存储大实验报告

计23

鲁逸沁

2012011314

# Detours的学习经历

## 使用detours.h编写工程的失败经历

我下载的版本是Detours Express 3.0。

根据网上的资料，在VC目录下对Detours进行编译，生成detours.lib，以便在Visual Studio 2012中使用。其间发生各种错误，详见遇到的问题。

接下来编写dll工程myHook32。首先我hook了CreateFileW这个函数，并对它进行重新编写Mine\_CreateFileW。

然后我又新建了一个Win32工程DetourCreateWithDllEx，用来把myHook32.dll注入可执行文件（我使用notepad.exe作为例子）当中，使用DetourCreateProcessWithDllEx函数进行注入。

可是在我给Mine\_CreateFileW函数添加更多功能的时候，却发生了问题：可执行文件一直跑在后台，进程里能发现，但是图形界面没有任何现象。在经过无数次调试和修改后，我放弃了这一方法。

## 使用samples里的工具进行开发

然后我重新阅读了Detours.chm中重要信息，并且学习了samples里的工具的用法。

将samples里的代码编译成可使用的工具这一步我花费了非常大的经历，曾经一度我以为是32位和64位的关系，使用虚拟机装了一个32位的Windows来跑，均无法编译成功，详见遇到的问题。

成功以后，我主要使用了里面的三个工具：

* syelogd.exe
* trcapi32.dll
* withdll.exe

syelogd.exe和trcapi32.dll是配套使用的。在samples\traceapi\\_win32.cpp下，已经有将所有API函数hook住并且重新定义为Mine\_...的新函数。因此我需要做的工作就是将\_win32.cpp中的某些新函数重新编写，实现文件虚拟化的目标。而在\_win32.cpp中，有函数\_PrintEnter会将hook信息输出，但是输出到了一个非正常cmd的地方，因此需要使用syelogd.exe来查看这些信息。

withdll.exe的用途就是将一个dll注入可执行文件中，这里我当然注入trcapi32.dll。

# 具体实现

## 注入目标游戏

这个实验中我选择的游戏是“A Slower Speed of Light”。这款游戏是MIT游戏实验室的研究人员开发了的一款免费游戏，精确模拟爱因斯坦的相对论效应。在游戏中，随着分数的累计，光速会变得越来越慢，游戏者是主要体验这种感觉。



这个游戏的文件主要集中在与exe同目录下的A Slower Speed of Light\_Data文件夹下。同时使用了一些系统的文件。

我的目标是虚拟化其所有调用的dll和一般游戏文件，让其访问我L盘下的文件夹as，而不是继续在其E盘源目录下工作。

## 确定需要hook的API

这里我使用了dumpbin工具，观察A Slower Speed of Light.exe中的系统API调用。大部分与文件无关，我通过一下步骤确定需要修改哪些函数：

* 与文件相关（参考网上资料）
* 系统调用参数涉及路径
* 在\_win32.cpp中将其输出，其他函数不输出，运行游戏观察是否存在调用以及文件操作

通过以上步骤，我挑选出了若干需要hook的函数：

88 CreateFileA

8F CreateFileW

D6 DeleteFileW

133 FindFirstFileExA

139 FindFirstFileW

33C LoadLibraryA

33F LoadLibraryW

但是这些函数新的代码并不是完全不同，有些函数几乎使用完全一样的修改即可。

## 流程

* 将其余输出注释，只剩需要hook的函数的输出，运行游戏，观察输出的规律（或文件的规律），指定修改策略
* 修改\_win32.cpp中对应的函数
* 在detours的根目录下nmake，编译出trcapi32.dll
* 打开syelogd.exe观察输出
* 使用withdll.exe，将trcapi32.dll注入A Slower Speed of Light.exe
* 观察输出，检验是否出现问题

## hook函数LoadLibrary

LoadLibrary函数是用来加载库函数的，通过输出我发现该游戏一般添加的都是C:\Windows\system32下的dll文件（也有例外）。

因此我使用的方法是，在新函数内使用CopyFile函数，将需要的dll的文件从相应路径复制到我指定的位置L:\as\lib下。

但是有些函数的路径是不完整的，因为其目录在环境变量下有定义，直接使用了其名字即可调用。我发现这些dll不超过10个，因此手工枚举了环境变量所在的文件夹，把这些dll手动复制了出来。

接下来L:\as\lib下就是全部我需要LoadLibrary的文件。所以我只需在函数中将其路径字符串进行修改，修改为我指定的路径，再调用原版的LoadLibrary，将其修改后的路径作为参数传入即可。

