_Detours技术介绍

detours.cpp源码分析总结

格式要求：

intro :函数功能介绍

param：参数介绍

return：返回值

示例：

C++

```
LONG WINAPI DetourUpdateThread(_In_ HANDLE hThread)
  /*intro: 刷新线程，如果联合GetCurrentThread()函数，可以更新当前的线程
    param:handle hThread 线程句柄
    return 如果成功，则返回NO_ERROR；否则，返回错误代码。return error 错误代码：ERROR_NOT_ENOUGH_MEMORY:没有足够的内存来记录线程的标识。
  */
```

总结

第一部分：苏震

C++

```
int Detour_AssertExprWithFunctionName(int reportType, const char* filename,
                                      int linenumber, const char* FunctionName,

                                      const char* msg)
/*
intro: 该函数用于声明带有函数名称的表达式，输出一些信息
param: int reportType 报告类型：_CRT_WARN、_CRT_ERROR和_CRT_ASSERT。
        char *filename 文件名字符串
        int linenumber 发生断言/报表的源文件中的行号或者NULL
return nRet 带有函数名字的表达式输出
*/

static bool detour_is_imported(PBYTE pbCode, PBYTE pbAddress)
/*
intro: 用于判断detour是否引入
param: PBYTE pbCode
```

```
param: PBYTE pbCode
        PBYTE pbAddress
return true 代表detour被引入，否则没被引入
*/

inline ULONG_PTR detour_2gb_below(ULONG_PTR address)
/*
intro: 蹦床需要放置于-2GB和2GB之间，below就是-2GB范围，above同理
param: ULONG_PTE address
return
*/

struct _DETOUR_TRAMPOLINE
{
    BYTE            rbCode[30];     // target code + jmp to pbRemain
    BYTE            cbCode;         // size of moved target code.
    BYTE            cbCodeBreak;    // padding to make debugging easier.
    BYTE            rbRestore[22];  // original target code.
    BYTE            cbRestore;      // size of original target code.
    BYTE            cbRestoreBreak; // padding to make debugging easier.
    _DETOUR_ALIGN   rAlign[8];      // instruction alignment array.
    PBYTE           pbRemain;       // first instruction after moved code.
[free list]
    PBYTE           pbDetour;       // first instruction of detour function.
};

inline PBYTE detour_gen_jmp_indirect(PBYTE pbCode, PBYTE *ppbJmpVal)
/*
intro:间接跳转
param:pbCode 目标函数
param:ppbJmpVal 蹦床的detour
return: 跳转后的地址
*/
//指针运算 pbCode[1] 指的是pbCode的接下来一个地址，赋值成0xff
//0xff++ 赋值成0x25
//0x25++处被复制成ppbJMpVal

inline PBYTE detour_gen_brk(PBYTE pbCode, PBYTE pbLimit)
//param:pbCode 目标函数
//param:pbLimit 蹦床的Remain( first instruction after moved code)


inline PBYTE detour_skip_jmp(PBYTE pbCode, PVOID *ppGlobals)
//如果有导入向量就跳过，最后，如果它是补丁跳转的目标，就跳过一个长距离跳转。

inline BOOL detour_does_code_end_function(PBYTE pbCode)
//param:pbCode:目标函数指针
//该函数用于判断是否detour起作用?
```

第二部分：姜志凯

C++

```
inline PBYTE detour_skip_jmp(PBYTE pPointer, PVOID *ppGlobals)
/*
intro:跳过import表，或者调试用间接跳转指令
param:PBYTE pPointer：指向函数的指针
PVOID *ppGlobals：变量，用于接收函数的全局数据的地址
return：返回跳过之后的地址
*/

inline void detour_find_jmp_bounds(PBYTE pbCode, PDETOUR_TRAMPOLINE
*ppLower,PDETOUR_TRAMPOLINE *ppUpper)
/*
intro:设置蹦床的低地址和高地址
param:PBYTE pbCode：目标函数地址
PDETOUR_TRAMPOLINE *ppLower：蹦床低地址
PDETOUR_TRAMPOLINE *ppUpper：蹦床高地址
*/

inline BOOL detour_does_code_end_function(PBYTE pbCode)
/*
intro:判断目标地址是否可用?
IA64里不需要
*/

inline ULONG detour_is_code_filler(PBYTE pbCode)
/*
intro:IA64不需要
*/

struct _DETOUR_TRAMPOLINE
/*
intro:蹦床结构
*/
```

return:

C++

```
*/
```

第三部分：臧国盛

C++

```
//DETOUR_TRACE 相当于 printf
// Starting at pbHi, try to allocate a memory region, continue until pbLo.
//分配一个内存区域，pbHi? pbLo?
//PVOID相当于 void*
//有关指针类型void*:
//所有指针都是一个32位二进制数（32位系统下），这个意义上说所有指针都是一样的，它们的大小一样
//用于指向内存中的某处地址，然而指针为什么要有类型之分呢？答案是指针偏移。例如p为一个指针，
//它指向内存某处地址，那么p+1（或者写p[1]）是什么意思呢？答案是p指向地址的后面那个地址，那么
//后面多少呢？这就看指针类型了，假如它是字符指针，那么就是后面一个字节，假如它是整型指针，那就
//是后面第四字节，假如它是一个结构体，那就是后面sizeof(结构体)个字节。可以说，指针有类型之分，
//完全就是为了计算地址偏移。这一区别到了汇编级就没有分别了，汇编级不存在指针类型，只有指针偏移
//数。
//那么void 指针是啥呢？答案是无类型指针。干啥用呢？它只是一个地址指向，从不用计算偏移，
//它只能指向一整块内存，只能通过它来访问这块内存，不能用偏移访问（p 1，p[1]等，千万不要用在
//void指针上）。它的好处是什么呢？答案是不用强制转换，任何类型指针都可直接赋值给一个void

//指针，而不用转换。
static PVOID detour_alloc_region_from_hi(PBYTE pbLo, PBYTE pbHi)
{
```

C++

```
//给蹦床函数分配空间
//PDETOUR_TRAMPOLINE自定义的蹦床结构体
static PVOID detour_alloc_trampoline_allocate_new(PBYTE pbTarget,
                                                  PDETOUR_TRAMPOLINE pLo,
                                                  PDETOUR_TRAMPOLINE pHi)
{
```

C++

```
PVOID WINAPI DetourAllocateRegionWithinJumpBounds(_In_ LPCVOID pbTarget,
                                                  _Out_ PDWORD
pcbAllocatedSize)
{
//在给定地址附近分配一个可执行区域。
//pbTarget:开始搜索可分配空间的地址
//pcbAllocatedSize:接收以字节为单位的已分配区域大小的变量
//如果返回成功，返回所分配的页面区域的基址;否则,返回NULL。
//要释放由detourallocateregionwithinjumpounds分配的区域，使用VirtualFree函数。
```

C++

```
static PDETOUR_TRAMPOLINE detour_alloc_trampoline(PBYTE pbTarget)
{
//申请蹦床
//在目标函数+/-2GB范围内放置蹦床
//确保存在默认区域。
//首先检查默认区域是否有有效的空闲块。
//然后检查现有区域是否有有效的空闲块。
//需要分配一个新的区域
//将pbTarget四舍五入到64KB块。
```

C++

```
static void detour_free_trampoline(PDETOUR_TRAMPOLINE pTrampoline)
{
//释放蹦床
//memset() 的作用是在一段内存块中填充某个给定的值。因为它只能填充一个值，所以该函数的初始化为
//原始初始化，无法将变量初始化为程序中需要的数据。用memset初始化完后，后面程序中再向该内存空间
//中存放需要的数据。
```

C++

```
static BOOL detour_is_region_empty(PDETOUR_REGION pRegion)
{
//判断某个内存区域是否为空
```

C++

```cpp
static void detour_free_unused_trampoline_regions()
{
//释放未使用的蹦床空间
```

C++

```cpp
//detour线程结构定义
struct DetourThread
{
```

C++

```cpp
//detour操作结构定义
struct DetourOperation
{
```

C++

```cpp
PVOID WINAPI DetourCodeFromPointer(_In_ PVOID pPointer,
                                   _Out_opt_ PVOID *ppGlobals)
{
//用于查找目标函数的api
/*
intro:DetourCodeFromPointer返回实际目标函数的地址，而不是跳转语句。
param:PVOID pPointer 指向函数的指针。
param:PVOID *ppGlobals 用于接收函数的全局数据的地址。
return detour_skip_jmp 返回一个指向实现函数指针的代码的指针。
*/
```

C++

```cpp
BOOL WINAPI DetourSetIgnoreTooSmall(_In_ BOOL fIgnore)
{
//在附加或分离单独的绕道功能失败时启用或禁用事务中止。
//如果Detours之前忽略了目标函数太小而不能绕道，则返回TRUE;否则,返回FALSE。
//Ignore:指定是否忽略过小而不能绕道的函数。如果该参数设置为TRUE，则遇到这些函数时将被忽略。
//如果将此参数设置为FALSE，则遇到太小而不能绕道的函数将导致DetourTransactionCommit失败。
```

C++

```
BOOL WINAPI DetourSetRetainRegions(_In_ BOOL fRetain)
{
//强制Detours保留Trampline分配区域，即使已经释放了蹦床。
//如果Detours之前保留了未使用的trampolines区域，则返回TRUE;否则,返回FALSE。
/*fRetain:指定在区域中的所有trampolines被释放后，是否应该保留(并重用)trampolines内存分配区域。
如果设置为TRUE，则保留这些区域。如果该参数设置为FALSE，则不保留region。*/
/*Detours从64KB内存的连续区域分配trampolines。默认情况下，当区域中的所有trampolines都被释放(分离)后，这些区域将返回给操作系统。然而，在某些情况下，例如当程序频繁地附加、分离和重新附加时，可能需要保留内存区域。*/
```

C++

```
//设置不能用于Trameplnes的内存区域的下界，因为它是为系统dll保留的。
//返回上一个下界值。
//pSystemRegionLowerBound:指定Detours必须避免放置Trameplenes的系统区域的下界。
PVOID WINAPI DetourSetSystemRegionLowerBound(_In_ PVOID pSystemRegionLowerBound)
{
```

C++

```
//设置不能用于Tramplines的内存区域的上界，因为它是为系统dll保留的。
//返回上一个上界值。
//pSystemRegionUpperBound:指定Detours必须避免放置蹦床的系统区域的上界。
PVOID WINAPI DetourSetSystemRegionUpperBound(_In_ PVOID pSystemRegionUpperBound)
{
```

第四部分：于文明

C++

```
LONG WINAPI DetourUpdateThread(_In_ HANDLE hThread)
  /*intro: 刷新线程，如果联合GetCurrentThread()函数，可以更新当前的线程
    param:handle hThread 线程句柄
    return 如果成功，则返回NO_ERROR；否则，返回错误代码。return error 错误代码:ERROR_NOT_ENOUGH_MEMORY:没有足够的内存来记录线程的标识。
  */
```

C++

```
LONG WINAPI DetourAttach(_Inout_ PVOID *ppPointer,
                         _In_ PVOID pDetour)
/*
param:PVOID ppPointer 指向将要被挂接函数地址的函数指针(被hook函数)
param:PVOID pDetour 指向实际运行的函数的指针
return 调用DetourAttachEX函数
*/
```

C++

```
LONG WINAPI DetourAttachEx(_Inout_ PVOID *ppPointer,
                           _In_ PVOID pDetour,
                           _Out_opt_ PDETOUR_TRAMPOLINE *ppRealTrampoline,
                           _Out_opt_ PVOID *ppRealTarget,
                           _Out_opt_ PVOID *ppRealDetour)
/*
intro:DetourAttachEx 将绕行附加到目标函数，并检索有关最终目标的其他详细信息
param:PVOID ppPointer:指向绕道将附加到的目标指针的指针。
param:pDetour:指向detour函数的指针。
ppRealTrampoline:可选接收蹦床地址的变量。
ppRealTarget:可选接收目标函数的最终地址的变量。
ppRealDetour:可选接收detour函数最终地址的变量。
return 如果成功返回NO_ERROR;否则，返回错误代码。
错误代码
ERROR_INVALID_BLOCK:被引用的函数太小，不能绕道。

ERROR_INVALID_HANDLE: ppPointer形参为NULL或指向NULL指针。

ERROR_INVALID_OPERATION:不存在挂起的事务。

ERROR_NOT_ENOUGH_MEMORY:没有足够的内存来完成操作。
说明: ppPointer参数所指向的变量必须在事务期间保持活动，直到DetourTransactionCommit、
DetourTransactionCommitEx或DetourTransactionAbort被调用。
*/
```

C++

```
DetourDetach(_Inout_ PVOID *ppPointer,
                         _In_ PVOID pDetour)
/*
intro:恢复拦截函数
参数和DetourAttach相同
*/
```

C++

```cpp
BOOL WINAPI DetourVirtualProtectSameExecuteEx(_In_  HANDLE hProcess,
                                              _In_  PVOID pAddress,
                                              _In_  SIZE_T nSize,
                                              _In_  DWORD dwNewProtect,
                                              _Out_ PDWORD pdwOldProtect)
// Some systems do not allow executability of a page to change. This function applies
// dwNewProtect to [pAddress, nSize), but preserving the previous executability.
// This function is meant to be a drop-in replacement for some uses of VirtualProtectEx.
// When "restoring" page protection, there is no need to use this function.
/*
intro:某些系统不允许更改页面的可执行性。
此函数将 dwNewProtect 应用于 [pAddress， nSize），但保留以前的可执行性。
此功能旨在替代VirtualProtectEx的某些用途。"恢复"页面保护时，无需使用此功能。
param: HANDLE hProcess更改其内存保护的进程的句柄。句柄必须具有PROCESS_VM_OPERATION访问权限。
return VirtualProtectEX()
*/
{
  return VirtualProtectEx(hProcess, pAddress, nSize,
                          DetourPageProtectAdjustExecute(mbi.Protect, dwNewProtect),
                          pdwOldProtect);
    /*VirtualProtectEx
    intro:更改对指定进程的虚拟地址空间中已提交页区域的保护。
    //param:hProcess要更改其内存保护的进程的句柄。句柄必须具有PROCESS_VM_OPERATION访问权限。有关详细信息
    param:pAddress指向要更改其访问保护属性的页面区域的基址的指针。
    指定区域中的所有页面必须位于使用MEM_RESERVE调用 VirtualAlloc 或 VirtualAllocEx 函数时分配的同一保留区域内。这些页面不能跨越相邻的保留区域，这些保留区域是通过使用 MEM_RESERVE对 VirtualAlloc 或 VirtualAllocEx 的单独调用而分配的。
    param:[in] dwSize
   访问保护属性已更改的区域的大小（以字节为单位）。受影响页面的区域包括包含从 lpAddress 参数到 的范围内的一个或多个字节的所有页面。这意味着跨越页面边界的 2 字节范围会导致两个页面的保护属性发生更改。(lpAddress+dwSize)
    param:[in] flNewProtect
    内存保护选项。此参数可以是内存保护常量之一。
    对于映射视图，此值必须与映射视图时指定的访问保护兼容
    param:[out] lpflOldProtect
    指向一个变量的指针，该变量接收指定页区域中第一页的上一个访问保护。如果此参数为 NULL 或未指向有效变量，则函数将失败。
    return:如果函数成功，则返回值为非零。如果函数失败，则返回值为零。
```

```
34      */
35  }
```

C++

```
1  BOOL WINAPI DetourVirtualProtectSameExecute(_In_  PVOID pAddress,
2                                              _In_  SIZE_T nSize,
3                                              _In_  DWORD dwNewProtect,
4                                              _Out_ PDWORD pdwOldProtect)
5  /*intro对当前进程的去保护
6  return DetourVitualProtectSameExectuteEx*/
```

C++

```
1  BOOL WINAPI DetourAreSameGuid(_In_ REFGUID left, _In_ REFGUID right)
2  /*
3  intro:比较GUID是否相同
4  param:两个rguid值
5  return bool值 如果相等返回 true 否则 返回false
6  */
```

