文章编号:1001 - 9081(2009)05 - 1393 - 04

MFC 消息响应函数的逆向定位

谢裕敏,舒 辉,陈建敏,熊小兵
(信息工程大学 信息工程学院,郑州 450002)
(xieyuminhao@163.com)

摘 要:定位程序中各种关键函数的位置是软件逆向分析的一个重要工作。针对封装技术的不同设计特点采用不同的逆向分析方法,通过分析 MFC 程序的消息处理机制,提出了一种针对 MFC 程序消息处理函数地址的快速定位技术。最后,对该定位技术进行实例测试,结果表明,该技术能快速准确定位出 MFC 的目标函数,有效提高了程序逆向分析效率。

关键词:微软基础类库;逆向分析;函数定位;消息处理
中图分类号: TP311 **文献标志码**:A

Location of MFC messages processing functions in converse analysis

XIE Yu-min, SHU Hui, CHEN Jian-min, XIONG Xiao-bing
(*Institute of Information Engineering, Information Engineering University, Zhengzhou Henan* 450002, *China*)

Abstract: It is important to locate all kinds of key-functions in the program in software reverse analysis. Instructed by the idea that using different converse analysis technique for different encapsulated technology, the theory and implementation of message process mechanism of MFC programs were analyzed, and speedy search and location for all kinds of message-processing functions of MFC were realized. From the result in actual tests, this method can speedily search and locate the functions, and raise the efficiency of reverse analysis.

Key words: Microsoft Fundation Class (MFC); reverse analysis; locate function; message process

0 引言

随着计算机技术的快速发展,编程技术的逐渐普及以及编程软件功能的日趋完善,各种类型的应用程序越来越多,软件规模也越来越大,软件开发工具多种多样,软件安全工作越来越受重视[1-2],对这些软件的编写以及运行情况的研究和测试也非常困难,而且代价昂贵。如何自动且高效地挖掘和分析软件中存在的关键代码也成为逆向工程的研究者面临的一项十分严峻的课题。

在软件逆向分析工作中,一个重要的技术即是对程序处理特定事件代码的搜索和定位。对 Windows 窗口应用程序而言,即是对窗口中各种界面元素消息响应事件的处理函数进行定位。但目前基于 Windows 的软件开发工具为了使用户开发更加方便,减少用户编写的代码量,都采取了封装的方法。如今流行的 VB、.Net、CSharp、C++ Builder、Dephi 等开发工具即都采取了这种方法[3]。如果采用通常的方法对框架类的程序进行跟踪调试,不但会被众多自动生成的各种代码影响查找效率,而且还可能会被相似代码所干扰甚至误导。如果能够通过各种程序封装的特点和规律来逆向分析程序,可以加快分析速度,提高分析效率和准确率。本文重点研究如何利用 MFC(Microsoft Fundation Class)的封装特性和 MFC 代码以及数据结构的特征来快速提取 MFC 程序中消息响应函数的地址。

1 MFC 基本原理

MFC 即微软基础类库,是微软开发的一个 FrameWork 类库。可以通过 MFC 框架类库搭建程序软件结构框架,处理消息循环,并构建一些常用的应用程序模板等。

MFC 从最开始设计出来以后一直在改进,但更新到 MFC 4.2 之后其框架就基本稳定。由于现在 MFC 程序多为 VC 6.0 编写,所以本文大部分内容都是基于 VC 6.0 的开发环境。

本文将首先分析与 MFC 消息函数定位技术密切相关的 MFC 的运行时类型识别、消息映射和消息传递这几个关键技术[4-7]。

1.1 运行时类型识别和动态对象创建

运行时类型识别(RunTime Type Indentificaion, RTTI)指程序在运行过程中具备识别内存中的对象是哪个类的实例的能力。动态对象创建是指程序在运行时动态产生某个类的对象的能力。

