Intercettare le chiamate alle funzioni di sistema

È descritta la tecnica di intercettazione delle funzioni di sistema con l'ausilio di una libreria denominata Detours, realizzata nei laboratori di Microsoft Research.

di Antonino Calderone

Qualunque programmatore, almeno una volta nella vita, ha avuto l'esigenza di intercettare chiamate a funzioni di sistema per modificarne o estenderne la semantica.

Forse stiamo esagerando, ma è innegabile che intercettare le funzioni di sistema è un esercizio quantomeno interessante.

Se l'esercizio consiste nell'intercettare una chiamata a funzione Win32 in un processo del quale non si abbiano i sorgenti, allora la soluzione non è neanche banale. A meno di non utilizzare *Detours* ([1]).

Questa libreria non è il prodotto del solito *hacker* burlone, ma il risultato del prestigioso *Microsoft Research* (si veda il **Riquadro 1**).

Detours nasce dunque come strumento per la ricerca, per l'analisi del software, per il debugging.

Antonino Calderone acalderone@infomedia.it

Lavora nella R&D per Ericsson, con la qualifica di Technical Development Engineer. Progetta e sviluppa software, device driver e firmware per i sistemi per le telecomunicazioni, sistemi di controllo e automazione. Lo interessano in modo particolare gli internals dei sistemi operativi, la programmazione nello spazio di nucleo, e gli aspetti implementativi del software per il networking, in relazione soprattutto alle problematiche di interprocess communication per sistemi eterogenei, multiprogrammati e real-time.

In particolare i ricercatori di Microsoft mettono in rilievo che oggigiorno è pressoché impossibile disporre del codice sorgente di tutte le librerie che si usano, da cui l'esigenza di disporre di uno strumento come quello che hanno realizzato.

Detours nasce come strumento per la ricerca, per l'analisi del software, per il debugging

Chi conosce il mondo dell'*Open Source* potrebbe essere tentato di sollevare obiezioni alla precedente affermazione. Alle stesse si potrebbe tuttavia rispondere che, pur disponendo dei sorgenti, non è detto che sia agevole modificarli e/o compilarne il relativo eseguibile. Questo ci riporta al punto di partenza.

Installare Detours

Al momento nel quale scriviamo, la versione di *Detours* disponibile è la 2.1. Questa versione supporta le architetture IA-32 e IA-64.

Microsoft Research (http://research.microsoft.com/) è il nome dato da Microsoft a una serie di laboratori di ricerca creati a partire dal 1991 e situati in America, Asia ed Europa, il cui obiettivo è quello di ricercare le tecnologie innovative pensando agli sviluppi che l'informatica avrà nei prossimi anni.
Tra le principali aree di ricerca e i relativi gruppi vi sono Algorithms and Theory, Intelligent Systems, Communication and Collaboration Systems, Machine Learning and Applied Statistics, Cryptography and Anti-Piracy, Security, Digital Geographics, Foundations of Software Engineering.
All'url http://research.microsoft.com/displayArticle.aspx?id=1538 è possibile trovare informazioni sui progetti ai quali i laboratori Microsoft Research stanno lavorando.
Ne riportiamo alcuni esempi.
Al Microsoft Research India, il Digital Geographics sta sviluppando una serie di innovative tecnologie di mapping in stretta collaborazione con il Windows Live Local team (si veda l'URL http://local.live.com/).
Il Computer Vision group nel laboratorio Microsoft Research Cambridge sta lavorando a tecniche per il riconoscimento e la manipolazione di classi di oggetti in un'immagine; in particolare alle tecniche per creare immagini e modelli tridimensionali a partire da immagini bidimensionali.
Al Microsoft Research's Silicon Valley lab, il Distributed Systems group sta sviluppando nuove tecnologie per la fornitura di servizi Internet su larga scala.
I laboratori Microsoft Research collaborano con importanti laboratori di ricerca di prestigiosi enti nel mondo. È possibile accedere a buon numero di pubblicazioni, frutto di queste collaborazioni, partendo dall'URL http://research.microsoft.com/research/pubs/default.aspx

RIQUADRO 1 A proposito di Microsoft Research

Il package di *Detours* è distribuito completo dei sorgenti, che per uno sviluppatore equivalgono, si sa, a documentazione allo stato puro, più un nutrito numero di esempi d'uso e una guida completa che descrive in modo esaustivo le API della libreria.