SOURCE CODE

PART_1_ @苏震 _BEGIN

C++

```
1  //////////////////////////////////////////////////////////////////////////////
2  //
3  //  Core Detours Functionality (detours.cpp of detours.lib)
4  //
5  //  Microsoft Research Detours Package, Version 4.0.1
6  //
7  //  Copyright (c) Microsoft Corporation.  All rights reserved.
8  //
9  //#define DETOUR_DEBUG 1
10 #define DETOURS_INTERNAL
11 #include "detours.h"
12
13 #if DETOURS_VERSION != 0x4c0c1   // 0xMAJORcMINORcPATCH
14 #error detours.h version mismatch
15 #endif
16
17 #define NOTHROW
18
19 //////////////////////////////////////////////////////////////////////////////
```

```
19 ////////////////////////////////////////////////////////////////////////////
20 //
21
22 #ifdef _DEBUG
23 /*ifdef是条件编译，Debug模式下执行代码直到#endif*/
24 extern "C" IMAGE_DOS_HEADER __ImageBase;
25 /*一个C++程序包含其它语言编写的部分代码
26 类似的，C++编写的代码片段可能被使用在其它语言编写的代码中
27 不同语言编写的代码互相调用是困难的，甚至是同一种编写的代码但不同的编译器编译的代码
28 为了使它们遵守统一规则，可以使用extern指定一个编译和连接规约*/
29 /*PE文件的第一个部分就是IMAGE_DOS_HEADER，大小为64B。ImageBase为基地址*/
30
31 //声明伴有函数名字的表达式
32 int Detour_AssertExprWithFunctionName(int reportType, const char* filename,
   int linenumber, const char* FunctionName, const char* msg)
33 {
34 /*
35 : param:int reportType 报告类型：_CRT_WARN、_CRT_ERROR和_CRT_ASSERT。
36         char *filename 文件名字符串
37         int linenumber 发生断言/报表的源文件中的行号或者NULL
38 return 带有函数名字的表达式输出
39 */
40     int nRet = 0;
41     DWORD dwLastError = GetLastError();
42     CHAR szModuleNameWithFunctionName[MAX_PATH * 2];
43     szModuleNameWithFunctionName[0] = 0;
44     GetModuleFileNameA((HMODULE)&__ImageBase, szModuleNameWithFunctionName,
   ARRAYSIZE(szModuleNameWithFunctionName));
45     //GetModuleFileName获取当前进程已加载模块的文件的完整路径，该模块必须由当前进程加
   载
46     StringCchCatNA(szModuleNameWithFunctionName,
   ARRAYSIZE(szModuleNameWithFunctionName), ",",
   ARRAYSIZE(szModuleNameWithFunctionName) - strlen(szModuleNameWithFunctionName)
   - 1);
47     StringCchCatNA(szModuleNameWithFunctionName,
   ARRAYSIZE(szModuleNameWithFunctionName), FunctionName,
   ARRAYSIZE(szModuleNameWithFunctionName) - strlen(szModuleNameWithFunctionName)
   - 1);
48     SetLastError(dwLastError);
49     nRet = _CrtDbgReport(reportType, filename, linenumber,
   szModuleNameWithFunctionName, msg);
50    //_CrtDbgReport:生成具有调试消息的报告并将该报告发送到三个可能的目标
51    //_CrtDbgReport(报告类型，文件名，linenumber,moduleName,format)
52     SetLastError(dwLastError);
53     return nRet;
54 }
55 #endif// _DEBUG
56
```

```
57  //////////////////////////////////////////////////////////////////////////////
58  //
59  struct _DETOUR_ALIGN
60  {
61  //BYTE:unsigned char
62      BYTE    obTarget        : 3;    //3bit的BYTE变量obTarger
63      BYTE    obTrampoline    : 5;    //5bit的BYTE变量obTrampoline
64  };
65
66  //assert是编写代码时作出一些假设，断言就是用于在代码中捕捉这些假设，可以将断言看作是异常处理的一种高级形式
67  //如果assert的条件返回错误，终止程序执行
68  C_ASSERT(sizeof(_DETOUR_ALIGN) == 1);
69
70  //////////////////////////////////////////////////////////////////////////////
71  //
72  // Region reserved for system DLLs, which cannot be used for trampolines.
73  //为系统动态链接库保留的区域，不能用于蹦床
74  static PVOID    s_pSystemRegionLowerBound   = (PVOID)(ULONG_PTR)0x70000000;
75  static PVOID    s_pSystemRegionUpperBound   = (PVOID)(ULONG_PTR)0x80000000;
76
77  //////////////////////////////////////////////////////////////////////////////
78  //
79  static bool detour_is_imported(PBYTE pbCode, PBYTE pbAddress)
80  {
81   //MEMORY_BASIC_INFORMATION: 用来存放虚拟地址空间或虚拟页的属性或状态的结构
82   /*typedef struct _MEMORY_BASIC_INFORMATION {
83  PVOID BaseAddress;
84  PVOID AllocationBase;
85  DWORD AllocationProtect;
86  SIZE_T RegionSize;
87  DWORD State;
88  DWORD Protect;
89  DWORD Type;
90  }*/
91  /*e_magic 字段是一个DOS可执行文件的标识符，该字段占用两个字节，该位置保存着的字符是“MZ”*/
92      MEMORY_BASIC_INFORMATION mbi;
93
94      /*DWORD VirtualQuery(
95      LPCVOID lpAddress,    // address of region
96      PMEMORY_BASIC_INFORMATION lpBuffer,    // address of information buffer
97      DWORD dwLength     // size of buffer
98     );*/
99      VirtualQuery((PVOID)pbCode, &mbi, sizeof(mbi));
100
101     __try {
102         PIMAGE_DOS_HEADER pDosHeader = (PIMAGE_DOS_HEADER)mbi.AllocationBase;
```

```
        if (pDosHeader->e_magic != IMAGE_DOS_SIGNATURE) {
            return false;
        }

        PIMAGE_NT_HEADERS pNtHeader = (PIMAGE_NT_HEADERS)((PBYTE)pDosHeader +
                                                          pDosHeader-
>e_lfanew);
        if (pNtHeader->Signature != IMAGE_NT_SIGNATURE) {
            return false;
        }

        if (pbAddress >= ((PBYTE)pDosHeader +
                          pNtHeader->OptionalHeader

.DataDirectory[IMAGE_DIRECTORY_ENTRY_IAT].VirtualAddress) &&
            pbAddress < ((PBYTE)pDosHeader +
                         pNtHeader->OptionalHeader

.DataDirectory[IMAGE_DIRECTORY_ENTRY_IAT].VirtualAddress +
                         pNtHeader->OptionalHeader
                         .DataDirectory[IMAGE_DIRECTORY_ENTRY_IAT].Size)) {
            return true;
        }
    }
#pragma prefast(suppress:28940, "A bad pointer means this probably isn't a PE
header.")
    __except(GetExceptionCode() == EXCEPTION_ACCESS_VIOLATION ?
             EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH) {
        return false;
    }
    return false;
}

inline ULONG_PTR detour_2gb_below(ULONG_PTR address)
{//param:address地址
//return返回一个低地址保证trampoline放置的2GB空间
    return (address > (ULONG_PTR)0x7ff80000) ? address - 0x7ff80000 : 0x80000;
}

inline ULONG_PTR detour_2gb_above(ULONG_PTR address)
{
#if defined(DETOURS_64BIT)
    return (address < (ULONG_PTR)0xffffffff80000000) ? address + 0x7ff80000 :
(ULONG_PTR)0xfffffffffff80000;
#else
    return (address < (ULONG_PTR)0x80000000) ? address + 0x7ff80000 :
(ULONG_PTR)0xfff80000;
#endif
```

```
144 #endif
145 }
146
147 ////////////////////////////////////////////////////////////////////////// X86.
148 //
149 #ifdef DETOURS_X86
150
151 struct _DETOUR_TRAMPOLINE
152 {
153     BYTE           rbCode[30];     // target code + jmp to pbRemain
154     BYTE           cbCode;         // size of moved target code.
155     BYTE           cbCodeBreak;    // padding to make debugging easier.
156     BYTE           rbRestore[22];  // original target code.
157     BYTE           cbRestore;      // size of original target code.
158     BYTE           cbRestoreBreak; // padding to make debugging easier.
159     _DETOUR_ALIGN  rAlign[8];      // instruction alignment array.
160     PBYTE          pbRemain;       // first instruction after moved code.
    [free list]
161     PBYTE          pbDetour;       // first instruction of detour function.
162 };
163
164 C_ASSERT(sizeof(_DETOUR_TRAMPOLINE) == 72);
165
166 enum {
167     SIZE_OF_JMP = 5
168 };
169
170 inline PBYTE detour_gen_jmp_immediate(PBYTE pbCode, PBYTE pbJmpVal)
171 {
172     PBYTE pbJmpSrc = pbCode + 5;
173     *pbCode++ = 0xE9;   // jmp +imm32
174     *((INT32*&)pbCode)++ = (INT32)(pbJmpVal - pbJmpSrc);
175     return pbCode;
176 }
177
178 inline PBYTE detour_gen_jmp_indirect(PBYTE pbCode, PBYTE *ppbJmpVal)
179 {/*
180 intro:间接跳转
181 param:pbCode 目标函数
182 param:ppbJmpVal 蹦床的detour
183 return: 跳转后的地址
184 */
185 //指针运算 pbCode[1] 指的是pbCode的接下来一个地址，赋值成0xff
186 //0xff++ 赋值成0x25
187 //0x25++处被复制成ppbJMpVal
188     *pbCode++ = 0xff;   // jmp [+imm32] pbCode下一个地址存0xfff
189     *pbCode++ = 0x25;    //pbCode下两个地址存0x25
190     *((INT32*&)pbCode)++ = (INT32)((PBYTE)ppbJmpVal);    //将ppbJampVal存入
```

```
191     return pbCode;
192 }
193
194 inline PBYTE detour_gen_brk(PBYTE pbCode, PBYTE pbLimit)
195 {
196 //param:pbCode 目标函数
197 //param:pbLimit 蹦床的Remain( first instruction after moved code)
198     while (pbCode < pbLimit) {
199         *pbCode++ = 0xcc;   // brk;
200     }
201     return pbCode;
202 }
203
204 inline PBYTE detour_skip_jmp(PBYTE pbCode, PVOID *ppGlobals)
205 {
206     if (pbCode == NULL) {
207         return NULL;
208     }
209     if (ppGlobals != NULL) {
210         *ppGlobals = NULL;
211     }
212
213     // First, skip over the import vector if there is one.
214     if (pbCode[0] == 0xff && pbCode[1] == 0x25) {   // jmp [imm32]
215         // Looks like an import alias jump, then get the code it points to.
216         PBYTE pbTarget = *(UNALIGNED PBYTE *)&pbCode[2];
217         if (detour_is_imported(pbCode, pbTarget)) {
218             PBYTE pbNew = *(UNALIGNED PBYTE *)pbTarget;
219             DETOUR_TRACE(("%p->%p: skipped over import table.\n", pbCode,
    pbNew));
220             pbCode = pbNew;
221         }
222     }
223
224     // Then, skip over a patch jump
225     if (pbCode[0] == 0xeb) {   // jmp +imm8
226         PBYTE pbNew = pbCode + 2 + *(CHAR *)&pbCode[1];
227         DETOUR_TRACE(("%p->%p: skipped over short jump.\n", pbCode, pbNew));
228         pbCode = pbNew;
229
230         // First, skip over the import vector if there is one.
231         //首先，跳过导入向量(如果有的话)。
232         if (pbCode[0] == 0xff && pbCode[1] == 0x25) {   // jmp [imm32]
233             // Looks like an import alias jump, then get the code it points
    to.
234             PBYTE pbTarget = *(UNALIGNED PBYTE *)&pbCode[2];
235             if (detour_is_imported(pbCode, pbTarget)) {
236                 pbNew = *(UNALIGNED PBYTE *)pbTarget;
```

```
237                 DETOUR_TRACE(("%p->%p: skipped over import table.\n", pbCode,
    pbNew));
238                 pbCode = pbNew;
239             }
240         }
241         // Finally, skip over a long jump if it is the target of the patch
    jump.
242         //最后，如果它是补丁跳转的目标，就跳过一个长距离跳转。
243         else if (pbCode[0] == 0xe9) {   // jmp +imm32
244             pbNew = pbCode + 5 + *(UNALIGNED INT32 *)&pbCode[1];
245             DETOUR_TRACE(("%p->%p: skipped over long jump.\n", pbCode,
    pbNew));
246             pbCode = pbNew;
247         }
248     }
249     return pbCode;
250 }
251
252 inline void detour_find_jmp_bounds(PBYTE pbCode,
253                                    PDETOUR_TRAMPOLINE *ppLower,
254                                    PDETOUR_TRAMPOLINE *ppUpper)
255 {
256     // We have to place trampolines within +/- 2GB of code.
257     ULONG_PTR lo = detour_2gb_below((ULONG_PTR)pbCode);//最低地址
258     ULONG_PTR hi = detour_2gb_above((ULONG_PTR)pbCode);//最高地址
259     DETOUR_TRACE(("[%p..%p..%p]\n", (PVOID)lo, pbCode, (PVOID)hi));
260
261     // And, within +/- 2GB of relative jmp targets.
262     if (pbCode[0] == 0xe9) {   // jmp +imm32
263         PBYTE pbNew = pbCode + 5 + *(UNALIGNED INT32 *)&pbCode[1];
264
265         if (pbNew < pbCode) {
266             hi = detour_2gb_above((ULONG_PTR)pbNew);
267         }
268         else {
269             lo = detour_2gb_below((ULONG_PTR)pbNew);
270         }
271         DETOUR_TRACE(("[%p..%p..%p] +imm32\n", (PVOID)lo, pbCode, (PVOID)hi));
272     }
273
274     *ppLower = (PDETOUR_TRAMPOLINE)lo;
275     *ppUpper = (PDETOUR_TRAMPOLINE)hi;
276 }
277
278 inline BOOL detour_does_code_end_function(PBYTE pbCode)
279 {
280 //param:pbCode:目标函数指针
```

```
//该函数用于判断是否detour起作用!
    if (pbCode[0] == 0xeb ||    // jmp +imm8
        pbCode[0] == 0xe9 ||    // jmp +imm32
        pbCode[0] == 0xe0 ||    // jmp eax
        pbCode[0] == 0xc2 ||    // ret +imm8
        pbCode[0] == 0xc3 ||    // ret
        pbCode[0] == 0xcc) {    // brk
        return TRUE;
    }
    else if (pbCode[0] == 0xf3 && pbCode[1] == 0xc3) {  // rep ret
        return TRUE;
    }
    else if (pbCode[0] == 0xff && pbCode[1] == 0x25) {  // jmp [+imm32]
        return TRUE;
    }
    else if ((pbCode[0] == 0x26 ||      // jmp es:
              pbCode[0] == 0x2e ||      // jmp cs:
              pbCode[0] == 0x36 ||      // jmp ss:
              pbCode[0] == 0x3e ||      // jmp ds:
              pbCode[0] == 0x64 ||      // jmp fs:
              pbCode[0] == 0x65) &&     // jmp gs:
             pbCode[1] == 0xff &&       // jmp [+imm32]
             pbCode[2] == 0x25) {
        return TRUE;
    }
    return FALSE;
}

inline ULONG detour_is_code_filler(PBYTE pbCode)
{
    // 1-byte through 11-byte NOPs.
    if (pbCode[0] == 0x90) {
        return 1;
    }
    if (pbCode[0] == 0x66 && pbCode[1] == 0x90) {
        return 2;
    }
    if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x00) {
        return 3;
    }
    if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x40 &&
        pbCode[3] == 0x00) {
        return 4;
    }
    if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x44 &&
        pbCode[3] == 0x00 && pbCode[4] == 0x00) {
        return 5;
    }
```

```
    if (pbCode[0] == 0x66 && pbCode[1] == 0x0F && pbCode[2] == 0x1F &&
        pbCode[3] == 0x44 && pbCode[4] == 0x00 && pbCode[5] == 0x00) {
        return 6;
    }
    if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x80 &&
        pbCode[3] == 0x00 && pbCode[4] == 0x00 && pbCode[5] == 0x00 &&
        pbCode[6] == 0x00) {
        return 7;
    }
    if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x84 &&
        pbCode[3] == 0x00 && pbCode[4] == 0x00 && pbCode[5] == 0x00 &&
        pbCode[6] == 0x00 && pbCode[7] == 0x00) {
        return 8;
    }
    if (pbCode[0] == 0x66 && pbCode[1] == 0x0F && pbCode[2] == 0x1F &&
        pbCode[3] == 0x84 && pbCode[4] == 0x00 && pbCode[5] == 0x00 &&
        pbCode[6] == 0x00 && pbCode[7] == 0x00 && pbCode[8] == 0x00) {
        return 9;
