Detours: Binary Interception of Win32 Functions

Galen Hunt and Doug Brubacher
Microsoft Research
One Microsoft Way
Redmond, WA 98052
detours@microsoft.com
http://research.microsoft.com/sn/detours

Abstract

Innovative systems research hinges on the ability to easily instrument and extend existing operating system and application functionality. With access to appropriate source code, it is often trivial to insert new instrumentation or extensions by rebuilding the OS or application. However, in today's world of commercial software, researchers seldom have access to all relevant source code.

We present Detours, a library for instrumenting arbitrary Win32 functions on x86 machines. Detours intercepts Win32 functions by re-writing target *function images. The Detours package also contains utilities to attach arbitrary DLLs and data segments (called* payloads*) to any Win32 binary.*

While prior researchers have used binary rewriting to insert debugging and profiling instrumentation, to our knowledge, Detours is the first package on any platform to logically preserve the un-instrumented target function (callable through a trampoline) as a subroutine for use by the instrumentation. Our unique trampoline design is crucial for extending existing binary software.

We describe our experiences using Detours to create an automatic distributed partitioning system, to instrument and analyze the DCOM protocol stack, and to create a thunking layer for a COM-based OS API. Micro-benchmarks demonstrate the efficiency of the Detours library.

Published in *Proceedings of the 3rd USENIX Windows NT Symposium*. Seattle, WA, July 1999.

1. Introduction

Innovative systems research hinges on the ability to easily instrument and extend existing operating system and application functionality whether in an application, a library, or the operating system DLLs. Typical reasons to intercept functions are to add functionality, modify returned results, or insert instrumentation for debugging or profiling. With access to appropriate source code, it is often trivial to insert new instrumentation or extensions by rebuilding the OS or application. However, in today's world of commercial development and binary-only releases, researchers seldom have access to all relevant source code.

Detours is a library for intercepting arbitrary Win32 binary functions on *x86* machines. Interception code is applied dynamically at runtime. Detours replaces the first few instructions of the *target function* with an unconditional jump to the user-provided *detour function*. Instructions from the target function are preserved in a *trampoline function*. The trampoline function consists of the instructions removed from the target function and an unconditional branch to the remainder of the target function. The detour function can either replace the target function or extend its semantics by invoking the target function as a subroutine through the trampoline.

Detours are inserted at execution time. The code of the target function is modified in memory, not on disk, thus facilitating interception of binary functions at a very fine granularity. For example, the procedures in a DLL can be detoured in one execution of an application, while the original procedures are not detoured in another execution

running at the same time. Unlike DLL re-linking or static redirection, the interception techniques used in the Detours library are guaranteed to work regardless of the method used by application or system code to locate the target function.

While others have used binary rewriting for debugging and to inline instrumentation, Detours is a general-purpose package. To our knowledge, Detours is the first package on any platform to logically preserve the un-instrumented target function as a subroutine callable through the trampoline. Prior systems logically prepended the instrumentation to the target, but did not make the original target's functionality available as a general subroutine. Our unique trampoline design is crucial for extending existing binary software.

In addition to basic detour functionality, Detours also includes functions to edit the DLL import table of any binary, to attach arbitrary data segments to existing binaries, and to inject a DLL into either a new or an existing process. Once injected into a process, the instrumentation DLL can detour any Win32 function, whether in the application or the system libraries.

The following section describes how Detours works. Section 0 outlines the usage of the Detours library. Section 4 describes alternative function-interception techniques and presents a micro-benchmark evaluation of Detours. Section 5 details the usage of Detours to produce distributed applications from local applications, to quantify DCOM overheads, to create a thunking layer for a new COM-based Win32 API, and to implement first chance exception handling. We compare Detours with related work in Section 6 and summarize our contributions in Section 7.

2. Implementation

Detours provides three important sets of functionality: the ability to intercept arbitrary Win32 binary functions on *x86* machines, the ability to edit the import tables of binary files, and the ability to attach arbitrary data segments to binary files. We will describe the implementation of each of these functionalities.

2.1. Interception of Binary Functions

The Detours library facilitates the interception of function calls. Interception code is applied dynamically at runtime. Detours replaces the first few instructions of the *target function* with an unconditional jump to the user-provided *detour function*. Instructions from the target function are preserved in a *trampoline function*. The trampoline consists of the instructions removed from the target function and an unconditional branch to the remainder of the target function.

