall约定调用函数test2(Par1, Par2)

```
push par2 ; 参数2
push par1 ; 参数1
call test2;
{
push ebp ; 保护现场原先的EBP指针
mov ebp, esp ; 设置新的EBP指针，指向栈顶
mov eax, [ebp+0C] ; 调用参数2
mov ebx, [ebp+08] ; 调用参数1
sub esp, 8 ; 若函数要用局部变量，则要在堆栈中留出点空间
…
add esp, 8 ; 释放局部变量占用的堆栈
pop ebp ; 恢复现场的ebp指针
ret 8 ; 返回（相当于ret; add esp,8）
}
```

其堆栈调用示意图：

1.2 局部变量

在子程序内部说明的变量称为局部变量，局部变量的作用域是其所在的子程序。从汇编角度来看，局部变量就是一个临时堆栈缓存，用完释放。
例如这个实例：附件:local.zip（http://bbs.pediy.com/upload/bbs/faq/local.zip）

其反汇编代码如下（红体字为局部变量）：

```
00401000 >/$ 6A 04 push 4 ; /Arg2 = 00000004
00401002 |. 6A 03 push 3 ; |Arg1 = 00000003
00401004 |. E8 16000000 call 0040101F ; /Add.0040101F
00401009 |. 8BD8 mov ebx, eax
0040100B |. 6A 00 push 0 ; /ExitCode = 0
0040100D /. FF15 00204000 call [<&KERNEL32.ExitProcess>] ; /ExitProcess


0040101F /$ 55 push ebp ; 保护现场原先的EBP指针
00401020 |. 8BEC mov ebp, esp ; 设置新的EBP指针，指向栈顶
00401022 |. 83EC 04 sub esp, 4 ; 分配局部变量所有空间
00401025 |. 8B45 0C mov eax, [ebp+C] ; 调用参数2
00401028 |. 8B5D 08 mov ebx, [ebp+8] ; 调用参数1
0040102B |. 895D FC mov [ebp-4] , ebx ; 参数1放局部变 量里
0040102E |. 0345 FC add eax,[ebp-4] ; 参数2与局部变量相 加
00401031 |. 83C4 04 add esp, 4 ; 释放局部变量所有空间
00401034 |. 5D pop ebp ; 恢复现场的ebp指针
00401035 /. C2 0800 retn 8
```

1.3 返回值

在调试程序时，不要见Call就跟进，在Call之前所做的所有PUSH动作以及对寄存器的操作都可能是在给函数传递参数，而函数的返回值一般都放在 EAX里面，当然这个值可能是一个指针，指向一个数据结构。从汇编角度来看，主要有如下形式：

1)通过寄存器返回函数值；
2)通过参数按引用方式返回函数值；
3)通过全局变量返回函数值；
4)通过处理器标志返回函数值；
一般情况下，由retrun操作符返回的值放在EAX寄存器之中，如果结果超过这个寄存器的位容量，那么该结果的高32位会加载到EDX寄存器中。 如果 返回一个含有几百个字节的结构或者一个近似大小的对象，编译器会在不告诉程序的情况下，给函数传递一个隐式参数，这个指针指向保存的返回结果。

1.4 启动函数

在编写Win32应用程序时，都必须在源码里实现一个WinMain函数。但Windows程序执行并不是从WinMain函数开始的，首先被执行的是启 动函数相关代码，这段代码是编译器生成的。启动代码完成初始化进程，再调用WinMain。标准编译器通常包含启动代码在内的库文件源码，例如 Visual C++中，启动代码存放在CRT/SRC/crt0.c文件中。

所有的C/C++运行时启动函数的作用基本都是相同的：检索指向新进程的命令行指针，检索指向新进程的环境变量指针，全局变量初始化，内存堆栈初始化等。 当所有的初始化操作完毕后，启动函数就调用应用程序的进入点函数。
调用WinMain如下所示：
GetStartupInfo (&StartupInfo);
Int nMainRetVal = WinMain(GetModuleHandle(NULL),NULL,pszCommandLineAnsi,
(StartupInfo.dwFlags&STARTF_USESHOWWINDOW)?StartupInfo.wShowWindow:SW__SHOWDEFAULT);
当进入点返回时，启动函数便调用C运行库期的exit函数，将返回值（nMainRetVal）传递给它，进行一些必要处理，最后调用系统函数 ExitProcess退出。 其他一些编译器，如Delphi、BorLand C++开发包中都有相应的启动代码。
在绝大数情况下，我们对启动代码并不需要关心。 对于逆向分析人员来说，首要的任务是找到Winmain函数。

WinMain函数原型如下：
int WINAPI WinMain(
HINSTANCE hInstance, // 当前实例的句柄
HINSTANCE hPrevInstance, // 前一个实例的句柄
LPSTR lpCmdLine, // 命令行的指针
int nCmdShow // 窗口的显示状态
);
其中参数hInstance一般通过GetModuleHandleA函数进行获取的，这对识别WinMain函数有些帮助。另外，对WinMain的调 用通常放在启动函数代码结尾部分，后面通常跟着诸如exit或XcptFilter之内的两、三个函数。例如下面这段代码：
.text:004010DC push eax ; nShowCmd
.text:004010DD push [ebp+lpCmdLine] ; lpCmdLine
.text:004010E0 push esi ; hPrevInstance
.text:004010E1 push esi ; lpModuleName
.text:004010E2 call ds:GetModuleHandleA
.text:004010E8 push eax ; hInstance
.text:004010E9 call WinMain(x,x,x,x)
.text:004010EE mov [ebp+var_60], eax
.text:004010F1 push eax ; int
.text:004010F2 call _exit

许多开发人员可以得到启动源代码的情况下对启动代码进行修改，这样，程序的执行可能不是从WinMain开始，而是从任何其他的函数开始。

1.5 API函数

现在很多讲Windows程序设计的书都是讲基于MFC库和OWL库的Windows设计，对Windows实现的细节都鲜有讨论，而调试程序都是和系统 底层打交道，所以有必要掌握一些Win32 API函数的知识，这样我们可快捷地找出程序调用错在哪？是哪个参数出了问题。
Windows程序模块包括KERNEL、USER和GDI，其中KERNEL完成内存管理、程序的装人与执行和任务调度等功能，它需要调用原 MS?DOS中的文件管理、磁盘输入输出和程序执行等功能；USER是一个程序库，它用来对声音、时钟、鼠标器及键盘输入等操作进行管理；GDI是一功能 十分丰富的子程序库，它提供了图形与文字输出、图象操作和窗口管理等各种与显示和打印有关的功能。上述KERNEL、USER和GDI模块中的库函数可被应用程序调用，也可被其他程序模块调用。把包含库函数的模块称为输出者（export）。你应明白为什么跟踪软件时经常在KERNEL32!.text和 USER32.text等系统领空转的问题吧。