È possibile prelevare il file d'installazione di Detours (*DetoursExpress.msi*) partendo dal sito all'URL [2]. Il risultato dell'installazione è la creazione di una directory (tipicamente) denominata "*Microsoft Research\Detours Express 2.1*" generata a partire da quella dei programmi all'interno della quale vengono installati la guida alle API in formato *.chm* e i sorgenti.

Dopo l'installazione è possibile compilare la libreria e gli esempi usando un compilatore C/C++ di Microsoft (noi abbiamo usato quello fornito con Visual Studio 6.0), lanciando l'utility *nmake* a partire dalla directory di installazione.

Per quanto riguarda gli aspetti legali, Detours è fornito con una licenza d'uso che ne consente l'impiego gratuito per scopi didattici e di ricerca (per i dettagli vi rimandiamo a un'attenta lettura della licenza stessa). È possibile richiedere altri tipi di licenze (per esempio per scopi commerciali) contattando la stessa Microsoft.

Inside Detours

Detours fornisce un set di API attraverso le quali è possibili intercettare le funzioni invocate da un programma in esecuzione, modificando lo strato di codice responsabile delle chiamate direttamente in memoria.

Detours non ha dunque bisogno di modificare il file eseguibile su disco, ma può operare direttamente nello spazio di indirizzamento del processo, consentendo di alterare la semantica di una qualunque funzione (eventualmente di sistema) invocata dal processo. Questo risultato è raggiunto modificando la normale sequenza di chiamata della funzione.

Attraverso un set di API è possibile intercettare le funzioni invocate da un programma in esecuzione

La tecnica utilizzata consiste nell'individuare e sostituire le prime istruzioni macchina della funzione *target* con un'istruzione di salto incondizionato a una *user function*, o come viene definita nella documentazione, *detour function*. Le istruzioni originali sono preservate in una funzione cosiddetta trampolino (*trampoline function*).

La funzione trampolino incorpora le istruzioni rimosse dalla funzione target, culminando con un salto incondizionato verso quest'ultima, per riprenderne l'esecuzione oltre i byte delle istruzioni macchina rimpiazzati dall'istruzione di salto che realizzano la "deviazione". Quando una funzione viene intercettata il suo codice viene alterato, in modo simile a come illustrato nel pseudo-codice del **Listato 1**.

L'intercettazione di una funzione consiste più precisamente nei seguenti passi:

- la creazione della funzione trampolino che preserva i byte originali della funzione target sovrascritti con l'istruzione di salto verso la *detour function*;
- l'alterazione dei permessi di accesso alla memoria della funzione target;
- la modifica della funzione target;
- il ripristino dei permessi in memoria in modo da consentire l'esecuzione del codice aggiunto e di quello modificato;
- il *flush* dell'*instruction cache* della CPU.

LISTATO 1 Pseudo codice che mostra come venga alterato un processo con Detours

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
TargetFunction:
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; push ebp    ;;; Codice originale
;;; mov ebp,esp ;;; sovrascritto dalla
;;; push ebx    ;;; istruzione
;;; push esi    ;;; jmp DetourFunction
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    jmp DetourFunction

;;; Da qui in avanti riprende il codice
;;; originale della TargetFunction
;;;
TargetFunction+5:   ;;; <- La funzione
     push edi       ;;;    trampolino
     ...            ;;;    riprende da qui
     ret

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TrampolineFunction:
;;; Le istruzioni originali rimosse dalla
;;; TargetFunction sono copiate nella
;;; funzione trampolino
     push ebp
     mov ebp,esp
     push ebx
     push esi