    }
    if (pbCode[0] == 0x66 && pbCode[1] == 0x66 && pbCode[2] == 0x0F &&
        pbCode[3] == 0x1F && pbCode[4] == 0x84 && pbCode[5] == 0x00 &&
        pbCode[6] == 0x00 && pbCode[7] == 0x00 && pbCode[8] == 0x00 &&
        pbCode[9] == 0x00) {
        return 10;
    }
    if (pbCode[0] == 0x66 && pbCode[1] == 0x66 && pbCode[2] == 0x66 &&
        pbCode[3] == 0x0F && pbCode[4] == 0x1F && pbCode[5] == 0x84 &&
        pbCode[6] == 0x00 && pbCode[7] == 0x00 && pbCode[8] == 0x00 &&
        pbCode[9] == 0x00 && pbCode[10] == 0x00) {
        return 11;
    }

    // int 3.
    if (pbCode[0] == 0xcc) {
        return 1;
    }
    return 0;
}

#endif // DETOURS_X86

//////////////////////////////////////////////////////////////////////////// X64.
//
#ifdef DETOURS_X64

struct _DETOUR_TRAMPOLINE
{
    // An X64 instuction can be 15 bytes long.
```

```
    // An X64 instruction can be 15 bytes long.
    // In practice 11 seems to be the limit.
    BYTE            rbCode[30];     // target code + jmp to pbRemain.
    BYTE            cbCode;         // size of moved target code.
    BYTE            cbCodeBreak;    // padding to make debugging easier.
    BYTE            rbRestore[30];  // original target code.
    BYTE            cbRestore;      // size of original target code.
    BYTE            cbRestoreBreak; // padding to make debugging easier.
    _DETOUR_ALIGN   rAlign[8];      // instruction alignment array.
    PBYTE           pbRemain;       // first instruction after moved code. [free list]
    PBYTE           pbDetour;       // first instruction of detour function.
    BYTE            rbCodeIn[8];    // jmp [pbDetour]
};

C_ASSERT(sizeof(_DETOUR_TRAMPOLINE) == 96);

enum {
    SIZE_OF_JMP = 5
};

inline PBYTE detour_gen_jmp_immediate(PBYTE pbCode, PBYTE pbJmpVal)
{
    PBYTE pbJmpSrc = pbCode + 5;
    *pbCode++ = 0xE9;   // jmp +imm32
    *((INT32*&)pbCode)++ = (INT32)(pbJmpVal - pbJmpSrc);
    return pbCode;
}

inline PBYTE detour_gen_jmp_indirect(PBYTE pbCode, PBYTE *ppbJmpVal)
{
    PBYTE pbJmpSrc = pbCode + 6;
    *pbCode++ = 0xff;   // jmp [+imm32]
    *pbCode++ = 0x25;
    *((INT32*&)pbCode)++ = (INT32)((PBYTE)ppbJmpVal - pbJmpSrc);
    return pbCode;
}

inline PBYTE detour_gen_brk(PBYTE pbCode, PBYTE pbLimit)
{
    while (pbCode < pbLimit) {
        *pbCode++ = 0xcc;   // brk;
    }
    return pbCode;
}

inline PBYTE detour_skip_jmp(PBYTE pbCode, PVOID *ppGlobals)
{
```

```
423     if (pbCode == NULL) {
424         return NULL;
425     }
426     if (ppGlobals != NULL) {
427         *ppGlobals = NULL;
428     }
429
430     // First, skip over the import vector if there is one.
431     if (pbCode[0] == 0xff && pbCode[1] == 0x25) {   // jmp [+imm32]
432         // Looks like an import alias jump, then get the code it points to.
433         PBYTE pbTarget = pbCode + 6 + *(UNALIGNED INT32 *)&pbCode[2];
434         if (detour_is_imported(pbCode, pbTarget)) {
435             PBYTE pbNew = *(UNALIGNED PBYTE *)pbTarget;
436             DETOUR_TRACE(("%p->%p: skipped over import table.\n", pbCode,
    pbNew));
437             pbCode = pbNew;
438         }
439     }
440
441     // Then, skip over a patch jump
442     if (pbCode[0] == 0xeb) {   // jmp +imm8
443         PBYTE pbNew = pbCode + 2 + *(CHAR *)&pbCode[1];
444         DETOUR_TRACE(("%p->%p: skipped over short jump.\n", pbCode, pbNew));
445         pbCode = pbNew;
446
447         // First, skip over the import vector if there is one.
448         if (pbCode[0] == 0xff && pbCode[1] == 0x25) {   // jmp [+imm32]
449             // Looks like an import alias jump, then get the code it points
    to.
450             PBYTE pbTarget = pbCode + 6 + *(UNALIGNED INT32 *)&pbCode[2];
451             if (detour_is_imported(pbCode, pbTarget)) {
452                 pbNew = *(UNALIGNED PBYTE *)pbTarget;
453                 DETOUR_TRACE(("%p->%p: skipped over import table.\n", pbCode,
    pbNew));
454                 pbCode = pbNew;
455             }
456         }
457         // Finally, skip over a long jump if it is the target of the patch
    jump.
458         else if (pbCode[0] == 0xe9) {   // jmp +imm32
459             pbNew = pbCode + 5 + *(UNALIGNED INT32 *)&pbCode[1];
460             DETOUR_TRACE(("%p->%p: skipped over long jump.\n", pbCode,
    pbNew));
461             pbCode = pbNew;
462         }
463     }
464     return pbCode;
465 }
```

```
466
467 inline void detour_find_jmp_bounds(PBYTE pbCode,
468                                    PDETOUR_TRAMPOLINE *ppLower,
469                                    PDETOUR_TRAMPOLINE *ppUpper)
470 {
471     // We have to place trampolines within +/- 2GB of code.
472     ULONG_PTR lo = detour_2gb_below((ULONG_PTR)pbCode);
473     ULONG_PTR hi = detour_2gb_above((ULONG_PTR)pbCode);
474     DETOUR_TRACE(("[%p..%p..%p]\n", (PVOID)lo, pbCode, (PVOID)hi));
475
476     // And, within +/- 2GB of relative jmp vectors.
477     if (pbCode[0] == 0xff && pbCode[1] == 0x25) {   // jmp [+imm32]
478         PBYTE pbNew = pbCode + 6 + *(UNALIGNED INT32 *)&pbCode[2];
479
480         if (pbNew < pbCode) {
481             hi = detour_2gb_above((ULONG_PTR)pbNew);
482         }
483         else {
484             lo = detour_2gb_below((ULONG_PTR)pbNew);
485         }
486         DETOUR_TRACE(("[%p..%p..%p] [+imm32]\n", (PVOID)lo, pbCode,
    (PVOID)hi));
487     }
488     // And, within +/- 2GB of relative jmp targets.
489     else if (pbCode[0] == 0xe9) {   // jmp +imm32
490         PBYTE pbNew = pbCode + 5 + *(UNALIGNED INT32 *)&pbCode[1];
491
492         if (pbNew < pbCode) {
493             hi = detour_2gb_above((ULONG_PTR)pbNew);
494         }
495         else {
496             lo = detour_2gb_below((ULONG_PTR)pbNew);
497         }
498         DETOUR_TRACE(("[%p..%p..%p] +imm32\n", (PVOID)lo, pbCode, (PVOID)hi));
499     }
500
501     *ppLower = (PDETOUR_TRAMPOLINE)lo;
502     *ppUpper = (PDETOUR_TRAMPOLINE)hi;
503 }
504
505 inline BOOL detour_does_code_end_function(PBYTE pbCode)
506 {
507     if (pbCode[0] == 0xeb ||    // jmp +imm8
508         pbCode[0] == 0xe9 ||    // jmp +imm32
509         pbCode[0] == 0xe0 ||    // jmp eax
510         pbCode[0] == 0xc2 ||    // ret +imm8
511         pbCode[0] == 0xc3 ||    // ret
512         pbCode[0] == 0xcc) {    // brk
```

```
512         pbCode[0] == 0xcc) {    // brk
513         return TRUE;
514     }
515     else if (pbCode[0] == 0xf3 && pbCode[1] == 0xc3) {  // rep ret
516         return TRUE;
517     }
518     else if (pbCode[0] == 0xff && pbCode[1] == 0x25) {  // jmp [+imm32]
519         return TRUE;
520     }
521     else if ((pbCode[0] == 0x26 ||      // jmp es:
522               pbCode[0] == 0x2e ||      // jmp cs:
523               pbCode[0] == 0x36 ||      // jmp ss:
524               pbCode[0] == 0x3e ||      // jmp ds:
525               pbCode[0] == 0x64 ||      // jmp fs:
526               pbCode[0] == 0x65) &&     // jmp gs:
527              pbCode[1] == 0xff &&       // jmp [+imm32]
528              pbCode[2] == 0x25) {
529         return TRUE;
530     }
531     return FALSE;
532 }
533
534 inline ULONG detour_is_code_filler(PBYTE pbCode)
535 {
536     // 1-byte through 11-byte NOPs.
537     if (pbCode[0] == 0x90) {
538         return 1;
539     }
540     if (pbCode[0] == 0x66 && pbCode[1] == 0x90) {
541         return 2;
542     }
543     if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x00) {
544         return 3;
545     }
546     if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x40 &&
547         pbCode[3] == 0x00) {
548         return 4;
549     }
550     if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x44 &&
551         pbCode[3] == 0x00 && pbCode[4] == 0x00) {
552         return 5;
553     }
554     if (pbCode[0] == 0x66 && pbCode[1] == 0x0F && pbCode[2] == 0x1F &&
555         pbCode[3] == 0x44 && pbCode[4] == 0x00 && pbCode[5] == 0x00) {
556         return 6;
557     }
558     if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x80 &&
559         pbCode[3] == 0x00 && pbCode[4] == 0x00 && pbCode[5] == 0x00 &&
```

```
560         pbCode[6] == 0x00) {
561         return 7;
562     }
563     if (pbCode[0] == 0x0F && pbCode[1] == 0x1F && pbCode[2] == 0x84 &&
564         pbCode[3] == 0x00 && pbCode[4] == 0x00 && pbCode[5] == 0x00 &&
565         pbCode[6] == 0x00 && pbCode[7] == 0x00) {
566         return 8;
567     }
568     if (pbCode[0] == 0x66 && pbCode[1] == 0x0F && pbCode[2] == 0x1F &&
569         pbCode[3] == 0x84 && pbCode[4] == 0x00 && pbCode[5] == 0x00 &&
570         pbCode[6] == 0x00 && pbCode[7] == 0x00 && pbCode[8] == 0x00) {
571         return 9;
572     }
573     if (pbCode[0] == 0x66 && pbCode[1] == 0x66 && pbCode[2] == 0x0F &&
574         pbCode[3] == 0x1F && pbCode[4] == 0x84 && pbCode[5] == 0x00 &&
575         pbCode[6] == 0x00 && pbCode[7] == 0x00 && pbCode[8] == 0x00 &&
576         pbCode[9] == 0x00) {
577         return 10;
578     }
579     if (pbCode[0] == 0x66 && pbCode[1] == 0x66 && pbCode[2] == 0x66 &&
580         pbCode[3] == 0x0F && pbCode[4] == 0x1F && pbCode[5] == 0x84 &&
581         pbCode[6] == 0x00 && pbCode[7] == 0x00 && pbCode[8] == 0x00 &&
582         pbCode[9] == 0x00 && pbCode[10] == 0x00) {
583         return 11;
584     }
585
586     // int 3.
587     if (pbCode[0] == 0xcc) {
588         return 1;
589     }
590     return 0;
591 }
592
593 #endif // DETOURS_X64
594 //////////////////////////////////////////////////////////////////////////// IA64.
595 //
596 #ifdef DETOURS_IA64
597
598 struct _DETOUR_TRAMPOLINE
599 {
600     // On the IA64, a trampoline is used for both incoming and outgoing calls.
601     //
602     // The trampoline contains the following bundles for the outgoing call:
603     //      movl gp=target_gp;
604     //      <relocated target bundle>
605     //      brl  target_code;
606     //
607     // The trampoline contains the following bundles for the incoming call:
```

```
608     //      alloc  r41=ar.pfs, b, 0, 8, 0
609     //      mov    r40=rp
610     //
611     //      adds   r50=0, r39
612     //      adds   r49=0, r38
613     //      adds   r48=0, r37 ;;
614     //
615     //      adds   r47=0, r36
616     //      adds   r46=0, r35
617     //      adds   r45=0, r34
618     //
619     //      adds   r44=0, r33
620     //      adds   r43=0, r32
621     //      adds   r42=0, gp ;;
622     //
623     //      movl   gp=ffffffff`ffffffff ;;
624     //
625     //      brl.call.sptk.few rp=disas!TestCodes+20e0 (00000000`00404ea0) ;;
626     //
627     //      adds   gp=0, r42
628     //      mov    rp=r40, +0 ;;
629     //      mov.i  ar.pfs=r41
630     //
631     //      br.ret.sptk.many rp ;;
632     //
633     // This way, we only have to relocate a single bundle.
634     //
635     // The complicated incoming trampoline is required because we have to
636     // create an additional stack frame so that we save and restore the gp.
637     // We must do this because gp is a caller-saved register, but not saved
638     // if the caller thinks the target is in the same DLL, which changes
639     // when we insert a detour.
640     //
641     DETOUR_IA64_BUNDLE  bMovlTargetGp;  // Bundle which sets target GP
642     BYTE                rbCode[sizeof(DETOUR_IA64_BUNDLE)]; // moved bundle.
643     DETOUR_IA64_BUNDLE  bBrlRemainEip;  // Brl to pbRemain
644     // This must be adjacent to bBranchIslands.
645
646     // Each instruction in the moved bundle could be a IP-relative chk or
    branch or call.
647     // Any such instructions are changed to point to a brl in bBranchIslands.
648     // This must be adjacent to bBrlRemainEip -- see "pbPool".
649     DETOUR_IA64_BUNDLE bBranchIslands[DETOUR_IA64_INSTRUCTIONS_PER_BUNDLE];
650
651     // Target of brl inserted in target function
652     DETOUR_IA64_BUNDLE  bAllocFrame;    // alloc frame
653     DETOUR_IA64_BUNDLE  bSave37to39;    // save r37, r38, r39.
654     DETOUR_IA64_BUNDLE  bSave34to36;    // save r34, r35, r36
```

```
654     DETOUR_IA64_BUNDLE  bSave34to36;    // save r34, r35, r36.
655     DETOUR_IA64_BUNDLE  bSaveGPto33;    // save gp, r32, r33.
656     DETOUR_IA64_BUNDLE  bMovlDetourGp;  // set detour GP.
657     DETOUR_IA64_BUNDLE  bCallDetour;    // call detour.
658     DETOUR_IA64_BUNDLE  bPopFrameGp;    // pop frame and restore gp.
659     DETOUR_IA64_BUNDLE  bReturn;        // return to caller.
660
661     PLABEL_DESCRIPTOR   pldTrampoline;
662
663     BYTE                rbRestore[sizeof(DETOUR_IA64_BUNDLE)]; // original
    target bundle.
664     BYTE                cbRestore;      // size of original target code.
665     BYTE                cbCode;         // size of moved target code.
666     _DETOUR_ALIGN       rAlign[14];     // instruction alignment array.
667     PBYTE               pbRemain;       // first instruction after moved code.
    [free list]
668     PBYTE               pbDetour;       // first instruction of detour
    function.
669     PPLABEL_DESCRIPTOR  ppldDetour;     // [pbDetour,gpDetour]
670     PPLABEL_DESCRIPTOR  ppldTarget;     // [pbTarget,gpDetour]
671 };
672
673 C_ASSERT(sizeof(DETOUR_IA64_BUNDLE) == 16);
674 C_ASSERT(sizeof(_DETOUR_TRAMPOLINE) == 256 +
    DETOUR_IA64_INSTRUCTIONS_PER_BUNDLE * 16);
675
676 enum {
677     SIZE_OF_JMP = sizeof(DETOUR_IA64_BUNDLE)
678 };
```

