2021/03/18_x86逆向_第11课_函数、作用域

笔记本： x86逆向-C
创建时间： 2021/3/18 星期四 10:01
作者： ileemi

函数

高版本的 VS 编译的程序是使用ebp进行寻址的。"var_xx" 表示为局部变量，"arg_xx"表示参数。而VC++6.0 编译的程序，函数的规模比较小的时候，使用esp进行寻址的，函数对面较大时会使用ebp进行变量以及参数寻址。

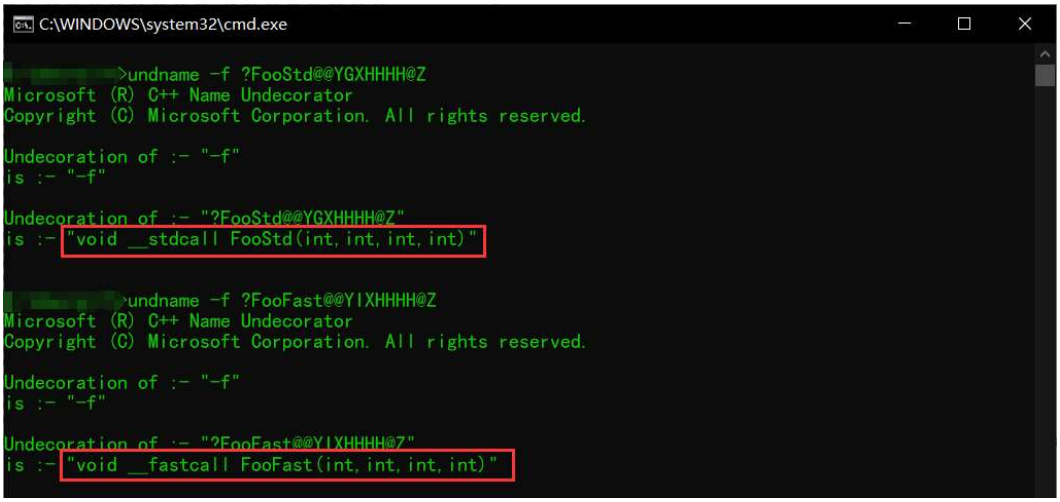
函数的调用约定

1. **__cdecl**：函数内部有参数且内部没有进行平栈（例：retn），函数调用后面的汇编代码有平栈情况。
   编译器对其函数做的符号名称格式为 "_FooC"（FooC为函数名称）。
2. **__stdcall**：函数内部有参数且函数内部进行了平栈（例：retn 10H），平栈的总字节数和参数的总字节数一致。
   编译器对其函数做的符号名称格式为 " _@参数总字节数"，例如函数 FooStd(int, int, int, int) 的符号名称为 _FooStd@16
3. **__fastcall**：函数内有对寄存器 ecx 和 edx 未赋值的使用（ecx和edx的赋值在函数调用前进行，通过寄存器传参）。分别参与参数1和参数2的信息传递工作，更多的参数通过栈传递，在函数内部有参数的访问（使用寄存器传参）。
   编译器对其函数做的符号名称格式为 " _@函数名@参数总字节数"，例如函数 FooFast(int, int, int, int) 的符号名称为 @FooFast@16。

> 情况1：在对应的 ".obj" 文件中可进行查看（函数返回值类型前有添加 extern "C"）。
> 情况2：如果 ".obj" 文件中存储的是名称粉碎后的函数名，可通过 "undname" 工具解析源函数的名称以及调用约定参数的类型，如下所示：

```
000580  00 00 00 3F 3F 5F 43 40  5F 30 34 47 4E 4C 50 40  ...??_C@_04GNLP@
000590  3F 24 43 46 64 3F 24 41  4E 3F 36 3F 24 41 41 40  ?$CFd?$AN?6?$AA@
0005a0  00 5F 46 6F 6F 53 74 64  40 31 36 00 40 46 6F 6F  ._FooStd@16.@Foo
0005b0  46 61 73 74 40 31 36 00                          Fast@16.
```