LoadLibrary代码如下：

HMODULE \_\_stdcall Mine\_LoadLibraryA**(**LPCSTR a0**)**

**{**

// 将a0复制

char a0Buff**[**999**];**

memset**(**a0Buff**,** 0**,** **sizeof(**a0Buff**));**

sprintf\_s**(**a0Buff**,** "%s"**,** a0**);**

// 获取文件名（通过识别符号\）

size\_t a0BuffLen **=** strlen**(**a0Buff**);**

size\_t p **=** a0BuffLen **-** 1**;**

**while** **(**p **&&** a0Buff**[**p**]** **!=** '\\'**)** **--**p**;**

**if** **(**p **!=** 0**)** p**++;**

// 修改新的路径

char newDic**[**999**];**

memset**(**newDic**,** 0**,** **sizeof(**newDic**));**

memcpy**(**newDic**,** "L:\\as\\lib\\"**,** **sizeof(**"L:\\as\\lib\\"**));**

strcat\_s**(**newDic**,** a0Buff **+** p**);**

// 复制文件的代码

/\*int res = Real\_CopyFileA(a0, newDic, true);

if (res) {

Real\_MessageBoxA(NULL, newDic, "xx", MB\_OK);

}\*/

//\_PrintEnter("LoadLibraryA(%hs)\n", newDic);

HMODULE rv **=** 0**;**

\_\_try **{**

rv **=** Real\_LoadLibraryA**(**newDic**);**

**}** \_\_finally **{**

//\_PrintExit("LoadLibraryA() -> %p\n", rv);

**};**

**return** rv**;**

**}**

这样就hook好了LoadLibraryA函数。

而LoadLibraryW函数与其类似，只不过类型全都从char变成了wchar\_t，相应的字符串处理函数也变了名字，代码如下：

HMODULE \_\_stdcall Mine\_LoadLibraryW**(**LPCWSTR a0**)**

**{**

// 将a0复制

wchar\_t a0Buff**[**999**];**

memset**(**a0Buff**,** 0**,** **sizeof(**a0Buff**));**

swprintf\_s**(**a0Buff**,** 999**,** L"%ls"**,** a0**);**

// 获取文件名（通过识别符号\）

size\_t a0BuffLen **=** wcslen**(**a0Buff**);**

size\_t p **=** a0BuffLen **-** 1**;**

**while** **(**p **&&** a0Buff**[**p**]** **!=** L'\\'**)** **--**p**;**

**if** **(**p **!=** 0**)** p**++;**

// 修改新的路径

wchar\_t newDic**[**999**];**

memset**(**newDic**,** 0**,** **sizeof(**newDic**));**

memcpy**(**newDic**,** L"L:\\as\\lib\\"**,** **sizeof(**L"L:\\as\\lib\\"**));**

wcscat\_s**(**newDic**,** a0Buff **+** p**);**

// 复制的代码

/\*int res = Real\_CopyFileW(a0, newDic, true);

if (res) {

Real\_MessageBoxW(NULL, newDic, L"xx", MB\_OK);

}\*/

//\_PrintEnter("LoadLibraryW(%ls)\n", newDic);

HMODULE rv **=** 0**;**

\_\_try **{**

rv **=** Real\_LoadLibraryW**(**newDic**);**

**}** \_\_finally **{**

//\_PrintExit("LoadLibraryW() -> %p\n", rv);

**};**

**return** rv**;**

**}**

## hook函数CreateFile

观察一开始CreateFile的函数，发现其五花八门，有C:\Windows\system32文件夹下的文件，有游戏根目录A Slower Speed of Light\_Data文件夹下的文件，还有若干字体文件。

由于CreateFile不像LoadLibrary那样可以同时CopyFile，此时文件是不可拷贝的，因此直接把文件复制下来放到一起的想法不现实。

经过长时间的排除和归类，可以得出结论：

* C:\Windows\system32文件夹下的文件类似LoadLibrary
* A Slower Speed of Light\_Data文件夹下的文件直接拷贝即可
* 字体文件又大有多，不宜作为虚拟化的对象，也没有意义

然后工作重点就成了分类文件和重定向。我写了transA/transW函数分别作为CreateFileA和CreateFileW修改路径的函数，transA的代码如下：