PART_1_ @苏震 _END

PART_2_ @姜志凯 _BEGIN

Objective-C

```
1  inline PBYTE detour_skip_jmp(PBYTE pPointer, PVOID *ppGlobals)
2  {/*
3
4  intro:跳过import表，或者调试用间接跳转指令
5  param:PBYTE pPointer：指向函数的指针
6  PVOID *ppGlobals：变量，用于接收函数的全局数据的地址
7  return 返回一个跳转之后的地址
8  */
9      PBYTE pGlobals = NULL;
10     PBYTE pbCode = NULL;
11
12     if ( P i n t e r   ! =   N U L L ) {
```

```
    if (pPointer != NULL) {
        PPLABEL_DESCRIPTOR ppld = (PPLABEL_DESCRIPTOR)pPointer;
        pbCode = (PBYTE)ppld->EntryPoint;
        pGlobals = (PBYTE)ppld->GlobalPointer;
    }
    if (ppGlobals != NULL) {
        *ppGlobals = pGlobals;
    }
    if (pbCode == NULL) {
        return NULL;
    }

    DETOUR_IA64_BUNDLE *pb = (DETOUR_IA64_BUNDLE *)pbCode;

    // IA64 Local Import Jumps look like:
    //      addl   r2=ffffffff`ffe021c0, gp ;;
    //      ld8    r2=[r2]
    //      nop.i  0 ;;
    //
    //      ld8    r3=[r2], 8 ;;
    //      ld8    gp=[r2]
    //      mov    b6=r3, +0
    //
    //      nop.m  0
    //      nop.i  0
    //      br.cond.sptk.few b6
    //

    //                      002024000200100b
    if ((pb[0].wide[0] & 0xfffffc000603ffff) == 0x002024000200100b &&
        pb[0].wide[1] == 0x0004000000203008 &&
        pb[1].wide[0] == 0x001014180420180a &&
        pb[1].wide[1] == 0x07000830c0203008 &&
        pb[2].wide[0] == 0x0000000100000010 &&
        pb[2].wide[1] == 0x0080006000000200) {

        ULONG64 offset =
            ((pb[0].wide[0] & 0x0000000001fc0000) >> 18) |  // imm7b
            ((pb[0].wide[0] & 0x000001ff00000000) >> 25) |  // imm9d
            ((pb[0].wide[0] & 0x00000000f8000000) >> 11);   // imm5c
        if (pb[0].wide[0] & 0x0000020000000000) {           // sign
            offset |= 0xfffffffffe00000;
        }
        PBYTE pbTarget = pGlobals + offset;
        DETOUR_TRACE(("%p: potential import jump, target=%p\n", pb,
pbTarget));

        if (detour_is_imported(pbCode, pbTarget) && *(PBYTE*)pbTarget != NULL)
```

```
    {
59              DETOUR_TRACE(("%p: is import jump, label=%p\n", pb, *(PBYTE
    *)pbTarget));
60
61              PPLABEL_DESCRIPTOR ppld = (PPLABEL_DESCRIPTOR)*(PBYTE *)pbTarget;
62              pbCode = (PBYTE)ppld->EntryPoint;
63              pGlobals = (PBYTE)ppld->GlobalPointer;
64              if (ppGlobals != NULL) {
65                  *ppGlobals = pGlobals;
66              }
67          }
68      }
69      return pbCode;
70  }
71
72
73  inline void detour_find_jmp_bounds(PBYTE pbCode,
74                                     PDETOUR_TRAMPOLINE *ppLower,
75                                     PDETOUR_TRAMPOLINE *ppUpper)
76  {/*
77  param:**ppLower 蹦床的低地址
78  param:**ppUpper:蹦床的高地址
79
80  */
81      (void)pbCode;
82      *ppLower = (PDETOUR_TRAMPOLINE)(ULONG_PTR)0x0000000000080000;
83      *ppUpper = (PDETOUR_TRAMPOLINE)(ULONG_PTR)0xfffffffffff80000;
84  }
85
86  inline BOOL detour_does_code_end_function(PBYTE pbCode)
87  {
88      // Routine not needed on IA64.
89      (void)pbCode;
90      return FALSE;
91  }
92
93  inline ULONG detour_is_code_filler(PBYTE pbCode)
94  {
95      // Routine not needed on IA64.
96      (void)pbCode;
97      return 0;
98  }
99
100 #endif // DETOURS_IA64
101
102 #ifdef DETOURS_ARM
103
104 struct  DETOUR_TRAMPOLINE
```

```
{//蹦床结构体
    // A Thumb-2 instruction can be 2 or 4 bytes long.
    BYTE            rbCode[62];     // target code + jmp to pbRemain
    BYTE            cbCode;         // size of moved target code.
    BYTE            cbCodeBreak;    // padding to make debugging easier.
    BYTE            rbRestore[22];  // original target code.
    BYTE            cbRestore;      // size of original target code.
    BYTE            cbRestoreBreak; // padding to make debugging easier.
    _DETOUR_ALIGN   rAlign[8];      // instruction alignment array.
    PBYTE           pbRemain;       // first instruction after moved code.
[free list]
    PBYTE           pbDetour;       // first instruction of detour function.
};

C_ASSERT(sizeof(_DETOUR_TRAMPOLINE) == 104);

enum {
    SIZE_OF_JMP = 8
};

inline PBYTE align4(PBYTE pValue)
{//一种对齐操作
    return (PBYTE)(((ULONG)pValue) & ~(ULONG)3u);
}

inline ULONG fetch_thumb_opcode(PBYTE pbCode)
{
    ULONG Opcode = *(UINT16 *)&pbCode[0];
    if (Opcode >= 0xe800) {
        Opcode = (Opcode << 16) | *(UINT16 *)&pbCode[2];
    }
    return Opcode;
}

inline void write_thumb_opcode(PBYTE &pbCode, ULONG Opcode)
{
    if (Opcode >= 0x10000) {
        *((UINT16*&)pbCode)++ = Opcode >> 16;
    }
    *((UINT16*&)pbCode)++ = (UINT16)Opcode;
}

PBYTE detour_gen_jmp_immediate(PBYTE pbCode, PBYTE *ppPool, PBYTE pbJmpVal)
{
    PBYTE pbLiteral;
    if (ppPool != NULL) {
        *ppPool = *ppPool - 4;
```

```
151         pbLiteral = *ppPool;
152     }
153     else {
154         pbLiteral = align4(pbCode + 6);
155     }
156 
157     *((PBYTE*&)pbLiteral) = DETOURS_PBYTE_TO_PFUNC(pbJmpVal);
158     LONG delta = pbLiteral - align4(pbCode + 4);
159 
160     write_thumb_opcode(pbCode, 0xf8dff000 | delta);     // LDR PC,[PC+n]
161 
162     if (ppPool == NULL) {
163         if (((ULONG)pbCode & 2) != 0) {
164             write_thumb_opcode(pbCode, 0xdefe);         // BREAK
165         }
166         pbCode += 4;
167     }
168     return pbCode;
169 }
170 
171 inline PBYTE detour_gen_brk(PBYTE pbCode, PBYTE pbLimit)
172 {
173     while (pbCode < pbLimit) {
174         write_thumb_opcode(pbCode, 0xdefe);
175     }
176     return pbCode;
177 }
178 
179 inline PBYTE detour_skip_jmp(PBYTE pbCode, PVOID *ppGlobals)
180 {
181     if (pbCode == NULL) {
182         return NULL;
183     }
184     if (ppGlobals != NULL) {
185         *ppGlobals = NULL;
186     }
187 
188     // Skip over the import jump if there is one.
189     pbCode = (PBYTE)DETOURS_PFUNC_TO_PBYTE(pbCode);
190     ULONG Opcode = fetch_thumb_opcode(pbCode);
191 
192     if ((Opcode & 0xfbf08f00) == 0xf2400c00) {          // movw r12,#xxxx
193         ULONG Opcode2 = fetch_thumb_opcode(pbCode+4);
194 
195         if ((Opcode2 & 0xfbf08f00) == 0xf2c00c00) {      // movt r12,#xxxx
196             ULONG Opcode3 = fetch_thumb_opcode(pbCode+8);
197             if (Opcode3 == 0xf8dcf000) {                 // ldr  pc,[r12]
198                 PBYTE pbTarget = (PBYTE)(((Opcode2 << 12) & 0xf7000000) |
```

```
                                        ((Opcode2 <<  1) & 0x08000000) |
                                        ((Opcode2 << 16) & 0x00ff0000) |
                                        ((Opcode  >>  4) & 0x0000f700) |
                                        ((Opcode  >> 15) & 0x00000800) |
                                        ((Opcode  >>  0) & 0x000000ff));
                if (detour_is_imported(pbCode, pbTarget)) {
                    PBYTE pbNew = *(PBYTE *)pbTarget;
                    pbNew = DETOURS_PFUNC_TO_PBYTE(pbNew);
                    DETOUR_TRACE(("%p->%p: skipped over import table.\n",
pbCode, pbNew));
                    return pbNew;
                }
            }
        }
    }
    return pbCode;
}

inline void detour_find_jmp_bounds(PBYTE pbCode,
                                   PDETOUR_TRAMPOLINE *ppLower,
                                   PDETOUR_TRAMPOLINE *ppUpper)
{
    // We have to place trampolines within +/- 2GB of code.
    ULONG_PTR lo = detour_2gb_below((ULONG_PTR)pbCode);
    ULONG_PTR hi = detour_2gb_above((ULONG_PTR)pbCode);
    DETOUR_TRACE(("[%p..%p..%p]\n", (PVOID)lo, pbCode, (PVOID)hi));

    *ppLower = (PDETOUR_TRAMPOLINE)lo;
    *ppUpper = (PDETOUR_TRAMPOLINE)hi;
}


inline BOOL detour_does_code_end_function(PBYTE pbCode)
{
    ULONG Opcode = fetch_thumb_opcode(pbCode);
    if ((Opcode & 0xffffff87) == 0x4700 ||          // bx <reg>
        (Opcode & 0xf800d000) == 0xf0009000) {      // b <imm20>
        return TRUE;
    }
    if ((Opcode & 0xffff8000) == 0xe8bd8000) {      // pop {...,pc}
        __debugbreak();
        return TRUE;
    }
    if ((Opcode & 0xffffff00) == 0x0000bd00) {      // pop {...,pc}
        __debugbreak();
        return TRUE;
    }
    return FALSE;
```

```
    return FALSE;
}

inline ULONG detour_is_code_filler(PBYTE pbCode)
{
    if (pbCode[0] == 0x00 && pbCode[1] == 0xbf) { // nop.
        return 2;
    }
    if (pbCode[0] == 0x00 && pbCode[1] == 0x00) { // zero-filled padding.
        return 2;
    }
    return 0;
}

#endif // DETOURS_ARM

#ifdef DETOURS_ARM64

struct _DETOUR_TRAMPOLINE
{
    // An ARM64 instruction is 4 bytes long.
    //
    // The overwrite is always composed of 3 instructions (12 bytes) which perform an indirect jump
    // using _DETOUR_TRAMPOLINE::pbDetour as the address holding the target location.
    //
    // Copied instructions can expand.
    //
    // The scheme using MovImmediate can cause an instruction
    // to grow as much as 6 times.
    // That would be Bcc or Tbz with a large address space:
    //   4 instructions to form immediate
    //   inverted tbz/bcc
    //   br
    //
    // An expansion of 4 is not uncommon -- bl/blr and small address space:
    //   3 instructions to form immediate
    //   br or brl
    //
    // A theoretical maximum for rbCode is thefore 4*4*6 + 16 = 112 (another 16 for jmp to pbRemain).
    //
    // With literals, the maximum expansion is 5, including the literals: 4*4*5 + 16 = 96.
    //
    // The number is rounded up to 128. m_rbScratchDst should match this.
    //
```

```
    BYTE            rbCode[128];    // target code + jmp to pbRemain
    BYTE            cbCode;         // size of moved target code.
    BYTE            cbCodeBreak[3]; // padding to make debugging easier.
    BYTE            rbRestore[24];  // original target code.
    BYTE            cbRestore;      // size of original target code.
    BYTE            cbRestoreBreak[3]; // padding to make debugging easier.
    _DETOUR_ALIGN   rAlign[8];      // instruction alignment array.
    PBYTE           pbRemain;       // first instruction after moved code.
[free list]
    PBYTE           pbDetour;       // first instruction of detour function.
};

C_ASSERT(sizeof(_DETOUR_TRAMPOLINE) == 184);

enum {
    SIZE_OF_JMP = 12
};

inline ULONG fetch_opcode(PBYTE pbCode)
{
    return *(ULONG *)pbCode;
}

inline void write_opcode(PBYTE &pbCode, ULONG Opcode)
{
    *(ULONG *)pbCode = Opcode;
    pbCode += 4;
}

struct ARM64_INDIRECT_JMP {
    struct {
        ULONG Rd : 5;
        ULONG immhi : 19;
        ULONG iop : 5;
        ULONG immlo : 2;
        ULONG op : 1;
    } ardp;

    struct {
        ULONG Rt : 5;
        ULONG Rn : 5;
        ULONG imm : 12;
        ULONG opc : 2;
        ULONG iop1 : 2;
        ULONG V : 1;
        ULONG iop2 : 3;
        ULONG size : 2;
    } ldr;
```

```
336
337     ULONG br;
338 };
339
340 #pragma warning(push)
341 #pragma warning(disable:4201)
342
343 union ARM64_INDIRECT_IMM {
344     struct {
345         ULONG64 pad : 12;
346         ULONG64 adrp_immlo : 2;
347         ULONG64 adrp_immhi : 19;
348     };
349
350     LONG64 value;
351 };
352
353 #pragma warning(pop)
354
355 PBYTE detour_gen_jmp_indirect(BYTE *pbCode, ULONG64 *pbJmpVal)
356 {
357     // adrp x17, [jmpval]
358     // ldr x17, [x17, jmpval]
359     // br x17
360
361     struct ARM64_INDIRECT_JMP *pIndJmp;
362     union ARM64_INDIRECT_IMM jmpIndAddr;
363
364     jmpIndAddr.value = (((LONG64)pbJmpVal) & 0xFFFFFFFFFFFFF000) -
365                        (((LONG64)pbCode) & 0xFFFFFFFFFFFFF000);
366
367     pIndJmp = (struct ARM64_INDIRECT_JMP *)pbCode;
368     pbCode = (BYTE *)(pIndJmp + 1);
369
370     pIndJmp->ardp.Rd = 17;
371     pIndJmp->ardp.immhi = jmpIndAddr.adrp_immhi;
372     pIndJmp->ardp.iop = 0x10;
373     pIndJmp->ardp.immlo = jmpIndAddr.adrp_immlo;
374     pIndJmp->ardp.op = 1;
375
376     pIndJmp->ldr.Rt = 17;
377     pIndJmp->ldr.Rn = 17;
378     pIndJmp->ldr.imm = (((ULONG64)pbJmpVal) & 0xFFF) / 8;
379     pIndJmp->ldr.opc = 1;
380     pIndJmp->ldr.iop1 = 1;
381     pIndJmp->ldr.V = 0;
382     pIndJmp->ldr.iop2 = 7;
383     pIndJmp->ldr.size = 3;
```

```
383     pIndJmp->ldr.size = 3;
384
385     pIndJmp->br = 0xD61F0220;
386
387     return pbCode;
388 }
389
390 PBYTE detour_gen_jmp_immediate(PBYTE pbCode, PBYTE *ppPool, PBYTE pbJmpVal)
391 {
392     PBYTE pbLiteral;
393     if (ppPool != NULL) {
394         *ppPool = *ppPool - 8;
395         pbLiteral = *ppPool;
396     }
397     else {
398         pbLiteral = pbCode + 8;
399     }
400
401     *((PBYTE*&)pbLiteral) = pbJmpVal;
402     LONG delta = (LONG)(pbLiteral - pbCode);
403
404     write_opcode(pbCode, 0x58000011 | ((delta / 4) << 5));  // LDR X17,[PC+n]
405     write_opcode(pbCode, 0xd61f0000 | (17 << 5));           // BR X17
406
407     if (ppPool == NULL) {
408         pbCode += 8;
409     }
410     return pbCode;
411 }
412 inline PBYTE detour_gen_brk(PBYTE pbCode, PBYTE pbLimit)
413 {
414     while (pbCode < pbLimit) {
415         write_opcode(pbCode, 0xd4100000 | (0xf000 << 5));
416     }
417     return pbCode;
418 }
419
420 inline INT64 detour_sign_extend(UINT64 value, UINT bits)
421 {
422     const UINT left = 64 - bits;
423     const INT64 m1 = -1;
424     const INT64 wide = (INT64)(value << left);
425     const INT64 sign = (wide < 0) ? (m1 << left) : 0;
426     return value | sign;
427 }
428
429 inline PBYTE detour_skip_jmp(PBYTE pbCode, PVOID *ppGlobals)
430 {
```

```
431     if (pbCode == NULL) {
432         return NULL;
433     }
434     if (ppGlobals != NULL) {
435         *ppGlobals = NULL;
436     }
437 
438     // Skip over the import jump if there is one.
439     pbCode = (PBYTE)pbCode;
440     ULONG Opcode = fetch_opcode(pbCode);
441 
442     if ((Opcode & 0x9f00001f) == 0x90000010) {           // adrp  x16, IAT
443         ULONG Opcode2 = fetch_opcode(pbCode + 4);
444 
445         if ((Opcode2 & 0xffe003ff) == 0xf9400210) {      // ldr   x16, [x16,
    IAT]
446             ULONG Opcode3 = fetch_opcode(pbCode + 8);
447 
448             if (Opcode3 == 0xd61f0200) {                 // br    x16
449 
450 /* https://static.docs.arm.com/ddi0487/bb/DDI0487B_b_armv8_arm.pdf
451     The ADRP instruction shifts a signed, 21-bit immediate left by 12 bits,
    adds it to the value of the program counter with
452     the bottom 12 bits cleared to zero, and then writes the result to a
    general-purpose register. This permits the
453     calculation of the address at a 4KB aligned memory region. In conjunction
    with an ADD (immediate) instruction, or
454     a Load/Store instruction with a 12-bit immediate offset, this allows for
    the calculation of, or access to, any address
455     within +/- 4GB of the current PC.
456 
457 PC-rel. addressing
458     This section describes the encoding of the PC-rel. addressing instruction
    class. The encodings in this section are
459     decoded from Data Processing -- Immediate on page C4-226.
460     Add/subtract (immediate)
461     This section describes the encoding of the Add/subtract (immediate)
    instruction class. The encodings in this section
462     are decoded from Data Processing -- Immediate on page C4-226.
463     Decode fields
464     Instruction page
465     op
466     0 ADR
467     1 ADRP
468 
469 C6.2.10 ADRP
470     Form PC-relative address to 4KB page adds an immediate value that is
    shifted left by 12 bits, to the PC value to
```

```
471     form a PC-relative address, with the bottom 12 bits masked out, and writes
    the result to the destination register.
472     ADRP <Xd>, <label>
473     imm = SignExtend(immhi:immlo:Zeros(12), 64);
474
475     31  30 29 28 27 26 25 24 23 5    4 0
476     1   immlo  1  0  0  0  0  immhi  Rd
477          9            0
478
479 Rd is hardcoded as 0x10 above.
480 Immediate is 21 signed bits split into 2 bits and 19 bits, and is scaled by
    4K.
481 */
482                 UINT64 const pageLow2 = (Opcode >> 29) & 3;
483                 UINT64 const pageHigh19 = (Opcode >> 5) & ~(~0ui64 << 19);
484                 INT64 const page = detour_sign_extend((pageHigh19 << 2) |
    pageLow2, 21) << 12;
485
486 /* https://static.docs.arm.com/ddi0487/bb/DDI0487B_b_armv8_arm.pdf
487
488     C6.2.101 LDR (immediate)
489     Load Register (immediate) loads a word or doubleword from memory and
    writes it to a register. The address that is
490     used for the load is calculated from a base register and an immediate
    offset.
491     The Unsigned offset variant scales the immediate offset value by the size
    of the value accessed before adding it
492     to the base register value.
493
494 Unsigned offset
495 64-bit variant Applies when size == 11.
496     31 30 29 28  27 26 25 24  23 22  21   10   9 5   4 0
497      1  x  1  1   1  0  0  1   0  1  imm12      Rn    Rt
498           F            9       4             200    10
499
500 That is, two low 5 bit fields are registers, hardcoded as 0x10 and 0x10 << 5
    above,
501 then unsigned size-unscaled (8) 12-bit offset, then opcode bits 0xF94.
502 */
503                 UINT64 const offset = ((Opcode2 >> 10) & ~(~0ui64 << 12)) <<
    3;
504
505                 PBYTE const pbTarget = (PBYTE)((ULONG64)pbCode &
    0xfffffffffffff000ULL) + page + offset;
506
507                 if (detour_is_imported(pbCode, pbTarget)) {
508                     PBYTE pbNew = *(PBYTE *)pbTarget;
```

```
509                     DETOUR_TRACE(("%p->%p: skipped over import table.\n",
    pbCode, pbNew));
510                     return pbNew;
511                 }
512             }
513         }
514     }
515     return pbCode;
516 }
517 inline void detour_find_jmp_bounds(PBYTE pbCode,
518                                    PDETOUR_TRAMPOLINE *ppLower,
519                                    PDETOUR_TRAMPOLINE *ppUpper)
520 {
521     // The encoding used by detour_gen_jmp_indirect actually enables a
522     // displacement of +/- 4GiB. In the future, this could be changed to
523     // reflect that. For now, just reuse the x86 logic which is plenty.
524 
525     ULONG_PTR lo = detour_2gb_below((ULONG_PTR)pbCode);
526     ULONG_PTR hi = detour_2gb_above((ULONG_PTR)pbCode);
527     DETOUR_TRACE(("[%p..%p..%p]\n", (PVOID)lo, pbCode, (PVOID)hi));
528 
529     *ppLower = (PDETOUR_TRAMPOLINE)lo;
530     *ppUpper = (PDETOUR_TRAMPOLINE)hi;
531 }
532 
533 inline BOOL detour_does_code_end_function(PBYTE pbCode)
534 {/*
535 intro: 该函数用于判断是否detour起作用
536 
537 */
538     ULONG Opcode = fetch_opcode(pbCode);
539     if ((Opcode & 0xfffffc1f) == 0xd65f0000 ||      // br <reg>
540         (Opcode & 0xfc000000) == 0x14000000) {      // b <imm26>
541         return TRUE;
542     }
543     return FALSE;
544 }
545 
546 inline ULONG detour_is_code_filler(PBYTE pbCode)
547 {/*
548 intro:检查是否已经填充Detour代码
549 return:没有填充返回4，填充返回0
550 */
551     if (*(ULONG *)pbCode == 0xd503201f) {   // nop.
552         return 4;
553     }
554     if (*(ULONG *)pbCode == 0x00000000) {   // zero-filled padding.
555         return 4;
```

```
556     }
557     return 0;
558 }
559
560 #endif // DETOURS_ARM64
561 ////////////////////////////////////////////// Trampoline Memory Management.
562 //
563 struct DETOUR_REGION
564 {
565     ULONG               dwSignature;
566     DETOUR_REGION *     pNext;  // Next region in list of regions.
567     DETOUR_TRAMPOLINE * pFree;  // List of free trampolines in this region.
568 };
569 typedef DETOUR_REGION * PDETOUR_REGION;
570
571 const ULONG DETOUR_REGION_SIGNATURE = 'Rrtd';
572 const ULONG DETOUR_REGION_SIZE = 0x10000;
573 const ULONG DETOUR_TRAMPOLINES_PER_REGION = (DETOUR_REGION_SIZE
574                                              / sizeof(DETOUR_TRAMPOLINE)) - 1;
575 static PDETOUR_REGION s_pRegions = NULL;            // List of all regions.
576 static PDETOUR_REGION s_pRegion = NULL;             // Default region.
577
578 static DWORD detour_writable_trampoline_regions()
579 {/*intro: 将所有区域标记为可写文件*/
580     // Mark all of the regions as writable.
581     for (PDETOUR_REGION pRegion = s_pRegions; pRegion != NULL; pRegion =
    pRegion->pNext) {
582         DWORD dwOld;
583         if (!VirtualProtect(pRegion, DETOUR_REGION_SIZE,
    PAGE_EXECUTE_READWRITE, &dwOld)) {
584             return GetLastError();
585         }
586     }
587     return NO_ERROR;
588 }
589
590 static void detour_runnable_trampoline_regions()
591 {/*
592 intro: 将所有区域标记为可执行文件
593 */
594     HANDLE hProcess = GetCurrentProcess();
595
596     // Mark all of the regions as executable.
597     for (PDETOUR_REGION pRegion = s_pRegions; pRegion != NULL; pRegion =
    pRegion->pNext) {
598         DWORD dwOld;
599         VirtualProtect(pRegion, DETOUR_REGION_SIZE, PAGE_EXECUTE_READ,
    &dwOld);
```

```
        FlushInstructionCache(hProcess, pRegion, DETOUR_REGION_SIZE);
    }
}

static PBYTE detour_alloc_round_down_to_region(PBYTE pbTry)
{/*
intro&return:WinXP64 返回与 32 位应用程序不一致的区域的低地址。
*/
    // WinXP64 returns free areas that aren't REGION aligned to 32-bit applications.
    ULONG_PTR extra = ((ULONG_PTR)pbTry) & (DETOUR_REGION_SIZE - 1);
    if (extra != 0) {
        pbTry -= extra;
    }
    return pbTry;
}

static PBYTE detour_alloc_round_up_to_region(PBYTE pbTry)
{/*
intro&return: WinXP64 返回与 32 位应用程序不一致的区域的高地址。
param:pbTry 传入区域
*/
    // WinXP64 returns free areas that aren't REGION aligned to 32-bit applications.
    ULONG_PTR extra = ((ULONG_PTR)pbTry) & (DETOUR_REGION_SIZE - 1);
    if (extra != 0) {
        ULONG_PTR adjust = DETOUR_REGION_SIZE - extra;
        pbTry += adjust;
    }
    return pbTry;
}

// Starting at pbLo, try to allocate a memory region, continue until pbHi.
//从 pbLo 开始，尝试分配内存区域，继续直到 pbHi。
static PVOID detour_alloc_region_from_lo(PBYTE pbLo, PBYTE pbHi)
{/*
intro: 从 pbLo 开始，尝试分配内存区域，继续直到 pbHi。
param:pbLo -> pbHi 低地址-高地址
无返回值
*/
    PBYTE pbTry = detour_alloc_round_up_to_region(pbLo);

    DETOUR_TRACE((" Looking for free region in %p..%p from %p:\n", pbLo, pbHi, pbTry));

    for (; pbTry < pbHi;) {
        MEMORY_BASIC_INFORMATION mbi;
```

```
644
645         if (pbTry >= s_pSystemRegionLowerBound && pbTry <=
    s_pSystemRegionUpperBound) {
646             // Skip region reserved for system DLLs, but preserve address
    space entropy.
647             pbTry += 0x08000000;
648             continue;
649         }
650
651         ZeroMemory(&mbi, sizeof(mbi));
652         if (!VirtualQuery(pbTry, &mbi, sizeof(mbi))) {
653             break;
654         }
655
656         DETOUR_TRACE("  Try %p => %p..%p %6lx\n",
657                      pbTry,
658                      mbi.BaseAddress,
659                      (PBYTE)mbi.BaseAddress + mbi.RegionSize - 1,
660                      mbi.State));
661
662         if (mbi.State == MEM_FREE && mbi.RegionSize >= DETOUR_REGION_SIZE) {
663
664             PVOID pv = VirtualAlloc(pbTry,
665                                     DETOUR_REGION_SIZE,
666                                     MEM_COMMIT|MEM_RESERVE,
667                                     PAGE_EXECUTE_READWRITE);
668             if (pv != NULL) {
669                 return pv;
670             }
671             else if (GetLastError() == ERROR_DYNAMIC_CODE_BLOCKED) {
672                 return NULL;
673             }
674             pbTry += DETOUR_REGION_SIZE;
675         }
676         else {
677             pbTry = detour_alloc_round_up_to_region((PBYTE)mbi.BaseAddress +
    mbi.RegionSize);
678         }
679     }
680     return NULL;
681 }
```