如果一个对象为可以动态创建的对象,则需要在这个对象所在的类的定义中加上 DECLARE_DYNAMIC/IMPLEMENT_DYNAMIC 宏。DECLARE_DYNAMIC 这个宏替换后的代码添加了 3 个方法和一个 CRuntimeClass 类型的变量。这个 CRuntimeClass 类型是 MFC 预定义的结构体类型,该类型的变量是实现 RTTI 的关键所在。所有 CObject 派生的类都有一个这样的结构。因为该结构体成员变量被声明为 static 类型。在C++ 中,全局变量和静态变量都会在程序进入入口函数以前(main 函数或者是 WinMain 函数)进行分配空间实例化。这个 CRuntimeClass 类型的结构体变量成员,在程序执行用户代码之前就生成了。

```
struct CRuntimeClass
{                                          //Attributes
  LPCSTR m_lpszClassName;
```

收稿日期:2008 - 11 - 04;**修回日期**:2008 - 12 - 25。

作者简介:谢裕敏(1984 -),男,江西赣州人,硕士研究生,主要研究方向:网络信息安全; 舒辉(1974 -),男,江苏盐城人,副教授,博士,主要研究方向:网络信息安全、并行编译; 陈建敏(1984 -),男,江西萍乡人,硕士研究生,主要研究方向:网络信息安全; 熊小兵(1985 -),男,江西丰城人,硕士研究生,主要研究方向:网络信息安全。

```
    int m_nObjectSize;
    UINT m_wSchema;          //schema number of the loaded class
    CObject * (PASCAL * m_pfnCreateObject)();
                        //NULL => abstract class(抽象类为空)
    #ifdef _AFXDLL
    CRuntimeClass * (PASCAL * m_pfnGetBaseClass)();
    #else
    CRuntimeClass * m_pBaseClass;
    #endif
    …
}
```

由 CRuntimeClass 的结构可以看出它记录了一个类的类名、大小、构造方法以及其父类的相同结构。事实上这个结构按照类的继承关系组织成为一个根节点为 CObject 类中名为 classCObject 的 CRuntimeClass 结构体的多叉树。即只要获取到了这个结构，就可以获取这个类及其基类的各种相关信息。

1.2 MFC 消息映射机制

CCmdTarget 及其子类的定义中都有这样一个宏 DECLARE_MESSAGE_MAP()，所以所有这些类中都有这个宏展开的类成员及类方法，并且所有 CCmdTarget 类及其子类每次定义都添加了一组成员及成员变量。如 static const AFX_MSGMAP_ENTRY _messageEntries[] 成员在 CCmdTarget 类中第一次被定义，在 CWnd 定义时这个成员被继承，但是由于 CWnd 自身也有 DECLAARE_MESSAGE_MAP() 这个宏，所以 CWnd 自身也有一个_messageEntries[]成员，加上父类的那个同名的_messageEntries[]一共有两个_messageEntries[]成员。如果 new 出一个 CWnd 对象，则该对象的内存中实际上有父类 CCmdTarget 的 _messageEntries[] 成员，也有自己的 _messageEntries[]成员。_messageEntries 数组的元素记录该类要处理的消息映射相关的信息。而这实际上就是一个查找比对表，用这个表进行消息相应的处理。

```
struct AFX_MSGMAP_ENTRY
{ UINT nMessage;                    //Windows 消息 ID
  UINT nCode;                       //控制消息的通知码
  UINT nID;                         //Windows Control 的 ID
  UINT nLastID;
       //如果是一定范围的消息被映射，则 nLastID 指定其范围
  UINT nSig;                        //消息的动作标识
  AFX_PMSG pfn;
   //响应消息时应执行的函数(routine to call (or special value))
};
```

AFX_MSGMAP 类型结构体的 lpEntries 指向各自对应的 AFX_MSGMAP_ENTRY 数组，pBaseMap 指针指向父类中的相同结构。这个结构按照类的继承关系组织成为一个根节点为 CCmdTarget 类中名为 messageMap 的 AFX_MSGMAP 结构体的多叉树。即只要获取到了这个结构，就可以获取到这个类及其基类的所有消息映射，然后提取出所需要的各种消息及响应该消息的函数地址。