When execution reaches the target function, control jumps directly to the user-supplied detour function. The detour function performs whatever interception *preprocessing* is appropriate. The detour function can return control to the *source* function or it can call the trampoline function, which invokes the target function without interception. When the target function completes, it returns control to the detour function. The detour function performs appropriate *postprocessing* and returns control to the source function. Figure 1 shows the logical flow of control for function invocation with and without interception.

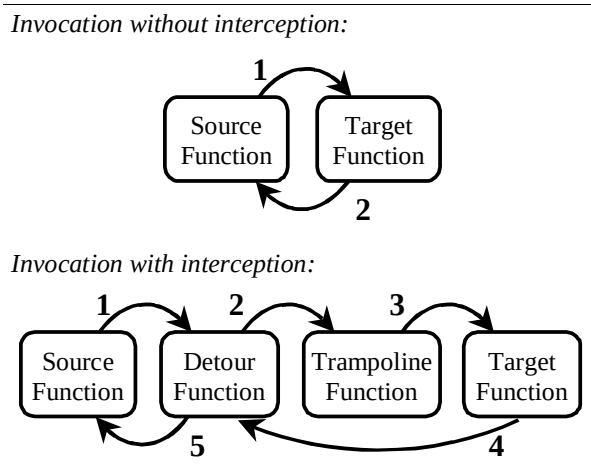

Figure 1. Invocation with and without interception.

The Detours library intercepts target functions by rewriting their in-process binary image. For each target function, Detours actual rewrites two functions: the target function and the matching trampoline function. The trampoline function can be allocated either dynamically or statically. A statically allocated trampoline always invokes the target function without the detour. Prior to insertion of a detour, the static trampoline contains a single jump to the target. After insertion, the trampoline contains the initial

instructions from the target function and a jump to the remainder of the target function.

Statically allocated trampolines are extremely useful for instrumentation programmers. For example, in Coign [7], invoking the `Coign_CoCreateInstance` trampoline is equivalent to invoking the original `CoCreateInstance` function without instrumentation. Coign internal functions can call `Count_CoCreateInstance` at any time to create a new component instance without concern for whether or not the original function has been rerouted with a detour.

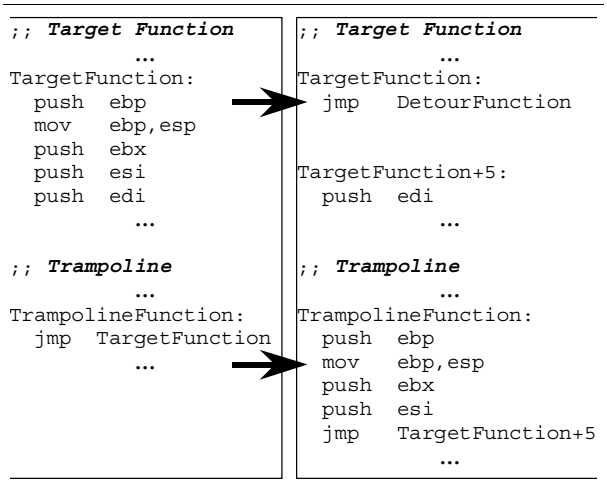

Figure 2. Trampoline and target functions, before and after insertion of the detour (left and right).

Figure 2 shows the insertion of a detour. To detour a target function, Detours first allocates memory for the dynamic trampoline function (if no static trampoline is provided) and then enables write access to both the target and the trampoline. Starting with the first instruction, Detours copies instructions from the target to the trampoline until at least 5 bytes have been copied (enough for an unconditional jump instruction). If the target function is fewer than 5 bytes, Detours aborts and returns an error code. To copy instructions, Detours uses a simple table-driven disassembler. Detours adds a jump instruction from the end of the trampoline to the first non-copied instruction of the target function. Detours writes an unconditional jump instruction to the detour function as the first instruction of the target function. To finish, Detours restores the original page permissions on both the target and trampoline functions and flushes the CPU instruction cache with a call to `FlushInstructionCache`.