PART_2_ @姜志凯 _END

PART_3_ @臧国盛 _BEGIN

C++

```
// Starting at pbHi, try to allocate a memory region, continue until pbLo.
//分配一个内存区域，pbHi? pbLo?
//PVOID相当于 void*
//有关指针类型void*:
//所有指针都是一个32位二进制数（32位系统下），这个意义上说所有指针都是一样的，它们的大小一样
//用于指向内存中的某处地址，然而指针为什么要有类型之分呢？答案是指针偏移。例如p为一个指针，
//它指向内存某处地址，那么p+1（或者写p[1]）是什么意思呢？答案是p指向地址的后面那个地址，那么
//后面多少呢？这就看指针类型了，假如它是字符指针，那么就是后面一个字节，假如它是整型指针，那就
//是后面第四字节，假如它是一个结构体，那就是后面sizeof(结构体)个字节。可以说，指针有类型之分，
//完全就是为了计算地址偏移。这一区别到了汇编级就没有分别了，汇编级不存在指针类型，只有指针偏移
//数。
//那么void 指针是啥呢？答案是无类型指针。干啥用呢？它只是一个地址指向，从不用计算偏移，
//它只能指向一整块内存，只能通过它来访问这块内存，不能用偏移访问（p 1，p[1]等，千万不要用在
//void指针上）。它的好处是什么呢？答案是不用强制转换，任何类型指针都可直接赋值给一个void
//指针，而不用转换。
static PVOID detour_alloc_region_from_hi(PBYTE pbLo, PBYTE pbHi)
{
    PBYTE pbTry = detour_alloc_round_down_to_region(pbHi - DETOUR_REGION_SIZE);
    //在pbLo中查找自由区域
    DETOUR_TRACE((" Looking for free region in %p..%p from %p:\n", pbLo, pbHi, pbTry));

    for (; pbTry > pbLo;) {
        MEMORY_BASIC_INFORMATION mbi;//包含有关进程的虚拟地址空间中的一系列信息，VirtualQuery 和 VirtualQueryEx 函数使用此结构。

        DETOUR_TRACE(("  Try %p\n", pbTry));
        if (pbTry >= s_pSystemRegionLowerBound && pbTry <= s_pSystemRegionUpperBound) {
            // Skip region reserved for system DLLs, but preserve address space entropy.
            //跳过为系统DLL保留的区域，但保留地址空间熵。
            pbTry -= 0x08000000;//为什么是0x08000000?
            continue;
        }

        ZeroMemory(&mbi, sizeof(mbi));//用0来填充一块内存区域
        if (!VirtualQuery(pbTry, &mbi, sizeof(mbi))) {//检索有关调用进程的虚拟地址
```

```
    空间中一系列页的信息。返回值是信息缓冲区中返回的实际字节数。
35              break;
36          }
37
38          DETOUR_TRACE(("  Try %p => %p..%p %6lx\n",
39                        pbTry,
40                        mbi.BaseAddress,
41                        (PBYTE)mbi.BaseAddress + mbi.RegionSize - 1,
42                        mbi.State));
43
44          if (mbi.State == MEM_FREE && mbi.RegionSize >= DETOUR_REGION_SIZE) {
45              //申请内存空间
46              //如果调用成功,返回分配的首地址
47              //调用失败,返回NULL 可以通过GetLastError函数来获取错误信息
48
49              PVOID pv = VirtualAlloc(pbTry,//地址
50                                      DETOUR_REGION_SIZE,//大小
51                                      MEM_COMMIT|MEM_RESERVE,//分配类型
52                                      PAGE_EXECUTE_READWRITE);//保护方式
53              if (pv != NULL) {
54                  return pv;
55              }
56              else if (GetLastError() == ERROR_DYNAMIC_CODE_BLOCKED) {
57                  return NULL;
58              }
59              pbTry -= DETOUR_REGION_SIZE;
60          }
61          else {
62              pbTry =
    detour_alloc_round_down_to_region((PBYTE)mbi.AllocationBase
63                                                      - DETOUR_REGION_SIZE);
64          }
65      }
66      return NULL;
67  }
68  //给蹦床函数分配空间
69  //PDETOUR_TRAMPOLINE自定义的蹦床结构
70  static PVOID detour_alloc_trampoline_allocate_new(PBYTE pbTarget,
71                                                    PDETOUR_TRAMPOLINE pLo,
72                                                    PDETOUR_TRAMPOLINE pHi)
73  {
74      PVOID pbTry = NULL;
75
76      // NB: We must always also start the search at an offset from pbTarget
77      //     in order to maintain ASLR entropy.
78      //为了保持ASLR熵，我们还必须始终从pbTarget的偏移量开始搜索。
79  #if defined(DETOURS_64BIT)
80      // Try looking 1GB below or lower.
```

```
80      // Try looking 1GB below or lower.
81      //0x40000000，即2的30次方，即1G
82      if (pbTry == NULL && pbTarget > (PBYTE)0x40000000) {
83          pbTry = detour_alloc_region_from_hi((PBYTE)pLo, pbTarget -
    0x40000000);
84      }
85      // Try looking 1GB above or higher.
86      if (pbTry == NULL && pbTarget < (PBYTE)0xffffffff40000000) {
87          pbTry = detour_alloc_region_from_lo(pbTarget + 0x40000000,
    (PBYTE)pHi);
88      }
89      // Try looking 1GB below or higher.
90      if (pbTry == NULL && pbTarget > (PBYTE)0x40000000) {
91          pbTry = detour_alloc_region_from_lo(pbTarget - 0x40000000, pbTarget);
92      }
93      // Try looking 1GB above or lower.
94      if (pbTry == NULL && pbTarget < (PBYTE)0xffffffff40000000) {
95          pbTry = detour_alloc_region_from_hi(pbTarget, pbTarget + 0x40000000);
96      }
97  #endif
98
99      // Try anything below.
100     if (pbTry == NULL) {
101         pbTry = detour_alloc_region_from_hi((PBYTE)pLo, pbTarget);
102     }
103     // try anything above.
104     if (pbTry == NULL) {
105         pbTry = detour_alloc_region_from_lo(pbTarget, (PBYTE)pHi);
106     }
107
108     return pbTry;
109 }
110 //在给定地址附近分配一个可执行区域。
111 //pbTarget:开始搜索可分配空间的地址
112 //pcbAllocatedSize:接收以字节为单位的已分配区域大小的变量
113 //如果返回成功，返回所分配的页面区域的基址;否则,返回NULL。
114 //要释放由detourallocateregionwithinjumpounds分配的区域，使用VirtualFree函数。
115 PVOID WINAPI DetourAllocateRegionWithinJumpBounds(_In_ LPCVOID pbTarget,
116                                                    _Out_ PDWORD
    pcbAllocatedSize)
117 {
118     PDETOUR_TRAMPOLINE pLo;
119     PDETOUR_TRAMPOLINE pHi;
120     detour_find_jmp_bounds((PBYTE)pbTarget, &pLo, &pHi);
121
122     PVOID pbNewlyAllocated =
123         detour_alloc_trampoline_allocate_new((PBYTE)pbTarget, pLo, pHi);
124     if (pbNewlyAllocated == NULL) {
```

```
125         DETOUR_TRACE(("Couldn't find available memory region!\n"));
126         *pcbAllocatedSize = 0;
127         return NULL;
128     }
129 
130     *pcbAllocatedSize = DETOUR_REGION_SIZE;
131     return pbNewlyAllocated;
132 }
133 //申请蹦床
134 //在目标函数+/-2GB范围内放置蹦床
135 //确保存在默认区域。
136 //首先检查默认区域是否有有效的空闲块。
137 //然后检查现有区域是否有有效的空闲块。
138 //需要分配一个新的区域
139 //将pbTarget四舍五入到64KB块。
140 static PDETOUR_TRAMPOLINE detour_alloc_trampoline(PBYTE pbTarget)
141 {
142     // We have to place trampolines within +/- 2GB of target.
143 
144     PDETOUR_TRAMPOLINE pLo;
145     PDETOUR_TRAMPOLINE pHi;
146 
147     detour_find_jmp_bounds(pbTarget, &pLo, &pHi);
148 
149     PDETOUR_TRAMPOLINE pTrampoline = NULL;
150     //确保存在默认区域。
151     // Insure that there is a default region.
152     if (s_pRegion == NULL && s_pRegions != NULL) {
153         s_pRegion = s_pRegions;
154     }
155     //首先检查默认区域是否有有效的空闲块。
156     // First check the default region for an valid free block.
157     if (s_pRegion != NULL && s_pRegion->pFree != NULL &&
158         s_pRegion->pFree >= pLo && s_pRegion->pFree <= pHi) {
159 
160       found_region:
161         pTrampoline = s_pRegion->pFree;
162         // do a last sanity check on region.
163         if (pTrampoline < pLo || pTrampoline > pHi) {
164             return NULL;
165         }
166         s_pRegion->pFree = (PDETOUR_TRAMPOLINE)pTrampoline->pbRemain;
167         memset(pTrampoline, 0xcc, sizeof(*pTrampoline));
168         return pTrampoline;
169     }
170     //然后检查现有区域是否有有效的空闲块。
171     // Then check the existing regions for a valid free block.
172     for (s_pRegion = s_pRegions; s_pRegion != NULL; s_pRegion = s_pRegion-
```

```
>pNext) {
173         if (s_pRegion != NULL && s_pRegion->pFree != NULL &&
174             s_pRegion->pFree >= pLo && s_pRegion->pFree <= pHi) {
175             goto found_region;
176         }
177     }
178     //我们需要分配一个新的区域
179     // We need to allocate a new region.
180     //将pbTarget四舍五入到64KB块。
181     // Round pbTarget down to 64KB block.
182     pbTarget = pbTarget - (PtrToUlong(pbTarget) & 0xffff);
183
184     PVOID pbNewlyAllocated =
185         detour_alloc_trampoline_allocate_new(pbTarget, pLo, pHi);
186     if (pbNewlyAllocated != NULL) {
187         s_pRegion = (DETOUR_REGION*)pbNewlyAllocated;
188         s_pRegion->dwSignature = DETOUR_REGION_SIGNATURE;
189         s_pRegion->pFree = NULL;
190         s_pRegion->pNext = s_pRegions;
191         s_pRegions = s_pRegion;
192         DETOUR_TRACE(("  Allocated region %p..%p\n\n",
193                       s_pRegion, ((PBYTE)s_pRegion) + DETOUR_REGION_SIZE -
    1));
194         //把所有东西都放在空列表上，除了第一个蹦床。
195         // Put everything but the first trampoline on the free list.
196         PBYTE pFree = NULL;
197         pTrampoline = ((PDETOUR_TRAMPOLINE)s_pRegion) + 1;
198         for (int i = DETOUR_TRAMPOLINES_PER_REGION - 1; i > 1; i--) {
199             pTrampoline[i].pbRemain = pFree;
200             pFree = (PBYTE)&pTrampoline[i];
201         }
202         s_pRegion->pFree = (PDETOUR_TRAMPOLINE)pFree;
203         goto found_region;
204     }
205
206     DETOUR_TRACE(("Couldn't find available memory region!\n"));
207     return NULL;
208 }
209 //释放蹦床
210 //memset() 的作用是在一段内存块中填充某个给定的值。因为它只能填充一个值，所以该函数的初
    始化为
211 //原始初始化，无法将变量初始化为程序中需要的数据。用memset初始化完后，后面程序中再向该内
    存空间
212 //中存放需要的数据。
213 static void detour_free_trampoline(PDETOUR_TRAMPOLINE pTrampoline)
214 {
215     PDETOUR_REGION pRegion = (PDETOUR_REGION)
216         ((ULONG_PTR)pTrampoline & (ULONG_PTR)0xffff);
```

```
216         ((ULONG_PTR)pTrampoline & ~(ULONG_PTR)0xFFFF);
217
218     memset(pTrampoline, 0, sizeof(*pTrampoline));
219     pTrampoline->pbRemain = (PBYTE)pRegion->pFree;
220     pRegion->pFree = pTrampoline;
221 }
222 //判断某个内存区域是否为空
223 static BOOL detour_is_region_empty(PDETOUR_REGION pRegion)
224 {
225     // Stop if the region isn't a region (this would be bad).
226     if (pRegion->dwSignature != DETOUR_REGION_SIGNATURE) {
227         return FALSE;
228     }
229
230     PBYTE pbRegionBeg = (PBYTE)pRegion;
231     PBYTE pbRegionLim  = pbRegionBeg + DETOUR_REGION_SIZE;
232
233     // Stop if any of the trampolines aren't free.
234     PDETOUR_TRAMPOLINE pTrampoline = ((PDETOUR_TRAMPOLINE)pRegion) + 1;
235     for (int i = 0; i < DETOUR_TRAMPOLINES_PER_REGION; i++) {
236         if (pTrampoline[i].pbRemain != NULL &&
237             (pTrampoline[i].pbRemain < pbRegionBeg ||
238              pTrampoline[i].pbRemain >= pbRegionLim)) {
239             return FALSE;
240         }
241     }
242
243     // OK, the region is empty.
244     return TRUE;
245 }
246 //释放未使用的蹦床空间
247 static void detour_free_unused_trampoline_regions()
248 {
249     PDETOUR_REGION *ppRegionBase = &s_pRegions;
250     PDETOUR_REGION pRegion = s_pRegions;
251
252     while (pRegion != NULL) {
253         if (detour_is_region_empty(pRegion)) {
254             *ppRegionBase = pRegion->pNext;
255
256             VirtualFree(pRegion, 0, MEM_RELEASE);
257             s_pRegion = NULL;
258         }
259         else {
260             ppRegionBase = &pRegion->pNext;
261         }
262         pRegion = *ppRegionBase;
263     }
```

```
264 }
265
266 ////////////////////////////////////////////////////////////// Transaction Structs.
267 //
268 //detour线程结构定义
269 struct DetourThread
270 {
271     DetourThread *      pNext;
272     HANDLE              hThread;
273 };
274 //detour操作结构定义
275 struct DetourOperation
276 {
277     DetourOperation *   pNext;
278     BOOL                fIsRemove;
279     PBYTE *             ppbPointer;
280     PBYTE               pbTarget;
281     PDETOUR_TRAMPOLINE  pTrampoline;
282     ULONG               dwPerm;
283 };
284
285 static BOOL                 s_fIgnoreTooSmall       = FALSE;
286 static BOOL                 s_fRetainRegions        = FALSE;
287
288 static LONG                 s_nPendingThreadId      = 0; // Thread owning
    pending transaction.
289 static LONG                 s_nPendingError         = NO_ERROR;
290 static PVOID *              s_ppPendingError        = NULL;
291 static DetourThread *       s_pPendingThreads       = NULL;
292 static DetourOperation *    s_pPendingOperations    = NULL;
293
294 ////////////////////////////////////////////////////////////////////////////
295 //
296 //用于查找目标函数的api
297 PVOID WINAPI DetourCodeFromPointer(_In_ PVOID pPointer,
298                                    _Out_opt_ PVOID *ppGlobals)
299 {/*
300 intro:DetourCodeFromPointer返回实际目标函数的地址，而不是跳转语句。
301 param:PVOID pPointer 指向函数的指针。
302 param:PVOID *ppGlobals 用于接收函数的全局数据的地址。
303 return detour_skip_jmp 返回一个指向实现函数指针的代码的指针。
304 */
305     return detour_skip_jmp((PBYTE)pPointer, ppGlobals);
306 }
307
308 ////////////////////////////////////////////////////////////// Transaction APIs.
309 //
310 //在附加或分离单独的绕道功能失败时启用或禁用事务中止。
```

```
//如果Detours之前忽略了目标函数太小而不能绕道，则返回TRUE;否则,返回FALSE。
//Ignore:指定是否忽略过小而不能绕道的函数。如果该参数设置为TRUE，则遇到这些函数时将被忽略。
//如果将此参数设置为FALSE，则遇到太小而不能绕道的函数将导致DetourTransactionCommit失败。
BOOL WINAPI DetourSetIgnoreTooSmall(_In_ BOOL fIgnore)
{
    BOOL fPrevious = s_fIgnoreTooSmall;
    s_fIgnoreTooSmall = fIgnore;
    return fPrevious;
}
//强制Detours保留Trampline分配区域，即使已经释放了蹦床。
//如果Detours之前保留了未使用的trampolines区域，则返回TRUE;否则,返回FALSE。
/*fRetain:指定在区域中的所有trampolines被释放后，是否应该保留(并重用)trampolines内存分配区域。
如果设置为TRUE，则保留这些区域。如果该参数设置为FALSE，则不保留region。*/
/*Detours从64KB内存的连续区域分配trampolines。默认情况下，当区域中的所有trampolines都被释放(分离)后，这些区域将返回给操作系统。然而，在某些情况下，例如当程序频繁地附加、分离和重新附加时，可能需要保留内存区域。*/
BOOL WINAPI DetourSetRetainRegions(_In_ BOOL fRetain)
{
    BOOL fPrevious = s_fRetainRegions;
    s_fRetainRegions = fRetain;
    return fPrevious;
}
//设置不能用于Trameplnes的内存区域的下界，因为它是为系统dll保留的。
//返回上一个下界值。
//pSystemRegionLowerBound:指定Detours必须避免放置Trameplenes的系统区域的下界。
PVOID WINAPI DetourSetSystemRegionLowerBound(_In_ PVOID pSystemRegionLowerBound)
{
    PVOID pPrevious = s_pSystemRegionLowerBound;
    s_pSystemRegionLowerBound = pSystemRegionLowerBound;
    return pPrevious;
}
//设置不能用于Tramplines的内存区域的上界，因为它是为系统dll保留的。
//返回上一个上界值。
//pSystemRegionUpperBound:指定Detours必须避免放置蹦床的系统区域的上界。
PVOID WINAPI DetourSetSystemRegionUpperBound(_In_ PVOID pSystemRegionUpperBound)
{
    PVOID pPrevious = s_pSystemRegionUpperBound;
    s_pSystemRegionUpperBound = pSystemRegionUpperBound;
    return pPrevious;
}

LONG WINAPI DetourTransactionBegin()
```

```
{
    //一次只允许进行一次处理
    // Only one transaction is allowed at a time.
_Benign_race_begin_
    if (s_nPendingThreadId != 0) {
        return ERROR_INVALID_OPERATION;
    }
_Benign_race_end_
    //确保只有一个线程可以启动处理
    // Make sure only one thread can start a transaction.
    if (InterlockedCompareExchange(&s_nPendingThreadId,
(LONG)GetCurrentThreadId(), 0) != 0) {
        return ERROR_INVALID_OPERATION;
    }

    s_pPendingOperations = NULL;
    s_pPendingThreads = NULL;
    s_ppPendingError = NULL;

    // Make sure the trampoline pages are writable.
    s_nPendingError = detour_writable_trampoline_regions();

    return s_nPendingError;
}

LONG WINAPI DetourTransactionAbort()
{/*
intro: 中止当前事务以附加或分离Detour，DetourTransactionAbort将终止使用
DetourTransactionBegin
创建的当前事务。中止一个事务将逆转事务中对DetourAttach、DetourAttachEx、DetourDetach
或DetourUpdateThread api的任何调用的效果。
return:  如果挂起的事务完全中止，则返回NO_ERROR;否则，返回错误代码。
错误代码 ERROR_INVALID_OPERATION:不存在挂起的事务
*/