;;; La successiva funzione riprende
;;; l'esecuzione della funzione target
;;; oltre i byte dell'istruzione
;;; di salto verso DetourFunction
;;;
     jmp TargetFunction+5
```

La copia delle istruzioni nella funzione trampolino è fatta in modo da preservare l'integrità delle istruzioni macchina spostate. A priori, infatti, non si sa quali istruzioni possano essere rimpiazzate, e questo implica dover effettuare un'analisi per capire come muovere il codice macchina in modo da non alterarne la semantica. Per ottenere questo risultato Detours ha un vero e proprio disassemblatore pilotato da una tabella, nella quale sono codificati gli *opcode* delle istruzioni macchina x86 e la relativa lunghezza. Le istruzioni rimpiazzate sono riconosciute grazie a questo strumento e modificate in modo consistente.

L'invocazione della funzione target originale è possibile solo attraverso la funzione trampolino

A questo punto ci si potrebbe chiedere se non esistano tecniche (e con quali vantaggi e/o svantaggi) per intercettare chiamate a funzioni di sistema alternative a quella illustrata. Gli stessi ricercatori di Microsoft ne forniscono una breve panoramica in [1].

La prima tecnica che può venire in mente, la soluzione banale, consiste nel fare un wrapping della funzione di sistema: operazione semplice a patto di disporre dei sorgenti, senza i quali non rimane che modificare il codice binario dell'eseguibile. In questo caso uno dei principali problemi, ma non il solo, è quello di individuare il codice delle chiamate da modificare.

Una soluzione elegante consiste invece nell'alterare le *entry* della *import table*. Sfortunatamente quest'approccio non consente di modificare le chiamate a funzioni ottenute con il caricamento della DLL per mezzo della funzione *LoadLibrary*.

Un'altra tecnica utilizzabile fa uso dei

breakpoint, ma rispetto quella usata in Detours ha almeno due svantaggi: il primo richiede che un secondo processo agganci e gestisca le *debug exception*; il secondo, le prestazioni sono estremamente penalizzate (in [1] sono riportati i risultati di test comparativi tra le varie tecniche viste).

Detours in azione

Se quanto discusso finora non risultasse del tutto chiaro (ce ne scusiamo in tal caso), riteniamo che un esempio d'uso possa quantomeno dimostrare che dietro alla complessità architetturale si nasconde uno strumento il cui utilizzo risulta estremamente semplice.

Prenderemo ora in esame uno degli esempi disponibili nel *package* di Detours, analizzandolo in dettaglio.

L'esempio in oggetto si chiama *simple* ed è contenuto nell'omonima directory a partire da quella che contiene i sorgenti di tutti gli esempi (denominata *samples*).

Codice alla mano, illustreremo l'utilizzo delle primitive di Detours che possono essere impiegate per intercettare una chiamata di sistema (nella fattispecie la funzione *Sleep*).

Gli attori sono una *console application* denominata *sleep5* e la DLL: *simple.dll*. Il codice sorgente di questa DLL è contenuto nel file *simple.cpp*. Quello del programma cavia (*sleep5.exe*) è contenuto invece nel file *sleep5.cpp*. I *binary* degli esempi vengono installati nella directory *bin* (stesso livello di *samples*).

Il programma *sleep5* si limita a invocare la funzione *Sleep*. Compilandolo e mandandolo in esecuzione senza parametri si ottiene un output simile al seguente:

```
sleep5.exe: Starting.
```

LISTATO 2 Frammento di codice sorgente della libreria simple.dll

```
//////////////////////////////////////////
//
//  Detours Test Program (simple.cpp of simple.dll)
//
//  Microsoft Research Detours Package, Version 2.1.
//  Copyright (c) Microsoft Corporation.  All rights reserved.
//
//  This DLL will detour the Windows Sleep API so that TimedSleep function
//  gets called instead. TimedSleep records the before and after times,
//  and calls the real Sleep API through the TrueSleep function pointer.
//

...

static VOID (WINAPI * TrueSleep)(DWORD dwMilliseconds) = Sleep;

VOID WINAPI TimedSleep(DWORD dwMilliseconds) { ... }

BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserved)
{
...
  if (dwReason == DLL_PROCESS_ATTACH) {
    DetourRestoreAfterWith();

    DetourTransactionBegin();

    DetourUpdateThread(GetCurrentThread());
    DetourAttach(&(PVOID&)TrueSleep, TimedSleep);

    error = DetourTransactionCommit();

    if (error == NO_ERROR) ...
   ...
  }
  else if (dwReason == DLL_PROCESS_DETACH) {
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());
    DetourDetach(&(PVOID&)TrueSleep, TimedSleep);
    error = DetourTransactionCommit();

    printf("simple.dll: Removed Sleep() (result=%d), slept %d ticks.\n",
                                                          error, dwSlept);
    fflush(stdout);
  }
}
```