输入命令：undname -f ?FooFast@@YIXHHHH@Z



在逆向动态库时，如果要使用其内部的函数，在使用时就需要准确的给出函数的调用约定。

函数的返回值

1. int：返回值保存在 eax 中。
2. char：返回值保存在 al 中。
3. short int：返回值保存在 ax 中。
4. __int64：返回值保存在 edx和eax 中。
5. float：VC++6.0，返回值保存在st0中（浮点协处理器第一个，st0~st7类似一个栈），访问值时使用 "dword ptr"。VS2019，会使用 "xmm0" 寄存器多媒体指令集。
6. double：返回值保存在ST0中，访问值时使用 "qword ptr"。也会使用 "xmm0" 寄存器。

作用域

局部变量和全局变量

立即数间接访问：可能是 全局变量、静态变量、全局常量。

全局变量存储在 ".data" 中（例：int g_nTest = 999;），const 全局常量存储在 ".rdata" 或者 ".text"中（例：const int g_nTest = 999;）。

参数常量和局部变量常量都是伪常量（本身的存储区域不在常量区，编译器会检查限制直接修改伪常量），可以通过指针运算以及访问技巧（越界访问）进行修改。但是直接访问这些参数常量和局部变量是会常量传播的。

全局变量的特征：

- 所在地址为数据区，声明周期与所在模块一致

- 使用立即数间接访问

局部变量的特征：

- 所在地址为栈区，生命周期与所在函数作用域一致
- 使用 ebp 或 esp 间接访问

全局变量初始化为函数的返回值

> 当全局变量以变量（或函数）给赋初值时，通过IDA静态分析该程序可以看出其初值为0，可在IDA中通过 "start" --> "__cinit" --> 点击第二个 "__initterm" 上方 push 的起始地址 --> 找到代理函数。

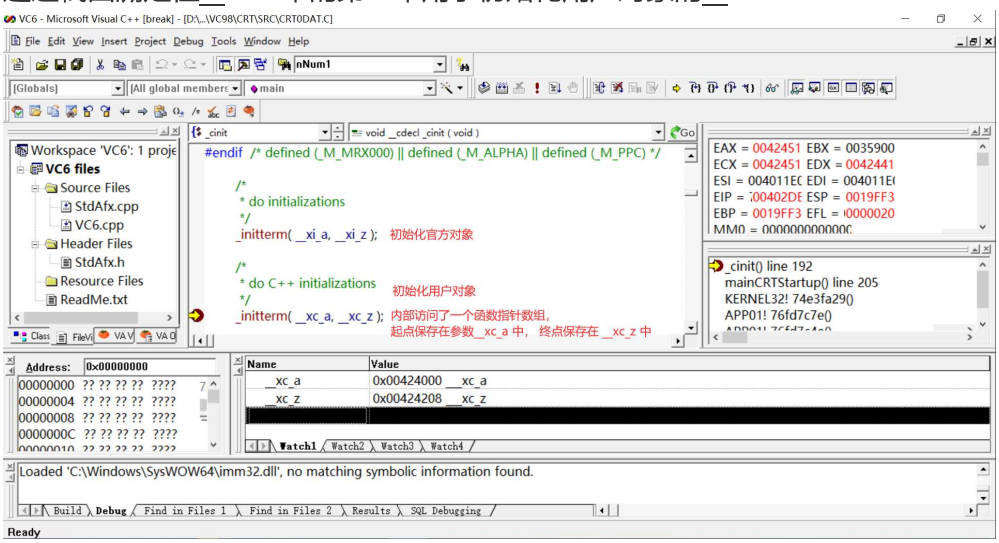
代码示例：

```
#include <stdio.h>
int GetValue(){
  return 999;
}
int g_nTest = GetValue();
int main(const int argc, char* argv[]) {
```