    if (s_nPendingThreadId != (LONG)GetCurrentThreadId()) {
        return ERROR_INVALID_OPERATION;
    }

    // Restore all of the page permissions.
    for (DetourOperation *o = s_pPendingOperations; o != NULL;) {
        // We don't care if this fails, because the code is still accessible.
        DWORD dwOld;
        VirtualProtect(o->pbTarget, o->pTrampoline->cbRestore,
                       o->dwPerm, &dwOld);

        if (!o->fIsRemove) {
            if (o->pTrampoline) {
                detour_free_trampoline(o->pTrampoline);
```

```
398                 o->pTrampoline = NULL;
399             }
400         }
401
402         DetourOperation *n = o->pNext;
403         delete o;
404         o = n;
405     }
406     s_pPendingOperations = NULL;
407
408     // Make sure the trampoline pages are no longer writable.
409     detour_runnable_trampoline_regions();
410
411     // Resume any suspended threads.
412     for (DetourThread *t = s_pPendingThreads; t != NULL;) {
413         // There is nothing we can do if this fails.
414         ResumeThread(t->hThread);
415
416         DetourThread *n = t->pNext;
417         delete t;
418         t = n;
419     }
420     s_pPendingThreads = NULL;
421     s_nPendingThreadId = 0;
422
423     return NO_ERROR;
424 }
425
426 LONG WINAPI DetourTransactionCommit()
427 {/*
428 intro:提交用DetourTransactionBegin创建的当前事务。
429 提交事务将在事务中调用DetourAttach、DetourAttachEx、DetourDetach或
    DetourUpdateThread api中指定所有更新。
430 return 如果成功返回NO_ERROR;否则，返回错误代码。
431 错误代码
432 ERROR_INVALID_DATA:目标函数在事务的各个步骤之间被第三方更改。
433
434 ERROR_INVALID_OPERATION:不存在挂起的事务。
435
436 Other: API在DetourAttach、DetourAttachEx或DetourDetach中返回的导致事务失败的错误代
    码。
437 */
438     return DetourTransactionCommitEx(NULL);
439 }
440
441 static BYTE detour_align_from_trampoline(PDETOUR_TRAMPOLINE pTrampoline, BYTE
    obTrampoline)
442 {
```

```
443     for (LONG n = 0; n < ARRAYSIZE(pTrampoline->rAlign); n++) {
444         if (pTrampoline->rAlign[n].obTrampoline == obTrampoline) {
445             return pTrampoline->rAlign[n].obTarget;
446         }
447     }
448     return 0;
449 }
450
451 static LONG detour_align_from_target(PDETOUR_TRAMPOLINE pTrampoline, LONG
    obTarget)
452 {/**/
453     for (LONG n = 0; n < ARRAYSIZE(pTrampoline->rAlign); n++) {
454         if (pTrampoline->rAlign[n].obTarget == obTarget) {
455             return pTrampoline->rAlign[n].obTrampoline;
456         }
457     }
458     return 0;
459 }
460
461 LONG WINAPI DetourTransactionCommitEx(_Out_opt_ PVOID **pppFailedPointer)
462 {/*
463 intro: 提交当前事务以附加或分离Detour。
464 param:pppFailedPointer:用于接收传递给导致最近事务失败的DetourAttach、DetourAttachEx
    或DetourDetach调用的目标指针的变量。
465 return 如果成功，则返回NO_ERROR;否则，返回错误代码
466 错误代码
467 ERROR_INVALID_DATA:在事务完成之前，目标函数被第三方更改。
468 ERROR_INVALID_OPERATION:不存在挂起的事务。
469 其他代码:API在DetourAttach、DetourAttachEx或DetourDetach中返回的导致事务失败的错误代
    码。
470 */
471     if (pppFailedPointer != NULL) {
472         // Used to get the last error.
473         *pppFailedPointer = s_ppPendingError;
474     }
475     if (s_nPendingThreadId != (LONG)GetCurrentThreadId()) {
476         return ERROR_INVALID_OPERATION;
477     }
478
479     // If any of the pending operations failed, then we abort the whole
    transaction.
480     if (s_nPendingError != NO_ERROR) {
481         DETOUR_BREAK();
482         DetourTransactionAbort();
483         return s_nPendingError;
484     }
485
```

```
    // Common variables.
    DetourOperation *o;
    DetourThread *t;
    BOOL freed = FALSE;

    // Insert or remove each of the detours.
    for (o = s_pPendingOperations; o != NULL; o = o->pNext) {
        if (o->fIsRemove) {
            CopyMemory(o->pbTarget,
                       o->pTrampoline->rbRestore,
                       o->pTrampoline->cbRestore);
#ifdef DETOURS_IA64
            *o->ppbPointer = (PBYTE)o->pTrampoline->ppldTarget;
#endif // DETOURS_IA64

#ifdef DETOURS_X86
            *o->ppbPointer = o->pbTarget;
#endif // DETOURS_X86

#ifdef DETOURS_X64
            *o->ppbPointer = o->pbTarget;
#endif // DETOURS_X64

#ifdef DETOURS_ARM
            *o->ppbPointer = DETOURS_PBYTE_TO_PFUNC(o->pbTarget);
#endif // DETOURS_ARM

#ifdef DETOURS_ARM64
            *o->ppbPointer = o->pbTarget;
#endif // DETOURS_ARM
        }
        else {
            DETOUR_TRACE(("detours: pbTramp =%p, pbRemain=%p, pbDetour=%p,
cbRestore=%u\n",
                          o->pTrampoline,
                          o->pTrampoline->pbRemain,
                          o->pTrampoline->pbDetour,
                          o->pTrampoline->cbRestore));

            DETOUR_TRACE(("detours: pbTarget=%p: "
                          "%02x %02x %02x %02x "
                          "%02x %02x %02x %02x "
                          "%02x %02x %02x %02x [before]\n",
                          o->pbTarget,
                          o->pbTarget[0], o->pbTarget[1], o->pbTarget[2], o-
>pbTarget[3],
                          o->pbTarget[4], o->pbTarget[5], o->pbTarget[6], o-
>pbTarget[7],
```

```
531                         o->pbTarget[8], o->pbTarget[9], o->pbTarget[10], o->pbTarget[11]));
532
533 #ifdef DETOURS_IA64
534             ((DETOUR_IA64_BUNDLE*)o->pbTarget)
535                 ->SetBrl((UINT64)&o->pTrampoline->bAllocFrame);
536             *o->ppbPointer = (PBYTE)&o->pTrampoline->pldTrampoline;
537 #endif // DETOURS_IA64
538
539 #ifdef DETOURS_X64
540             detour_gen_jmp_indirect(o->pTrampoline->rbCodeIn, &o->pTrampoline->pbDetour);
541             PBYTE pbCode = detour_gen_jmp_immediate(o->pbTarget, o->pTrampoline->rbCodeIn);
542             pbCode = detour_gen_brk(pbCode, o->pTrampoline->pbRemain);
543             *o->ppbPointer = o->pTrampoline->rbCode;
544             UNREFERENCED_PARAMETER(pbCode);
545 #endif // DETOURS_X64
546
547 #ifdef DETOURS_X86
548             PBYTE pbCode = detour_gen_jmp_immediate(o->pbTarget, o->pTrampoline->pbDetour);
549             pbCode = detour_gen_brk(pbCode, o->pTrampoline->pbRemain);
550             *o->ppbPointer = o->pTrampoline->rbCode;
551             UNREFERENCED_PARAMETER(pbCode);
552 #endif // DETOURS_X86
553
554 #ifdef DETOURS_ARM
555             PBYTE pbCode = detour_gen_jmp_immediate(o->pbTarget, NULL, o->pTrampoline->pbDetour);
556             pbCode = detour_gen_brk(pbCode, o->pTrampoline->pbRemain);
557             *o->ppbPointer = DETOURS_PBYTE_TO_PFUNC(o->pTrampoline->rbCode);
558             UNREFERENCED_PARAMETER(pbCode);
559 #endif // DETOURS_ARM
560
561 #ifdef DETOURS_ARM64
562             PBYTE pbCode = detour_gen_jmp_indirect(o->pbTarget, (ULONG64*)&(o->pTrampoline->pbDetour));
563             pbCode = detour_gen_brk(pbCode, o->pTrampoline->pbRemain);
564             *o->ppbPointer = o->pTrampoline->rbCode;
565             UNREFERENCED_PARAMETER(pbCode);
566 #endif // DETOURS_ARM64
567
568             DETOUR_TRACE(("detours: pbTarget=%p: "
569                           "%02x %02x %02x %02x "
570                           "%02x %02x %02x %02x "
571                           "%02x %02x %02x %02x [after]\n",
572                           o->pbTarget
```

```
                           o->pbTarget,
                           o->pbTarget[0], o->pbTarget[1], o->pbTarget[2], o->pbTarget[3],
                           o->pbTarget[4], o->pbTarget[5], o->pbTarget[6], o->pbTarget[7],
                           o->pbTarget[8], o->pbTarget[9], o->pbTarget[10], o->pbTarget[11]));

            DETOUR_TRACE(("detours: pbTramp =%p: "
                           "%02x %02x %02x %02x "
                           "%02x %02x %02x %02x "
                           "%02x %02x %02x %02x\n",
                           o->pTrampoline,
                           o->pTrampoline->rbCode[0], o->pTrampoline->rbCode[1],
                           o->pTrampoline->rbCode[2], o->pTrampoline->rbCode[3],
                           o->pTrampoline->rbCode[4], o->pTrampoline->rbCode[5],
                           o->pTrampoline->rbCode[6], o->pTrampoline->rbCode[7],
                           o->pTrampoline->rbCode[8], o->pTrampoline->rbCode[9],
                           o->pTrampoline->rbCode[10], o->pTrampoline->rbCode[11]));

#ifdef DETOURS_IA64
            DETOUR_TRACE(("\n"));
            DETOUR_TRACE(("detours:  &pldTrampoline  =%p\n",
                           &o->pTrampoline->pldTrampoline));
            DETOUR_TRACE(("detours:  &bMovlTargetGp  =%p [%p]\n",
                           &o->pTrampoline->bMovlTargetGp,
                           o->pTrampoline->bMovlTargetGp.GetMovlGp()));
            DETOUR_TRACE(("detours:  &rbCode         =%p [%p]\n",
                           &o->pTrampoline->rbCode,
                           ((DETOUR_IA64_BUNDLE&)o->pTrampoline->rbCode).GetBrlTarget()));
            DETOUR_TRACE(("detours:  &bBrlRemainEip  =%p [%p]\n",
                           &o->pTrampoline->bBrlRemainEip,
                           o->pTrampoline->bBrlRemainEip.GetBrlTarget()));
            DETOUR_TRACE(("detours:  &bMovlDetourGp  =%p [%p]\n",
                           &o->pTrampoline->bMovlDetourGp,
                           o->pTrampoline->bMovlDetourGp.GetMovlGp()));
            DETOUR_TRACE(("detours:  &bBrlDetourEip  =%p [%p]\n",
                           &o->pTrampoline->bCallDetour,
                           o->pTrampoline->bCallDetour.GetBrlTarget()));
            DETOUR_TRACE(("detours:  pldDetour       =%p [%p]\n",
                           o->pTrampoline->ppldDetour->EntryPoint,
```

```
610                         o->pTrampoline->ppldDetour->GlobalPointer));
611             DETOUR_TRACE(("detours:  pldTarget       =%p [%p]\n",
612                         o->pTrampoline->ppldTarget->EntryPoint,
613                         o->pTrampoline->ppldTarget->GlobalPointer));
614             DETOUR_TRACE(("detours:  pbRemain        =%p\n",
615                         o->pTrampoline->pbRemain));
616             DETOUR_TRACE(("detours:  pbDetour        =%p\n",
617                         o->pTrampoline->pbDetour));
618             DETOUR_TRACE(("\n"));
619 #endif // DETOURS_IA64
620         }
621     }
622
623     // Update any suspended threads.
624     for (t = s_pPendingThreads; t != NULL; t = t->pNext) {
625         CONTEXT cxt;
626         cxt.ContextFlags = CONTEXT_CONTROL;
627
628 #undef DETOURS_EIP
629
630 #ifdef DETOURS_X86
631 #define DETOURS_EIP         Eip
632 #endif // DETOURS_X86
633
634 #ifdef DETOURS_X64
635 #define DETOURS_EIP         Rip
636 #endif // DETOURS_X64
637
638 #ifdef DETOURS_IA64
639 #define DETOURS_EIP         StIIP
640 #endif // DETOURS_IA64
641
642 #ifdef DETOURS_ARM
643 #define DETOURS_EIP         Pc
644 #endif // DETOURS_ARM
645
646 #ifdef DETOURS_ARM64
647 #define DETOURS_EIP         Pc
648 #endif // DETOURS_ARM64
649
650 typedef ULONG_PTR DETOURS_EIP_TYPE;
651
652         if (GetThreadContext(t->hThread, &cxt)) {
653             for (o = s_pPendingOperations; o != NULL; o = o->pNext) {
654                 if (o->fIsRemove) {
655                     if (cxt.DETOURS_EIP >= (DETOURS_EIP_TYPE)(ULONG_PTR)o-
    >pTrampoline &&
656                         cxt.DETOURS_EIP < (DETOURS_EIP_TYPE)((ULONG_PTR)o-
```

```
    >pTrampoline
657                                                     + sizeof(o-
    >pTrampoline))
658                         ) {
659
660                          cxt.DETOURS_EIP = (DETOURS_EIP_TYPE)
661                              ((ULONG_PTR)o->pbTarget
662                               + detour_align_from_trampoline(o->pTrampoline,
663                                                              (BYTE)
    (cxt.DETOURS_EIP
664                                                                   -
    (DETOURS_EIP_TYPE)(ULONG_PTR)
665                                                                   o-
    >pTrampoline)));
666
667                          SetThreadContext(t->hThread, &cxt);
668                      }
669                  }
670                  else {
671                      if (cxt.DETOURS_EIP >= (DETOURS_EIP_TYPE)(ULONG_PTR)o-
    >pbTarget &&
672                          cxt.DETOURS_EIP < (DETOURS_EIP_TYPE)((ULONG_PTR)o-
    >pbTarget
673                                                               + o->pTrampoline-
    >cbRestore)
674                          ) {
675
676                          cxt.DETOURS_EIP = (DETOURS_EIP_TYPE)
677                              ((ULONG_PTR)o->pTrampoline
678                               + detour_align_from_target(o->pTrampoline,
679                                                          (BYTE)(cxt.DETOURS_EIP
680                                                                 -
    (DETOURS_EIP_TYPE)(ULONG_PTR)
681                                                                 o->pbTarget)));
682
683                          SetThreadContext(t->hThread, &cxt);
684                      }
685                  }
686              }
687          }
688  #undef DETOURS_EIP
689      }
690
691      // Restore all of the page permissions and flush the icache.
692      HANDLE hProcess = GetCurrentProcess();
693      for (o = s_pPendingOperations; o != NULL;) {
694          // We don't care if this fails, because the code is still accessible.
695          DWORD dwOld;
```

```
695         DWORD dwOld;
696         VirtualProtect(o->pbTarget, o->pTrampoline->cbRestore, o->dwPerm,
    &dwOld);
697         FlushInstructionCache(hProcess, o->pbTarget, o->pTrampoline-
    >cbRestore);
698
699         if (o->fIsRemove && o->pTrampoline) {
700             detour_free_trampoline(o->pTrampoline);
701             o->pTrampoline = NULL;
702             freed = true;
703         }
704
705         DetourOperation *n = o->pNext;
706         delete o;
707         o = n;
708     }
709     s_pPendingOperations = NULL;
710
711     // Free any trampoline regions that are now unused.
712     if (freed && !s_fRetainRegions) {
713         detour_free_unused_trampoline_regions();
714     }
715
716     // Make sure the trampoline pages are no longer writable.
717     detour_runnable_trampoline_regions();
718
719     // Resume any suspended threads.
720     for (t = s_pPendingThreads; t != NULL;) {
721         // There is nothing we can do if this fails.
722         ResumeThread(t->hThread);
723
724         DetourThread *n = t->pNext;
725         delete t;
726         t = n;
727     }
728     s_pPendingThreads = NULL;
729     s_nPendingThreadId = 0;
730
731     if (pppFailedPointer != NULL) {
732         *pppFailedPointer = s_ppPendingError;
733     }
734
735     return s_nPendingError;
736 }
```