1.3 MFC 消息传递机制

在 MFC 中，窗口的消息在经过一些基本的处理后的最初进入点是 CWnd::WindowProc。CWnd 是所有窗口类的基类。WindowProc 的代码很简单，调用 OnWndMsg 进行消息分派，如果没有处理，则调用 DefWindowProc 作默认处理。最重要的是 OnWndMsg 成员函数，去掉了命令通知消息和一些优化代码后得到该函数简化的代码如下：

```
BOOL CWnd:: OnWndMsg(UINT message, WPARAM wParam,
LPARAM lParam, LRESULT * pResult)
{ LRESULT lResult=0;
  const AFX_MSGMAP * pMessageMap;
          //取得消息映射结构，GetMessageMap 为虚函数，
              //所以实际是当前窗口所在类的消息映射
  pMessageMap=GetMessageMap();
                                  //查找对应的消息处理函数
  for(pMessageMap!=NULL; pMessageMap=pMessageMap->
    pBaseMap)
    if(message<0xC000)
      if((lpEntry=AfxFindMessageEntry(pMessageMap->
        lpEntries, message, 0, 0))!=NULL)
        goto LDispatch;
        …
        LDispatch:
        …
}
```

接下来根据当前的消息表寻找对应的消息处理函数，调用 AfxFindMessageEntry 查找消息映射表，如果找不到就继续到基类去找，这就是 For 循环做的事情。如果找到处理函数，则调用 goto LDispatch 跳到下面的 Case 语句，根据 nSig 判断函数指针的实际类型，最后转换并调用，此时相应的消息处理函数就被调用到了。

也就是说 MFC 中处理消息的顺序是：先由当前类进行处理，如果当前类不处理，则将其交给其父类进行处理，直到搜索到 CCmdTarget 类，如果还不处理，则交给 windows 进行默认的处理。

2 MFC 程序特点分析

作为一类由固定框架类生成的程序，在具体分析程序结构时首先可以找到该类程序的诸多特点。这些特点为简化分析方法，加快分析速度提供了线索。

2.1 MFC 程序的识别

由 MFC 的代码可知，AfxWndProc 是 MFC 中唯一的窗口处理程序，所有 CWnd 类及其派生类的消息都从这里进入，并通过 MFC 维护的句柄对象的对照表来获得消息所对应的对象指针，然后把指针和消息参数一并传给 AfxCallWndProc(事实上这个函数初始化后对 WM_INITDIALOG 和 WM_DESTROY 做了默认处理才把所有的消息传递给了前文提到的 CWnd::WindowProc)。而这个 AfxWndProc 首先对消息进行检查，只要是 WM_QUERYAFXWNDPROC 消息则立即返回 1，也就是说这个消息是否使用了 AfxWndProc 函数的标志。因此可以通过发送这个消息得到的返回值来检查该窗口是否由 MFC 编写。

```
LRESULT CALLBACK
AfxWndProc(HWND hWnd, UINT nMsg, WPARAM wParam,
LPARAM lParam)
{ //special message which identifies the window as using
  AfxWndProc
  if(nMsg==WM_QUERYAFXWNDPROC)
    return 1;
                  //all other messages route through message map
  CWnd * pWnd=CWnd::FromHandlePermanent(hWnd);
                                        //注意这个函数
  ASSERT(pWnd!=NULL);
  ASSERT(pWnd->m_hWnd==hWnd);
  return AfxCallWndProc(pWnd, hWnd, nMsg, wParam, lParam);
}
```