```
sleep5.exe: Done sleeping.
```

Tra i due messaggi di output è eseguita la funzione *Sleep* che sospende il processo per 5 secondi.

La DLL *simple.dll*, se iniettata nello spazio di indirizzamento di *sleep5* - operazione che può essere effettuata mediante l'utility di Detours *withdll* - altera la funzione *Sleep* modificandone la semantica.

Provando a dare il seguente comando da console

```
withdll /d:"simple.dll" sleep5.exe
```

si dovrebbe ottenere un output come quello riportato di seguito:

```
withdll.exe: Starting: `sleep5.exe'
withdll.exe:   with `simple.dll'
withdll.exe:   marked by `detoured.dll'
simple.dll: Starting.
simple.dll: Detoured Sleep().
sleep5.exe: Starting.
sleep5.exe: Done sleeping.
simple.dll: Removed Sleep() (result=0),
                            slept 5000 ticks.
```

Le chiamate alle API usate per agganciare e deviare la funzione *Sleep*, e più in generale qualunque altra primitiva di sistema, sono illustrate in un frammento di codice tratto dai sorgenti di *simple.dll* e riportati nel **Listato 2**.

In **Figura 1** sono invece rappresentate le sequenze delle chiamate effettuate eseguendo *sleep5* senza e con la DLL *simple*.

I PE è un formato di file derivante dal formato Unix COFF

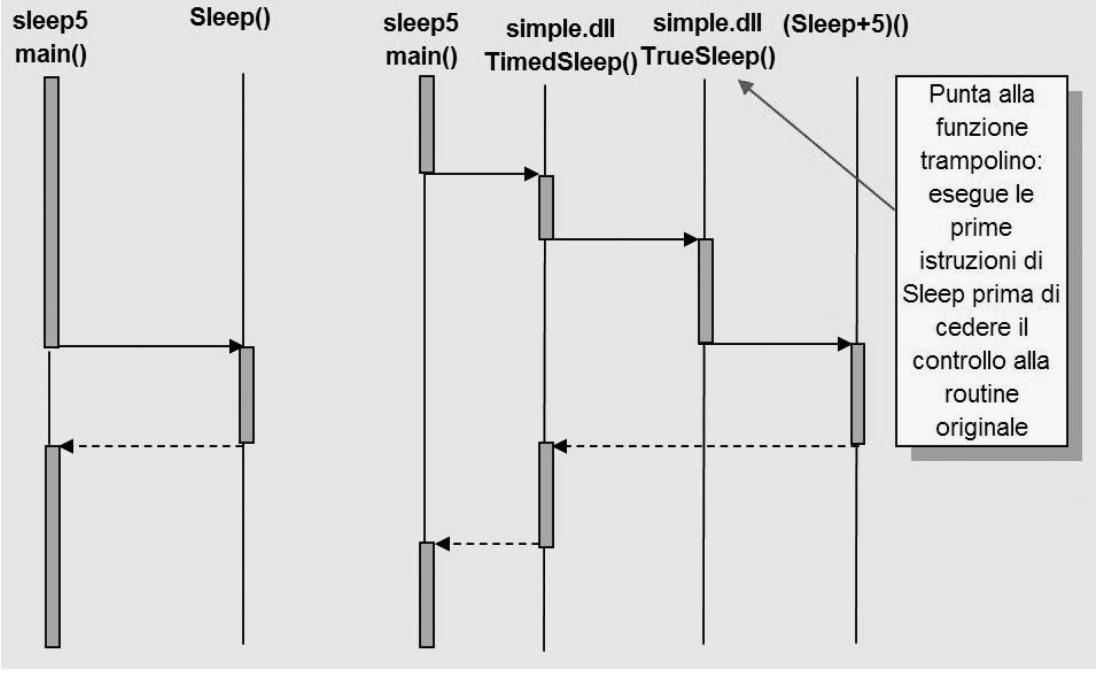
La funzione *Sleep* è intercettata quando la *DLL* viene collegata al processo chiamante *sleep5*.

Il codice è eseguito all'interno della funzione *DllMain*, invocata con il parametro *dwReason* uguale a *DLL_PROCESS_ATTACH*.

In maniera simmetrica lo stato originale del processo è ripristinato quando la *DLL* viene "scollegata" (*dwReason* uguale a *DLL_PROCESS_DETACH*).

La funzione *DetourAttach* alloca e prepara lo spazio in memoria per ospitare il codice della

FIGURA 1 Sequenza delle chiamate di Sleep del programma sleep5.exe senza e con la DLL simple.dll



funzione trampolino, ed è inserita all'interno di una cosiddetta *detour transaction*, delimitata dalle funzioni *DetourTransactionBegin* e *DetourTransactionCommit*. Le modifiche al codice della funzione target sono effettuate solo al momento del *commit* della *detour transaction*, cioè quando è invocata la funzione *DetourTransactionCommit*.

La funzione *DetourAttach* ha il seguente prototipo:

```
LONG DetourAttach( PVOID * ppPointer,
                                PVOID pDetour );
```

Il primo parametro è l'indirizzo a un puntatore a funzione che deve essere inizializzato con l'indirizzo della funzione da intercettare. Il secondo deve puntare la *detour function* verso la quale deviare la chiamata della funzione intercettata.

Dal momento che la funzione target punta alla *detour function*, l'invocazione della funzione target originale è possibile solo attraverso la funzione trampolino: il puntatore originale alla funzione *Sleep* assegnato alla variabile *TrueSleep* viene aggiornato perché punti proprio a questa funzione.

Osserviamo che prima di iniziare la transazione, è chiamata la funzione *DetourRestoreAfterWith* che assolve il compito di ripristinare i contenuti in memoria della *import table*, alterata dalla funzione *DetourCreateProcessWithDll*, impiegata dall'utility *withdll* per iniettare *simple.dll* nel processo che esegue il programma *sleep5*.

L'implementazione della funzione *TimedSleep*, verso la quale la chiamata alla funzione *Sleep* è *deviata*, è mostrata nel **Listato 3**. Si noti che tale funzione chiama la funzione *Sleep* attraverso il puntatore a funzione *TrueSleep* che punta in quel momento alla funzione trampolino. Come già detto, quest'ultima esegue le istruzioni spostate dalla funzione target originale e in ultimo riprende, con un salto incondizionato, l'esecuzione della funzione originale stessa oltre i byte dell'istruzione di salto iniziale.

LISTATO 3 Implementazione della funzione TimedSleep fatta in simple.dll

```
static LONG dwSlept = 0;
...

VOID WINAPI TimedSleep(DWORD dwMilliseconds)
{
  DWORD dwBeg = GetTickCount();
  TrueSleep(dwMilliseconds);
  DWORD dwEnd = GetTickCount();