PART_3_ @臧国盛 _END

PART_4_ @于文明 _BEGIN

C++

```cpp
LONG WINAPI DetourUpdateThread(_In_ HANDLE hThread)
{//刷新线程，如果联合GetCurrentThread()函数，可以更新当前的线程
    /*
    param:handle hThread 线程句柄
    return 如果成功，则返回NO_ERROR；否则，返回错误代码。return error 错误代码:ERROR_NOT_ENOUGH_MEMORY:没有足够的内存来记录线程的标识。
    */
    LONG error;

    // If any of the pending operations failed, then we don't need to do this.
    if (s_nPendingError != NO_ERROR) {
        return s_nPendingError;
    }

    // Silently (and safely) drop any attempt to suspend our own thread.
    if (hThread == GetCurrentThread()) {//偷偷的（以及安全的）放弃任何试图挂起我们自己线程的操作
        return NO_ERROR;
    }

    DetourThread *t = new NOTHROW DetourThread;
    if (t == NULL) {
        error = ERROR_NOT_ENOUGH_MEMORY;
      fail:
        if (t != NULL) {
            delete t;
            t = NULL;
        }
        s_nPendingError = error;
        s_ppPendingError = NULL;
        DETOUR_BREAK();
        return error;
    }

    if (SuspendThread(hThread) == (DWORD)-1) {
        error = GetLastError();
        DETOUR_BREAK();
        goto fail;
    }

    t->hThread = hThread;
    t->pNext = s_pPendingThreads;
    s_pPendingThreads = t;