2.2 MFC 程序类型的区分

通常的 MFC 程序，按编译方式分有 Debug 版和 Release 版，按链接方式有静态链接和共享链接。这两种方式组合起

来就可以形成四种 MFC 程序,它们之间在代码编译和结构组织上又各自都有微小的差异,因此在分析程序之前需要判定其种类。

一种常规判定 MFC 应用程序是否为共享链接的方法就是检查其是否加载了 MFC 的 dll。因为共享链接的程序在运行时必须加载 MFC 的 dll,并且共享链接的 Debug 版本必须加载 mfc42d. dll,而共享链接的 Release 版本加载 mfc42. dll。如果都没有加载则可以判定为静态链接,然后根据查找节表中是否有 Debug 节来判断是否为 Debug 版本,否则为 Release 版本。在后面的分析中也可以通过应用程序代码的特征来判定与验证。

2.3　MFC 程序代码与结构的特点

2.3.1　debug 和 release 版本的区别

对于 debug 版本,VC 编译时在函数入口处为函数预留了栈空间之后一般都会再把 ecx,ebx,esi,edi 这几个寄存器压栈保存,而 release 则一般无此操作。如上面的 AfxWndProc 的起始部分在 vc6.0 的不同版本中汇编结果如下:

Debug 版:

```
PUSH    EBP
MOV     EBP, ESP
PUSH    ECX
PUSH    EBX
PUSH    ESI
PUSH    EDI
CMP     DWORD PTR SS:[EBP + C],360
JNZ     addr1
```

Release 版:

```
PUSH    EBP
MOV     EBP, ESP
CMP     DWORD PTR SS:[EBP + C],360
JNZ     addr1
```

其实在不同版本的 VC 环境下,Debug 版本的汇编结果略有差异,但 Release 版本基本不变。在后文中可以根据搜索到该函数所使用的特征串来确定 VC 的编译版本。

2.3.2　MFC 中的类信息的存储

MFC 中的类信息的存储一般是按照如下的顺序:

1)该类的 MessageMap;

2)该类的 MessageEntry;

3)该类的 RTTI Complete Object Locator(如果该类有 RTTI 信息);

4)该类的虚函数表。

跟进这个 RIIT 链表指针“virtual CRuntimeClass * GetRuntimeClass( ) const;”可以快速定位到它的类名和父类名称。查找虚函数表时可以直接搜索 GetRunTimeClass 函数,该函数必然是各个虚函数表的开始。然后可以根据 MFC 中定义虚函数的次序确定各个已知的虚函数的函数地址,当在获取到父类中对应的相关函数地址时,如果发现某个相同的虚函数的地址不相等,则可以确定这个虚函数在子类中被重载了。

2.4　MFC 程序的一些特殊控件

在实际测试中发现,如果 ID 为 0x1、0x2、0xE146 时,其对应的消息处理函数并不在 AFX_MSGMAP 的数组当中。也就是说即使遍历整个类及其基类的 AFX_MSGMAP 也发现不了对应以上 ID 的消息处理函数。实际上这些 ID 分别为 IDOK、IDCANCEL、IDHELP,对应了对话框的确定、取消和帮助按钮,它们的地址也相应的为 CDialog 类的 OnOK、OnCancel 和 CWnd 类的 WinHelpA 成员函数的地址。因此如果查找到的是对话框中以上控件的 ID,则需要转化成对这几个成员函数的查找。这样的 ID 还有如 CPropertySheet 类中的 ID_WIZNEXT、ID_WIZBACK 等。但更新版本如 MFC7.1 则有些情况不需要进行转换。具体情况本文在此不进行深入讨论。