2.2. Payloads and DLL Import Editing

While a number of tools exist for editing binary files [10, 12, 13, 17], most systems research doesn't require such heavy-handed access to binary files. Instead, it is often sufficient to add an extra DLL or data segment to an application or system binary file. In addition to detour functions, the Detours library also contains fully reversible support for attaching arbitrary data segments, called *payloads*, to Win32 binary files and for editing DLL import tables.

Figure 3 shows the basic structure of a Win32 Portable Executable (PE) binary file. The PE format for Win32 binaries is an extension of COFF (the Common Object File Format). A Win32 binary consists of a DOS compatible header, a PE header, a text section containing program code, a data section containing initialized data, an import table listing any imported DLLS and functions, an export table listing functions exported by the code, and debug symbols. With the exception of the two headers, each of the other sections of the file is optional and may not exist in a given binary.

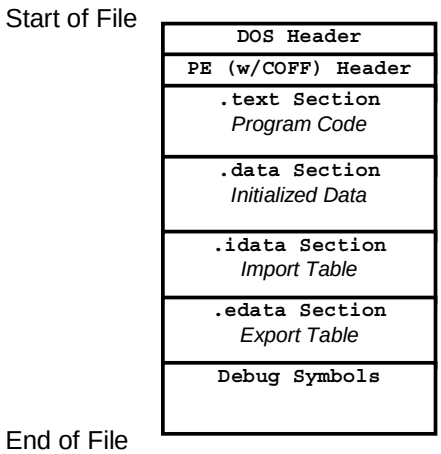

Figure 3. Format of a Win32 PE binary file.

To modify a Win32 binary, Detours creates a new `.detours` section between the export table and the debug symbols. Note that debug symbols must always reside last in a Win32 binary. The new section contains a detours header record and

a copy of the original PE header. If modifying the import table, Detours creates the new import table, appends it to the copied PE header, then modifies the original PE header to point to the new import table. Finally, Detours writes any user payloads at the end of the `.detours` section and appends the debug symbols to finish the file. Detours can reverse modifications to the Win32 binary by restoring the original PE header from the `.detours` section and removing the `.detours` section. Figure 4 shows the format of a Detours-modified Win32 binary.

Creating a new import table serves two purposes. First, it preserves the original import table in case the programmer needs to reverse all modifications to the Win32 file. Second, the new import table can contain renamed import DLLs and functions or entirely new DLLs and functions. For example, Coign [7] uses Detours to insert an initial entry for `coignrte.dll` into each instrumented application. As the first entry in the applications import table, `coignrte.dll` always is the first DLL to run in the application's address space.

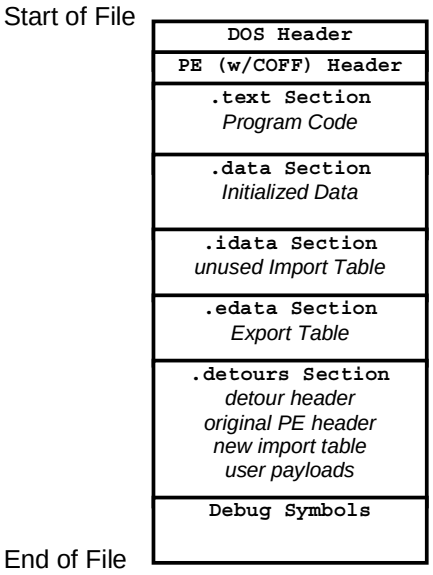

Figure 4. Format of a Detours-modified binary file.

Detours provides functions for editing import tables, adding payloads, enumerating payloads, removing payloads, and rebinding binary files. Detours also provides routines for enumerating the binary files mapped into an address space and locating payloads within those mapped binaries. Each payload is identified by a 128-bit globally unique identifier (GUID). Coign uses Detours to attach per-application configuration data to application binaries.

In cases where instrumentation need be inserted into an application without modifying binary files, Detours provides functions to inject a DLL into either a new or an existing process. To inject a DLL, Detours writes a `LoadLibrary` call into the target process with the `VirtualAllocEx` and `WriteProcessMemory` APIs then invokes the call with the `CreateRemoteThread` API.