```

```
43      return NO_ERROR;
44  }
45
46  ////////////////////////////////////////////////////////////// Transacted APIs.
47  //
48  LONG WINAPI DetourAttach(_Inout_ PVOID *ppPointer,
49                           _In_ PVOID pDetour)
50  {
51  /*
52  param:PVOID ppPointer 指向将要被挂接函数地址的函数指针(被hook函数)
53  param:PVOID pDetour 指向实际运行的函数的指针
54  return 调用DetourAttachEX函数
55  */
56      return DetourAttachEx(ppPointer, pDetour, NULL, NULL, NULL);
57  }
58
59  LONG WINAPI DetourAttachEx(_Inout_ PVOID *ppPointer,
60                             _In_ PVOID pDetour,
61                             _Out_opt_ PDETOUR_TRAMPOLINE *ppRealTrampoline,
62                             _Out_opt_ PVOID *ppRealTarget,
63                             _Out_opt_ PVOID *ppRealDetour)
64  {/*
65  intro:DetourAttachEx 将绕行附加到目标函数，并检索有关最终目标的其他详细信息
66  param:PVOID ppPointer:指向绕道将附加到的目标指针的指针。
67  param:pDetour:指向detour函数的指针。
68  ppRealTrampoline:可选接收蹦床地址的变量。
69  ppRealTarget:可选接收目标函数的最终地址的变量。
70  ppRealDetour:可选接收detour函数最终地址的变量。
71  return 如果成功返回NO_ERROR;否则，返回错误代码。
72  错误代码
73  ERROR_INVALID_BLOCK:被引用的函数太小，不能绕道。
74
75  ERROR_INVALID_HANDLE: ppPointer形参为NULL或指向NULL指针。
76
77  ERROR_INVALID_OPERATION:不存在挂起的事务。
78
79  ERROR_NOT_ENOUGH_MEMORY:没有足够的内存来完成操作。
80  说明: ppPointer参数所指向的变量必须在事务期间保持活动，直到DetourTransactionCommit、
    DetourTransactionCommitEx或DetourTransactionAbort被调用。
81  */
82      LONG error = NO_ERROR;
83
84      if (ppRealTrampoline != NULL) {
85          *ppRealTrampoline = NULL;
86      }
87      if (ppRealTarget != NULL) {
88          *ppRealTarget = NULL;
```

```
 89      }
 90      if (ppRealDetour != NULL) {
 91          *ppRealDetour = NULL;
 92      }
 93      if (pDetour == NULL) {
 94          DETOUR_TRACE(("empty detour\n"));
 95          return ERROR_INVALID_PARAMETER;
 96      }
 97
 98      if (s_nPendingThreadId != (LONG)GetCurrentThreadId()) {
 99          DETOUR_TRACE(("transaction conflict with thread id=%ld\n",
    s_nPendingThreadId));
100          return ERROR_INVALID_OPERATION;
101      }
102
103      // If any of the pending operations failed, then we don't need to do this.
104      if (s_nPendingError != NO_ERROR) {
105          DETOUR_TRACE(("pending transaction error=%ld\n", s_nPendingError));
106          return s_nPendingError;
107      }
108
109      if (ppPointer == NULL) {
110          DETOUR_TRACE(("ppPointer is null\n"));
111          return ERROR_INVALID_HANDLE;
112      }
113      if (*ppPointer == NULL) {
114          error = ERROR_INVALID_HANDLE;
115          s_nPendingError = error;
116          s_ppPendingError = ppPointer;
117          DETOUR_TRACE(("*ppPointer is null (ppPointer=%p)\n", ppPointer));
118          DETOUR_BREAK();
119          return error;
120      }
121
122      PBYTE pbTarget = (PBYTE)*ppPointer;
123      PDETOUR_TRAMPOLINE pTrampoline = NULL;
124      DetourOperation *o = NULL;
125
126  #ifdef DETOURS_IA64
127      PPLABEL_DESCRIPTOR ppldDetour = (PPLABEL_DESCRIPTOR)pDetour;
128      PPLABEL_DESCRIPTOR ppldTarget = (PPLABEL_DESCRIPTOR)pbTarget;
129      PVOID pDetourGlobals = NULL;
130      PVOID pTargetGlobals = NULL;
131
132      pDetour = (PBYTE)DetourCodeFromPointer(ppldDetour, &pDetourGlobals);
133      pbTarget = (PBYTE)DetourCodeFromPointer(ppldTarget, &pTargetGlobals);
134      DETOUR_TRACE((" ppldDetour=%p, code=%p [gp=%p]\n",
135                    ppldDetour, pDetour, pDetourGlobals));
```

```
136     DETOUR_TRACE(("  ppldTarget=%p, code=%p [gp=%p]\n",
137                   ppldTarget, pbTarget, pTargetGlobals));
138 #else // DETOURS_IA64
139     pbTarget = (PBYTE)DetourCodeFromPointer(pbTarget, NULL);
140     pDetour = DetourCodeFromPointer(pDetour, NULL);
141 #endif // !DETOURS_IA64
142
143     // Don't follow a jump if its destination is the target function.
144     // This happens when the detour does nothing other than call the target.
145     if (pDetour == (PVOID)pbTarget) {
146         if (s_fIgnoreTooSmall) {
147             goto stop;
148         }
149         else {
150             DETOUR_BREAK();
151             goto fail;
152         }
153     }
154
155     if (ppRealTarget != NULL) {
156         *ppRealTarget = pbTarget;
157     }
158     if (ppRealDetour != NULL) {
159         *ppRealDetour = pDetour;
160     }
161
162     o = new NOTHROW DetourOperation;
163     if (o == NULL) {
164         error = ERROR_NOT_ENOUGH_MEMORY;
165       fail:
166         s_nPendingError = error;
167         DETOUR_BREAK();
168       stop:
169         if (pTrampoline != NULL) {
170             detour_free_trampoline(pTrampoline);
171             pTrampoline = NULL;
172             if (ppRealTrampoline != NULL) {
173                 *ppRealTrampoline = NULL;
174             }
175         }
176         if (o != NULL) {
177             delete o;
178             o = NULL;
179         }
180         if (ppRealDetour != NULL) {
181             *ppRealDetour = NULL;
182         }
183         if (ppRealTarget != NULL) {
```

```
            if (ppRealTarget != NULL) {
                *ppRealTarget = NULL;
            }
            s_ppPendingError = ppPointer;
            return error;
        }

        pTrampoline = detour_alloc_trampoline(pbTarget);
        if (pTrampoline == NULL) {
            error = ERROR_NOT_ENOUGH_MEMORY;
            DETOUR_BREAK();
            goto fail;
        }

        if (ppRealTrampoline != NULL) {
            *ppRealTrampoline = pTrampoline;
        }

        DETOUR_TRACE(("detours: pbTramp=%p, pDetour=%p\n", pTrampoline, pDetour));

        memset(pTrampoline->rAlign, 0, sizeof(pTrampoline->rAlign));

        // Determine the number of movable target instructions.
        PBYTE pbSrc = pbTarget;
        PBYTE pbTrampoline = pTrampoline->rbCode;
#ifdef DETOURS_IA64
        PBYTE pbPool = (PBYTE)(&pTrampoline->bBranchIslands + 1);
#else
        PBYTE pbPool = pbTrampoline + sizeof(pTrampoline->rbCode);
#endif
        ULONG cbTarget = 0;
        ULONG cbJump = SIZE_OF_JMP;
        ULONG nAlign = 0;

#ifdef DETOURS_ARM
        // On ARM, we need an extra instruction when the function isn't 32-bit
    aligned.
        // Check if the existing code is another detour (or at least a similar
        // "ldr pc, [PC+0]" jump.
        if ((ULONG)pbTarget & 2) {
            cbJump += 2;

            ULONG op = fetch_thumb_opcode(pbSrc);
            if (op == 0xbf00) {
                op = fetch_thumb_opcode(pbSrc + 2);
                if (op == 0xf8dff000) { // LDR PC,[PC]
                    *((PUSHORT&)pbTrampoline)++ = *((PUSHORT&)pbSrc)++;
                    *((PULONG&)pbTrampoline)++ = *((PULONG&)pbSrc)++;
```

```
                *((PULONG&)pbTrampoline)++ = *((PULONG&)pbSrc)++;
                cbTarget = (LONG)(pbSrc - pbTarget);
                // We will fall through the "while" because cbTarget is now >=
    cbJump.
            }
        }
    }
    else {
        ULONG op = fetch_thumb_opcode(pbSrc);
        if (op == 0xf8dff000) { // LDR PC,[PC]
            *((PULONG&)pbTrampoline)++ = *((PULONG&)pbSrc)++;
            *((PULONG&)pbTrampoline)++ = *((PULONG&)pbSrc)++;
            cbTarget = (LONG)(pbSrc - pbTarget);
            // We will fall through the "while" because cbTarget is now >=
    cbJump.
        }
    }
#endif

    while (cbTarget < cbJump) {
        PBYTE pbOp = pbSrc;
        LONG lExtra = 0;

        DETOUR_TRACE((" DetourCopyInstruction(%p,%p)\n",
                      pbTrampoline, pbSrc));
        pbSrc = (PBYTE)
            DetourCopyInstruction(pbTrampoline, (PVOID*)&pbPool, pbSrc, NULL,
    &lExtra);
        DETOUR_TRACE((" DetourCopyInstruction() = %p (%d bytes)\n",
                      pbSrc, (int)(pbSrc - pbOp)));
        pbTrampoline += (pbSrc - pbOp) + lExtra;
        cbTarget = (LONG)(pbSrc - pbTarget);
        pTrampoline->rAlign[nAlign].obTarget = cbTarget;
        pTrampoline->rAlign[nAlign].obTrampoline = pbTrampoline - pTrampoline-
    >rbCode;
        nAlign++;

        if (nAlign >= ARRAYSIZE(pTrampoline->rAlign)) {
            break;
        }

        if (detour_does_code_end_function(pbOp)) {
            break;
        }
    }

    // Consume, but don't duplicate padding if it is needed and available.
    //使用，但不要重复填充（如果需要且可用）。
```

```
274     while (cbTarget < cbJump) {
275         LONG cFiller = detour_is_code_filler(pbSrc);
276         if (cFiller == 0) {
277             break;
278         }
279 
280         pbSrc += cFiller;
281         cbTarget = (LONG)(pbSrc - pbTarget);
282     }
283 
284 #if DETOUR_DEBUG
285     {
286         DETOUR_TRACE((" detours: rAlign ["));
287         LONG n = 0;
288         for (n = 0; n < ARRAYSIZE(pTrampoline->rAlign); n++) {
289             if (pTrampoline->rAlign[n].obTarget == 0 &&
290                 pTrampoline->rAlign[n].obTrampoline == 0) {
291                 break;
292             }
293             DETOUR_TRACE((" %u/%u",
294                           pTrampoline->rAlign[n].obTarget,
295                           pTrampoline->rAlign[n].obTrampoline
296                           ));
297 
298         }
299         DETOUR_TRACE((" ]\n"));
300     }
301 #endif
302 
303     if (cbTarget < cbJump || nAlign > ARRAYSIZE(pTrampoline->rAlign)) {
304         // Too few instructions.
305 
306         error = ERROR_INVALID_BLOCK;
307         if (s_fIgnoreTooSmall) {
308             goto stop;
309         }
310         else {
311             DETOUR_BREAK();
312             goto fail;
313         }
314     }
315 
316     if (pbTrampoline > pbPool) {
317         __debugbreak();
318     }
319 
320     pTrampoline->cbCode = (BYTE)(pbTrampoline - pTrampoline->rbCode);
321     pTrampoline->cbRestore = (BYTE)cbTarget;
```

```
321     pTrampoline->cbRestore = (BYTE)cbTarget;
322     CopyMemory(pTrampoline->rbRestore, pbTarget, cbTarget);
323 
324 #if !defined(DETOURS_IA64)
325     if (cbTarget > sizeof(pTrampoline->rbCode) - cbJump) {
326         // Too many instructions.
327         error = ERROR_INVALID_HANDLE;
328         DETOUR_BREAK();
329         goto fail;
330     }
331 #endif // !DETOURS_IA64
332 
333     pTrampoline->pbRemain = pbTarget + cbTarget;
334     pTrampoline->pbDetour = (PBYTE)pDetour;
335 
336 #ifdef DETOURS_IA64
337     pTrampoline->ppldDetour = ppldDetour;
338     pTrampoline->ppldTarget = ppldTarget;
339     pTrampoline->pldTrampoline.EntryPoint = (UINT64)&pTrampoline-
    >bMovlTargetGp;
340     pTrampoline->pldTrampoline.GlobalPointer = (UINT64)pDetourGlobals;
341 
342     ((DETOUR_IA64_BUNDLE *)pTrampoline->rbCode)->SetStop();
343 
344     pTrampoline->bMovlTargetGp.SetMovlGp((UINT64)pTargetGlobals);
345     pTrampoline->bBrlRemainEip.SetBrl((UINT64)pTrampoline->pbRemain);
346 
347     // Alloc frame:      alloc r41=ar.pfs,11,0,8,0; mov r40=rp
348     pTrampoline->bAllocFrame.wide[0] = 0x00000580164d480c;
349     pTrampoline->bAllocFrame.wide[1] = 0x00c4000500000200;
350     // save r36, r37, r38.
351     pTrampoline->bSave37to39.wide[0] = 0x031021004e019001;
352     pTrampoline->bSave37to39.wide[1] = 0x8401280600420098;
353     // save r34,r35,r36: adds r47=0,r36; adds r46=0,r35; adds r45=0,r34
354     pTrampoline->bSave34to36.wide[0] = 0x02e0210048017800;
355     pTrampoline->bSave34to36.wide[1] = 0x84011005a042008c;
356     // save gp,r32,r33"  adds r44=0,r33; adds r43=0,r32; adds r42=0,gp ;;
357     pTrampoline->bSaveGPto33.wide[0] = 0x02b0210042016001;
358     pTrampoline->bSaveGPto33.wide[1] = 0x8400080540420080;
359     // set detour GP.
360     pTrampoline->bMovlDetourGp.SetMovlGp((UINT64)pDetourGlobals);
361     // call detour:      brl.call.sptk.few rp=detour ;;
362     pTrampoline->bCallDetour.wide[0] = 0x0000000100000005;
363     pTrampoline->bCallDetour.wide[1] = 0xd000001000000000;
364     pTrampoline->bCallDetour.SetBrlTarget((UINT64)pDetour);
365     // pop frame & gp:   adds gp=0,r42; mov rp=r40,+0;; mov.i ar.pfs=r41
366     pTrampoline->bPopFrameGp.wide[0] = 0x4000210054000802;
367     pTrampoline->bPopFrameGp.wide[1] = 0x00aa029000038005;
```

```
368     // return to caller: br.ret.sptk.many rp ;;
369     pTrampoline->bReturn.wide[0] = 0x0000000100000019;
370     pTrampoline->bReturn.wide[1] = 0x0084000880000200;
371
372     DETOUR_TRACE(("detours: &bMovlTargetGp=%p\n", &pTrampoline-
    >bMovlTargetGp));
373     DETOUR_TRACE(("detours: &bMovlDetourGp=%p\n", &pTrampoline-
    >bMovlDetourGp));
374 #endif // DETOURS_IA64
375
376     pbTrampoline = pTrampoline->rbCode + pTrampoline->cbCode;
377 #ifdef DETOURS_X64
378     pbTrampoline = detour_gen_jmp_indirect(pbTrampoline, &pTrampoline-
    >pbRemain);
379     pbTrampoline = detour_gen_brk(pbTrampoline, pbPool);
380 #endif // DETOURS_X64
381
382 #ifdef DETOURS_X86
383     pbTrampoline = detour_gen_jmp_immediate(pbTrampoline, pTrampoline-
    >pbRemain);
384     pbTrampoline = detour_gen_brk(pbTrampoline, pbPool);
385 #endif // DETOURS_X86
386
387 #ifdef DETOURS_ARM
388     pbTrampoline = detour_gen_jmp_immediate(pbTrampoline, &pbPool,
    pTrampoline->pbRemain);
389     pbTrampoline = detour_gen_brk(pbTrampoline, pbPool);
390 #endif // DETOURS_ARM
391
392 #ifdef DETOURS_ARM64
393     pbTrampoline = detour_gen_jmp_immediate(pbTrampoline, &pbPool,
    pTrampoline->pbRemain);
394     pbTrampoline = detour_gen_brk(pbTrampoline, pbPool);
395 #endif // DETOURS_ARM64
396
397     (void)pbTrampoline;
398
399     DWORD dwOld = 0;
400     if (!VirtualProtect(pbTarget, cbTarget, PAGE_EXECUTE_READWRITE, &dwOld)) {
401         error = GetLastError();
402         DETOUR_BREAK();
403         goto fail;
404     }
405
406     DETOUR_TRACE(("detours: pbTarget=%p: "
407                   "%02x %02x %02x %02x "
408                   "%02x %02x %02x %02x "
409                   "%02x %02x %02x %02x\n",
```

```
                    pbTarget,
                    pbTarget[0], pbTarget[1], pbTarget[2], pbTarget[3],
                    pbTarget[4], pbTarget[5], pbTarget[6], pbTarget[7],
                    pbTarget[8], pbTarget[9], pbTarget[10], pbTarget[11]));
    DETOUR_TRACE(("detours: pbTramp =%p: "
                    "%02x %02x %02x %02x "
                    "%02x %02x %02x %02x "
                    "%02x %02x %02x %02x\n",
                    pTrampoline,
                    pTrampoline->rbCode[0], pTrampoline->rbCode[1],
                    pTrampoline->rbCode[2], pTrampoline->rbCode[3],
                    pTrampoline->rbCode[4], pTrampoline->rbCode[5],
                    pTrampoline->rbCode[6], pTrampoline->rbCode[7],
                    pTrampoline->rbCode[8], pTrampoline->rbCode[9],
                    pTrampoline->rbCode[10], pTrampoline->rbCode[11]));

    o->fIsRemove = FALSE;
    o->ppbPointer = (PBYTE*)ppPointer;
    o->pTrampoline = pTrampoline;
    o->pbTarget = pbTarget;
    o->dwPerm = dwOld;
    o->pNext = s_pPendingOperations;
    s_pPendingOperations = o;

    return NO_ERROR;
}

LONG WINAPI DetourDetach(_Inout_ PVOID *ppPointer,
                         _In_ PVOID pDetour)
{/*
intro:恢复拦截函数
参数和DetourAttach相同
*/
    LONG error = NO_ERROR;

    if (s_nPendingThreadId != (LONG)GetCurrentThreadId()) {
        return ERROR_INVALID_OPERATION;
    }

    // If any of the pending operations failed, then we don't need to do this.
    if (s_nPendingError != NO_ERROR) {
        return s_nPendingError;
    }

    if (pDetour == NULL) {
        return ERROR_INVALID_PARAMETER;
    }
```

```
457     if (ppPointer == NULL) {
458         return ERROR_INVALID_HANDLE;
459     }
460     if (*ppPointer == NULL) {
461         error = ERROR_INVALID_HANDLE;
462         s_nPendingError = error;
463         s_ppPendingError = ppPointer;
464         DETOUR_BREAK();
465         return error;
466     }
467
468     DetourOperation *o = new NOTHROW DetourOperation;
469     if (o == NULL) {
470         error = ERROR_NOT_ENOUGH_MEMORY;
471       fail:
472         s_nPendingError = error;
473         DETOUR_BREAK();
474       stop:
475         if (o != NULL) {
476             delete o;
477             o = NULL;
478         }
479         s_ppPendingError = ppPointer;
480         return error;
481     }
482
483
484 #ifdef DETOURS_IA64
485     PPLABEL_DESCRIPTOR ppldTrampo = (PPLABEL_DESCRIPTOR)*ppPointer;
486     PPLABEL_DESCRIPTOR ppldDetour = (PPLABEL_DESCRIPTOR)pDetour;
487     PVOID pDetourGlobals = NULL;
488     PVOID pTrampoGlobals = NULL;
489
490     pDetour = (PBYTE)DetourCodeFromPointer(ppldDetour, &pDetourGlobals);
491     PDETOUR_TRAMPOLINE pTrampoline = (PDETOUR_TRAMPOLINE)
492         DetourCodeFromPointer(ppldTrampo, &pTrampoGlobals);
493     DETOUR_TRACE(("  ppldDetour=%p, code=%p [gp=%p]\n",
494                   ppldDetour, pDetour, pDetourGlobals));
495     DETOUR_TRACE(("  ppldTrampo=%p, code=%p [gp=%p]\n",
496                   ppldTrampo, pTrampoline, pTrampoGlobals));
497
498
499     DETOUR_TRACE(("\n"));
500     DETOUR_TRACE(("detours:  &pldTrampoline  =%p\n",
501                   &pTrampoline->pldTrampoline));
502     DETOUR_TRACE(("detours:  &bMovlTargetGp  =%p [%p]\n",
503                   &pTrampoline->bMovlTargetGp,
504                   pTrampoline->bMovlTargetGp.GetMovlGp()));
```

```
505     DETOUR_TRACE(("detours:  &rbCode         =%p [%p]\n",
506                   &pTrampoline->rbCode,
507                   ((DETOUR_IA64_BUNDLE&)pTrampoline->rbCode).GetBrlTarget()));
508     DETOUR_TRACE(("detours:  &bBrlRemainEip  =%p [%p]\n",
509                   &pTrampoline->bBrlRemainEip,
510                   pTrampoline->bBrlRemainEip.GetBrlTarget()));
511     DETOUR_TRACE(("detours:  &bMovlDetourGp  =%p [%p]\n",
512                   &pTrampoline->bMovlDetourGp,
513                   pTrampoline->bMovlDetourGp.GetMovlGp()));
514     DETOUR_TRACE(("detours:  &bBrlDetourEip  =%p [%p]\n",
515                   &pTrampoline->bCallDetour,
516                   pTrampoline->bCallDetour.GetBrlTarget()));
517     DETOUR_TRACE(("detours:  pldDetour       =%p [%p]\n",
518                   pTrampoline->ppldDetour->EntryPoint,
519                   pTrampoline->ppldDetour->GlobalPointer));
520     DETOUR_TRACE(("detours:  pldTarget       =%p [%p]\n",
521                   pTrampoline->ppldTarget->EntryPoint,
522                   pTrampoline->ppldTarget->GlobalPointer));
523     DETOUR_TRACE(("detours:  pbRemain        =%p\n",
524                   pTrampoline->pbRemain));
525     DETOUR_TRACE(("detours:  pbDetour        =%p\n",
526                   pTrampoline->pbDetour));
527     DETOUR_TRACE(("\n"));
528 #else // !DETOURS_IA64
529     PDETOUR_TRAMPOLINE pTrampoline =
530         (PDETOUR_TRAMPOLINE)DetourCodeFromPointer(*ppPointer, NULL);
531     pDetour = DetourCodeFromPointer(pDetour, NULL);
532 #endif // !DETOURS_IA64
533
534     ////////////////////////////////////// Verify that Trampoline is in place.
535     //
536     LONG cbTarget = pTrampoline->cbRestore;
537     PBYTE pbTarget = pTrampoline->pbRemain - cbTarget;
538     if (cbTarget == 0 || cbTarget > sizeof(pTrampoline->rbCode)) {
539         error = ERROR_INVALID_BLOCK;
540         if (s_fIgnoreTooSmall) {
541             goto stop;
542         }
543         else {
544             DETOUR_BREAK();
545             goto fail;
546         }
547     }
548
549     if (pTrampoline->pbDetour != pDetour) {
550         error = ERROR_INVALID_BLOCK;
551         if (s_fIgnoreTooSmall) {
552             goto stop;
```

```
553         }
554         else {
555             DETOUR_BREAK();
556             goto fail;
557         }
558     }
559
560     DWORD dwOld = 0;
561     if (!VirtualProtect(pbTarget, cbTarget,
562                         PAGE_EXECUTE_READWRITE, &dwOld)) {
563         error = GetLastError();
564         DETOUR_BREAK();
565         goto fail;
566     }
567
568     o->fIsRemove = TRUE;
569     o->ppbPointer = (PBYTE*)ppPointer;
570     o->pTrampoline = pTrampoline;
571     o->pbTarget = pbTarget;
572     o->dwPerm = dwOld;
573     o->pNext = s_pPendingOperations;
574     s_pPendingOperations = o;
575
576     return NO_ERROR;
577 }
578
579 ////////////////////////////////////////////////////////////////////////////
580 //
581 // Helpers for manipulating page protection.
582 //
583
584 // For reference:
585 //   PAGE_NOACCESS          0x01
586 //   PAGE_READONLY          0x02
587 //   PAGE_READWRITE         0x04
588 //   PAGE_WRITECOPY         0x08
589 //   PAGE_EXECUTE           0x10
590 //   PAGE_EXECUTE_READ      0x20
591 //   PAGE_EXECUTE_READWRITE 0x40
592 //   PAGE_EXECUTE_WRITECOPY 0x80
593 //   PAGE_GUARD             ...
594 //   PAGE_NOCACHE           ...
595 //   PAGE_WRITECOMBINE      ...
596
597 #define DETOUR_PAGE_EXECUTE_ALL    (PAGE_EXECUTE |              \
598                                     PAGE_EXECUTE_READ |         \
599                                     PAGE_EXECUTE_READWRITE |    \
```

```
600                                     PAGE_EXECUTE_WRITECOPY)
601
602 #define DETOUR_PAGE_NO_EXECUTE_ALL (PAGE_NOACCESS |             \
603                                     PAGE_READONLY |             \
604                                     PAGE_READWRITE |            \
605                                     PAGE_WRITECOPY)
606
607 #define DETOUR_PAGE_ATTRIBUTES     (~(DETOUR_PAGE_EXECUTE_ALL |
    DETOUR_PAGE_NO_EXECUTE_ALL))
608
609 C_ASSERT((DETOUR_PAGE_NO_EXECUTE_ALL << 4) == DETOUR_PAGE_EXECUTE_ALL);
610
611 static DWORD DetourPageProtectAdjustExecute(_In_  DWORD dwOldProtect,
612                                             _In_  DWORD dwNewProtect)
613 //  Copy EXECUTE from dwOldProtect to dwNewProtect.
614 {//从dwOldProtect拷贝EXECUTE到dwNewProtect。
615     bool const fOldExecute = ((dwOldProtect & DETOUR_PAGE_EXECUTE_ALL) != 0);
616     bool const fNewExecute = ((dwNewProtect & DETOUR_PAGE_EXECUTE_ALL) != 0);
617
618     if (fOldExecute && !fNewExecute) {
619         dwNewProtect = ((dwNewProtect & DETOUR_PAGE_NO_EXECUTE_ALL) << 4)
620             | (dwNewProtect & DETOUR_PAGE_ATTRIBUTES);
621     }
622     else if (!fOldExecute && fNewExecute) {
623         dwNewProtect = ((dwNewProtect & DETOUR_PAGE_EXECUTE_ALL) >> 4)
624             | (dwNewProtect & DETOUR_PAGE_ATTRIBUTES);
625     }
626     return dwNewProtect;
627 }
628
629 _Success_(return != FALSE)
630 BOOL WINAPI DetourVirtualProtectSameExecuteEx(_In_  HANDLE hProcess,
631                                               _In_  PVOID pAddress,
632                                               _In_  SIZE_T nSize,
633                                               _In_  DWORD dwNewProtect,
634                                               _Out_ PDWORD pdwOldProtect)
635 // Some systems do not allow executability of a page to change. This function
    applies
636 // dwNewProtect to [pAddress, nSize), but preserving the previous
    executability.
637 // This function is meant to be a drop-in replacement for some uses of
    VirtualProtectEx.
638 // When "restoring" page protection, there is no need to use this function.
639 /*
640 intro:某些系统不允许更改页面的可执行性。
641 此函数将 dwNewProtect 应用于 [pAddress， nSize），但保留以前的可执行性。
642 此功能旨在替代VirtualProtectEx的某些用途。"恢复"页面保护时，无需使用此功能。
643 param: HANDLE hProcess更改其内存保护的进程的句柄。句柄必须具有PROCESS_VM_OPERATION访
```

```
    问权限。
644 */
645 {
646     MEMORY_BASIC_INFORMATION mbi;
647 
648      //MEMORY_BASIC_INFORMATION: 包含有关进程的虚拟地址空间中的一系列页的信息。
649      /*typedef struct _MEMORY_BASIC_INFORMATION {
650     PVOID BaseAddress;
651     PVOID AllocationBase;
652     DWORD AllocationProtect;
653     SIZE_T RegionSize;
654     DWORD State;
655     DWORD Protect;
656     DWORD Type;
657     }*/
658     /*
659     memb:BaseAddress指向页面区域的基址的指针。
660     memb:AllocationBase指向由 VirtualAlloc 函数分配的页的一系列页的基址的指针。
661     BaseAddress 成员所指向的页面包含在此分配范围内。
662     memb:AllocationProtect
663     最初分配区域时的内存保护选项。此成员可以是内存保护常量之一，如果调用方没有访问权限，
    则可以是 0。
664     memb:PartitionId
665     RegionSize
666     从所有页面具有相同属性的基址开始的区域大小（以字节为单位）。
667     ...
668     Protect 区域中页面的访问保护。此成员是为"分配保护"成员列出的值之一。
669     */
670     // Query to get existing execute access.
671 
672     ZeroMemory(&mbi, sizeof(mbi));
673 
674     if (VirtualQueryEx(hProcess, pAddress, &mbi, sizeof(mbi)) == 0) {
675         return FALSE;
676     }
677     return VirtualProtectEx(hProcess, pAddress, nSize,
678                             DetourPageProtectAdjustExecute(mbi.Protect,
    dwNewProtect),
679                             pdwOldProtect);
680     /*VirtualProtectEx
681     intro:更改对指定进程的虚拟地址空间中已提交页区域的保护。
682     //param:hProcess要更改其内存保护的进程的句柄。句柄必须具有PROCESS_VM_OPERATION访
    问权限。有关详细信息
683     param:pAddress指向要更改其访问保护属性的页面区域的基址的指针。
684     指定区域中的所有页面必须位于使用MEM_RESERVE调用 VirtualAlloc 或 VirtualAllocEx
    函数时分配的同一保留区域内。这些页面不能跨越相邻的保留区域，这些保留区域是通过使用
    MEM_RESERVE对 VirtualAlloc 或 VirtualAllocEx 的单独调用而分配的。
685     param:[in] dwSize
```

```
    param:[in] dwSize
    访问保护属性已更改的区域的大小（以字节为单位）。受影响页面的区域包括包含从 lpAddress 参数到 的范围内的一个或多个字节的所有页面。这意味着跨越页面边界的 2 字节范围会导致两个页面的保护属性发生更改。(lpAddress+dwSize)
    param:[in] flNewProtect
    内存保护选项。此参数可以是内存保护常量之一。
    对于映射视图，此值必须与映射视图时指定的访问保护兼容
    param:[out] lpflOldProtect
    指向一个变量的指针，该变量接收指定页区域中第一页的上一个访问保护。如果此参数为 NULL 或未指向有效变量，则函数将失败。
    return:如果函数成功，则返回值为非零。如果函数失败，则返回值为零。
    */
}

_Success_(return != FALSE)
BOOL WINAPI DetourVirtualProtectSameExecute(_In_  PVOID pAddress,
                                            _In_  SIZE_T nSize,
                                            _In_  DWORD dwNewProtect,
                                            _Out_ PDWORD pdwOldProtect)
{/*intro对当前进程的去保护
return DetourVitualProtectSameExectuteEx*/
    return DetourVirtualProtectSameExecuteEx(GetCurrentProcess(),
                                             pAddress, nSize, dwNewProtect, pdwOldProtect);
}

BOOL WINAPI DetourAreSameGuid(_In_ REFGUID left, _In_ REFGUID right)
{/*
比较GUID是否相同
return bool值 如果相等返回 true 否则 返回false
*/
    return
        left.Data1 == right.Data1 &&
        left.Data2 == right.Data2 &&
        left.Data3 == right.Data3 &&
        left.Data4[0] == right.Data4[0] &&
        left.Data4[1] == right.Data4[1] &&
        left.Data4[2] == right.Data4[2] &&
        left.Data4[3] == right.Data4[3] &&
        left.Data4[4] == right.Data4[4] &&
        left.Data4[5] == right.Data4[5] &&
        left.Data4[6] == right.Data4[6] &&
        left.Data4[7] == right.Data4[7];
}

//  End of File
```

PART_4_@于文明_END

THE END