3　消息函数查找设计与关键技术

为了简化分析过程,避免压缩加壳等手段对程序造成的影响,同时也为了方便逆向分析员的使用,本文将实现代码注入到正在正常运行的目标程序中进行动态分析查找。

3.1　查找过程设计

该设计以指定的控件及消息为输入,查找其对应的消息处理函数的地址。其设计流程如图 1 所示。

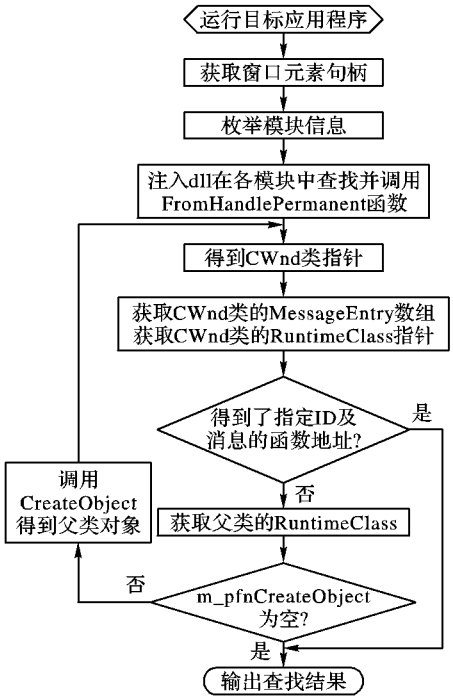

图 1　消息处理函数查找设计流程

3.1.1　获取窗口元素句柄

首先要获取需要分析的窗口的句柄,一般可以采用以下两种方式:一种是先通过穷举当前的所有窗口元素的信息,然后由用户选取需要分析的窗口元素的句柄;另一种就是由用户指定窗口的所在位置,然后通过 GetWindowFromPointEx 获取指定位置的窗口元素句柄。在获得句柄后,首先分析窗户的类型。由于按钮等特殊窗口的消息处理函数由其父窗口实现,因此应当把通过 GetParent 获取的父窗口的句柄作为研究目标,并保存在通过 GetMenu 获取按钮 ID 的同时。

3.1.2　模块枚举分析

根据获取到的窗口元素句柄可以通过 GetWindowThreadProcessId 函数获取窗口所在的进程及线程 ID,然后依据 CreateToolhelp32Snapshot 以及 TH32CS_SNAPMODULE 参数获得该进程所有模块的信息。如果模块中没有包含 MFC 的动态链接 dll(如 VC 6.0 中 Debug 对应的 mfc42d. dll 和 Release 版对应的 mfc42. dll),则不用考虑存在动态链接的情况,否则可以依据各个模块的导入表信息确定其中使用了动态链接的模块。然后根据结果优先选择使用的搜索方式。

3.1.3　由句柄获取窗口类指针

CWnd 类中的 FromHandlePermanent 函数可以将给定的句柄转换成一个 CWnd 类的指针,该指针实际指向的是该句

柄所在窗口类的对象。所以只要窗口元素是 MFC 中 CWnd 的子类的一个对象,就能获取到该类的指针,而非 CWnd 子类的则能够直接返回空。这一步的关键就在于获取这个函数。

3.1.4 获取类信息及函数信息

利用类指针可以获得类的各种信息。首先可以获取 CWnd 类的各种成员变量的值以及虚函数数组,再利用 GetRuntimeClass 函数获得该类的 RuntimeClass 类成员,利用 GetMessageMap 函数获得该类的 MessageEntry 结构数组。由前面的介绍可以知道,在这一步时根据该结构基本已经获取到了绝大部分事件所对应的 ID 以及函数地址,所以最后可以通过需要查找的控件 ID 返回相应得函数地址。

3.1.5 获取更多的信息

如果在该类中没有得到与该控件的 ID 相对应的消息处理函数,则需要在其父类中搜索。通过 RuntimeClass 类 m_pfnGetBaseClass 或者 m_pBaseClass 成员获取到父类的 RuntimeClass 类成员后,如果 m_pfnCreateObject 不为空,则可以通过调用这个成员获取父类的对象。继而获得父类的 MessageEntry 结构数组。这里还可以通过与父类的虚函数表比较,得出哪些虚函数在编写时被重载了。最后可以根据函数的地址,通过 VirtualQuery 获取到该函数所在的模块,从而判断该函数在哪个文件中。