3. Using Detours

The code fragment in Figure 5 illustrates the usage of the Detours library. User code must include the `detours.h` header file and link with the `detours.lib` library.

```
#include <windows.h>
#include <detours.h>

VOID (*DynamicTrampoline)(VOID) = NULL;

DETOUR_TRAMPOLINE(
  VOID WINAPI SleepTrampoline(DWORD),
  Sleep
);

VOID WINAPI SleepDetour(DWORD dw)
{
  return SleepTrampoline(dw);
}

VOID DynamicDetour(VOID)
{
  return DynamicTrampoline();
}

void main(void)
{
  VOID (*DynamicTarget)(VOID) = SomeFunction;

  DynamicTrampoline
    =(FUNCPTR)DetourFunction(
       (PBYTE)DynamicTarget,
       (PBYTE)DynamicDetour);

  DetourFunctionWithTrampoline(
    (PBYTE)SleepTrampoline,
    (PBYTE)SleepDetour);

  // Execute the remainder of program.

  DetourRemoveTrampoline(SleepTrampoline);
  DetourRemoveTrampoline(DynamicTrampoline);
}
```

Figure 5. Sample Instrumentation Program.

Trampolines may be created either statically or dynamically. To intercept a target function with a static trampoline, the application must create the trampoline with the `DETOUR_TRAMPOLINE` macro. `DETOUR_TRAMPOLINE` takes two arguments: the prototype for the static trampoline and the name of the target function.

Note that for proper interception the prototype, target, trampoline, and detour functions must all have exactly the same call signature including number of arguments and calling convention. It is the responsibility of the detour function to copy arguments when invoking the target function through the trampoline. This is intuitive as the target function is just a subroutine callable by the detour function.

Using the same calling convention insures that registers will be properly preserved and that the stack will be properly aligned between detour and target functions.

Interception of the target function is enabled by invoking the `DetourFunctionWithTrampoline` function with two arguments: the trampoline and the pointer to the detour function. The target function is not given as an argument because it is already encoded in the trampoline.

A dynamic trampoline is created by calling `DetourFunction` with two arguments: a pointer to the target function and a pointer to the detour function. `DetourFunction` allocates a new trampoline and inserts the appropriate interception code in the target function.

Static trampolines are extremely easy to use when the target function is available as a link symbol. When the target function is not available for linking, a dynamic trampoline can be used. Often a function pointer to the target function can be acquired from a second function. For those times, when a pointer to the target function is not readily available, `DetourFindFunction` can find the pointer to a function when it is either exported from a known DLL, or if debugging symbols are available for the target function's binary[1].

`DetourFindFunction` accepts two arguments, the name of the binary and the name of the function. `DetourFindFunction` returns either a valid pointer to the function or `NULL` if the symbol for the function could not be found. `DetourFindFunction` first attempts to locate the function using the Win32 `LoadLibrary` and `GetProcAddress` APIs. If the function is not found in the export table of the DLL, `DetourFindFunction` uses the `ImageHlp` library to search available debugging symbols. The function pointer returned by `DetourFindFunction` can be given to `DetourFunction` to create a dynamic trampoline.

Interception of a target function can be removed by invoking the `DetourRemoveTrampoline` function.

Note that because the functions in the Detours library modify code in the application address space, it is the programmer's responsibility to ensure that no other threads are executing in the address space while a detour is inserted or removed. An easy way to insure single-threaded execution is to call functions in the Detours library from a `DllMain` routine.

4. Evaluation

Several alternative techniques exist for intercepting function calls. Alternative interception techniques include:

Call replacement in application source code. Calls to the target function are replaced with calls to the detour function by modifying application source code. The major drawback of this technique is that it requires access to source code.

Call replacement in application binary code. Calls to the target function are replaced with calls to the detour function by modifying application binaries. While this technique does not require source code, replacement in the application binary does require the ability to identify all applicable call sites. This requires substantial symbolic information that is not generally available for binary software.

DLL redirection. If the target function resides in a DLL, the DLL import entries in the binary can be modified to point to a detour DLL. Redirection to the detour DLL can be achieved by either replacing the name of the original DLL in the import table before load time or replacing the function addresses in the indirect import jump

[1] Microsoft ships debugging symbols for the entire Windows NT operation system as part of the retail release. These symbols can be found in the `\support\symbols` directory on the OS distribution media.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.