LPCSTR transA**(**LPCSTR a0**)** **{**

// 复制a0

char a0Buff**[**999**];**

memset**(**a0Buff**,** 0**,** **sizeof(**a0Buff**));**

sprintf\_s**(**a0Buff**,** "%s"**,** a0**);**

// 分离文件名

size\_t a0BuffLen **=** strlen**(**a0Buff**);**

size\_t p **=** a0BuffLen **-** 1**;**

**while** **(**p **&&** a0Buff**[**p**]** **!=** '\\'**)** **--**p**;**

**if** **(**p **!=** 0**)** p**++;**

// 新建新路径

char **\***newDic **=** **new** char**[**999**];**

memset**(**newDic**,** 0**,** **sizeof(**char**)** **\*** 999**);**

// 分类（是否在根目录下）

char **\***data **=** "A Slower Speed of Light\_Data"**;**

size\_t dataLen **=** **sizeof(**"A Slower Speed of Light\_Data"**);**

char **\***q **=** strstr**(**a0Buff**,** data**);**

**if** **(**q **==** **NULL)** **{**

// 系统文件，已处理

memcpy**(**newDic**,** a0Buff**,** **sizeof(**a0Buff**));**

**}** **else** **{**

// 重定向

memcpy**(**newDic**,** "L:\\as\\file\\"**,** **sizeof(**"L:\\as\\file\\"**));**

strcat\_s**(**newDic**,** 899**,** q **+** dataLen **/** **sizeof(**char**));**

**}**

**return** newDic**;**

**}**

同样transW就是把char改成了wchar\_t：

LPCWSTR transW**(**LPCWSTR a0**)** **{**

// 复制a0

wchar\_t a0Buff**[**999**];**

memset**(**a0Buff**,** 0**,** **sizeof(**a0Buff**));**

swprintf\_s**(**a0Buff**,** 999**,** L"%ls"**,** a0**);**

// 分离文件名

size\_t a0BuffLen **=** wcslen**(**a0Buff**);**

size\_t p **=** a0BuffLen **-** 1**;**

**while** **(**p **&&** a0Buff**[**p**]** **!=** L'\\'**)** **--**p**;**

**if** **(**p **!=** 0**)** p**++;**

// 新建新路径

wchar\_t **\***newDic **=** **new** wchar\_t**[**999**];**

memset**(**newDic**,** 0**,** **sizeof(**newDic**));**

// 分类（是否在根目录下）

wchar\_t **\***data **=** L"A Slower Speed of Light\_Data"**;**

size\_t dataLen **=** **sizeof(**L"A Slower Speed of Light\_Data"**);**

wchar\_t **\***q **=** wcsstr**(**a0Buff**,** data**);**

**if** **(**q **==** **NULL)** **{**

// 系统文件，已处理

memcpy**(**newDic**,** a0Buff**,** **sizeof(**a0Buff**));**

**}** **else** **{**

// 重定向

memcpy**(**newDic**,** L"L:\\as\\file\\"**,** **sizeof(**L"L:\\as\\file\\"**));**

wcscat\_s**(**newDic**,** 899**,** q **+** dataLen **/** **sizeof(**wchar\_t**));**

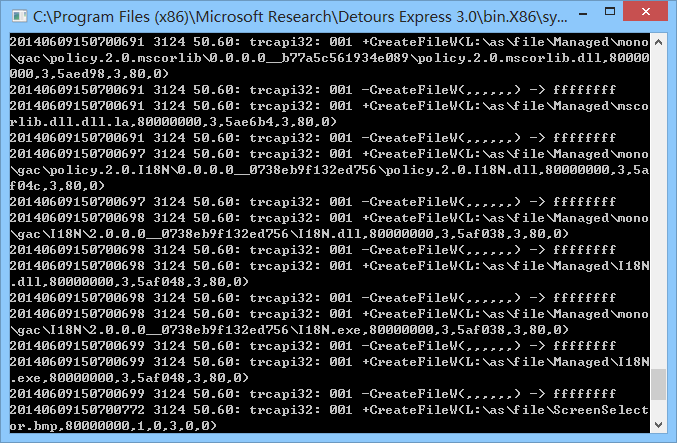
**}**

**return** newDic**;**

**}**

这样CreateFileA和CreateFileW只要把路径一改，调用原来的CreateFile，就hook好了。

可以发现，文件已经被重新定向了，实现虚拟化。



## hook函数FindFirstFile、FindNextFile

与CreateFile类似，使用trans函数就能搞定。

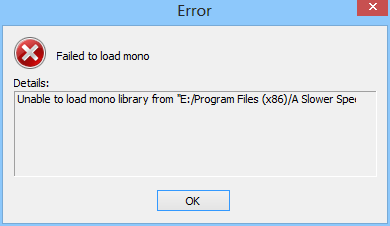
## 测试

全部hook好以后，我进行了测试。

在非detours下游戏进行很流畅，但是在detours后，游戏画面明显变卡。虽然还是可以操作，但是能明显感觉detours减缓了速度。我想这和我将游戏和文件放在两个不同的磁盘有一定关系。