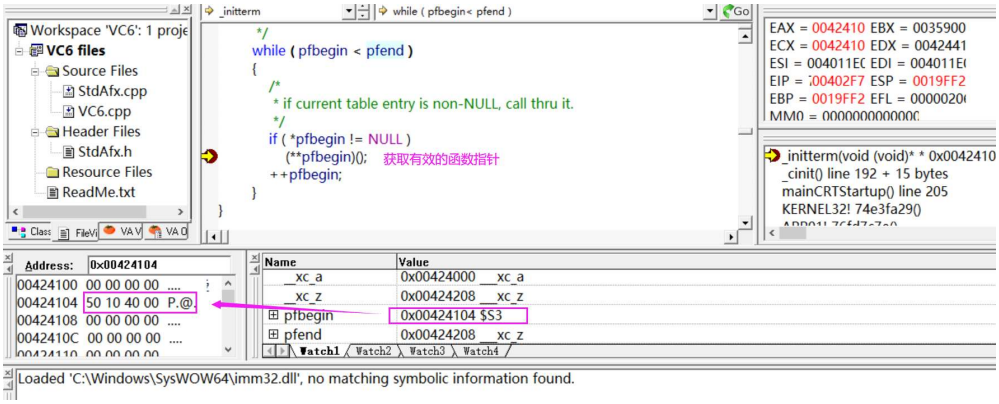
```
    printf("%d\r\n", g_nTest);
    return 0;
}
```

在VC++6.0 中调试Debug版进行分析：

1. 通过栈回溯定位__cinit中的第二个用于初始化用户对象的__initterm



2. 在__initterm中获取有效的函数指针



3. 通过函数指针的间接访问执行代理函数，通过代理函数执行目标函数，将函数的返回值赋值给全局变量。

> 在 VS 高版本中，开启优化后，上述问题的代理函数可能会被内联，函数的参数为常量时可能会进行传播以及折叠。在Release版中， initerm 初始化函数在 "__scrt_common_main_seh" 函数中。

局部静态变量

> **静态全局变量**和**全局变量**在静态分析中分辨不出来，静态全局变量在语法上是文件作用域，文件作用域的判定以及限制是编译器再编译检查期间内做的事情。所以在静态分析反汇编代码的时候将静态全局变量和全局变量都还原为全局变量。

局部静态变量不会随着作用域的结束而消失，在未进入作用域之前就已经存在，其生命周期和和全局变量相同。局部静态变量和群居变量都保存在可执行文件中的数据区，其会预先被作为全局变量处理，而它的初始化部分只是在做赋值操作而已。

局部静态变量会有一个标志（1个字节），通过位运算将标志中的一位数据置1，以此判断局部静态变量是否已经被初始化。由于一个静态变量只是用了1位，而一个字节数据有8位，所以一个1字节的标志可以同时表示8个局部静态变量的初始化状态。

在 VC++ 6.0 中，该标志所在内存地址在最先定义局部静态变量地址的附近，当同一作用域内超过8个局部静态变量时，下一个标记位会在第9个定义的局部静态变量地址附近。

1. 以常量初始化时，变量存放在初始化区，在初始化过程中不产生代码。多次进行初始化不会产生变化，这样就无需再做初始化标志，编译器采用直接以全局变量方式处理。转换位全局变量，仍然不可以超出作用域访问，通过名称粉碎，编译器在编译期间将静态变量重新命名（在原有的名称中加入其所在的作用域，以及类型等信息）。
2. 以变量或者函数的返回值初始化时，变量放在未初始化区。会产生代码，通过标志来进行限制其是否进行初始化（若为未初始化状态，将未初始化改为已初始化状态，执行初始化操作）。VC++ 6.0 通过全局变量作为标志，高版本的 VS 将标志存在 "TLS" 中（考虑到多线程存在同步问题）。

堆变量

堆变量即不在数据区也不再堆栈中。通过函数签名识别或者存储位置（地址）进行识别（静态分析靠签名，动态分析通过内存位置）。