3.2 相关关键技术

3.2.1 函数查找代码的注入

为了加快查找函数的速度,并且方便代码的编写,程序将把部分需要运行的代码注入到目标程序的运行空间内,使得这部分代码可以直接访问目标程序的内存空间并调用目标程序的各种函数来获取信息。

本文采用的是通过使用钩子注入 dll 的方法,通过 SetWindowsHookEx 使用 WH_CALLWNDPROC 类型的钩子替换目标线程的消息处理函数,这样在收到指定消息时新的消息处理函数就可以运行 dll 中的函数查找代码。最后查找完毕需要卸载钩子。

3.2.2 各模块类型的具体分析

这个分析工作应当在注入 dll 之前完成,以减少分析量,避免额外分析注入的 dll。由于目标程序在运行时由多个模块组成,不仅拥有各种系统提供的动态链接库,而且也可能有用户自己编写的 dll。由于窗口元素对应的消息处理函数未必位于主模块中,所以在 WM_QUERYAFXWNDPROC 消息返回为 1 后需要在所有的 MFC 模块中进行查找。为了简单快速地分析出各模块的类型,实际上可以采用 AfxWndProc 函数代码固定的这个特征。前面已经提到了这个函数在两种编译条件下固定得出两种结果,因此只需要在各个模块的代码段中搜索匹配这个函数。如果匹配成功则可以在得到 AfxWndProc 函数地址的同时根据匹配结果得出模块的 MFC 类型。

3.2.3 CWnd::FromHandlePermanent 函数的获取

在前面的分析中可以发现,整个查找过程中最关键的一步是通过 FromHandlePermanent 函数获取到窗口类的指针。因此获取到 CWnd::FromHandlePermanent 函数的地址至关重要。但在上一步的分析中,已经通过查找函数的方式获取了 AfxWndProc 函数的地址,由该函数的源代码可以看到其函数内部的第一个函数调用就是 CWnd::FromHandlePermanent。所以只要从 AfxWndProc 函数起始位置开始,继续搜索到它调用的第一个函数的地址就是 FromHandlePermanent 函数的地址。

4 测试及结果

下面对一些常用的工具进行测试与分析,以验证方法的正确性,具体分析以迅雷(版本 5.7.4.401)为例。

为分析迅雷的登录过程,需要对左侧的登录按钮进行研究。首先根据位置获取其句柄(测试中为 0x001007C0),通过句柄得到此按钮类名为 Button,控件 ID 为 0x042e,并获得父窗口为 CDialog 类,句柄为 0x0056074E。发送 WM_QUERYAFXWNDPROC 消息确认该程序为 MFC 编写。通过枚举该进程的模块,根据其加载了 MFC42.DLL 初步推断该程序为采用 MFC4.2 框架的共享链接 Release 版本。然后便可以注入 DLL 到该进程,在 DLL 中调用 MFC42.DLL 中的 CWnd::FromHandlePermanent 函数得到父窗口句柄对应的 CWnd 型指针。该指针存放在栈中,测试中其地址为 0x010223F0,指向 0x2267FA04。调用或分析 GetMessageMap 成员函数得到该类的 AFX_MSGMAP 结构,再顺次得到 AFX_MSGMAP_ENTRY 数组。在数组中查找按钮的 ID,仅有{0x111,0x0,0x42E,0x42E,0xC,0x226045DE}满足要求。继续在基类中查找,没有更多发现(如果没有修改 MFC 框架代码,显然不应该有更多发现)。所以点击登录后,最终的功能实现代码位于 0x226045DE 处,在 XLCommunity.dll 的代码段范围之内。

根据相同的方法,得到迅雷右上方的"资源搜索"按钮 ID 为 0x03eb,父窗口为 CWnd 类,测试中类指针指向 0x22F699A0。GetMessageMap 成员函数地址为 0x22F62023,AFX_MSGMAP 结构地址为 0x22F69848,AFX_MSGMAP_ENTRY 数组中{0x111,0x0,0x03eb,0x03eb,0xC,0x22F626D6}满足要求,位于 XLSearch.dll 的代码段范围之内。