  InterlockedExchangeAdd(&dwSlept,
                                  dwEnd-dwBeg);
}
```

Altra operazione preliminare è assolta dalla funzione *DetourUpdateThread*, che ha lo scopo di informare Detours dei thread in esecuzione: questi devono essere sospesi da Detours prima che venga alterato il codice del processo in memoria. In realtà l'operazione mostrata nell'esempio è superflua poiché il thread corrente chiaramente non può essere sospeso: un opportuno controllo all'interno della funzione *DetourUpdateThread* si accerta sempre che l'handle passato come argomento non sia proprio quello del thread corrente, evitando di intraprendere azioni per questo.

Inoltre va osservato che se le transazioni sono eseguite all'interno della *DllMain* della libreria iniettata (come nell'esempio preso in considerazione), in questo contesto è assicurato che non vi siano altri thread in esecuzione.

Quando la libreria *simple.dll* è "sganciata" dal processo è eseguita la transazione che consente di ripristinare lo stato originale della funzione Sleep. In questo caso è usata la funzione *DetourDetach*.

Aggiungere i payload e modificare le import table delle DLL

Anche se la caratteristica principale di Detours è quella di intercettare le funzioni di sistema, non meno interessanti sono altre funzionalità messe a disposizione in questo *package*.

Con Detours è possibile collegare segmenti dati (chiamati *payload*) a un processo e si possono modificare le *import table* direttamente su file; nella nuova *import table* possono essere aggiunti o ridenominati i riferimenti a DLL e funzioni. È interessante notare che la tabella originale viene preservata in modo che tale operazione sia reversibile.

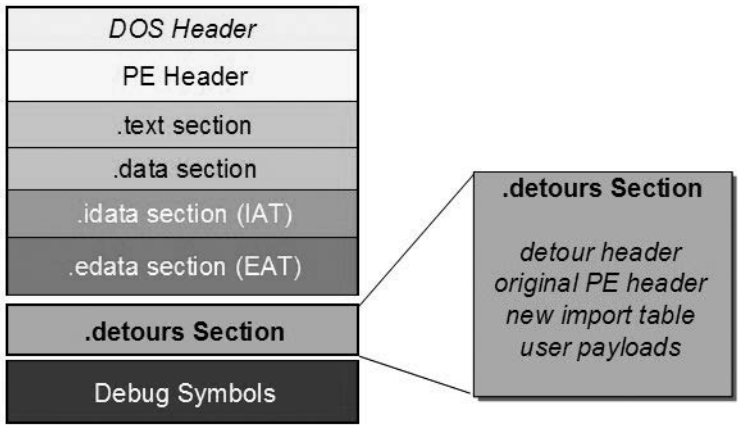
Le due caratteristiche appena elencate sono ottenute da Detours agendo sulla struttura del file binario, organizzata nel formato noto come *Portable Executable* (PE).

Il PE è un formato di file derivante dal formato Unix COFF (*Common Object File Format*). Sono PE gli eseguibili di Windows (.EXE), i file oggetto (.OBJ), le librerie a link dinamico (.DLL), i device driver (.SYS).

Questo formato è usato dalle diverse versioni di Windows, da cui il termine *portable*.

Esistono estensioni del formato PE, a supporto delle nuove tecnologie Microsoft come il *.NET PE*, la versione a 64-bit denominata *PE32+* o *PE+*, e caratterizzazioni per i sistemi che derivano da *Windows CE* (in [3] il riferimento al documento *Microsoft Portable Executable and Common Object File Format Specification*).

FIGURA 2 Schema di un eseguibile Win32 modificato da Detours



Un file binario di Windows è costituito da un header compatibile con il DOS seguito da un header nel formato PE.

Il PE contiene le informazioni usate dal *loader* del S/O per caricare ed eseguire il codice. All'interno del PE sono presenti i riferimenti alle DLL utilizzate, le *import* ed *export address table*, i dati delle risorse, i dati della *thread-local storage* (TLS).

Un eseguibile mappato in memoria viene suddiviso in sezioni. In un file eseguibile di norma esistono: una sezione detta *.text*, che contiene il codice del programma; una sezione dati (*.data*) che contiene i dati inizializzati; la *import table* che mantiene i riferimenti alle DLL utilizzate dal processo e le relative funzioni; la *export table* che riporta le funzioni esportate dal modulo (si pensi a una DLL) e in ultimo possono esservi i simboli di *debug*.

In generale le sezioni che seguono i due header iniziali sono tutte opzionali: dipendono dalla destinazione ultima del file binario.

Per modificare un file binario nel formato PE, Detours crea una nuova sezione denominata *.detours* tra la *export table* e i *simboli di debug* (che devono trovarsi sempre nella parte finale del file). La nuova sezione creata da Detours è usata per contenere una copia originale dell'header del PE. I *payload* inseriti nel file vengono posizionati dopo la sezione *.detours*. Quindi i simboli di debugging vengono "spostati" alla fine del file (come schematizzato in **Figura 2**).

Quale che sia la ragione per voler modificare un PE, Detours consente di farlo in modo estremamente efficace. Esiste un set di API utilizzabili per modificare le *import tables* (*DetourBinaryEditImports*) e aggiungere *payload* (*DetourBinaryEnumeratePayloads*), alle quali si aggiungono altre interessanti funzionalità "minori".

Lasciamo al lettore il piacere di scoprire quali.

Conclusioni

Abbiamo descritto quello che riteniamo uno strumento potente e allo stesso tempo semplice da usare, ricco di molte caratteristiche interessanti (vi abbiamo parlato in effetti solo delle principali).

Spesso il modo migliore di documentarsi è analizzare il codice (tanto meglio se non debba farsi in *assembly*). In questo senso i sorgenti di *Detours* sono una valida fonte di documentazione tecnica.

Se vi abbiamo almeno incuriositi, vi consigliamo di leggere gli articoli e i documenti ai quali ci siamo riferiti, e ovviamente scaricare, installare e usare *Detours*.

BIBLIOGRAFIA & RIFERIMENTI

[1] G. Hunt, D. Brubacher - "Detours: Binary Interception of Win32 Functions", Microsoft Research
[2] http://research.microsoft.com/sn/detours
[3] http://www.microsoft.com/whdc/system/platform/firmware/PECOFFdwn.mspx