我再将游戏跟目录下文件删除，但是这个游戏自身有一个检查根目录文件的功能，因此不能正常运行。于是我只能删除其内部部分文件，发现游戏可以正常运行，说明其不再依赖根目录下文件。

另外一个角度，我删除L:\as\下的文件，运行后游戏会报错：



至此可以判断，游戏虚拟化完成。

# 遇到的问题

## Detours编译不通过

这个是我调试时间最长，花费经历最多，也最虐心的一个步骤，因为它的存在我的工作根本无法开展。

首先我在写工程前配置VC下detours.lib时就遇到编译不通过的情况，不过后来通过执行VC\bin下的vcvars32.bat文件后终于可以编译。

然后遇到一定问题后我转变用samples中的代码。但是编译samples的问题出现更多，也更广：

* NMAKE : fatal error U1077: “if”: 返回代码“0x1”
* NMAKE : fatal error U1077: “"E:\Program Files (x86)\Microsoft Visual Studio 11.0\VC\bin\nmake.exe"”: 返回代码“0x2”

……

我还以为是32位程序和64为系统不兼容的缘故，特地去安装了32位系统的虚拟机，结果还是类似，根本编译不通过。

最后在通过不知什么原因的情况下终于编译成功了一次，以下是疑似情况：

* 开启管理员权限
* 设定环境变量
* 运行VC下的某个bat
* 给detours.cpp加入#include <windows.h>

……

然后用nmake终于可以编译了，但是编译到一半出现错误：samples中的member.cpp怎么都编译不通过。我最后只能将member的makefile给清空跳过了它的编译才最终生成了bin.X86文件夹，以及里面的各种工具。

## 各种C++问题

由于我是第一次接触wchar\_t，因此很多字符串函数都不知道，在处理LoadLibraryW、CreateFileW等函数的时候非常棘手。最后通过各种搜索引擎+尝试编译才完成。

在使用原来C++字符串函数的时候也会出现问题，比如函数sprintf不能用了，因为新版C++换成了sprintf\_s这个更安全的函数。

其他错误就更多了，比如sizeof(char\*)得到的不是char\*这个字符串的大小而是一个char\*的大小等等。

## 修改\_win32.cpp遇到的问题

\_win32.cpp优点在于已经hook住了所有函数，而这也成为它的缺点：它实在是太大了，以至于修改特别不方便。

一个典型的例子就是一开始它输出了所有被hook的函数，而我只需要跟文件相关的函数，因此我需要将其他函数注释。\_win32.cpp有些函数的输出是单行，有些是多行，无法直接替换，因此我不得不再写一个程序给它加注释。

寻找需要的函数也要经过Ctrl+F几下才能找到，修改复杂度极高。我的detours还安装在系统盘里，不能直接修改，只能通过外面修改好再覆盖，更增加了修改复杂度。

另外，如我2.3中的流程所示，我一整套流程下来其实要很多步骤，这比直接写工程要耗时很多，还不能操作失误。

# 感想

* 深切体会到Windows系统的劣势。各种API，各种复杂的命名，各种不知所云的函数，各种字符转化……在编译过程中就各种出问题，问题还不明确，需要尝试各种方法修修补补，非常恶心
* 第一次使用detours，觉得挺好的一个软件，至少能了解一个程序在干什么，还能控制它，修改它。虽然力量有限，限制无数，但是作为一个体验底层操作系统API的软件，我在使用过程中收获还是很丰富的
* 从一无所知到最终完成大作业，整个过程就是一个艰苦而充满好奇的体验。我在完成的过程中有无能为力的无助，也有运行成功的狂喜，最重要的是体验到了探索的魅力，通过自己的努力一步步获得信息，一步步完成目标。虽然偶尔有问同学一些细节问题，但是最终所有的代码和方法都是自己弄出来的，很有成就感。我也觉得老师布置的这次大作业很有意义，开发了我们的创新性，让我们在探索的过程中进步，而不是硬啃老师灌输的知识