为了验证得到的结论,利用 WinDbg 采用附加的方式调试 Thunder 进程,分别在 0x226045DE、0x22F626D6 处下断点,然后在运行时点击按钮,发现程序在断点处断住,继续调试后可以确定该处确实为实现目标功能的代码段。

表 1 是通过测试得到的其他一些常用工具中经常用到的按钮的分析结果。经过分别调试验证,均证明本文所述方法准确可靠。

表 1 常用按钮的测试结果

程序(版本)	按钮名称(ID)	CWnd 类地址	函数地址
金山词霸(10.0.0.4)	本地查词(0x3F0)	0x4CFAC0	0x43C1DA
金山词霸(10.0.0.4)	帮助(0x9)	0x4D07DC	0x447242
QQ2008(8.0.1253.201)	登录(0x3ea0)	0x6090BDE0	0x608EEC51
QQ2008(8.0.1253.201)	发送(0x77e)	0x317C53C	0x30B32C8
QQ2008(8.0.1253.201)	修改密码(0x840)	0x6250CB90	0x624E4BF2
vs 平台(2.5.1025.2135)	登录(0x414)	0x511C88	0x47B979

(下转第 1400 页)

395 002,单约束 2 = 485 003、并约束 = 395 002 ∨ 485 003),PCCL 和 CCLA 两种算法的构造时间比较。

表1 不同的数据集和约束条件下约束概念格节点数

约束条件	数据集条数			
	2000	4000	6000	8315
395 002 ∧ 485 003	177	319	367	599
395 002	609	1 129	1 256	2 138
485 003	3 528	6 038	6 699	10 858
395 002 ∨ 485 003	3 960	6 848	7 588	12 397
∅	17 504	33225	40 849	63 139

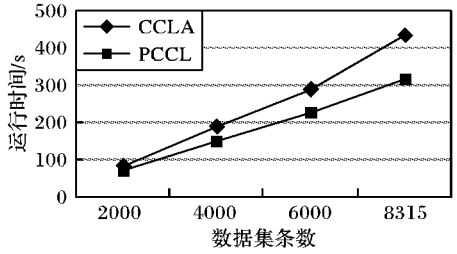

图1 实验结果1(P =395 002 ∧ 485 003)

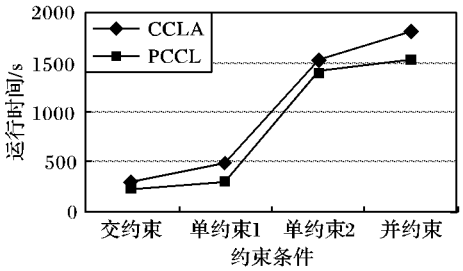

图2 实验结果2(| DB | =6 000)

由表1、图1、2可以看出:1)相同背景下,CCLA 算法构造出的约束概念格与 PCCL 算法构造出的约束概念格是相同的,从而验证了剪枝过程不改变格的结构;2)PCCL 算法比 CCLA 算法构造效率高,表明对于约束概念格渐进式构造,确实存在冗余信息,可以通过剪枝消除冗余信息,提高约束概念格的构造效率;3)随着数据集大小的增加,格节点个数和相应的建格时间也明显增加。随着约束程度的减弱,格节点个数和相应的建格时间明显增加;4)由于背景的不同,PCCL 算法的优势不同,这取决于形式背景中所包含冗余信息的比例。

5 结语

本文提出了一种基于剪枝的约束概念格渐进式构造算法,该算法是对 CCLA 算法的一种改进,消除了影响构造效率的冗余信息,达到约束概念格的剪枝目的,并以恒星光谱数据作为形式背景,实验证明了 PCCL 构造算法比 CCLA 构造算法的效率高。

参考文献:

[1] WILLE R. Restructuring lattice theory: An approach based on hierarchies of concepts [M]// Ordered Sets. Berlin: Dordrecht Reidel, 1982: 415 - 470.

[2] 张继福,蒋义勇,胡立华,等.基于概念格的天体光谱离群数据识别方法[J].自动化学报,2008,34(9):1060 - 1066.

[3] TROY A D, ZHANG GUO-QIANG, TIAN YE. Faster concept analysis [C]// Proceedings of the 15th International Conference on Conceptual Structures: Knowledge Architectures for Smart Applications. Berlin: Springer-Verlag, 2007: 206 - 219.

[4] EKUZNETSOV S O, OBIEDKOV S A. Comparing performance of algorithms for generating concept lattices[J]. Journal of Experimental and Theoretical Artificial Intelligence, 2002, 14 (23): 189 - 216.

[5] GODIN R, MISSAOUE R, ALAUI H. Incremental concept formation algorithms based on Galois (concept) lattice [J]. Computational Intelligence, 1995, 11(2): 246 - 267.

[6] 谢志鹏,刘宗田.概念格的快速渐进式构造算法[J].计算机学报,2002,25(5):490 - 496.

[7] 曲立平,刘大昕,杨静,等.基于属性的概念格快速渐进式构造算法[J].计算机研究与发展,2007,44 (z3):251 - 256.

[8] van de MERWE D, OBIEDKOV S, KOURIE D. AddIntent: A new incremental algorithm for constructing concept lattices [C]// Concept Lattices: the Second International Conference on Formal Concept Analysis. Sydney: Die Deutsche Bibliothek, 2004: 372 - 385.

[9] 曲立平,刘大昕,杨静.基于属性的相对约简格快速渐进式构造算法[J].计算机科学,2008,35(4):135 - 138.

[10] 张继福,张素兰,胡立华.约束概念格及其构造方法[J].智能系统学报,2006,2(1):31 - 38.

[11] 刘宗田,强宇,周文,等.一种模糊概念格模型及其渐进式构造算法[J].计算机学报,2007,30(2):184 - 188.

[12] 胡立华,张继福,张素兰.基于剪枝的概念格渐进式构造[J].计算机应用,2006,26(7):1659 - 1661.

(上接第1396页)

5 结语

本文介绍了 MFC 的封装原理,将 MFC 框架类构建的一些关键技术作了说明。通过深入分析 MFC 程序的特点,设计并实现了 MFC 消息处理函数的查找。该方法简单方便、迅捷可靠,能立即帮助逆向分析员给出 MFC 程序各个按钮的点击响应函数地址,并给出相应窗口类的各种信息。

本文以多个复杂的 MFC 窗口程序如腾迅的 QQ、vs 对战平台、金山词霸等作为测试目标,对各个按钮进行了测试,结果表明该方法均能够立即给出正确的响应函数地址及所在模块,对逆向分析研究 MFC 程序的运行情况起了很好的辅助作用。

如果针对各种常用编程工具的不同封装方法,分别研究其特点,这对逆向分析相应的程序能节省很多重复操作,提高逆向分析的效率。

参考文献:

[1] CERT/CC. CERT/CCS tatistics1988-2006 [EB/OL]. [2008 - 09 - 12]. http://www.cert.org/stats. 2006.

[2] 孙蔚敏.2006年网络安全工作报告[R]. 2008.

[3] BHANSALI S, CHEN WEN-KE, de JONG S, *et al*. Framework for instruction-level tracing and analysis of programs [C]// Proceedings of the 2nd International Conference on Virtual Execution Environments. New York: ACM Press, 2006: 154 - 163.

[4] Microsoft. Microsoft Developers'Network [EB/OL]. [2008 - 09 - 12]. http://msdn.microsoft.com/.

[5] 王孝喜. Windows 应用程序编程技术[M]. 天津:南开大学出版社,1998.

[6] 侯俊杰.深入浅出 MFC[M].2版.湖北:华中科技大学出版社,2001.

[7] 任哲. MFC Windows 应用程序设计[M].北京:清华大学出版社,2004.